Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

_____Інститут електроенергетики_____
(навчально-науковий інститут)

_____факультет інформаційних технологій_____
(факультет)

Кафедра_____інформаційних технологій та комп'ютерної інженерії_____
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи ступеня __бакалавра_

(бакалавра, магістра)

Здобувача вищої освіти_____Полішка Антона Мирославовича_____
(ПІБ)

академічної групи_____126-21-2_____
(шифр)

спеціальності _____126 «Інформаційні системи та технології»_____
(код і назва спеціальності)

спеціалізації за освітньо-професійною (освітньо-науковою) програмою _____
(за наявності)

_____
(офіційна назва)

на тему _____Розробка високонавантаженого інтернет-каталогу товарів_____
_____з опрацюванням задач пошуку та фільтрації товарів під Android_____
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи				
розділів:				

Рецензент				

Нормоконтролер	проф. Коротенко Г. М.			

Дніпро
2025

ЗАТВЕРДЖЕНО:
завідувач кафедри
інформаційних технологій та комп'ютерної інженерії
(повна назва)

_____ __В.В. Гнатушенко__
(підпис) (ініціали та прізвище)

«_____» _____20___року

ЗАВДАННЯ

на кваліфікаційну роботу
ступеня _бакалавра_
(бакалавра, магістра)

здобувача вищої освіти ____Полішко А.М.____ **академічної групи**___126-21-2____
(прізвище та ініціали) (шифр)

спеціальності_____126 «Інформаційні системи та технології»_____

спеціалізації за освітньою-професійною програмою _____
(за наявності)

на тему_____Розробка високонавантаженого інтернет-каталогу товарів_____
_____ з опрацюванням задач пошуку та фільтрації товарів під Android_____

затверджену наказом ректора НТУ «Дніпровська політехніка» від _____ № _____

Розділ	Зміст	Термін виконання
Розділ 1. Аналіз стану області рішення завдання	Огляд існуючих аналогів, літературних джерел, статтей, огляд інформаціних технологій та засобів вирішення завдання, методи кешування даних, архітектурні патерни програмного забезпечення	28.04.2025 – 14.05.2025
Розділ 2. Проєктні рішення	1. Проєктування архітектури високонавантажених Android-застосунків. 2. Опис ключових алгоритмів та вмісту модулів проєкту. 3. Опис взаємодії клієнта та сервера мобільного додатку. 4. Створення інтерфейсу користувача для мобільного застосунку 5. Опис результатів тестування застосунку та подальший аналіз результатів роботи програми.	15.05.2025 – 15.06.2025

Завдання видано _____ _____
(підпис керівника) (ініціали та прізвище)

Дата видачі _____

Дата подання до екзаменаційної комісії _____

Прийнято до виконання _____ _____Полішко А.М._____
(підпис здобувача вищої освіти)

(ініціали та прізвтще)

РЕФЕРАТ

Пояснювальна записка: 83 с., 21 рис., 4 табл., 2 додатки, 23 джерел.

ANDROID-ЗАСТОСУНОК, MVC, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, МОНОЛІТНА АРХІТЕКТУРА, API, ФРЕЙМВОРК JAVA SPRING, MYSQL, HTTP.

Об'єкт кваліфікаційної роботи: клієнт-серверна архітектура високонавантаженого інтернет-каталогу товарів.

Предмет кваліфікаційної роботи: взаємодія Android-застосунку з backend на Java Spring.

Мета роботи: розробка високонавантаженого мобільного інтернет-каталог товарів, що функціонує на платформі Android, та архітектура якого оптимізована для роботи в умовах високих навантажень і забезпечує ефективну реалізацію задач пошуку та фільтрації.

У вступі наведено інформацію про стан проблеми, здійснено аналіз відомих аналогів, а також обґрунтовано актуальність теми, підкреслюючи, що від швидкості та точності пошуку й фільтрації на платформі Android безпосередньо залежить конверсія та успішність бізнесу. Доведено, що дослідження та застосування сучасних архітектурних патернів, технологій кешування та асинхронної обробки є важливою та практично значущою задачею в галузі інформаційних систем та технологій.

У першому розділі наведені основні відомості про рівень розвитку інформаційних систем та технологій – зокрема, виконано огляд існуючих високонавантажених інтернет-каталогів (Prom, Rozetka, OLX) та їхніх архітектурних підходів, таких як використання пошукових рушіїв та багаторівневого кешування. Проаналізовано ключові літературні джерела та статті, що стосуються побудови масштабованих, надійних і відмовостійких систем. Обґрунтовано актуальність розробки під платформу Android на основі аналізу статистичних даних ринку операційних систем. Крім того, наведено огляд та порівняння релевантних інструментів розробки, мов програмування,

систем управління базами даних, архітектурних шаблонів програмного забезпечення (SOA, моноліт) та методів кешування даних.

У другому розділі наведена проєктна складова вирішення завдання. Описані вимоги до програмної архітектури , ключові алгоритми обробки високого навантаження , процес створення користувацького інтерфейсу , а також програмування клієнтської та серверної частин. Розглянута обрана клієнт-серверна архітектура , архітектурні шаблони MVC для клієнта та монолітна архітектура для сервера , модель даних, реалізована в СУБД MySQL , та принципи взаємодії компонентів системи. Подані результати тестування та роботи розробленого програмного забезпечення, що включають опис методології , планування та обробку результатів тестування, а також інструкцію для користувача з описом основних функцій.

У висновках узагальнені результати роботи розробленого програмного забезпечення. Надані пропозиції щодо подальшого розвитку проєкту, зокрема реалізації повноцінного прототипу, інтеграції спеціалізованого пошукового рушія для складних запитів та розробки системи персоналізованих рекомендацій.

Практичне значення кваліфікаційної роботи полягає у створенні архітектурного рішення, що може слугувати прототипом та шаблоном для розробки комерційних високонавантажених мобільних додатків у сфері e-commerce.

Розроблене програмне забезпечення може бути запроваджено у компаніях, що займаються розробкою мобільних додатків для інтернет-магазинів та маркетплейсів, для підвищення продуктивності, надійності та покращення користувацького досвіду їхніх продуктів.

ABSTRACT

Explanatory note: 83 p., 21 fig., 4 tables, 2 appendices, 23 sources.

ANDROID APPLICATION, MVC, CLIENT-SERVER ARCHITECTURE, MONOLITHIC ARCHITECTURE, REST API, JAVA SPRING FRAMEWORK, MYSQL, HTTP.

The object of the qualification work: the client-server architecture of a high-load online product catalog.

The subject of the qualification work: the interaction of an Android application with a backend developed using Java Spring.

The aim of the work: the development of a high-load mobile product catalog that operates on the Android platform, with an architecture optimized to operate under high-load conditions and to provide an effective implementation of search and filtering tasks.

The introduction outlines the problem statement, analyzes well-known analogues, and substantiates the relevance of the topic, emphasizing that conversion rates and business success directly depend on the speed and accuracy of search and filtering on the Android platform. It is proven that the research and application of modern architectural patterns, caching technologies, and asynchronous processing are an important and practically significant task in the field of information systems and technologies.

The first chapter provides an analysis of the problem domain. It reviews existing high-load e-commerce platforms (Prom, Rozetka, OLX) and their architectural solutions, such as search engines and multi-level caching. Key literature and articles concerning the development of scalable, reliable, and fault-tolerant systems are analyzed. The relevance of developing for the Android platform is justified based on market statistics. Furthermore, an overview and comparison of relevant development tools, programming languages, database management systems, software architectural patterns (SOA, monolith), and data caching methods are provided.

The second chapter describes the project implementation. It details the software architecture requirements, key algorithms for high-load processing, the user interface design process, and the programming of both client and server components. The chosen client-server architecture is examined, including the MVC pattern for the client and a monolithic architecture for the server, the data model implemented in MySQL, and the principles of component interaction. The results of software testing are presented, including the methodology, planning, and results processing, along with a user manual describing the main functions.

The conclusions summarize the results of the work. Proposals for the further development of the project are provided, including the implementation of a full-fledged prototype, the integration of a specialized search engine for complex queries, and the development of a personalized recommendation system.

The practical significance of the qualification work lies in the creation of an architectural solution that can serve as a prototype and template for the development of commercial high-load mobile applications in the e-commerce sphere.

The developed software can be implemented in companies involved in the development of mobile applications for online stores and marketplaces to increase the performance, reliability, and user experience of their products.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БД – База даних

HTTP – HyperText Transfer Protocol

API – Application Programming Interface

MVC – Model-View-Controller

UI – User Interface

ПЗ – Програмне забезпечення

AVD – Android Virtual Device

CDN – Content Delivery Network

HTTPS – HyperText Transfer Protocol Secure

JSON – JavaScript Object Notation

ОС – Операційна система

IDE – Integrated Development Environment

JVM – Java Virtual Machine

SDK – Software Development Kit

UML – Unified Modeling Language

ВСТУП

Сьогодні люди звикли отримувати все, що їм потрібно, практично миттєво. Це стосується і мобільних додатків, особливо тих, що призначені для покупок. Сфера мобільної комерції, тобто купівля товарів через телефон, стрімко розвивається і вже випереджає торгівлю через звичайні комп'ютери. Кількість товарів в онлайн-каталогах постійно зростає і може налічувати мільйони одиниць, а кількість людей, які одночасно користуються додатком, може сягати тисяч. Це створює серйозне технічне завдання для розробників: як зробити так, щоб додаток працював швидко і стабільно, незважаючи на величезну кількість даних та користувачів.

Старі підходи до створення програм уже не справляються з такими навантаженнями. Вони призводять до того, що додатки починають «гальмувати», пошук видає не ті результати, які потрібні, а в моменти найбільшої активності користувачів, наприклад, під час розпродажів, система може взагалі перестати працювати. Це дуже дратує людей і шкодить бізнесу.

Особливо гостро ця проблема стоїть для додатків на платформі Android, оскільки ця операційна система встановлена на переважній більшості смартфонів у світі. При цьому існує величезна кількість різних моделей телефонів з різною потужністю, і додаток має добре працювати на кожному з них. Для інтернет-каталогу швидкість і точність пошуку та фільтрації товарів є критично важливими. Якщо людина не може швидко знайти те, що їй потрібно, вона, найімовірніше, закриє додаток і нічого не купить. Таким чином, від продуктивності програми безпосередньо залежить успішність продажів.

Щоб вирішити цю проблему, потрібен комплексний підхід. Необхідно оптимізувати як сам мобільний додаток (клієнтську частину), так і створити надійну та гнучку серверну частину (бекенд), яка зможе обробляти тисячі запитів одночасно без збоїв. Тому вивчення та застосування сучасних способів побудови програм, методів прискорення роботи та спеціальних

інструментів для пошуку є дуже важливою і потрібною задачею для фахівців у галузі інформаційних технологій.

РОЗДІЛ 1
АНАЛІЗ СТАНУ ОБЛАСТІ РІШЕННЯ ЗАВДАННЯ

1.1 Огляд існуючих існуючих інтернет-каталогів товарів з точки зору високонавантаженості

Мобільний застосунок Prom (рис. 1.1) функціонує як програмний клієнт, що надає кінцевому користувачеві інтерфейс для взаємодії з торговою платформою Prom.ua. Застосунок реалізує функціонал для виконання пошукових запитів у каталозі товарної номенклатури та аналізу відгуків інших споживачів. Окрім цього, він забезпечує повний цикл оформлення замовлення, що включає проведення платіжних транзакцій та подальший моніторинг статусу доставки придбаних товарів.

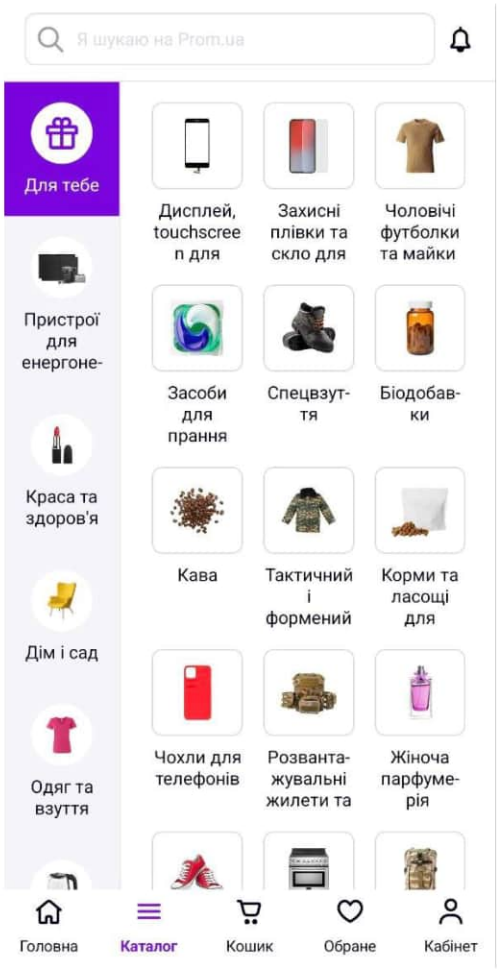

Рисунок 1.1 — Каталог товарів мобільного застосунку Prom

Однією з найбільш обчислювально складних задач у будь-якому інтернет-каталозі є пошук та фільтрація. Для її вирішення розробники Prom відмовились від використання стандартних запитів до реляційної бази даних на користь спеціалізованого пошукового рушія Elasticsearch. Його ефективність зумовлена механізмом зворотного індексування, завдяки якому дані про товари попередньо обробляються та структуруються. Це дозволяє системі виконувати складні повнотекстові пошукові запити та застосовувати багатоаспектну фільтрацію з надзвичайно низькою затримкою, надаючи користувачеві релевантні результати практично миттєво. Для мінімізації навантаження на основні сервіси та бази даних впроваджена комплексна стратегія багаторівневого кешування. На першому рівні, мережа доставки контенту (CDN) використовується для глобальної дистрибуції статичних файлів, таких як зображення товарів, що прискорює їх завантаження у клієнтів за рахунок географічної близькості серверів. На другому рівні, для динамічних даних застосовуються розподілені системи кешування в оперативній пам'яті (in-memory cache), як-от Redis, де зберігаються результати часто виконуваних запитів. Нарешті, на третьому рівні, сам мобільний застосунок локально кешує певну інформацію на пристрої, щоб уникнути зайвих мережевих запитів.

Тепер розглянуто мобільний застосунок Rozetka (рис. 1.2). Це робиться для кращого розуміння того, як саме розробники, що працювали над цим застосунком, підійшли до вирішення проблеми високонавантаженості. Аналіз його архітектури може пролити світло на практичні підходи до забезпечення масштабованості та продуктивності в умовах значного навантаження користувачів.

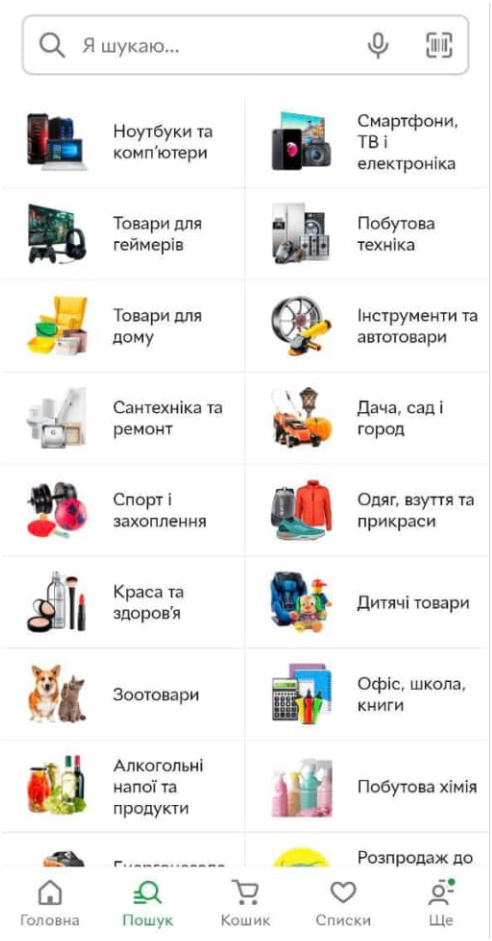

Рисунок 1.2 — Каталог товарів мобільного застосунку Rozetka

Основою сучасної інфраструктури Rozetka є поступовий, але послідовний перехід від монолітної архітектури до мікросервісної. Ця архітектурна парадигма є галузевим стандартом для побудови складних, високонавантажених систем. Декомпозиція системи на сукупність невеликих, автономних сервісів дозволяє досягти низки критичних переваг. По-перше, це гранулярне горизонтальне масштабування, за якого ресурси додаються точково лише тим компонентам, що зазнають пікового навантаження. По-друге, це підвищена відмовостійкість: збій одного з сервісів не призводить до відмови всієї платформи, а лише тимчасово обмежує частину її функціональності. По-третє, це технологічна незалежність, що дозволяє обирати оптимальний стек технологій для кожної конкретної задачі.

