МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

ННІ/факультет	Інформаційних технологій
Кафедра	Інформатики і прикладного програмного забезпечення
Спеціальність	Інженерія програмного забезпечення
Форма навчання	Денна

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Лупач Павло Андрійович
(прізвище, ім'я, по батькові здобувача)

на тему **Розробка програмного забезпечення продажу смартфонів**
(повна назва теми)

за матеріалами праць провідних спеціалістів з розробки ПЗ та проектування БД
(повна назва бази дослідження)

науковий керівник **к.т.н., доцент** _____ **Медведєв Д.Г.**
(наук. ступінь, вчене звання) *(підпис)* *(прізвище, ініціали)*

Робота допущена до захисту в ЕК
Протокол засідання кафедри
від 11.06.2025 № 12
Завідувач кафедри _____
(підпис)
д.т.н., професор Зеленський О.С.
Наук. ступень, вчене звання *Ініціали, прізвище*

Кривий Ріг – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

ННІ/факультет	Інформаційних технологій
Кафедра	Інформатики і прикладного програмного забезпечення
Спеціальність	Інженерія програмного забезпечення
Форма навчання	Денна

«ЗАТВЕРДЖУЮ»
Завідувач кафедри _____ Зеленський О.С.
(підпис) (Прізвище, ініціали)
«11» червня 2025 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ

1. Тема роботи Розробка програмного забезпечення замовлень клієнта у ресторані

Керівник роботи _к.т.н. доцент_Медведєв Д.Г._____
затверджені наказом закладу вищої освіти від «04» квітня 2025 р. № 222-ст

2. Строк подання здобувачем роботи до «09» червня 2025 р.

3. Зміст кваліфікаційної роботи, об’єкт, предмет та мета дослідження:

Розділ 1. Постановка задачі

Розділ 2. Розробка алгоритму розв’язання задачі

Розділ 3.Проектування бази даних для проекту

Розділ 4.Розробка програмного забезпечення

Об’єкт дослідження: продаж смартфонів

Предмет дослідження: автоматизацію процесів продажу смартфонів

Мета кваліфікаційної роботи:створення програмного забезпечення магазину смартфонів та автоматизація його процесів

5. Дата видачі завдання «04» квітня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів МДР	Строк виконання етапів роботи	Відмітка керівника про виконання етапів (дата, підпис)
1	Підготовка розділу 1	04.04.2025-13.04.2025	
2	Підготовка розділу 2	14.04.2025-26.04.2025	
3.	Підготовка розділу 3	27.04.2025-15.05.2025	
4	Підготовка розділу 4	16.05.2025-08.06.2025	
5	Реєстрація завершеної кваліфікаційної роботи	09.06.2025	Реєстраційний №___ «09»червня 2025 р.
6	Отримання відгуку від наукового керівника	10.06.2025	
7	Подання кваліфікаційної роботи на перегляд завідувачу кафедри	11.06.2025	
8	Отримання зовнішньої рецензії	12.06.2025	
9	Попередній захист кваліфікаційної роботи на кафедрі	13.06.2025	
10	Підготовка до захисту в ЕК	16.06.2025-21.06.2025	

Завдання підготував науковий керівник _____ ___Медведєв Д.Г.____

(підпис) (прізвище та ініціали)

Завдання одержав _____ _____Лупач П.А____

(підпис) (прізвище та ініціали)

ЗГОДА здобувача вищої освіти

Державного університету економіки і технологій про перевірку кваліфікаційної роботи на прояви академічного плагіату та розміщення в Репозитарії Університету

Я, _____Лупач Павло Андрійович_____, підтримую політику Державного університету економіки і технологій з академічної доброчесності і відкритого доступу.

Засвідчую, що кваліфікаційна бакалаврська робота Розробка програмного забезпечення продажу смартфонів_____
виконана самостійно та не містить академічного плагіату. Я не надавав і не одержував недозволену допомогу під час підготовки цієї роботи. Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Із чинним Положенням про запобігання та виявлення академічного плагіату в роботах здобувачів вищої освіти Державного університету економіки і технологій ознайомлений. Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі порушення норм академічної доброчесності робота не допускається до захисту або оцінюється незадовільно.

Також я поінформований, що відповідно до «Положення про Репозитарій (електронну базу даних) Державного університету економіки і технологій» зазначена робота буде розміщена в Електронному архіві Університету (Репозитарії ДУЕТ). З умовами такого розміщення ознайомлений.