Надалі розглянуто OLX для кращого розуміння стану області рішення завдання та вибору подальшого проєктного рішення. OLX (рис. 1.3) — це одна

з найбільших платформ онлайн-оголошень в Україні, якою користуються мільйони покупців та продавців. У цього сервісу простий та інтуїтивний інтерфейс, але за ним стоїть складна інфраструктура, призначена для екстремальних навантажень.

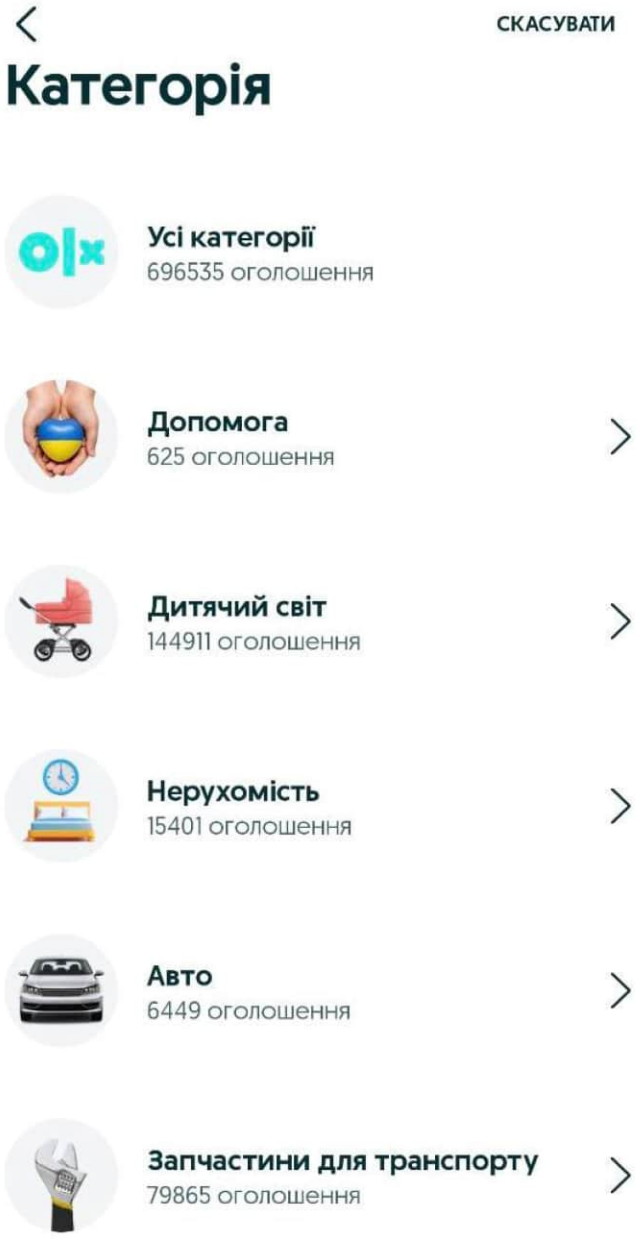

Рисунок 1.3 — Категорії мобільного застосунку OLX.ua

По-перше, бувають пікові навантаження, які виникають у вечірній час та вихідні дні, коли активність користувачів є максимальною, або під час сезонних сплесків попиту на певні категорії товарів. По-друге, функція пошуку та персоналізовані рекомендації створюють колосальне навантаження, адже система одночасно обробляє мільйони запитів на фільтрацію та сортування мільйонів оголошень. По-третє, синхронізація даних в реальному часі є критичною: нові оголошення мають з'являтися миттєво, а повідомлення між користувачами — доставлятися без затримок.

Для вирішення цих проблем високонавантаженості, OLX використовує сучасні хмарні технології та масштабовану, а саме мікросервісну архітектуру. Цей підхід дозволяє розподілити функції системи на невеликі, незалежні сервіси, такі як управління оголошеннями, пошук, система обміну повідомленнями, автентифікація тощо. Завдяки хмарній інфраструктурі, кожен мікросервіс може масштабуватися окремо відповідно до поточного навантаження. Така гнучкість забезпечує ефективне розподілення ресурсів та високу доступність платформи навіть під час пікових навантажень, підвищуючи її відмовостійкість.

1.2 Огляд літературних джерел і статті про вирішення проблеми високонавантаженості

1.2.1 Книга «Designing Data-Intensive Applications: The big ideas behind reliable, scalable and maintanable systems» Мартін Клепман

Книга «Designing Data-Intensive Applications» Мартіна Клепмана глибоко занурює читача в основи побудови систем, здатних витримувати значні навантаження. З точки зору високонавантажених систем, книга є цінним ресурсом, оскільки вона детально розглядає ключові аспекти масштабованості. Автор пояснює, як важливо кількісно описувати навантаження за допомогою відповідних параметрів та вимірювати продуктивність, використовуючи метрики на кшталт часу відгуку та перцентилів.

Клепман досліджує різні підходи до подолання зростаючого навантаження, включаючи вертикальне (scale up) та горизонтальне масштабування (scale out або shared-nothing architecture). Він наголошує, що не існує універсального «магічного соусу» масштабованості, і вибір архітектури (рис. 1.1) залежить від специфіки навантаження та вимог додатку.

Книга також обговорює складнощі, пов'язані з розподіленими системами, які часто є необхідними для досягнення високої масштабованості та відмовостійкості. Розглядаються питання реплікації, партиціонування та обробки даних у розподіленому середовищі. Хоча книга не є покроковою інструкцією з використання конкретних інструментів, вона дає глибоке розуміння фундаментальних принципів, що лежать в основі високонавантажених систем. Це дозволяє інженерам приймати обґрунтовані рішення щодо архітектури та вибору технологій.

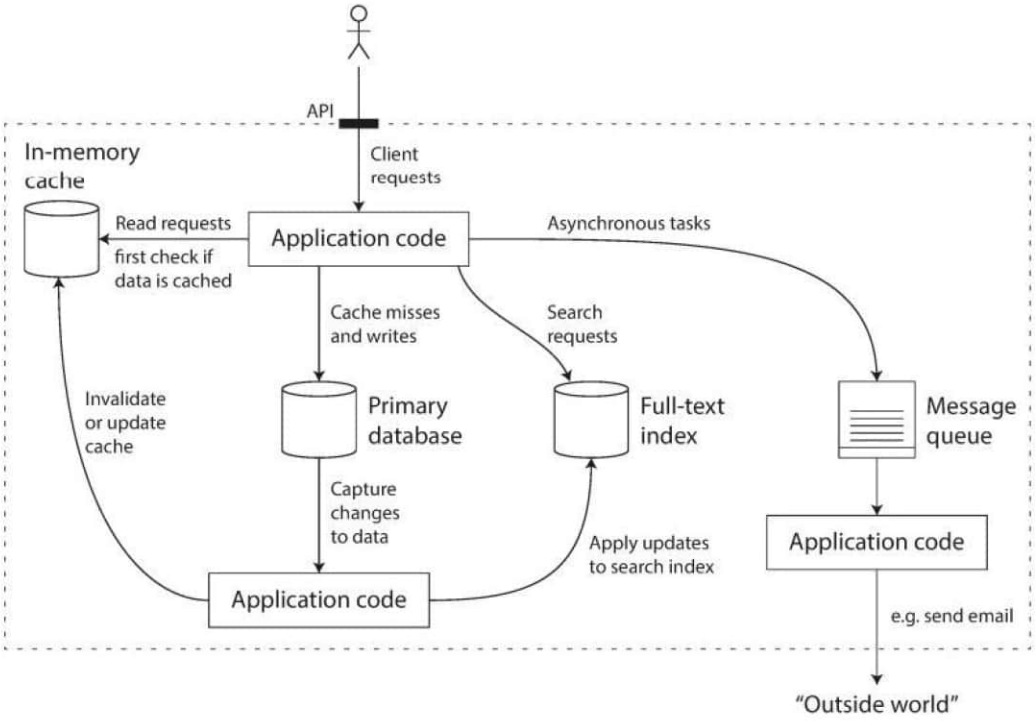

Рисунок 1.2 — Архітектура системи даних, яка включає декілька компонентів[1]

1.2.2 Книга «Release it! Design and deploy production-ready software» Майкл Найгард

Книга Майкла Нігардта «Release It! Design and Deploy Production-Ready Software» є надзвичайно корисною для спеціалістів з інформаційних технологій, які прагнуть створювати системи, здатні успішно працювати під високим навантаженням у реальних умовах експлуатації. Замість того, щоб зосереджуватися виключно на теорії, автор акцентує увагу на практичних аспектах надійності, стабільності та відмовостійкості, які є критично важливими при зростанні навантаження.

Книга представляє низку шаблонів стабільності, таких як Bulkhead, Circuit Breaker та Limiter, які допомагають захистити систему від каскадних збоїв та перевантаження. Ці шаблони є безпосередньо застосовними для проектування систем, що мають витримувати значні піки трафіку або тривале

1 Рисунок із зовнішнього джерела: https://unidel.edu.ng/focelibrary/books/Designing%20Data-Intensive%20Applications%20The%20Big%20Ideas%20Behind%20Reliable,%20Scalable,%20and%20Maintainable%20Systems%20by%20Martin%20Kleppmann%20(z-lib.org).pdf

високе навантаження, забезпечуючи при цьому доступність ключових функцій.

Нігардт також глибоко розглядає важливість тестування в умовах, близьких до використання, включаючи тестування навантаження та витривалості. Він пояснює, як виявити потенційні «вузькі місця» та точки відмови до того, як система потрапить в експлуатацію, що є життєво важливим для запобігання збоїв під високим навантаженням. Питання моніторингу, розгортання та управління змінами також висвітлюються з точки зору забезпечення стабільності та надійності системи під будь-яким навантаженням.

Таким чином, «Release It!» є цінним посібником для всіх, хто займається розробкою та експлуатацією високонавантажених систем. Вона надає практичні інструменти та підходи для побудови стійкого програмного забезпечення, яке не просто працює, але й надійно функціонує в найскладніших умовах.

1.2.3 Стаття «Problems and solutions in building highly loaded systems» Володимир Козуб

На основі аналізу статті Володимира Козуба «Problems and solutions in building highly loaded systems», з точки зору високонавантаженості, робота фокусується на ключових аспектах створення стійких та продуктивних систем. Автор висвітлює типові проблеми, що виникають при зростанні навантаження, та пропонує дієві рішення для їх подолання.

Основний акцент зроблено на важливості архітектурної гнучкості та масштабованості, зокрема за рахунок використання мікросервісної архітектури, недоліки та переваги якої наведені в табл. 1.1. Розглядаються методи ефективного управління даними в умовах високого навантаження, такі як шардинг та реплікація баз даних. Також стаття підкреслює значення кешування та балансування навантаження для оптимізації часу відгуку та розподілу трафіку.

Автор вказує на необхідність впровадження асинхронної обробки запитів для уникнення блокувань та підвищення загальної пропускної здатності системи. Крім того, значна увага приділяється питанням моніторингу, логування та побудови відмовостійких механізмів для забезпечення безперебійної роботи системи навіть під екстремальними навантаженнями. Стаття є цінним ресурсом для розуміння комплексного підходу до проектування високонавантажених систем.

Таблиця 1.1 — Недоліки та переваги мікросервісної архітектури

№ п/н	**Переваги**	**Недоліки**
1	Кожен мікросервіс відносно невеликий і виконує лише одну функцію	У розробників з'являється набагато більше роботи, пов'язаної з взаємодією численних сервісів
2	Кожна окрема частина відповідає за розгортання, розробку, тестування та масштабування частини програми	Тестування різних взаємодій між сервісами стає набагато складнішим черех складну структуру системи
3	Кожен мікросервіс має свій власний технологічний стек, і він не має впливу на інші частини	Розробникам потрібно більше часу на реалізацію механізмів взаємодії для вирішення деяких можливих проблем
4	Ізольований підхід архітектури мікросервісів також спрощує процес виправлення проблем, що виникають	Кожен член команди повинен бути повністю присутнім і глибоко розуміти процеси, інакше буде складно отримати ідеальний робочий процес

1.3 Актуальність вибору розробки високонавантаженого інтернет-каталогу товарів під Android

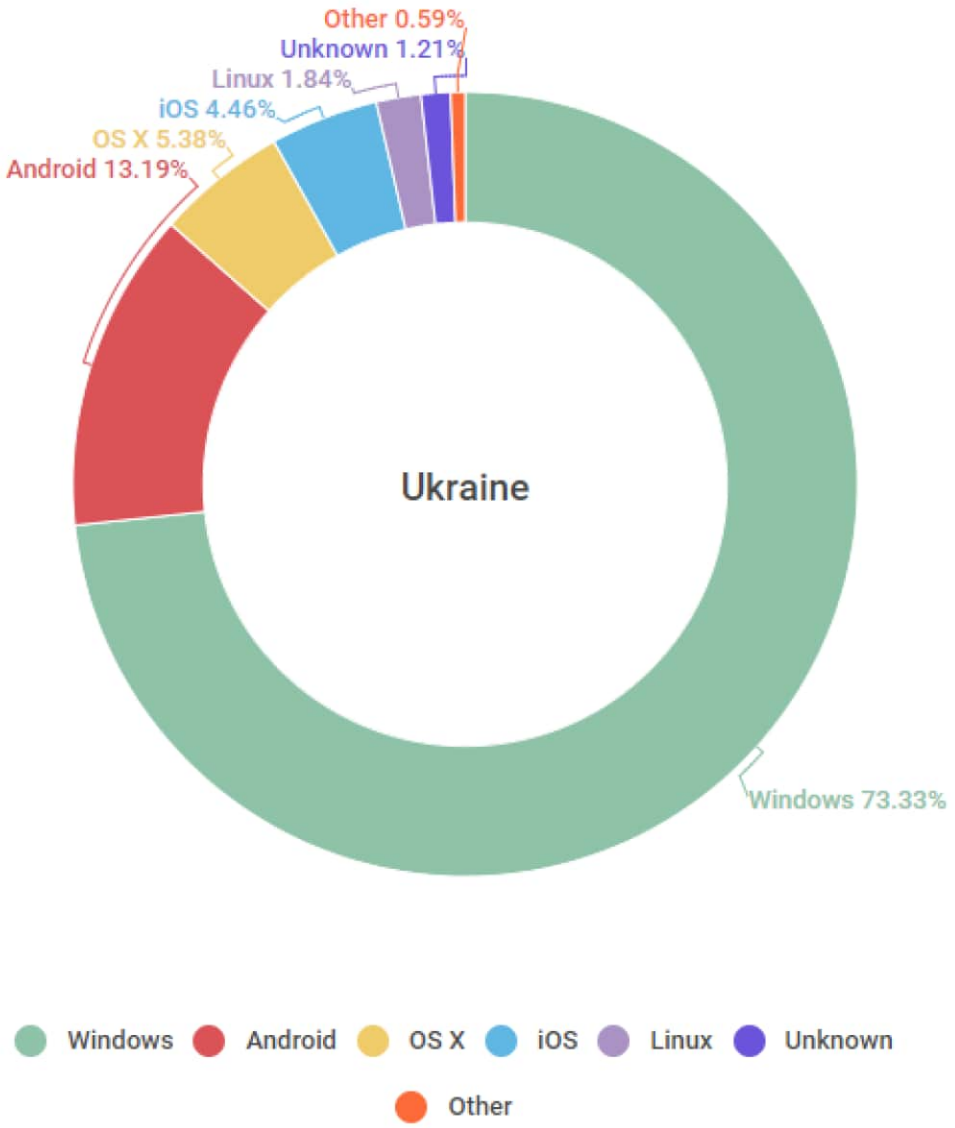

Рисунок 1.3 — Статистика використовуваних ОС від «Marketer» станом на 2017 рік[2]

По-перше, найбільш важливим є домінування операційної системи Android, частка якої становить 41.24%. Це робить її беззаперечним лідером на глобальному ринку. Такий високий показник пояснюється, головним чином, повсюдним поширенням мобільних пристроїв (смартфонів та планшетів) від

2 Рисунок із зовнішнього джерела: https://marketer.ua/ru/stats-operating-system-2017/

широкого спектра виробників, особливо у середньому та бюджетному цінових сегментах.

По-друге, другу за величиною частку займає операційна система Windows з показником 35.24%. Історично ця платформа домінувала на ринку персональних комп'ютерів (десктопів та ноутбуків). Незважаючи на стрімкий розвиток мобільних технологій, Windows зберігає міцні позиції, що свідчить про її триваючу релевантність у корпоративному секторі та для завдань, що вимагають потужних обчислювальних ресурсів.

По-третє, слід виділити екосистему компанії Apple. Частка IOS становить 13.20%, а частка OS X — 4.66%. Хоча кожна з них окремо значно поступається лідерам, їх сукупна частка (17.86%) є вельми істотною. З одного боку, це демонструє успішну стратегію створення закритої, вертикально інтегрованої екосистеми. З іншого боку, це підкреслює концентрацію продукції Apple у преміальному сегменті ринку.

Крім того, необхідно підкреслити сукупне домінування мобільних платформ. Якщо узагальнити частки Android та iOS, їх спільний показник сягає 54.44%. Для порівняння, сукупна частка класичних десктопних операційних систем (Windows, OS X, Linux) становить 40.67%. Це наочно ілюструє глобальний зсув парадигми споживання контенту в бік мобільних пристроїв (mobile-first).

До менш значних, але вартих уваги сегментів, відносяться Linux (0.77%), «Other» (Інші) (1.90%) та «Unknown» (Невідомі) (2.99%). Низька частка Linux у цьому контексті пояснюється її переважним використанням на серверах та у вузькому колі ентузіастів, а не на споживчих пристроях для веб-серфінгу. Категорії «Other» та «Unknown» включають різноманітні нішеві ОС та пристрої, чию операційну систему не вдалося ідентифікувати.

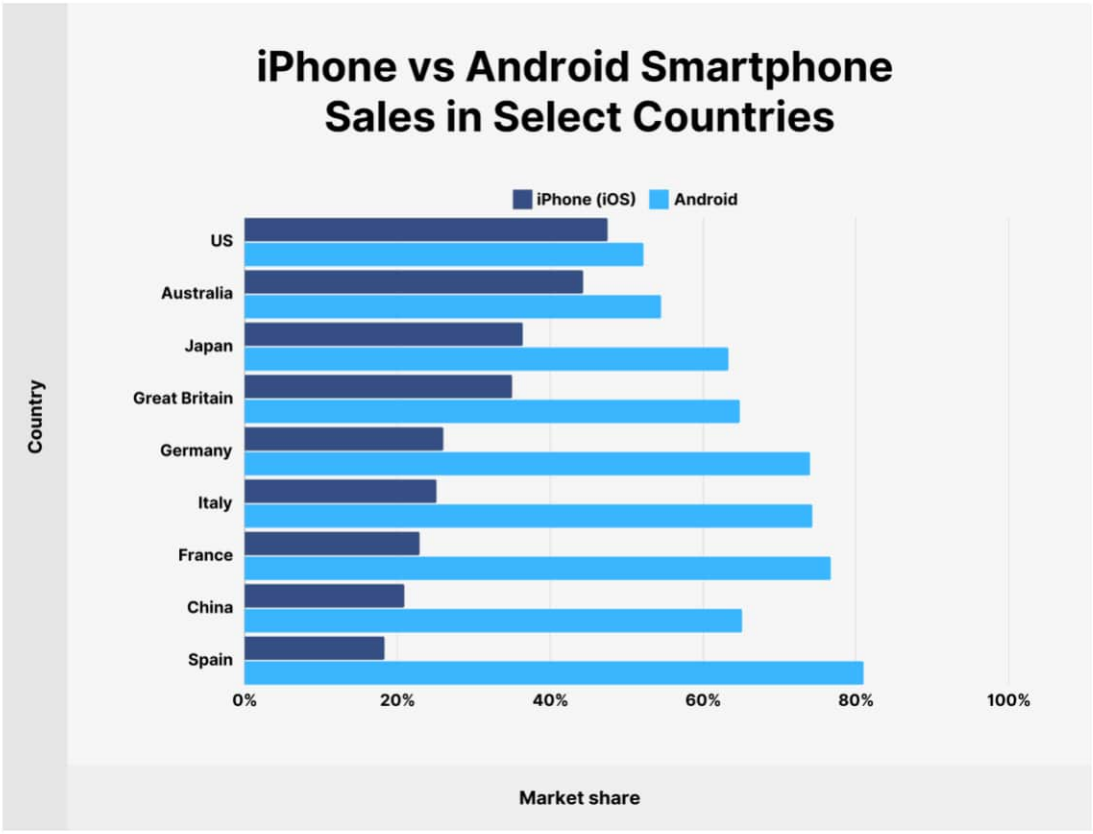

Рисунок 1.4 – Статистика Android/IOS станом на 2025 рік[3]

Проведений аналіз має велике практичне значення для різних суб'єктів ринку. Для розробників мобільних додатків зі сказаного раніше випливає, що стратегія виходу на ринок повинна бути географічно диференційованою. Для ринків континентальної Європи доцільно в першу чергу орієнтуватися на платформу Android (Android-first strategy). Водночас, при орієнтації на ринки США та Японії, розробка під iOS є не менш, а іноді й більш пріоритетною. Бажано також враховувати, що, незважаючи на меншу ринкову частку, користувачі iOS в середньому є більш платоспроможними та схильними до покупок всередині додатків.

3 Рисунок із зовнішнього джерела: https://coolest-gadgets.com/android-statistics/

1.4 Інформаційні технології та інструменти для рішення проблеми високонавантаженості

1.4.1 Середовище розробки під Android

Android Studio — це офіційне середовище розробки програмного забезпечення під Android, яке вийшло в реліз 8 грудня 2014 року. В табл. 1.2 наведені переваги та недоліки IDE. Компанія JetBrains розробила його на основі IntelliJ IDEA Community Edition і розвиває досі. Саме середовище є адаптованим до виконання типових задач, які постають перед розробником ПЗ в процесі створення застосунків під Android.

Таблиця 1.2 — Переваги та недоліки Android Studio

№ п/н	Переваги	Недоліки
1	Офіційна IDE від Google	Складність для новачків через велику кількість інструментів і функцій
2	Інструменти аналізу продуктивності, налагодження, тестування та оптимізації коду	Рекомендовано 8 ГБ оперативної пам’яті
3	Тестування додатків на різних віртуальних пристроях	Рекомендовано від 4 ядер процесор
4	Система збірки Gradle	Займає від 10 до 50 ГБ на жорсткому диску
5	Розширення функціоналу за допомогою плагінів	Перша збірка може бути більше декількох хвилин

Android Studio безсумнівно є найкращим середовищем розробки під Android, завдяки великої кількості інструментів і функцій, інтуїтивно зрозумілому інтерфейсу, широкими можливостями налаштувань та величезної

спільноти розробників, бо його розробив розробник Android — Google. При цьому потребує середнього за характеристиками комп'ютера — процесор Intel Core i3 або вище від 7 покоління, оперативна пам'ять від 8 ГБ з частотою не менше 2133 Ггц і жорсткий диск від 256 ГБ.

1.4.2 Мови програмування для розробки під Android

На даний момент існують дві найбільш поширені мови програмування під Android — Kotlin і Java. Крім цього для розробки під Android використовується мова програмування C#.

Java — об'єктно-орієнтована, структурна, імперативна мова програмування випущена в 1995 році компанією «Sun Microsystems». В 2009 році компанія «Oracle» придбала «Sun Microsystems». На мові Java розробляють програми, ігри під ОС Android, Windows, Linux і backend для веб-сайтів тощо. Станом на 2023 рік Java входить в топ-5 найпопулярніших мов програмування серед IT-спеціалістів згідно рейтингу IEEE Spectrum (рис. 1.5).

Kotlin був розроблений JetBrains у 2011 році, за цей час набув високої популярності завдяки гнучкості та безпеки роботи з ним, відносно простого синтаксиса, сумісністю зі старими базами даних Java, можливістю кросплатформеної компіляції і розробки серверних додатків. Він працює на Java Virtual Machine і створений під впливом таких мов програмування як Java, Scala, Groovy. Kotlin є статично типізованою мовою програмування, тобто в ній реалізований механізм, що дозволяє на етапі написання програми визначити через тип об'єкта програми множину припустимих значень та множину операцій над об'єктом так, що порушення вимог типізації призводитиме до попередження або помилки на етапі трансляції програми, а не на етапі її виконання.

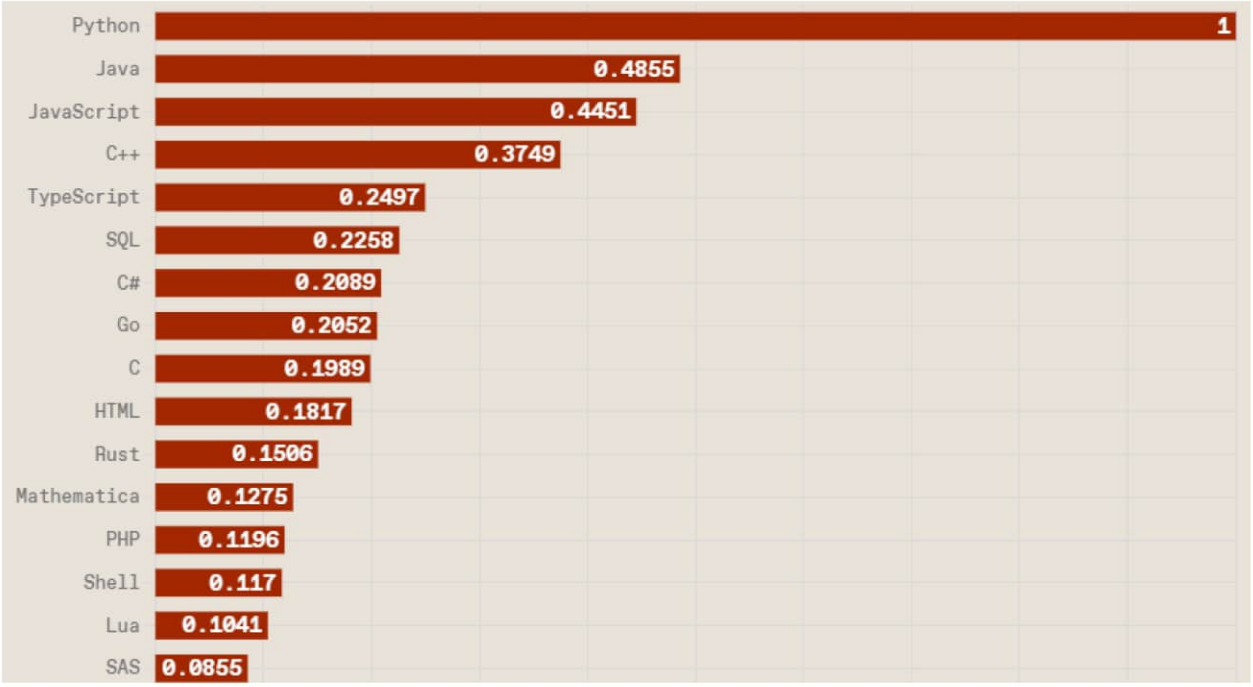

Рисунок 1.5 — Лінійчаста діаграма рейтигу мов програмування за 2024 рік [4]

Підсумовуючи, Java та Kotlin — домінуючі мови для розробки під Android, менше використовується C#. Kotlin позиціонується як сучасніша мова з перевагами у гнучкості, безпеці та лаконічному синтаксисі. Водночас, Java, як більш зріла мова з багаторічною історією з 1995 року, вирізняється величезною екосистемою, великою кількістю існуючих бібліотек та фреймворків, а також надзвичайно великою та активною спільнотою розробників. Ці фактори роблять Java надійним вибором з широкою підтримкою та величезною базою знань, що є її ключовими перевагами над Kotlin, незважаючи на зростаючу популярність останнього. Обидві мови працюють на JVM, забезпечуючи функціональність на різних платформах.

1.4.3 Графічні редактори для створення інтерфейсу користувача

На сьогоднішній день існують десятки інструментів для створення інтерфейсу користувача ПЗ. Найбільш популярними є Adobe Photoshop, Figma, Sketch та CorelDRAW.

Adobe Photoshop — це багатофункціональний растровий графічний редактор, розроблений і поширюваний компанією «Adobe Systems». Ключові

4 Рисунок із зовнішнього джерела: https://spectrum.ieee.org/top-programming-languages-2024

переваги: універсальність, розширюваність, популярність, інтеграція з Adobe Creative Cloud і висока якість зображень, а недоліки: висока ціна, вимогливість до відеопам’яті, складний інтерфейс, не підтримує векторну графіку та неможливо створювати інтерактивний дизайн.

Figma — це векторний графічний редактор, який доступний як онлайн так і офлайн. Застосунок забезпечує доступ до повного набору функцій Figma, орієнтованих на UI/UX дизайн. За допомогою нього можна використовувати різноманітні інструменти для редагування векторної графіки та прототипування. Десктопна і веб- версії підтримують спільну роботу над проектами. Далі розглянуто переваги десктопної програми Figma:

- працює без Інтернету;
- кросплатформеність;
- повний функціонал.

Недоліки десктопної програми Figma:

- не підходить для створення макетів для друку та професійної глибокої ретуші фотографії;
- недостатньо функцій для створення складної векторної графіки;
- режим «Dev Mode» (рис. 1.6) недоступний на Freemium.

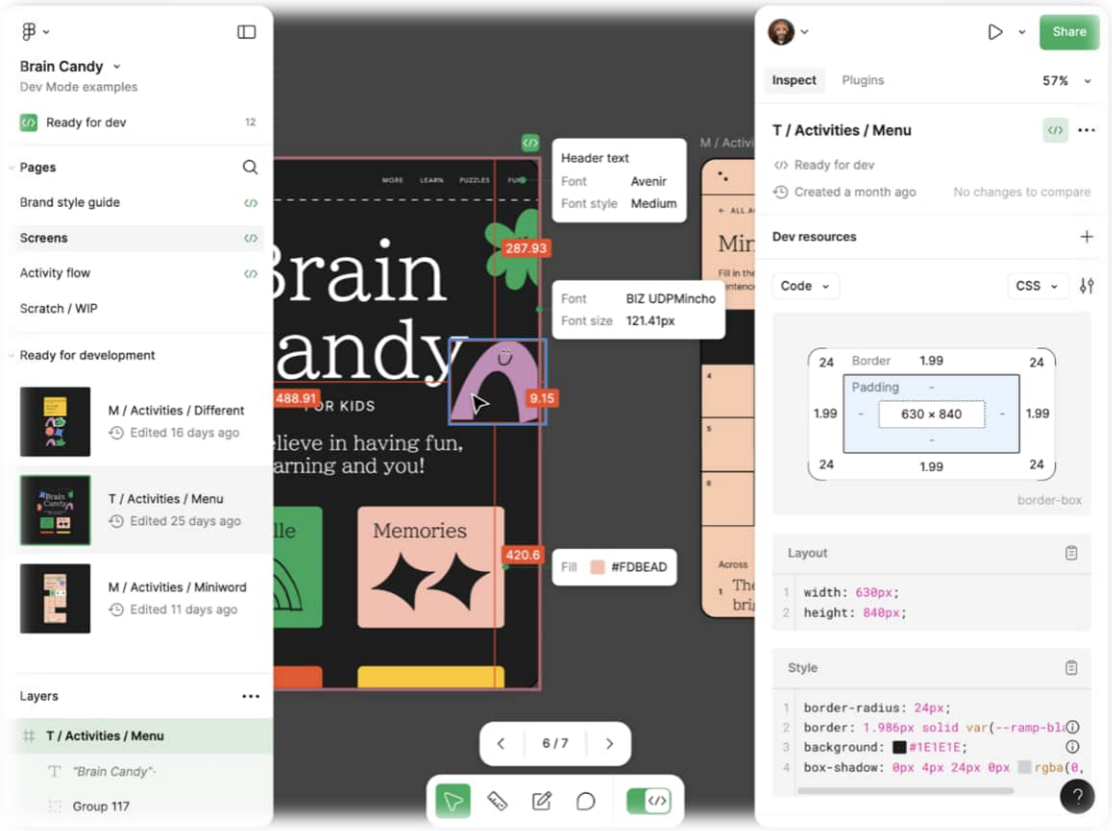

Рисунок 1.6 — Figma в режимі «Dev Mode»

CorelDRAW — це векторний графічний редактор, розроблений компанією Corel Corporation у 1989 році. CorelDRAW є потужним векторним графічним редактором, цінним для ілюстрацій та макетування завдяки гнучким інструментам і можливості одноразової покупки ліцензії, пропонуючи зручний інтерфейс для певних завдань дизайну. Його недоліки включають потенційно складний інтерфейс для початківців, меншу поширеність порівняно з конкурентами (як Illustrator), що може створювати проблеми сумісності файлів, та історично слабшу підтримку macOS. Незважаючи на це, CorelDRAW залишається популярним вибором у поліграфії та рекламній сфері, надаючи професійним дизайнерам альтернативний ефективний інструмент для векторної графіки та макетування.

1.4.4 Системи управління базами даних

Система управління базами даних — це набір програмного забезпечення, призначений для управління даними в базі даних, що охоплює функції їх внесення, зберігання, пошуку та обробки.

Одна з найпоширеніших реляційних СУБД — MySQL. Вона використовує діалогову мову запитів SQL для взаємодії користувача з базами даних, яка застосовується для отримання потрібної інформації з БД, додавання нових записів до таблиць, оновлення існуючих записів, видалення записів, створення нових баз даних і таблиць (рис. 1.7) та інше.