Дата підпис ініціали, прізвище (власноруч)

АНОТАЦІЯ

на бакалаврську кваліфікаційну роботу

«Розробка програмного забезпечення продажу смартфонів»

Лупача Павла Андрійовича

Кваліфікаційна бакалаврська робота на здобуття освітньо-кваліфікаційного рівня бакалавра зі спеціальності 121 «Інженерія програмного забезпечення» – Державний університет економіки і технологій – Кривий Ріг, 2025.

Основним завданням дипломної роботи було створення програмного комплексу, сайту, через який кожен день виконується більше десятка тридцяти замовленнь з різних куточків країни.

У бакалаврській дипломній роботі удосконалені теоретико-методичні підходи до створення проектів.

Доповнені підходи до створення проектів та використання їх на різних операційних системах та у різних браузерах.

У розробленому програмному забезпечені використана технологія Node.js Framwork для серверної частини інтернет магазину, та HTML, CSS, JavaScript для клієнтської частини.

В роботі запропоновано новий підхід до автоматизації процесів роботи інтернет-магазину. Завдяки яким для роботи сервісу залучено мінімальну кількість працівників.

Ключові слова: інтернет-магазин, автоматизація роботи, Node.js Framework, HTML, CSS, JavaScript.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Скорочення	Розшифрування
HTML	Мова розмітки, яка використовується для створення веб-сторінок
СУБД	Система управління базми даних
CSS	Мова опису стилів, яку використовують для оформлення веб-сторінок
Вебсайт або сайт	Сукупність вебсторінок та залежного вмісту, доступних у мережі Інтернет, які об'єднані як за змістом, так і за навігацією під єдиним доменним ім'ям
Алгоритм	Набір інструкцій, які описують порядок дій виконавця, щоб досягти результату розв'язання задачі за скінченну кількість дій
Структура даних	Спосіб організації даних в більш складні типи даних

ЗМІСТ

ВСТУП

У сучасному світі інформаційних технологій електронна комерція стрімко розвивається, стаючи невіддільною складовою бізнес-процесів. Особливу популярність набувають онлайн-магазини з продажу цифрової техніки, що пояснюється високим попитом на ці пристрої серед користувачів. Сучасний споживач потребує не лише зручного веб-інтерфейсу для швидкого та комфортного оформлення покупки, а й широких інтерактивних можливостей для детального ознайомлення з товаром ще до моменту його придбання.

Традиційні засоби представлення товарів, такі як статичні зображення чи текстові описи, не завжди можуть бути правдивими та дійсно допомогти ознайомитися з зовнішнім виглядом товару. Покупець прагне якомога точніше уявити товар, розміри, пропорції, текстуру та функціональні елементи смартфона до того, як зробити вибір. У зв’язку з цим впровадження візуалізації товару у вигляді тривимірних 3D моделей будуть перевагою та унікальною особливістю роботи.

Таким чином, актуальною стає розробка гібридного програмного забезпечення, яке поєднує сучасні вебтехнології для продажу та керування товарами з поєднанням десктопного застосунку, з підтримкою графічних бібліотек для візуалізації тривимірних об’єктів.

У межах цієї дипломної роботи буде реалізовано наступну систему: вебдодаток для онлайн-продажу смартфонів та застосунок з використанням Windows Forms та OpenGL для демонстрації 3D моделі смартфона. Це дозволить користувачу не тільки здійснити покупку, а й отримати уявлення про те як буде виглядати смартфон, з мінімальними похибками та хибним уявленням.

Існує потреба у вдосконаленні засобів онлайн-продажу, які б забезпечували користувачеві зручний, інформативний і візуально привабливий інтерфейс. Ідея проекту полягає в тому, щоб сатй надавав не

прості комерційні інструменти покупки, а й можливість ознайомитися з товаром максимально наближено до реального споживчого досвіду.

Метою даної дипломної роботи є розробка програмного забезпечення для продажу смартфонів. В планах створення веб-сайту для онлайн-покупок та десктопного застосунку з можливістю перегляду тривимірної моделі смартфона.

Завдання, які необхідно вирішити в рамках роботи:

1. Проаналізувати сучасні засоби розробки веб-додатків і засоби тривимірної графіки.
2. Спроектувати архітектуру програмного комплексу.
3. Розробити веб-сайт для продажу смартфонів із функціоналом перегляду, пошуку та оформлення замовлення.
4. Реалізувати застосунок на платформі Windows Forms з інтеграцією 3D моделі смартфона за допомогою технологій OpenGL.

Об'єкт дослідження — процес автоматизації електронної комерції у сфері продажу смартфонів.