Менш популярною, але все ж таки розповсюдженою СКБД є PostgreSQL — це система управління об'єктно-реляційними базами даних з підтримкою ОС Windows, Linux, MacOS. Використовується для того, що було перелічено вище для SQL, але має наступні відмінності: підтримує більше типів даних, показує вищу продуктивність при складних запитах, реалізована можливість створювати власні функції та розповсюджується під вільною ліцензією.

Наступна достатньо розповсюджена СУБД — Oracle Database. Це комерційна реляційна СУБД компанії «Oracle» розроблена в 1979 році Ларрі Еллісоном, Бобом Майнером і Едом Оутсом. Сфера застосування Oracle Database — розробка високонавантажених критично важливих систем. Її головними перевагами є потужність, надійність, масштабовність і широкий набір функцій.

Рисунок 1.7 — Приклад таблиці реляційної БД з основними поняттями

1.4.5 Бібліотеки для тестування програмного коду Java

Розробка сучасного ПЗ не може обійтись без тестування програмного коду. Тестування проводиться з метою виявлення помилок і оцінки ризику відмови. Процес тестування поділяється на чотири кроки: тестування модулів, тестування збірки модулів, тестування правильності, тестування системи. ПЗ тестується за допомогою інструментів тестування, або тести пишуться з нуля. На сьогоднішній день існують десятки інструментів тестування з відкритим кодом. Серед них найчастіше використовуваними є інструменти для модульного тестування. Наприклад, Junit або TestNG.

JUnit — це бібліотека для тестування ПЗ для мови програмування Java з функціями для перевірки очікуваних результатів у тестах і для керування життєвим циклом тестів, в табл. 1.4 наведений опис перших.

TestNG — це біблітека для автоматизованого тестування програмного забезпечення, написана на мові програмування Java, підтримує функціональні, інтеграційні і навантажувальні тести.

Таблиця 1.3 — Опис функцій бібліотеки JUnit

№ п/н	**Назва функції**	**Опис**
1	assertTrue	Перевіряє, чи є задана булеве умова істинною (true)
2	assertFalse	Перевіряє, чи є задана булеве умова хибною (false)
3	assertNull	Перевіряє, чи є посилання на об'єкт нульовим (null)
4	assertSame	Перевіряє, чи посилаються два посилання на об'єкти на один і той самий об’єкт у пам'яті

Продовження табл. 1.3

5	assertNotSame	Перевіряє, чи посилаються два посилання на об'єкти на різні об’єкти у пам'яті
6	assertNotNull	Перевіряє, чи є посилання на об'єкт ненульовим (not null)

1.5 Архітектурні шаблони програмного забезпечення для розробки високонавантаженого мобільного інтернет-каталогу

Для розробки високонавантаженого мобільного інтернет-каталогу товарів найкраще підходять архітектурні шаблони, які забезпечують високу масштабованість, стійкість до навантажень та ефективну обробку запитів. З огляду на те, що каталог товарів є переважно «читаючою» системою важливими є архітектури, оптимізовані для швидкого пошуку та відображення даних.

SOA (Service-Oriented Architecture або укр. сервіс-орієнтована архітектура) — це архітектурний стиль проєктування ПЗ, який структурує додаток як набір автономних, слабозв'язаних сервісів, що взаємодіють між собою. Кожен сервіс інкапсулює окрему бізнес-функцію та надає до неї доступ через стандартизований інтерфейс. Такий підхід дозволяє розробляти, розгортати та масштабувати кожен сервіс незалежно, що підвищує гнучкість розробки, відмовостійкість та загальну продуктивність системи.

Рисунок 1.8 — Модель архітектури SOA[5]

5 Зображення залучено з зовнішнього джерела: https://softinnovation.site123.me/my-blog/arquitectura-orientada-a-servicios-soa

Монолітна архітектура — це традиційна модель проєктування ПЗ, яка історично передувала сервіс-орієнтованим підходам. Її суть полягає у побудові системи як єдиного, неподільного модуля, в якому всі компоненти тісно взаємопов'язані та функціонують як одне ціле. Цей підхід був домінуючим у розробці протягом десятиліть і часто асоціюється з класичними методологіями, такими як Waterfall, хоча й досі залишається актуальним для багатьох типів проєктів.

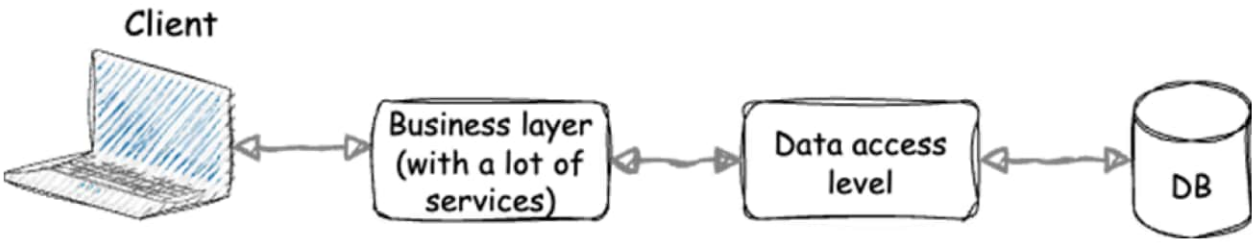

Рисунок 1.9 — Модель монолітної архітектури[6]

1.6 Методи кешування даних на мобільних пристроях

Для мобільних додатків існує кілька поширених методів кешування даних, які використовуються для покращення продуктивності, зменшення залежності від мережі та забезпечення роботи в офлайн-режимі.

In-Memory Cache, відоме також як локальне кешування, передбачає розміщення даних безпосередньо в RAM сервера додатку. Завдяки цьому досягається максимальна швидкість отримання даних, адже виключається затримка, пов'язана з передачею даних мережею або зчитуванням з диска. Це робить кешування в оперативній пам'яті дуже популярним для оптимізації продуктивності: воно прискорює виконання операцій, зменшує навантаження на базу даних та покращує загальну швидкодію системи.

Файлове кешування передбачає зберігання даних безпосередньо на фізичному накопичувачі мобільного пристрою (внутрішній або зовнішній пам'яті). Це дозволяє зберігати кешовану інформацію між сеансами роботи

6 Зображення залучено з зовнішнього джерела: https://www.artofba.com/uk/post/main-types-of-software-architecture

додатку, на відміну від кешування в пам'яті. Disk Cache (рис. 1.10) ідеально підходить для зберігання великих обсягів даних, таких як завантажені зображення, відеофайли або відповіді від мережевих запитів (наприклад, JSON чи XML). Доступ до даних з диска повільніший, ніж з RAM, але значно швидший, ніж повторне завантаження через мережу. Цей метод є ключовим для зменшення споживання трафіку та забезпечення часткової офлайн-функціональності додатку.

Кешування запитів (Query Cache) – це механізм, який реалізується на рівні системи управління базами даних (СУБД) або проксі-сервера перед нею. Його основне призначення – зберігати у пам'яті результати виконання раніше виконаних запитів до бази даних. Коли система отримує новий запит, вона спочатку перевіряє, чи є ідентичний запит вже в кеші. Якщо такий запит знайдено і відповідні дані в таблицях не змінювалися з моменту кешування, СУБД негайно повертає збережений результат з кешу, минаючи процес повторного виконання запиту, оптимізації та звернення до даних на диску.
Використання кешу запитів дозволяє значно прискорити обробку часто повторюваних запитів, особливо тих, що вибирають дані, які рідко змінюються. Це суттєво знижує навантаження на ресурси бази даних (CPU, диск) та покращує час відгуку для клієнтських застосунків.

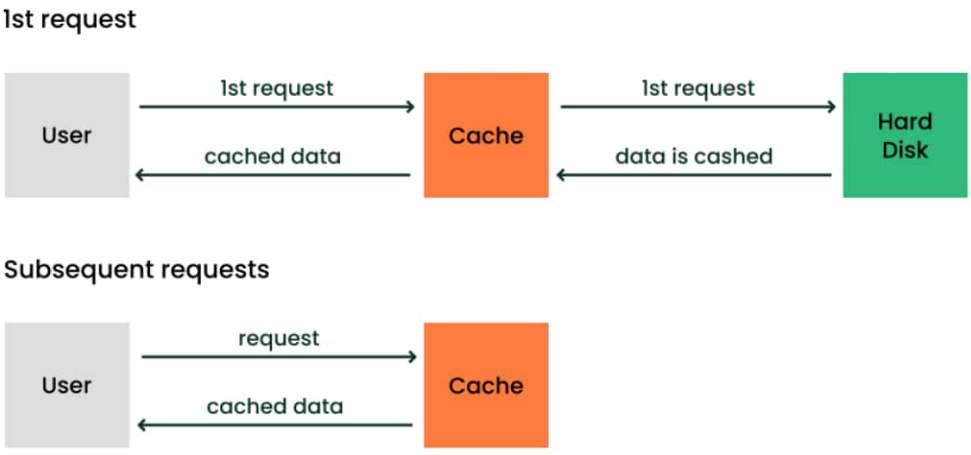

Рисунок 1.10 — Візуалізація файлового кешування (Disk Cache)[7]

7 Рисунок залучений із зовнішнього джерела:
https://doc.opensuse.org/documentation/leap/virtualization/html/book-virtualization/cha-cachemodes.html

РОЗДІЛ 2
ПРОЄКТНІ РІШЕННЯ ПО ЗАВДАННЮ

2.1 Проєктування програмної архітектури інтернет-каталогу товарів під Android

2.1.1 Вимоги до програмної архітектури

Функціональні вимоги:

- Перегляд каталогу товарів: Користувач має можливість переглядати частину UI, яка відповідає за відображення карток товарів;
- Характеристики товару: Користувач має можливість перейти на сторінку товару та переглянути його характеристики;
- Пошук товарів: Користувач може здійснювати пошук товарів за назвою;
- Фільтрація товарів: Користувач має можливість фільтрувати товари за категорією, ціною і брендом;
- Довантаження товарів: Користувач має можливість довантажити товари, якщо досягне кінця списку товарів.

Нефункціональні вимоги:

- Продуктивність: Час завантаження каталогу товарів не повинен перевищувати 2 секунди при стабільному інтернет-з’єднанні, пошук та фільтрація товарів повинні видавати результати не більше ніж за 4 секунди;
- Надійність: Android-застосунок повинен бути стійким до мережевих збоїв і містити в собі механізм повторних спроб;

- Масштабованість: Програмна архітектура мобільного-застосунку повинна бути гнучкою для додавання нового функціоналу або переходу на інший API;
- Ремонтопридатність: ПЗ повинно бути з вбудованою функцією відновлення до початкового стану;
- Зручність використання: Інтерфейс програми розроблений за підходом Human-centered design;

2.1.2 Вибір архітектури ПЗ

Для розробки ПЗ обрано архітектурний шаблон клієнт-сервер (рис. 2.1). По-перше, клієнт-серверна архітектура розподіляє frontend і backend на окремі сервери, що дозволяє жорстко не прив'язувати клієнт до сервера. По-друге, архітектурний шаблон передбачає нескінченну кількість клієнтів, які можуть працювати паралельно і взаємодіяти з одним сервером. По-третє, передача даних між клієнтським кодом і серверною частиною здійснюється по HTTP-протоколу.

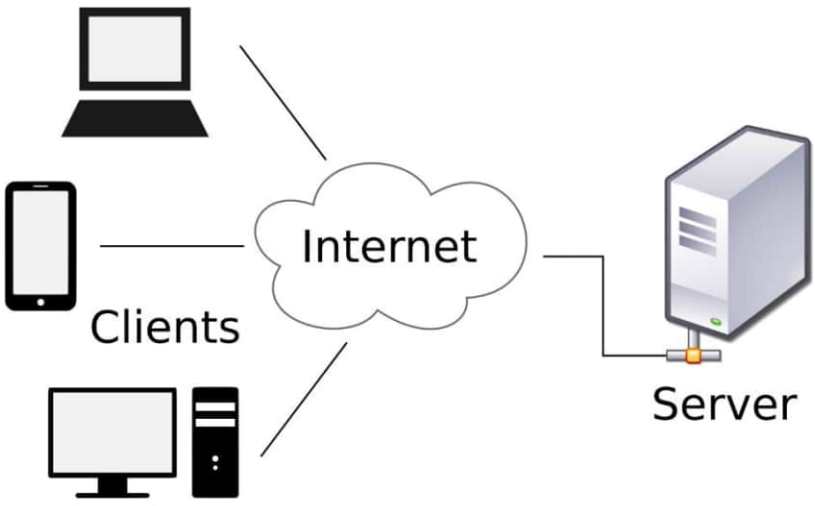

Рисунок 2.1 — Структура клієнт-серверної архітектури[8]

На моделі (рис. 2.1) представленій вище, Internet виступає як посередник між Clients і Server. Він потрібен для того, щоб передавати дані з серверу на

8 Рисунок залучений із зовнішнього джерела: https://upload.wikimedia.org/wikipedia/commons/c/c9/Client-server-model.svg?download

клієнт або навпаки, з метою їх відображення або обробки відповідно. Clients – це клієнти, які працюють на різноманітних пристроях, а Server – це сервер на якому відбувається обробка даних і обчислення.

2.1.3 Визначення архітектурних шаблонів клієнт-серверної архітектури ПЗ

Клієнт-серверна архітектура (рис. 2.2) потребує вибору протоколу передачі даних. Визначено, що HTTP-протокол є оптимальним рішенням для високонавантаженого Android-застосунку, тому що він призначений для передачі таких типів файлів як текст, зображення, відео і скрипти. Але він все ж таки не такий захищений як HTTPS, але для виконання завдання є достатньо. Цей протокол забезпечить передачу всіх необхідних даних: тексту, зображень, чисел.

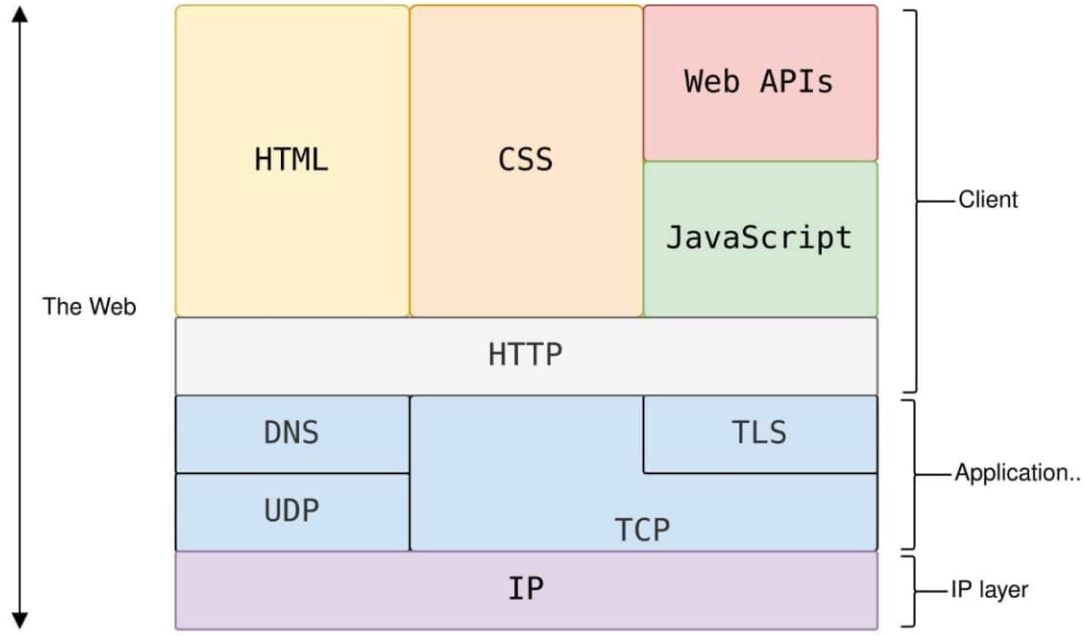

Рисунок 2.2 — Структура клієнту і серверу з використанням HTTP на прикладі Web[9]

Клієнт працює на Android, тому обрано архітектурний патерн MVC (рис. 2.3) – Model-View-Controller, який використовується для розподілення додатку на 3 логічні компонента. Це зручно як і в веб-розробці, так і в розробці мобільних застосунків під Android. Наприклад, якщо інтернет-каталог товарів

9 Рисунок залучений із зовнішнього джерела: https://mdn.github.io/shared-assets/images/diagrams/http/overview/http-layers.svg

виконує ті ж самі функції, що і веб-додаток, то різниця в реалізації клієнту такого Android-застосунку в порівнянні з веб-додатком буде в мові програмування, текстовій розмітці та середовищі розробки.

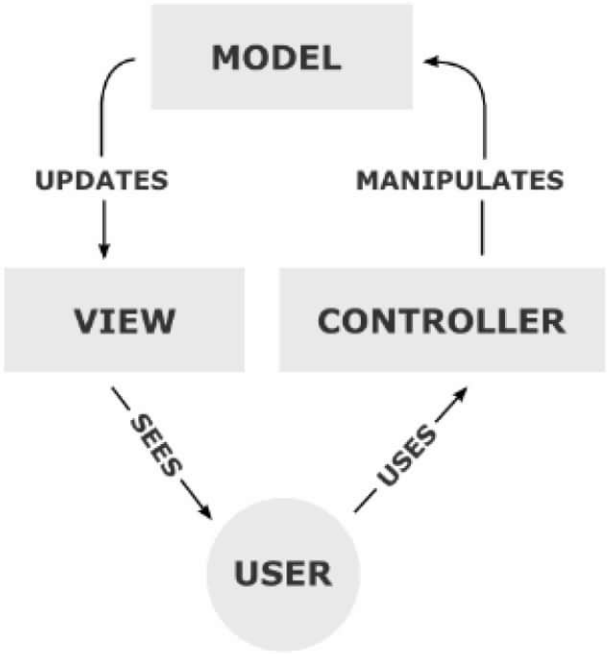

Рисунок 2.3 — Структура патерну MVC[10]

Сервер – backend, який надає API для клієнтського додатку, обробляє запити, керує базою даних та реалізує основну бізнес-логіку. В процесі виконання завдання кваліфікаційної роботи був обраний веб-сервер XAMPP, який потрібен для того, щоб з backend отримувати дані з MySQL, і розгорнути локальний веб-сервер Apache HTTP Server для того, щоб працювала БД. RestAPI слугує точкою входу для всіх клієнтських запитів, тому він є особливо важливим для високонавантажених систем.

Протягом аналізу стану області рішення завдання визначено, що монолітна архітектура (рис. 2.4) буде ідеальним варіантом для серверної частини, тому що вона є простою і ефективною для високонавантаженого інтернет-каталогу товарів під Android. Також це дозволить організувати всі

10 Рисунок залучений із зовнішнього джерела: https://ru.wikipedia.org/wiki/Model-View-Controller

методи і функції в одному модулі і спростить процес написання коду. Звичайно що надійність, масштабованість і ремонтопридатність цієї системи є гірше, ніж в порівнянні з SOA в табл. 2.1.

Таблиця 2.1 – Порівняльна таблиця SOA і монолітної архітектури

Критерій	**Сервісно-орієнтована архітектура (SOA)**	**Монолітна архітетура**
Надійність	Висока: відмова одного сервісу не призводить до збою всієї системи; легше ізолювати проблеми	Нижча: відмова в одному компоненті може призвести до збою всієї системи; складніше ізолювати помилки
Масштабованість	Горизонтальна та гнучка: окремі сервіси можуть масштабуватися незалежно від інших, дозволяючи ефективно використовувати ресурси та адаптуватися до навантаження	Переважно вертикальна: для збільшення продуктивності зазвичай потрібно масштабувати весь застосунок. Горизонтальне масштабування може бути складним і менш ефективним
Ремонтопридатність	Висока: невеликі, незалежні сервіси легко змінювати, тестувати та розгортати.	Нижча: зміни в одному компоненті можуть вимагати перекомпіляції та повторного розгортання всього застосунку

Сервісно-орієнтована архітектура виграє у моноліта за всіма трьома критеріями: надійності, масштабованості та ремонтопридатності. Це робить SOA привабливішою для сучасних складних систем, які вимагають високої стійкості, гнучкості до навантажень та швидких і безпечних змін.

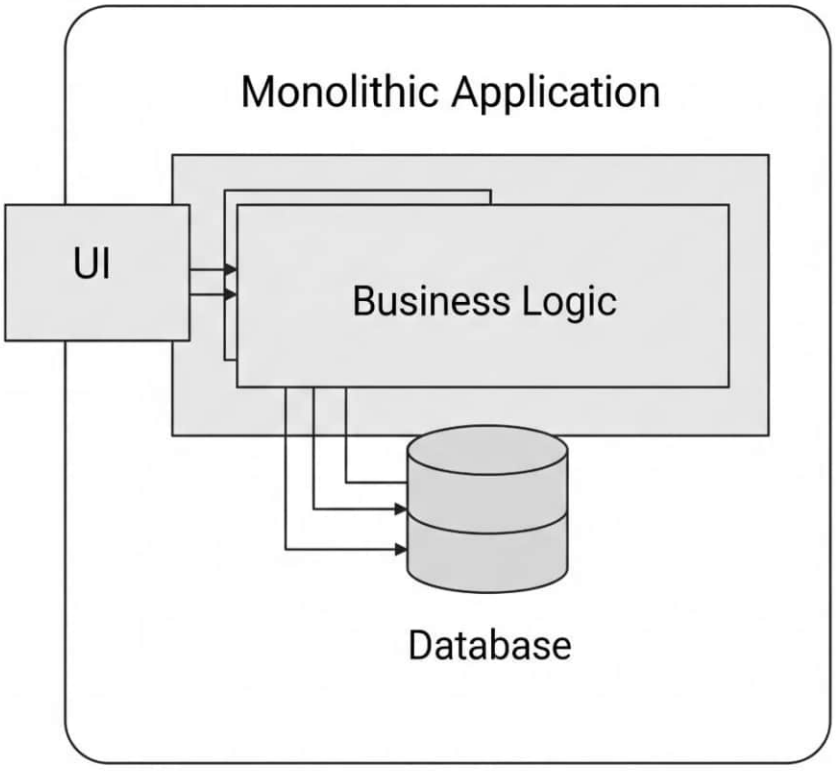

Рисунок 2.4 — Структура монолітної архітектури ПЗ

2.1.4 Опис взаємодії між класами MVC на клієнті

Щоб організувати роботу клієнтської частини програмного продукту, обрано архітектурний патерн MVC. Таке рішення (рис. 2.5) дозволяє чітко розмежувати відповідальність між тим, як виглядають дані, як вони обробляються, і як система реагує на дії користувача. Коли кожен компонент займається своєю справою, систему стає набагато простіше підтримувати,

тестувати та розвивати в майбутньому, що особливо важливо для складних проєктів.

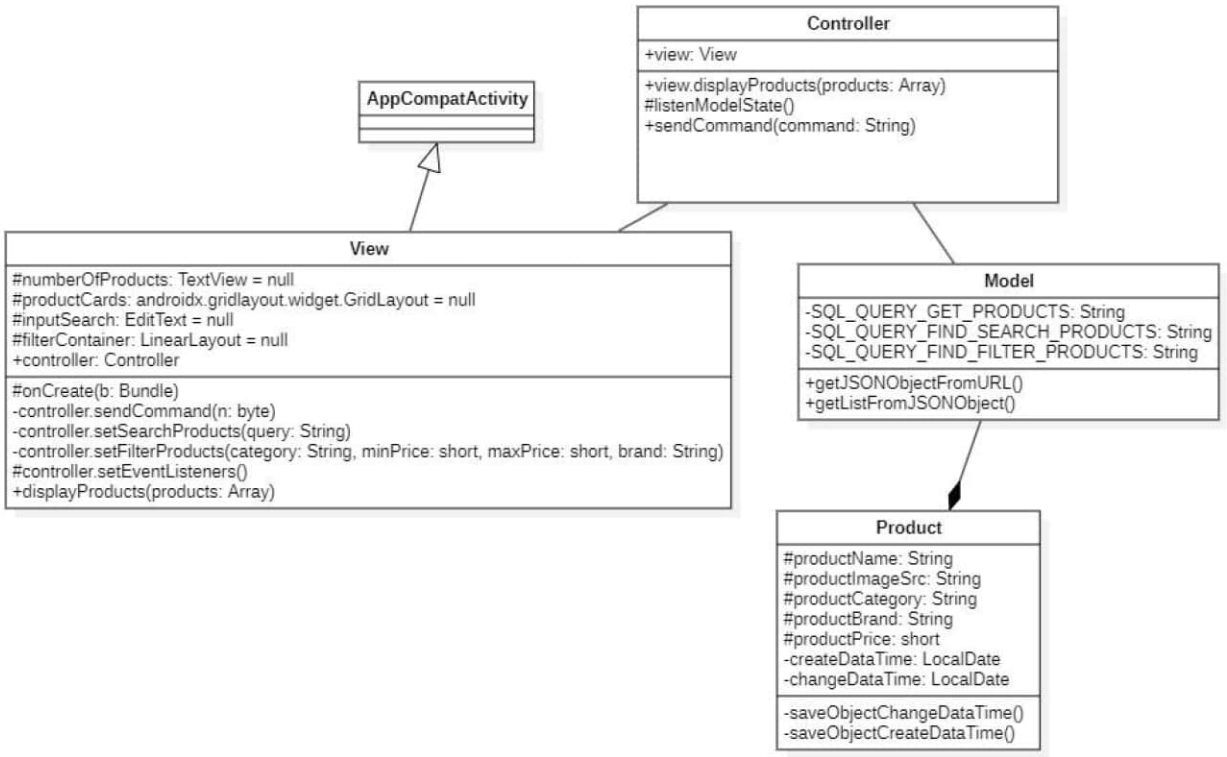

Рисунок 2.5 — UML-діаграма взаємодії MVC

В основі їхньої взаємодії лежить принцип мінімальної зв'язності. Кожен компонент має свою чітку роль. View займається виключно тим, що показує користувачеві графічний інтерфейс та реагує на його дотики чи кліки. Він не обробляє логіку самостійно, а натомість передає інформацію про всі дії користувача далі, до Controller.

Коли Controller отримує сигнал від View, він перетворює дію користувача на конкретну команду. Саме цю команду він і скеровує до Model. Model, у свою чергу, тримає в собі всі дані та бізнес-логіку. Він знає, як виконувати обчислення та працювати з сутностями, і при цьому йому абсолютно невідомо, як виглядає інтерфейс. Щойно Model змінює свій

внутрішній стан, вона повідомляє про це своїх спостерігачів. Controller, будучи одним з них, одразу дізнається про оновлення і дає View вказівку оновити екран, щоб користувач побачив актуальну інформацію.

Збирається вся система в єдине ціле у головній точці входу програми. Там послідовно створюються екземпляри Model, Controller та View. Щоб компоненти не були жорстко зв'язані, використано впровадження залежностей: передано Model в Controller, а Controller — у View. В результаті View спілкується лише з Controller, а Controller — з Model, тоді як Model залишається повністю автономною.

Під час роботи Model може виконувати операції, що тривають довго, наприклад, звертатися до віддалених серверів. Якщо викликати таку операцію напряму з графічного потоку, інтерфейс програми «зависне», чого не можна допускати. Щоб уникнути цього, реалізований асинхронний підхід. Щоразу, коли починається довготривала операція, Controller запускає її в окремому фоновому потоці. Завдяки цьому графічний потік залишається вільним і програма миттєво реагує на дії користувача.

2.1.5 Опис взаємодії між класами монолітної архітектури сервері

Структура монолітної архітектури (рис. 2.6) серверної частини складається з компонентів API і Logic. Головна функція UI прийняти HTTP-запит, обробити його і викликати функції завантаження, пошуку та фільтрації товарів у Logic, який відповідає за взаємодію з MySQL. Цей клас робить SQL-запити до БД. А MySQL повертає результати запитів з 3 таблиць.

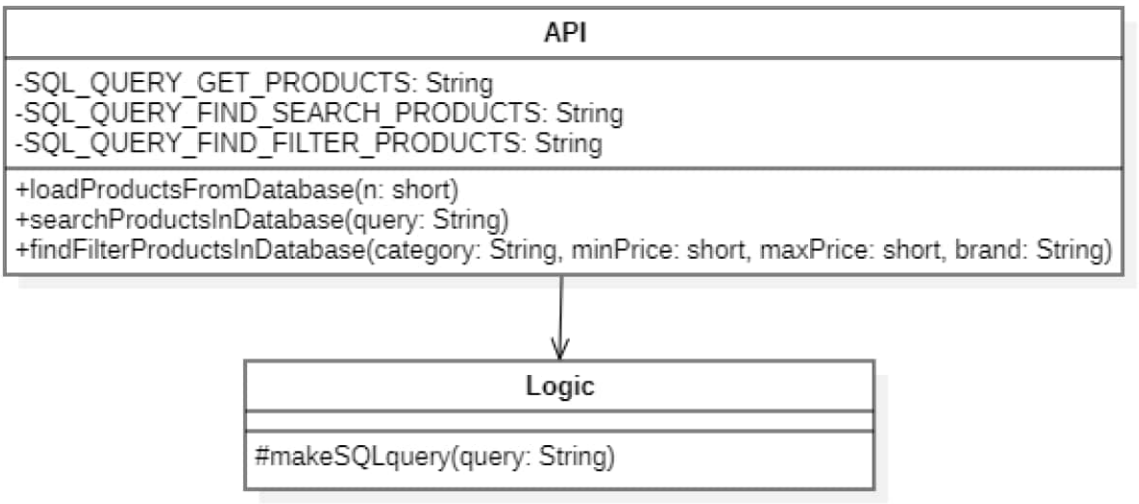

Рисунок 2.6 — UML-діаграма монолітної архітектури серверу

2.1.6 Модель даних і база даних

Вибір реляційної моделі даних (рис. 2.7) для високонавантаженого інтернет-каталогу є технічно обґрунтованим рішенням, що базується на її фундаментальних властивостях. По-перше, гарантування цілісності та консистентності даних забезпечується на рівні ядра системи завдяки транзакційній моделі ACID, що є критично важливим для управління цінами та товарними залишками. По-друге, нормалізована структура дозволяє адекватно моделювати складні ієрархічні зв'язки між сутностями, уникати аномалій оновлення та виконувати довільні реляційні запити для реалізації складної бізнес-логіки. Високе навантаження на читання ефективно нівелюється перевіреними архітектурними патернами, зокрема застосуванням індексації, кешування та масштабуванням через реплікацію.

Рисунок 2.7 — Структура реляційної моделі даних[11]

У базі даних products_computer_components, створеній на локальному сервері MySQL, було реалізовано структуру для зберігання інформації про комп'ютерні комплектуючі. Зокрема, було створено та наповнено даними три таблиці: cpus (472 записи), gpus (427 записів) та rams (247 записів). Загальна кількість записів у таблицях складає 1,146. Управління базою даних та її об'єктами здійснювалося за допомогою веб-інтерфейсу phpMyAdmin.

11 Рисунок залучений із зовнішнього джерела: https://surl.lu/arfesw

2.2 Розробка алгоритмів обробки високого навантаження

2.2.1 Стратегії оптимізації продуктивності та обробки високого навантаження

Масштабування є ключовим механізмом для обробки зростаючого навантаження. Існують два фундаментальні підходи:

- Вертикальне масштабування (Scale-Up): Передбачає збільшення потужності існуючого сервера (CPU, RAM, сховище). Цей підхід є відносно простим в реалізації, але має фізичні та фінансові обмеження.
- Горизонтальне масштабування (Scale-Out): Полягає у додаванні нових серверів (інстансів) до системи та розподілі навантаження між ними. Цей підхід забезпечує значно більшу гнучкість та потенціал для росту, але вимагає використання балансувальників навантаження та розробки додатків з урахуванням розподіленої природи.

2.2.2 Алгоритми обробки високого навантаження

Для забезпечення стабільної роботи системи в умовах високих навантажень розробляється комплекс алгоритмічних рішень, спрямованих на оптимізацію ключових операцій. Цей розділ присвячено детальному опису алгоритмів, що лежать в основі функціоналу пошуку, фільтрації та взаємодії мобільного застосунку з віддаленим сервером. Розглядаються стратегії оптимізації продуктивності, включаючи методи масштабування та кешування, що дозволяють мінімізувати час відгуку та раціонально використовувати обчислювальні ресурси.

2.2.3 Обробка HTTP-запитів

Взаємодія між клієнтським мобільним застосунком та серверною частиною реалізується через програмний інтерфейс додатку (API), що функціонує на основі протоколу HTTP. Для отримання даних з сервера застосунок відправляє HTTP GET-запити до відповідних кінцевих точок (endpoints) RESTful API. Наприклад, для завантаження списку товарів виконується запит до ендпоінту /products, а для отримання даних конкретного товару – до /products/{id}. Параметри для фільтрації та пошуку передаються як параметри запиту (query parameters), наприклад /products?category=cpus&price_max=5000. Сервер обробляє ці запити, взаємодіє з базою даних і повертає результат у форматі JSON, який є легкою та стандартизованою структурою для обміну даними. Клієнтський застосунок, у свою чергу, здійснює парсинг отриманого JSON-об'єкта та відображає дані у відповідних елементах інтерфейсу. Для обробки можливих мережевих помилок реалізовано механізм перевірки коду стану HTTP (наприклад, 200 OK, 404 Not Found, 500 Internal Server Error) та відповідної реакції інтерфейсу.