Предмет дослідження — методи та засоби програмної реалізації вебплатформи та десктопного додатку з візуалізацією 3D моделей у контексті продажу електронних пристроїв.

РОЗДІЛ І
ПОСТАНОВКА ЗАДАЧІ

1.1. Характеристика задачі

Задача розробки програмного забезпечення для продажу смартфонів полягає у створенні інтегрованого рішення, яке поєднує в собі функціональність веб застосунку для онлайн-торгівлі та десктопного застосунку з підтримкою 3D графіки для візуалізації товару.

Модуль веб застосунку повинен виконувати наступні задачі:

1. Забезпечення користувачеві доступу до асортименту смартфонів;
2. Пошук, фільтрація та сортування товарів за параметрами;
3. Оформлення замовлення;
4. Адаптивний дизайн для різних типів пристроїв;
5. Інтеграція з серверною частиною для зберігання та обробки даних.

Модуль десктопного застосунку реалізований за допомогою Windows Forms + OpenGL повинен виконувати наступні задачі:

1. Рендер 3D моделі смартфона;
2. Обертання моделі;
3. Завантаження характеристик смартфону з БД;

Технічні вимоги до розробки наступні:

1. Для веб-розробки буде використано мову JavaScript, HTML/CSS та відповідні фреймворки;
2. Для десктопного додатку використовується мова C# у середовищі Windows Forms;
3. Для візуалізації тривимірних моделей — OpenGL як інструмент графічного рендерингу;
4. Для зберігання даних — використання бази даних PostgeRQL;

Аналіз предметної області дозволяє визначити, що головною проблемою онлайн-продажів смартфонів є обмежена можливість ознайомлення з

зовнішнім виглядом товару до його придбання. В умовах, коли фізичний контакт із пристроєм відсутній, важливо створити інструменти, які дозволять потенційному покупцю детально розглянути товар, сформувати уявлення про його дизайн, пропорції та особливості конструкції. Реалізація можливості перегляду тривимірної моделі смартфона, яку можна обертати та вивчати у динаміці, є одним із шляхів вирішення цієї проблеми.

Окрім візуалізації, задача також включає забезпечення якісної взаємодії між вебінтерфейсом та десктопним додатком. Особливістю цієї задачі є необхідність реалізації двох різнорідних програмних модулів, які функціонують у різних середовищах, проте мають спільну логіку роботи та джерело даних. Це потребує побудови єдиної архітектури системи, в основі якої лежить база даних, яка забезпечуватиме доступ як із боку сатйу, так і з боку графічного модуля. Такий підхід обраний, задля досягнення мнти підтримувати актуальність і синхронність даних, незалежно від того, з якого застосунку користувач здійснює доступ.

З технічного погляду, реалізація веб-застосунку здійснюється з використанням мови JavaScript у поєднанні з HTML та CSS. Необхідно було вибрати такі інструменти розробки, які дозволять створити адаптивний інтерфейс, який буде коректно відображатися на пристроях з різними розмірами екранів, включаючи смартфони, планшети та десктопи. Десктопна частина реалізована на мові C# у середовищі Windows Forms з використанням OpenGL для рендерингу тривимірної моделі смартфона. Ці мови програмування дозволяють обробляти графічні сцени, налаштовувати камеру, світло, матеріали та інші компоненти 3D-відображення, щоб формувати реалістичне уявлення про товар. Уся інформація про товари, їхні характеристики та візуальні компоненти зберігається у реляційній базі даних PostgreSQL.

Дана задача охоплює кілька напрямків програмування — веб-розробку, клієнт-серверну архітектуру, роботу з базами даних, тривимірну графіку та розробку UI/UX дизайну. Такий міждисциплінарний характер проекту

дозволяє не лише реалізувати практичне рішення для ринку, але й продемонструвати комплексні навички з інженерії програмного забезпечення. Результати роботи мають не лише навчальну, а й практичну цінність, оскільки можуть бути застосовані для подальшого розширення функціоналу або адаптації до інших видів продукції.

1.2. Огляд існуючих аналогів магазинів смартфонів

У рамках аналізу предметної області було проведено дослідження вже існуючих рішень на ринку електронної комерції, що спеціалізуються на продажу смартфонів. Такий огляд дозволяє визначити загальні тенденції, переваги та недоліки популярних платформ, а також окреслити потенційні напрями вдосконалення, які можуть бути враховані під час розробки власного програмного забезпечення.

Rozetka (rozetka.com.ua) є одним із найпопулярніших інтернет-магазинів в Україні.