2.3 Створення інтерфейсу користувача для інтернет-каталогу товарів

2.3.1 Вимоги до користувацького інтерфейсу UI

Загальні вимоги:

- Інтерфейс має бути адаптивним для мобільних пристроїв;
- Дизайн має бути human center design, з акцентом на контенті;
- Кольорова схема повинна використовувати переважно нейтральні кольори (білий, сірий) з яскравими акцентами для інтерактивних елементів, червоні кнопки.

Компоненти інтерфейсу:

- Меню: Має містити іконку пошуку для швидкого доступу до функції пошуку; Має містити іконку «бургера» для виклику головного меню навігації.
- Основна частина: Повинен відображатися заголовок поточної категорії з вказанням загальної кількості товарів у ній; список товарів має бути представлений у вигляді сітки (grid); кожен товар у сітці повинен відображатися як картка.
- Картка товару: Має містити місце для зображення товару; під зображенням повинна відображатися назва товару; має містити кнопку із закликом до дії «Переглянути».
- Пагінація: Під списком товарів має бути блок пагінації для навігації по сторінках; пагінація повинна включати: кнопки для переходу до першої та останньої сторінки («Перша», «Остання»). кнопки для переходу до попередньої та наступної сторінки (іконки стрілок); номери сторінок, причому поточна сторінка має бути візуально виділена.

2.3.2 Визначення кольорової палітри

В процесі створення дизайну UI підібрані кольори і винесені окремою сторінкою в Figma. Макет кольорової палітри (рис. 2.9) включає кольори: сірий для тексту, красний для кнопок і чорний для меню.

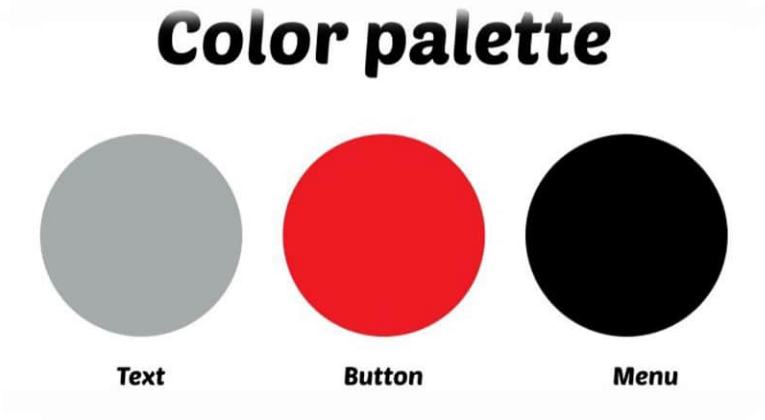

Рисунок 2.9 — Кольорова палітра для дизайну

По-перше, простота палітри (3 основні кольори) робить інтерфейс інтуїтивно зрозумілим. Користувачеві не потрібно розшифровувати значення різних кольорів; він швидко засвоює правило: сірий – інформація, червоний – дія. У високонавантажених системах, де користувач переглядає десятки сторінок, простота та передбачуваність інтерфейсу знижують втому та покращують загальний досвід взаємодії.

По-друге, яскравий червоний колір використовується виключно для кнопок. Це класичний і ефективний прийом. На тлі нейтральної схеми червона кнопка миттєво привертає погляд і спонукає до дії «Прибрати». Для високонавантаженого каталогу це критично, оскільки такий підхід допомагає підвищити коефіцієнт конверсії, чітко направляючи користувача до цільової дії.

По-третє, використання сірого тексту на білому фоні забезпечує хороший, але не надмірно різкий контраст, що комфортно для тривалого читання. Чорний колір для елементів меню також гарантує відмінну видимість.

Це відповідає базовим вимогам доступності (accessibility), роблячи сайт зручним для ширшого кола користувачів.

2.3.3 Вибір стилю шрифту

Для забезпечення візуальної ієрархії та високої читабельності інтерфейсу визначається єдиний стиль текстових елементів. Основним шрифтом обирається сучасний гротеск без засічок (sans-serif), такий як Roboto або Inter, що є стандартним для платформи Android та оптимізований для відображення на екранах різної щільності пікселів. Для заголовків екранів та назв товарів встановлюється товщина шрифту medium та розмір кегля 16pt. Для основного тексту, описів та характеристик використовується стандартне накреслення regular з розміром 14pt. Допоміжний текст, такий як ціни або інформаційні підписи, має менший розмір 12pt для візуального розмежування. Колір тексту визначається згідно з раніше розробленою палітрою: основний текст має темно-сірий колір для зменшення навантаження на очі, тоді як акцентні елементи можуть використовувати чорний або червоний кольори.

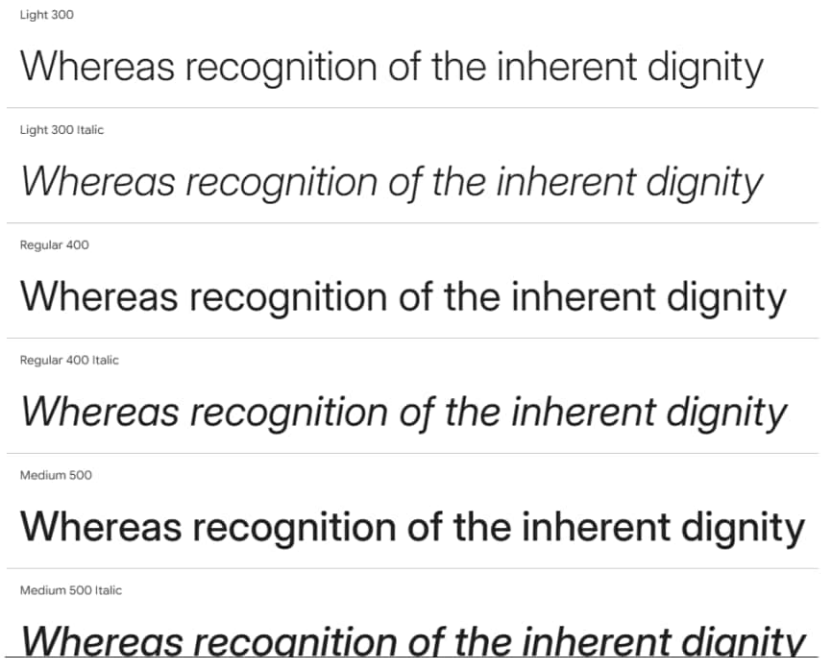

Рисунок 2.9 – Товщини шрифту Inter

2.3.4 Вибір бібліотеки іконок

Для візуального представлення функцій та дій в інтерфейсі застосовується єдиний набір іконок. Обрана бібліотека іконок Font Awesome (рис. 2.10), який гармонійно поєднується із загальним дизайном застосунку та не перевантажує інтерфейс зайвими деталями. Іконки проєктуються у векторному форматі (SVG), що забезпечує їх чітке відображення на екранах з будь-якою роздільною здатністю без втрати якості та дозволяє легко змінювати їх колір програмно. Основні іконки, такі як пошук і фільтр мають стандартизований та впізнаваний вигляд, що відповідає гайдлайнам Material Design. Це сприяє інтуїтивному розумінню їх призначення користувачем та прискорює взаємодію з додатком.

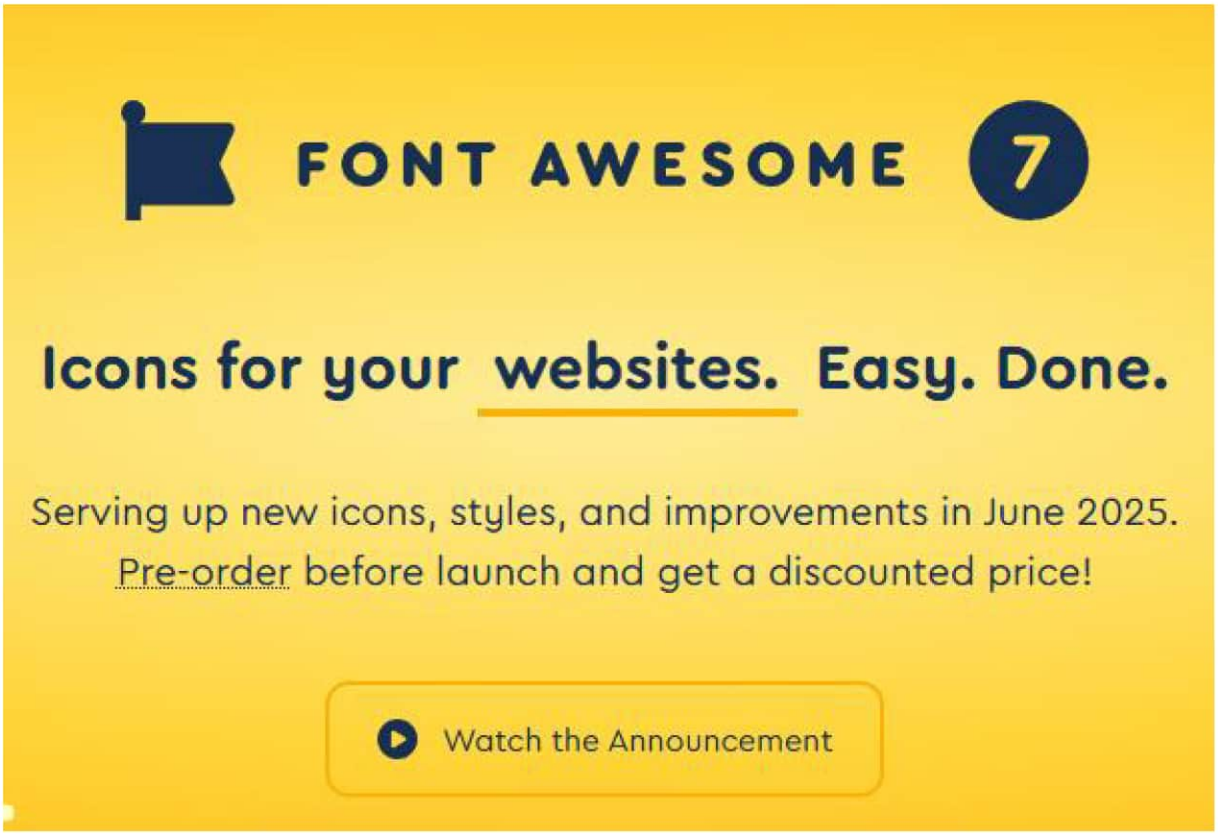

Рисунок 2.10 – Бібліотека іконок Font Awesome

2.3.5 Створення дизайну основного екрану

На макеті зображено екран мобільного додатку-каталогу відеокарт. У верхній частині інтерфейсу він розміщує темну панель із двома іконками — пошуку (лупа) та фільтрації (три горизонтальні лінії з крапками). Під панеллю він виводить заголовок «Всі комплектуючі», який інформує користувача про кількість доступних товарів у каталозі.

Основну частину екрана він формує у вигляді сітки з двома колонками, де кожен товар представлено у вигляді картки. Картка містить прямокутник-заглушку для зображення, демонстраційний текст із назвою товару та червону кнопку з написом «Зберегти», що привертає увагу користувача.

У нижній частині макету він реалізує панель пагінації. Вона включає посилання на першу й останню сторінки, кнопки переходу назад і вперед, а також номери сторінок. Активна сторінка візуально виділяється, що полегшує навігацію.

Інтерфейс виконано у простому й зрозумілому стилі, орієнтованому на зручність використання на мобільних пристроях.

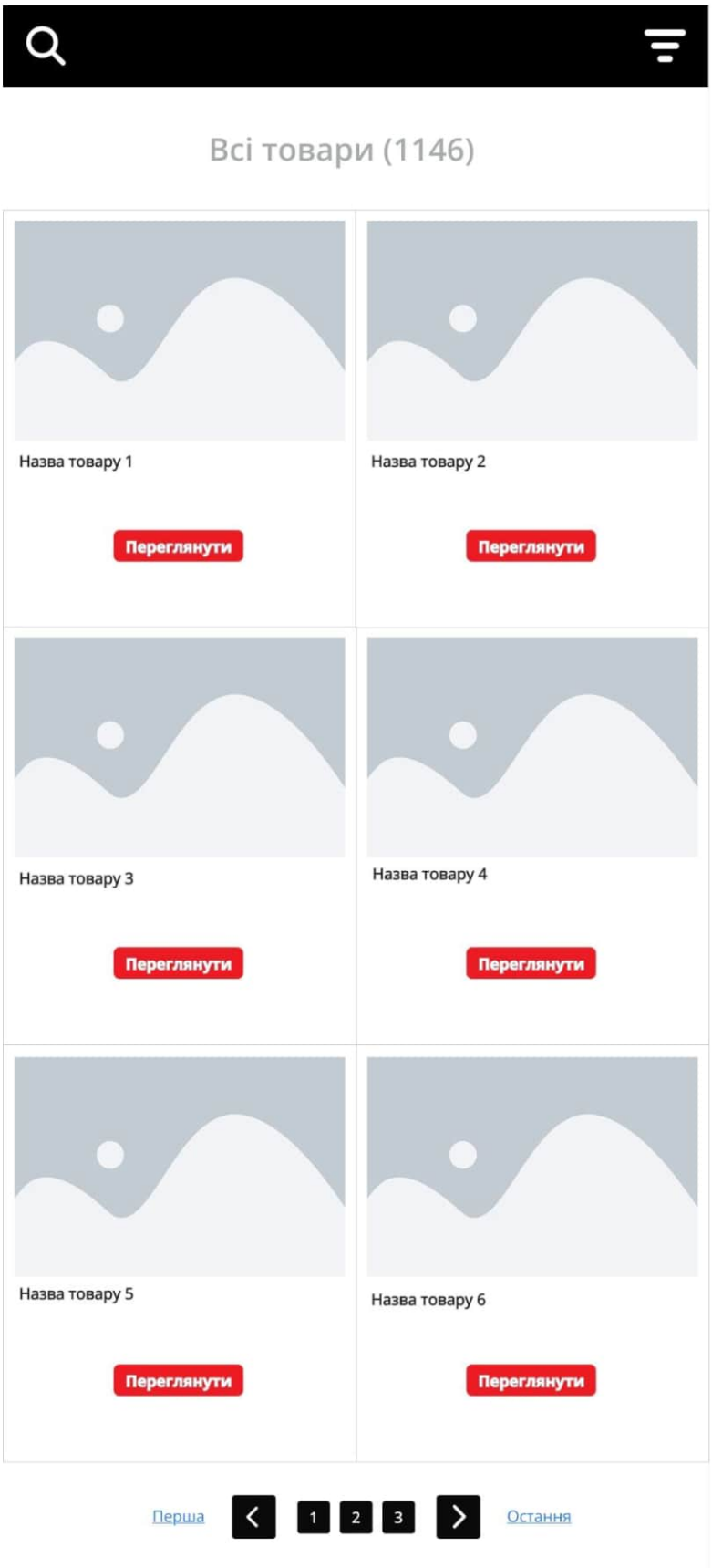

Рисунок 2.11 — Інтерфейс користувача високонавантаженого інтернет-каталогу товарів під Android

2.3.6 Створення макету дизайну меню

Дизайн меню навігації розробляється у вигляді висувної бічної панелі (Navigation Drawer), що активується натисканням на іконку «бургера» у верхній частині екрана. Такий підхід дозволяє розмістити значну кількість пунктів навігації, не захаращуючи при цьому основний екран. Верхня частина меню містить заголовок або логотип застосунку. Нижче розташовується вертикальний список основних розділів каталогу або функціональних блоків (наприклад, «Головна», «Каталог», «Обране», «Профіль», «Налаштування»). Кожен пункт меню складається з іконки та текстового підпису, що забезпечує швидку ідентифікацію. Активний розділ, в якому перебуває користувач, візуально виділяється іншим кольором фону або тексту для покращення навігаційної орієнтації.

2.3.7 Створення дизайну макету контенту

Дизайн сторінки контенту, що відображає деталі конкретного товару, проєктується з метою надання користувачеві вичерпної інформації у структурованому та легкому для сприйняття вигляді. Верхню частину екрана займає галерея зображень товару з можливістю гортання. Безпосередньо під нею розташовується назва товару, його ціна та статус наявності, що є ключовою інформацією для прийняття рішення. Нижче розміщуються окремі логічні блоки: «Характеристики», «Опис», «Відгуки». Кожен блок може бути згортаємим для економії простору. У блоці «Характеристики» дані представлені у вигляді двоколонкової таблиці (параметр-значення) для зручності порівняння. Кнопка цільової дії, наприклад «Додати до кошика», є фіксованою у нижній частині екрана, залишаючись видимою при прокручуванні сторінки.

2.3.8 Створення UX пошуку і фільтра

Дизайн екрана пошуку реалізовано як окрему сторінку, що активується при натисканні на відповідну іконку. Вона містить текстове поле вводу у верхній частині та може відображати історію попередніх запитів або популярні пошукові терміни. Результати пошуку виводяться динамічно в реальному часі

у вигляді списку товарів, аналогічного до основного каталогу. Інтерфейс фільтрації проєктується як модальне вікно або бічна панель, що з'являється поверх поточного екрана каталогу. Фільтри групуються за логічними категоріями: «Категорія» (вибір зі списку), «Ціна» (два поля вводу «від» і «до» або повзунок), «Бренд» (список з можливістю множинного вибору). У нижній частині панелі фільтрів розташовані дві кнопки: «Застосувати» для підтвердження вибору та «Скинути» для повернення до початкових значень.

2.4 Налаштування середовищ розробки і створення проєктів

2.4.1 Вибір та інсталяція інтегрованого середовища розробки (IDE)

Цей розділ описує процес підготовки та конфігурації програмного оточення, необхідного для реалізації проєкту. Правильне налаштування інструментарію є ключовою передумовою для ефективного процесу розробки, тестування та подальшої підтримки програмного продукту. Розглядається вибір та інсталяція інтегрованого середовища розробки (IDE), налаштування пакетів для розробки під платформу Android (SDK), а також конфігурація системи управління версіями для контролю змін у кодовій базі.

2.4.2 Налаштування Android SDK та інструментів

Після інсталяції Android Studio здійснюється налаштування Android SDK (Software Development Kit) за допомогою вбудованого SDK Manager. Обирається та встановлюється цільова версія Android API, на яку орієнтований застосунок, а також кілька попередніх версій для забезпечення зворотної сумісності. Додатково встановлюються необхідні інструменти збірки (Build-Tools), інструменти платформи (Platform-Tools) та образи системи для Android Emulator. Створюється та налаштовується віртуальний пристрій (AVD) з характеристиками, що відповідають типовому сучасному смартфону, для тестування та налагодження застосунку безпосередньо в середовищі розробки.

2.4.3 Система управління версіями

Для контролю за змінами вихідного коду, спільної роботи над проєктом та забезпечення можливості відкату до попередніх стабільних версій використовується розподілена система управління версіями Git. Створюється локальний Git-репозиторій в кореневій директорії проєкту. Також налаштовується віддалений репозиторій на одній з хмарних платформ (наприклад, GitHub, GitLab або Bitbucket). Початковий стан проєкту комітиться та відправляється до віддаленого репозиторію. У файлі .gitignore конфігуруються правила для ігнорування файлів, що генеруються

середовищем розробки та системою збірки, щоб уникнути їх потрапляння до системи контролю версій.

2.4.4 Створення нового Android-проєкту

Новий проєкт створюється безпосередньо в середовищі Android Studio за допомогою вбудованого майстра «New Project». В якості шаблону обирається «Empty Activity», що створює мінімальну базову структуру для подальшої розробки. Під час створення проєкту задаються ключові параметри: назва застосунку, ім'я пакету (package name), що є унікальним ідентифікатором застосунку, місце збереження проєкту, а також мова програмування. В рамках даної роботи в якості основної мови програмування обирається Java , зважаючи на її зрілість, величезну екосистему та широку базу знань. Також встановлюється мінімальна необхідна версія Android SDK (minSdkVersion), що визначає, на яких пристроях зможе функціонувати застосунок

2.4.5 Конфігурація інструментів збірки

Конфігурація процесу збірки здійснюється через файли build.gradle, які використовує система автоматичної збірки Gradle. У файлі build.gradle на рівні модуля (app) налаштовуються основні параметри, такі як applicationId, minSdkVersion, targetSdkVersion та versionCode/versionName. В цьому ж файлі, у секції dependencies, прописуються всі зовнішні бібліотеки та залежності, що використовуються у проєкті. Це дозволяє Gradle автоматично завантажувати, компілювати та підключати необхідні сторонні компоненти під час збірки проєкту.

2.4.6 Підключення та налаштування додаткових інструментів

Для підвищення якості коду та ефективності розробки до проєкту підключаються додаткові інструменти. Налаштовується статичний аналізатор коду Lint, вбудований в Android Studio, який допомагає виявляти потенційні помилки, проблеми з продуктивністю та вразливості безпосередньо в процесі написання коду. Також може бути інтегровано інструменти для автоматичного форматування коду, такі як ktlint або Checkstyle, для підтримки єдиного стилю кодування в усьому проєкті. Для спрощення роботи з HTTP-запитами до

проєкту підключається спеціалізована бібліотека, наприклад, Retrofit або Volley.

2.4.7 Управління залежностями

Управління залежностями є централізованим та здійснюється через систему Gradle. Всі необхідні бібліотеки, такі як бібліотеки підтримки AndroidX, бібліотеки для роботи з мережею, обробки зображень чи тестування, декларуються у файлі build.gradle(:app). Використання системи управління залежностями дозволяє уникнути необхідності вручну завантажувати та підключати JAR-файли, спрощує процес оновлення бібліотек до нових версій та вирішує потенційні конфлікти між різними залежностями. Кожна залежність вказується у форматі implementation 'group:name:version', що забезпечує прозорість та відтворюваність збірки проєкту.

2.5 Програмування клієнтської і серверної частини Android-застосунку

2.5.1 Програмування клієнтської частини (Frontend)

Програмування клієнтської частини здійснюється мовою Java в середовищі Android Studio. Розробка ведеться відповідно до архітектурного шаблону MVC (Model-View-Controller), що дозволяє чітко розмежувати логіку представлення даних, бізнес-логіку та управління подіями. Компонент View реалізується за допомогою XML-файлів розмітки, де декларативно описується структура та зовнішній вигляд кожного екрана. Класи Activity та Fragment виступають у ролі Controller, обробляючи дії користувача, такі як натискання на кнопки чи введення тексту, та ініціюючи відповідні запити до Model. Компонент Model інкапсулює логіку роботи з даними, включаючи відправку мережевих запитів до API, обробку отриманих даних та їх підготовку для відображення. Для виконання асинхронних мережевих операцій використовуються механізми, що не блокують основний потік UI, забезпечуючи плавність та чутливість інтерфейсу.

2.5.2 Програмування серверної частини (Backend)

Серверна частина, відповідальна за основну бізнес-логіку, реалізується у вигляді монолітного додатку. Дана архітектура була обрана для спрощення розробки та розгортання на початковому етапі. Backend розробляється з використанням скриптової мови PHP, яка добре інтегрується з веб-сервером Apache та системою управління базами даних MySQL. Створюється набір скриптів, що реалізують RESTful API. Кожен скрипт відповідає за певну кінцеву точку (endpoint) та обробляє відповідні HTTP-запити (GET, POST тощо). Логіка включає валідацію вхідних даних від клієнта, формування SQL-запитів до бази даних products_computer_components, обробку результатів та їх перетворення у формат JSON для відправки відповіді клієнту. Вся взаємодія з базою даних інкапсульована в окремому класі, що відповідає за з'єднання та виконання запитів, для підвищення модульності коду

2.5.3 Взаємодія клієнт-сервер

Взаємодія між клієнтським Android-застосунком та серверною частиною здійснюється за протоколом HTTP. Клієнт формує та відправляє HTTP-запити до API, розгорнутого на сервері. Для цього на клієнті використовується спеціалізована бібліотека, яка спрощує процес створення запитів та обробки відповідей. Сервер, отримавши запит, виконує необхідну логіку та повертає дані у форматі JSON. Дані, що передаються, включають списки товарів, деталі окремого товару, результати пошуку та фільтрації. Для забезпечення безпеки обміну даними у промисловій версії системи передбачається перехід на захищений протокол HTTPS, що забезпечує шифрування трафіку.

2.6 Інструкція користувача

2.6.1 Вступ

Дана інструкція розроблена для ознайомлення користувачів з основними функціональними можливостями мобільного застосунку «Вискоконавантаженрий інтернет-каталог товарів». Документ містить покрокові описи для виконання ключових операцій, таких як навігація по каталогу, пошук, фільтрація товарів та перегляд їх характеристик. Виконання рекомендацій, наведених у цій інструкції, дозволить максимально ефективно та комфортно використовувати всі переваги розробленого програмного продукту.

2.6.2 Початок роботи

Для початку роботи з додатком необхідно завантажити його з офіційного магазину додатків та встановити на ваш мобільний пристрій під управлінням ОС Android. Після першого запуску застосунок автоматично завантажить головний екран з каталогом товарів. Для роботи основних функцій не вимагається обов'язкова реєстрація чи авторизація. Переконайтеся, що ваш пристрій має активне підключення до мережі Інтернет для завантаження та оновлення даних каталогу.

2.6.3 Навігація по каталогу

Основний екран застосунку представляє собою каталог товарів у вигляді сітки. Для перегляду доступних товарів використовуйте вертикальну прокрутку екрана. Якщо товарів у категорії більше, ніж вміщується на одному екрані, у нижній частині з'явиться панель пагінації. Для переходу між сторінками використовуйте кнопки зі стрілками («назад» та «вперед») або натискайте безпосередньо на номер сторінки. Для перегляду детальної інформації про товар просто натисніть на його картку у списку.

2.6.4 Пошук товарів

Для швидкого пошуку конкретного товару за його назвою використовуйте іконку пошуку (у вигляді лупи), розташовану у верхній

частині екрана. Після натискання на іконку відкриється поле для вводу тексту. Введіть назву або частину назви товару, який ви шукаєте, і натисніть кнопку пошуку на віртуальній клавіатурі. Система автоматично відобразить список товарів, що відповідають вашому запиту. Щоб повернутися до повного каталогу, очистіть поле пошуку або скористайтеся системною кнопкою «Назад».

2.6.5 Фільтрація товарів

Для звуження списку товарів за певними критеріями використовуйте функцію фільтрації, доступну за іконкою меню («бургер» або іконка фільтра). Після її натискання відкриється панель з доступними параметрами фільтрації, такими як категорія, ціновий діапазон та бренд. Виберіть необхідні значення (наприклад, встановіть бажаний діапазон цін та оберіть один чи декілька брендів). Після вибору всіх критеріїв натисніть кнопку «Застосувати». Список товарів на головному екрані автоматично оновиться, відображаючи лише ті позиції, що відповідають заданим умовам.

2.6.6 Додаткові функції

Крім основних функцій перегляду, пошуку та фільтрації, застосунок може містити додаткові можливості. Наприклад, на сторінці детального перегляду товару користувач має можливість переглядати відгуки інших покупців, що допомагає у прийнятті рішення. Також може бути реалізована функція додавання товарів до списку «Обраного» для швидкого доступу до них у майбутньому.

2.6.7 Вирішення типових проблем

У випадку, якщо товари не завантажуються, перш за все перевірте наявність стабільного інтернет-з'єднання на вашому пристрої. Якщо проблема не зникає, спробуйте перезапустити застосунок. Якщо пошук або фільтрація не дають результатів, спробуйте спростити запит або розширити критерії фільтрації. Переконайтеся, що у вас встановлена остання версія застосунку, оскільки оновлення можуть містити виправлення помилок та покращення стабільності.

2.7 Тестування розробленого програмного забезпечення

2.7.1 Мета та завдання тестування

Основною метою тестування є забезпечення високої якості, надійності та продуктивності розробленого програмного забезпечення. До ключових завдань входить: перевірка коректності реалізації всього заявленого функціоналу (пошук, фільтрація, перегляд каталогу); виявлення та документування помилок у програмному коді та логіці роботи; оцінка продуктивності застосунку в умовах, наближених до реальних; перевірка стабільності роботи та коректної обробки помилкових ситуацій (наприклад, відсутність мережевого з'єднання); оцінка зручності та інтуїтивності користувацького інтерфейсу.

2.7.2 Методологія тестування

Тестування проводиться з використанням методології «сірої скриньки» (gray-box testing), що поєднує елементи тестування «чорної скриньки» (перевірка функціоналу без знання внутрішньої структури коду) та «білої скриньки» (аналіз коду для написання більш глибоких тестів). Функціональне тестування виконується на основі визначених вимог та сценаріїв використання. Нефункціональне тестування спрямоване на перевірку продуктивності, надійності та зручності використання.

2.7.3 Планування тестування

Процес тестування планується та розбивається на кілька етапів. На початковому етапі проводиться модульне тестування окремих функцій та класів. Далі слідує інтеграційне тестування для перевірки коректності взаємодії між клієнтською та серверною частинами. Після цього виконується системне тестування всього застосунку як єдиного цілого. Завершальним етапом є приймальне тестування (User Acceptance Testing), яке може проводитись фокус-групою для оцінки зручності використання. Для кожного етапу розробляються тестові сценарії (test cases) з очікуваними результатами.

2.7.4 Види тестування та їх реалізація

Всі виявлені під час тестування дефекти та невідповідності документуються у системі відстеження помилок (bug tracking system). Кожен звіт про помилку містить детальний опис проблеми, кроки для її відтворення, інформацію про середовище (версія ОС, модель пристрою) та пріоритет. Після виправлення дефекту розробником проводиться повторне регресійне тестування, щоб переконатися, що виправлення не спричинило нових помилок в інших частинах системи. За результатами всіх етапів тестування складається фінальний звіт, що містить аналіз якості продукту та рекомендації щодо його готовності до випуску.

2.7.5 Обробка результатів тестування

Всі виявлені під час тестування дефекти та невідповідності документуються у системі відстеження помилок (bug tracking system). Кожен звіт про помилку містить детальний опис проблеми, кроки для її відтворення, інформацію про середовище (версія ОС, модель пристрою) та пріоритет. Після виправлення дефекту розробником проводиться повторне регресійне тестування, щоб переконатися, що виправлення не спричинило нових помилок в інших частинах системи. За результатами всіх етапів тестування складається фінальний звіт, що містить аналіз якості продукту та рекомендації щодо його готовності до випуску.

ВИСНОВКИ

У результаті виконання завдання кваліфікаційної роботи була досягнута її головна мета: розроблено проєкт високонавантаженого мобільного інтернет-каталогу товарів, що функціонує на платформі Android. Спроєктована клієнт-серверна архітектура системи була цілеспрямовано оптимізована для стабільної роботи в умовах високих навантажень, забезпечення відмовостійкості та ефективної реалізації задач пошуку та фільтрації товарів, що повністю відповідає поставленим вимогам.

На першому етапі роботи було проведено комплексний аналіз предметної області. Було досліджено існуючі високонавантажені системи, такі як OLX, Rozetka та Prom, для виявлення ключових архітектурних викликів та передових практик їх вирішення. Особливу увагу було приділено аналізу архітектурних шаблонів, зокрема монолітного, сервіс-орієнтованого (SOA) та мікросервісного. На основі порівняльного аналізу було обґрунтовано вибір MVC і монолітиної архітектури як найбільш гнучкого та масштабованого рішення для поставленої задачі. Даний підхід дозволяє незалежно розробляти, розгортати та масштабувати окремі компоненти системи, що є критично важливим для забезпечення високої надійності та ремонтопригодності.

Наступним кроком стало проєктування серверної частини (backend). В її основу було покладено реляційну модель даних, реалізовану у вигляді схеми бази даних products_computer_components з таблицями для зберігання інформації про окремі категорії товарів (cpu, gpu, ram, power_supply, hdd, mother_board). Вибір реляційної СУБД обґрунтовується необхідністю забезпечення цілісності та консистентності даних, що є фундаментальною вимогою для систем. Для подальшої оптимізації роботи з даними в рамках монолітної архітектури було запропоновано застосування шаблону MVC (Model-View-Controller), який дозволяє розділити операції читання та запису, тим самим оптимізуючи продуктивність під кожен тип навантаження окремо.

Паралельно з серверною частиною велася розробка клієнтської частини — мобільного застосунку для платформи Android. Було спроєктовано користувацький інтерфейс (UI), орієнтований на інтуїтивність та зручність використання. Дизайн UI, представлений у макеті, включає grid-сітку для відображення товарів, що забезпечує легкість візуального сканування, та механізм пагінації для ефективної роботи з великими обсягами даних. Кольорова палітра була обрана з урахуванням психології сприйняття: нейтральні кольори для контенту не відвертають увагу від товарів, а яскравий акцентний колір використовується для кнопок цільових дій, що направляє користувача та підвищує конверсію.

Спроєктована система повністю відповідає меті кваліфікаційної роботи, оскільки її архітектура дозволяє витримувати пікові навантаження, забезпечує швидкий відгук при пошуку та фільтрації, а також є гнучкою для подальшого розвитку. В якості потенційних напрямків для подальшої роботи можна розглянути реалізацію повноцінного прототипу, інтеграцію спеціалізованого пошукового рушія для реалізації більш складних пошукових запитів, а також розробку системи персоналізованих рекомендацій для підвищення залученості користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Rozetka. URL: https://play.google.com/store/apps/details?id=ua.com.rozetka.shop&hl=uk&gl=US (дата звернення: 27.04.2025)
2. Знайомство з ROZETKA Маркетплейс. URL: https://sellerhelp.rozetka.com.ua/p272-get-to-know-us.html (дата звернення: 28.04.2025)
3. Як замовити товар в інших продавців. URL: https://help.rozetka.com.ua/p/18-yak-zamovyty-tovar-v-inshyh-prodavciv/ (дата звернення: 29.04.2025)
4. OLX: Buy & Sell Near. URL: https://play.google.com/store/apps/details?id=com.olx.olx&hl=uk&gl=US (дата звернення: 30.04.2025)
5. Про OLX - Центр підтримки. URL: https://help.olx.ua/olxuahelp/s/article/%D0%BF%D1%80%D0%BE-olx-V35 (дата звернення: 02.05.2025)
6. OLX Доставка. URL: https://business.olx.ua/olx-dostavka/ (дата звернення: 03.05.2025)
7. Prom.ua — інтернет-покупки. URL: https://play.google.com/store/apps/details?id=ua.prom.b2c&hl=uk (дата звернення: 04.05.2025)
8. Все про Prom.ua, або як продавати на 1 млн на українському маркетплейсі Пром. URL: https://blog.keycrm.app/uk/vse-pro-prom-ua-abo-yak-prodavati-na-1-mln-na-prom/ (дата звернення: 05.04.2025)
9. Як купити товар? - EVOService - Prom.ua. URL: https://help.prom.ua/hc/uk/articles/360009358458-%D0%AF%D0%BA-

%D0%BA%D1%83%D0%BF%D0%B8%D1%82%D0%B8-%D1%82%D0%BE%D0%B2%D0%B0%D1%80 (дата звернення: 06.05.2025)

10. Kleppmann Martin. Designing Data-Intensive Applications [Літер. джерело]. – O'Reilly Media, 2017. – 616 с. – Спосіб доступу: URL: https://surl.li/nctauk (дата звернення: 08.05.2025)
11. Nygard Michael. Release It! URL: https://surl.li/wcvlyq (дата звернення: 13.05.2025)
12. Volodymyr Kozub. Problems and Solutions in Building Highly Loaded Software. URL: https://theamericanjournals.com/index.php/tajet/article/view/5995 (дата звернення: 18.05.2025)
13. Android Studio. URL: https://developer.android.com/studio (дата звернення: 20.05.2025)
14. Top Programming Languages 2024. URL: https://spectrum.ieee.org/top-programming-languages-2024 (дата звернення: 31.05.2025)
15. Java. URL: https://www.java.com/ (дата звернення: 23.05.2025)
16. Kotlin. URL: https://kotlinlang.org/ (дата звернення: 31.05.2025)
17. Adobe Photoshop. URL: https://www.adobe.com/products/photoshop.html (дата звернення: 03.06.2025)
18. Figma. URL: https://www.figma.com/ (дата звернення: 05.05.2025)
19. CorelDRAW.URL: https://www.coreldraw.com/ (дата звернення: 07.05.2025)
20. MySQL. URL: https://www.mysql.com/ (дата звернення: 10.05.2025)
21. PostgreSQL. URL: https://www.postgresql.org/ (дата звернення: 13.05.2025)

22. Oracle Database. URL: https://www.oracle.com/database/ (дата звернення: 15.05.2025)

23. JUnit. URL: https://junit.org/junit5/ (дата звернення: 16.05.2025)

ДОДАТОК А.
ПРОГРАМНИЙ КОД КЛІЄНСЬКОЇ ЧАСТИНИ

А.1 Програмний код модуля View.java

```
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.ProgressBar;
import android.widget.TextView;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import androidx.gridlayout.widget.GridLayout;
import java.util.ArrayList;
import your.package.name.R;

public class View extends AppCompatActivity {

    // UI elements
    protected TextView numberOfProducts;
    protected GridLayout productCards;
    protected EditText inputSearch;
    protected LinearLayout filterContainer;
    protected ImageButton searchEnter;
    protected Button filterApply;
    protected ProgressBar loadingIndicator;

    // View has a reference to the Controller
    protected Controller controller;

    // Commands for Controller
    private static final String GET_N_PRODUCTS = "load";
    private static final String GET_SEARCH_PRODUCTS = "search";
    private static final String GET_FILTER_PRODUCTS = "filter";
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // 1. Initialize the Controller
    controller = new Controller(this);

    // 2. Initialize UI elements
    numberOfProducts = findViewById(R.id.number_of_products);
    productCards = findViewById(R.id.product_cards);
    inputSearch = findViewById(R.id.input_search);
    searchEnter = findViewById(R.id.btn_search);
     filterApply = findViewById(R.id.btn_filter);
    loadingIndicator = findViewById(R.id.progress_bar);

    // 3. Set up event listeners
    setEventListeners();

    // 4. Initial request to get data when the applicaiton starts
    controller.sendCommand("load", 10, "", null);
}

// Logic for setting listeners
private void setEventListeners() {
    // Search enter on click listener
    searchEnter.setOnClickListener(v -> {
        String query = inputSearch.getText().toString();
        controller.sendCommand("search", 10, query, null);
    });

    // Filter apply on click
   filterApply.setOnClickListener(v -> {
        String query = inputSearch.getText().toString();

         List filterParameters = new ArrayList(3);
        String category = filterContainer.findByView(R.id.filter_category);
         String minPrice = filterContainer.findByView(R.id.filter_min_price);
         String maxPrice = filterContainer.findByView(R.id.filter_max_price);
         String brand = filterContainer.findByView(R.id.filter_brand);
```

```
            filterParametrs.add(category);
            filterParametrs.add(minPrice);
            filterParametrs.add(maxPrice);
            filterParametrs.add(brand);

            controller.sendCommand("filter", 10, "", filterParameters);
        });


    }

    // Method to display the list of products in the UI
    public void displayProducts(ArrayList<Product> products) {
        productCards.removeAllViews();

        if (products == null || products.isEmpty()) {
            numberOfProducts.setText("No products found");
            return;
        }

        numberOfProducts.setText("Products found: " + products.size());

        // Dynamically create and add product cards to the GridLayout
        for (Product product : products) {
            // for each product and added to productCards.
            TextView productView = new TextView(this);
            productView.setText(String.format("%s - $%d", product.getProductName(),
product.getProductPrice()));
            productView.setPadding(8, 8, 8, 8);
            productCards.addView(productView);
        }
    }

    public void showLoadingIndicator(boolean show) {
        loadingIndicator.setVisibility(show ? android.view.View.VISIBLE :
android.view.View.GONE);
    }

    public void showError(String message) {
        Toast.makeText(this, message, Toast.LENGTH_SHORT).show();
    }
```

```
}
```

А.2 Програмний код модуля Controller.java

```
import java.util.ArrayList;

public class Controller {

    // Added references to View and Model
    private View view;
    private Model model;

    // Constructor to initialize the connections
    public Controller(View view) {
        this.view = view;
        this.model = new Model();
    }

    // The main method for handling commands from the View.
    public void sendCommand(String command, short n, String query, ArrayList
filterPars) {
        if ("load".equals(command)) {
            Model.loadProductsFromServer(n);
        }

        if ("search".equals(command)) {
            Model.searchProductsFromServer(query);
        }

         if ("filter".equals(command)) {
            Model.filterProductsFromServer(filterPars);
        }
    }

    // Method to load the initial list of products
    private void loadProducts() {
        view.showLoadingIndicator(true); // Show the loading indicator
        model.fetchProductsFromAPI(new Model.OnDataReadyCallback() {
            @Override
            public void onSuccess(ArrayList<Product> products) {
                // Once data is received, pass it to the View for display
```

```
                view.runOnUiThread(() -> {
                    view.showLoadingIndicator(false);
                    view.displayProducts(products);
                });
            }

            @Override
            public void onFailure(Exception e) {
                view.runOnUiThread(() -> {
                    view.showLoadingIndicator(false);
                    view.showError("Failed to load data.");
                });
            }
        });
    }
}
```

А.3 Програмний код модуля Model.java

```
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

public class Model {

    // SQL-requests
    private static final String SQL_QUERY_GET_PRODUCTS = "SELECT * FROM products;";
    private static final String SQL_QUERY_SEARCH_PRODUCTS = "SELECT * FROM products
WHERE name LIKE ?;";
    private static final String SQL_QUERY_FILTER_PRODUCTS = "SELECT * FROM products
WHERE category = ? AND brand = ? AND price BETWEEN ? AND ?;";

    // Interface to pass results of an asynchronous operation to the Controller
    public interface OnDataReadyCallback {
        void onSuccess(ArrayList<Product> products);
        void onFailure(Exception e);
    }

    // Fetching data from the network and parsing it
    public void fetchProductsFromAPI(OnDataReadyCallback callback) {
        // Asynchronous network request
```

```
        new Thread(() -> {
            try {
                // Step 1: Get JSON from URL
                String jsonResponse = getJSONObjectFromURL();
                Thread.sleep(1500); // Simulate network latency

                // Step 2: Parse JSON into a list of products
                ArrayList<Product> products = getListFromJSONObject(jsonResponse);

                // Return the result to the main thread (a Handler would be needed in
a real app)
                callback.onSuccess(products);
            } catch (InterruptedException e) {
                callback.onFailure(e);
            }
        }).start();
    }

    // Method for searching
    public void findProducts(String query, OnDataReadyCallback callback) {
        new Thread(() -> {
             try {
                String jsonResponse = getJSONObjectFromURL(query);
                ArrayList<Product> allProducts = getListFromJSONObject(jsonResponse);

                // Filtering
                List<Product> filteredList = allProducts.stream()
                        .filter(p ->
p.getProductName().toLowerCase().contains(query.toLowerCase()))
                        .collect(Collectors.toList());

                Thread.sleep(500); // Simulate delay
                callback.onSuccess(new ArrayList<>(filteredList));

            } catch (Exception e) {
                callback.onFailure(e);
            }
        }).start();
    }

    // Request JSONObject from URL
```

```
    public static JSONArray getJSONObjectFromURL(String urlPath) throws IOException,
JSONException {
        HttpURLConnection urlConnection = null;

        URL url = new URL(urlPath);

        // Send GET request to web-server
        urlConnection = (HttpURLConnection) url.openConnection();
        urlConnection.setRequestMethod("GET");
        urlConnection.setReadTimeout(10000 /* milliseconds */ );
        urlConnection.setConnectTimeout(15000 /* milliseconds */ );
        urlConnection.setDoOutput(true);
        urlConnection.connect();

        BufferedReader br = new BufferedReader(new
InputStreamReader(url.openStream()));
        StringBuilder sb = new StringBuilder();

        String line;
        while ((line = br.readLine()) != null) {
            sb.append(line + "\n");
        }
        br.close();

        String jsonString = sb.toString();
        System.out.println("JSON: " + jsonString);

        return new JSONArray(jsonString);
    }
```

```
// Get List from JSONObject
public static List<Product> getListFromJSONObject(JSONArray jsonArray)
    {
        List<Product> products = new ArrayList<>();

        for(int i = 0; i < jsonArray.length(); i++)
        {
```

```
            Product product = new Product();

            try {
                product.name = jsonArray.getJSONArray(i).getString(0);
            } catch (JSONException e) {
                throw new RuntimeException(e);
            }
            try {
                product.src = jsonArray.getJSONArray(i).getString(1);
            } catch (JSONException e) {
                throw new RuntimeException(e);
            }
            try {
                product.price = jsonArray.getJSONArray(i).getString(2);
            } catch (JSONException e) {
                throw new RuntimeException(e);
            }
        }

        return products;
    }
}
```

А.4 Програмний код модуля Product.java

```
import java.time.LocalDateTime;

public class Product {
    protected String productName;
    protected String productImageSrc;
    protected String productCategory;
    protected String productBrand;
    protected short productPrice;
    private LocalDateTime createDateTime;
    private LocalDateTime changeDateTime;

    // Constructor to create an object
    public Product(String productName, String productImageSrc, String
productCategory, String productBrand, short productPrice) {
        this.productName = productName;
        this.productImageSrc = productImageSrc;
        this.productCategory = productCategory;
```

```
        this.productBrand = productBrand;
        this.productPrice = productPrice;
        saveObjectCreateDataTime(); // Set the creation time
    }

    // Private methods for date management
    private void saveObjectCreateDataTime() {
        this.createDateTime = LocalDateTime.now();
    }

    private void saveObjectChangeDataTime() {
        this.changeDateTime = LocalDateTime.now();
    }

    @Override
    public String getAllData() {
        return “name=’” + productName + ‘\’’ +”, price=” + productPrice + “, brand=”
+ productPrice + “, category=”;
    }
}
```

ДОДАТОК Б.
ПРОГРАМНИЙ КОД СЕРВЕРНОЇ ЧАСТИНИ

Б.1 Програмний коду модуля API.java

```
package com.example.demo.controller;

import com.example.demo.model.Product;
import com.example.demo.repository.ProductLogicRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import java.math.BigDecimal;
import java.util.List;

// As a @RestController, it handles incoming HTTP requests and returns JSON data
@RestController
@RequestMapping("/api/products") // Base path for all endpoints in this controller
public class ProductApiController {

    private final ProductLogicRepository productRepository;

    // The dependency on the repository (Logic class) is injected here.
    @Autowired
    public ProductApiController(ProductLogicRepository productRepository) {
        this.productRepository = productRepository;
    }



    /*
        Corresponds to loadProductsFromDatabase(n).
        URL: GET /api/products?limit=5
     */
    @GetMapping
```

```
    public ResponseEntity<List<Product>> getProducts(
            @RequestParam(defaultValue = "10") int limit) {
        List<Product> products = productRepository.loadProductsFromDatabase(limit);
        return ResponseEntity.ok(products);
    }

    // URL: GET /api/products/search?query=laptop
    @GetMapping("/search")
    public ResponseEntity<List<Product>> searchProducts(
            @RequestParam String query) {
        List<Product> products = productRepository.searchProductsInDatabase(query);
        return ResponseEntity.ok(products);
    }

    //URL: GET
/api/products/filter?category=SSD&brand=transcent&minPrice=100&maxPrice=500
    @GetMapping("/filter")
    public ResponseEntity<List<Product>> filterProducts(
            @RequestParam String category,
            @RequestParam String brand,
            @RequestParam BigDecimal minPrice,
            @RequestParam BigDecimal maxPrice) {
        List<Product> products =
productRepository.findFilterProductsInDatabase(category, brand, minPrice, maxPrice);
        return ResponseEntity.ok(products);
    }
}
```

Б.2 Програмний код модуля Logic.java

```
/*
    This class corresponds to the 'Logic' part of the diagram
    The @Repository annotation marks it as a Spring Data Access Component
*/
@Repository
public class ProductLogicRepository {

    // SQL query constants are placed here, where they are used.
    private static final String SQL_QUERY_GET_PRODUCTS = "SELECT id, name, category,
brand, price FROM products LIMIT ?;";
```

```
    private static final String SQL_QUERY_FIND_SEARCH_PRODUCTS = "SELECT id, name,
category, brand, price FROM products WHERE LOWER(name) LIKE ? OR LOWER(brand)
LIKE ?;";
    private static final String SQL_QUERY_FIND_FILTER_PRODUCTS = "SELECT id, name,
category, brand, price FROM products WHERE category = ? AND brand = ? AND price
BETWEEN ? AND ?;";

    private final JdbcTemplate jdbcTemplate;

    // Dependency Injection of JdbcTemplate via constructor.
    @Autowired
    public ProductLogicRepository(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    // The methods from the diagram are implemented here.

    public List<Product> loadProductsFromDatabase(int n) {
        // The query method executes the SQL and maps results using a RowMapper.
        return jdbcTemplate.query(SQL_QUERY_GET_PRODUCTS, new ProductRowMapper(), n);
    }

    public List<Product> searchProductsInDatabase(String query) {
        String searchQuery = "%" + query.toLowerCase() + "%";
        return jdbcTemplate.query(SQL_QUERY_FIND_SEARCH_PRODUCTS, new
ProductRowMapper(), searchQuery, searchQuery);
    }

    public List<Product> findFilterProductsInDatabase(String category, String brand,
BigDecimal minPrice, BigDecimal maxPrice) {
        return jdbcTemplate.query(SQL_QUERY_FIND_FILTER_PRODUCTS, new
ProductRowMapper(), category, brand, minPrice, maxPrice);
    }

    /*
        This private inner class implements the RowMapper interface
       This logic corresponds to the protected '#makeSQLquery' method's
responsibility in the diagram
    */
    private static class ProductRowMapper implements RowMapper<Product> {
        @Override
```

```
        public Product mapRow(ResultSet rs, int rowNum) throws SQLException {
            return new Product(
                rs.getLong("id"),
                rs.getString("name"),
                rs.getString("category"),
                rs.getString("brand"),
                rs.getBigDecimal("price")
            );
        }
    }
}
```

Б.3 Програмний код Product.java

```
package com.example.demo.model;

import java.math.BigDecimal;

// A record to represent a Product
public record Product(
    Long id,
    String name,
    String category,
    String brand,
    BigDecimal price
) {}
```