Цей майданчик пропонує широкий вибір цифрової техніки від різних виробників. Основними перевагами платформи є гарна система фільтрації (рис. 1.1.), наявність рекомендацій (рис. 1.2.), рейтингової системи та відгуків (рис. 1.3.), інформативні сторінки товарів з фотографіями та характеристиками (рис. 1.4.), а також зручне оформлення замовлень (рис. 1.5.).

Проте Rozetka не пропонує інструментів для візуального ознайомлення з товаром: всі зображення — це статичні фотографії, користувач не має можливості переглянути товар у 3D чи взаємодіяти з ним інтерактивно.

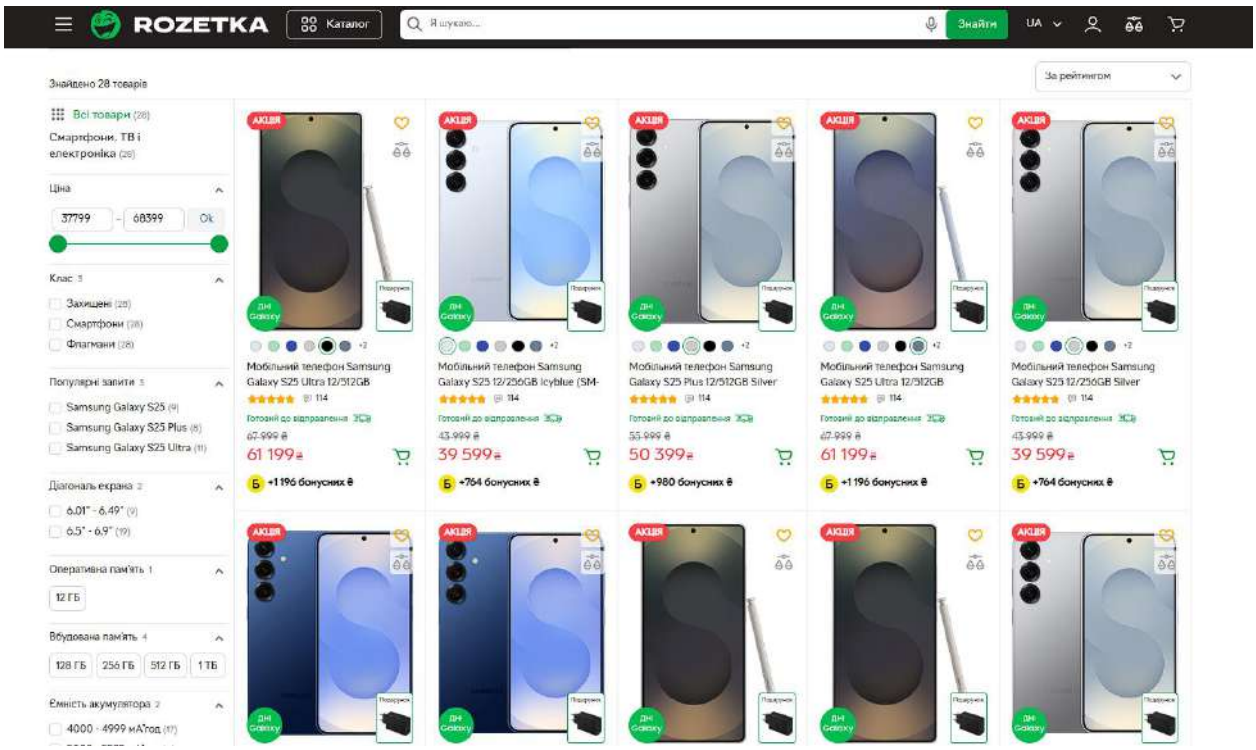

Рис. 1.1. Каталог товарів Rozetka

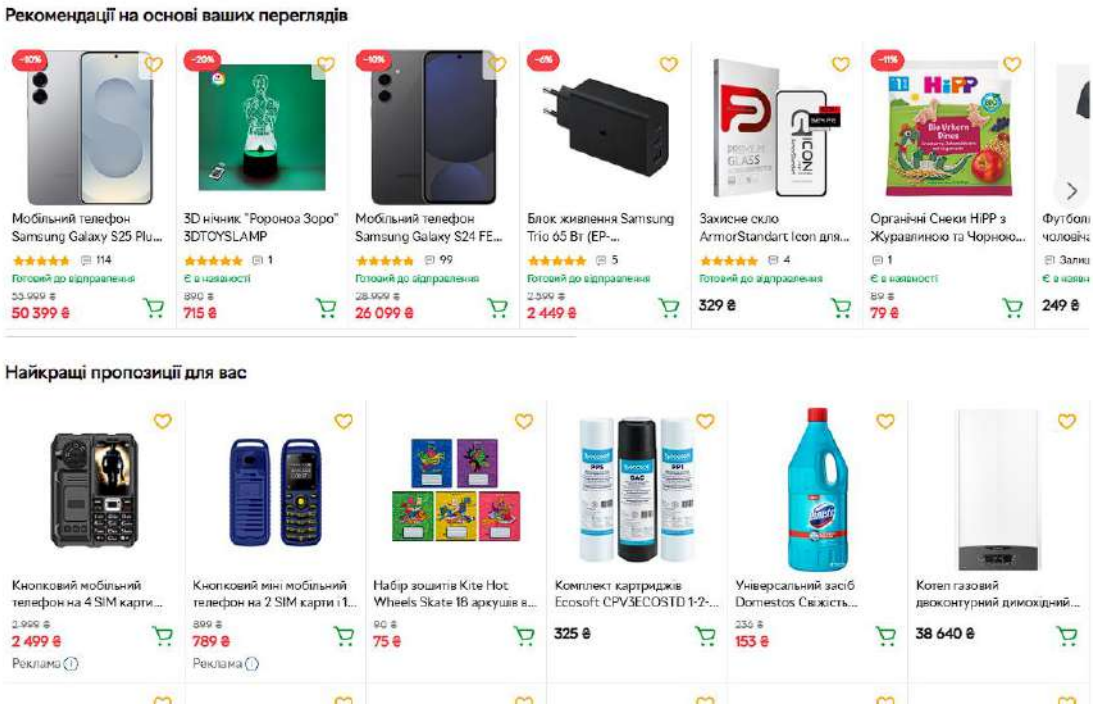

Рис. 1.2. Рекомендації Rozetka

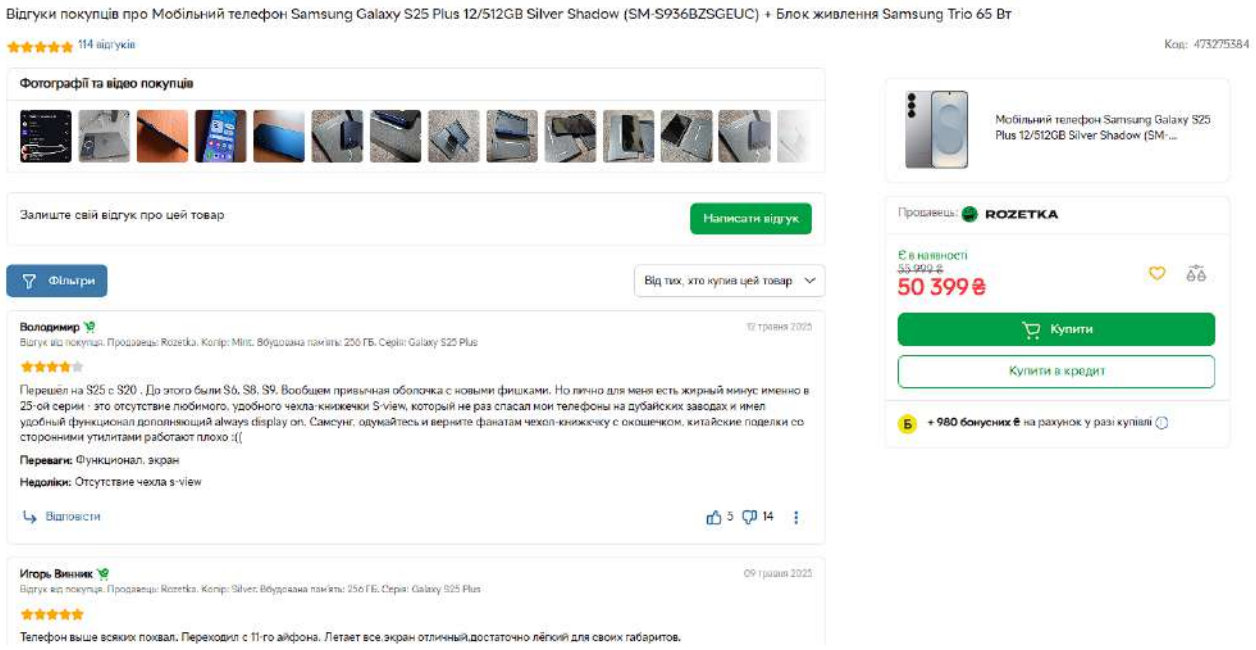

Рис. 1.3. Система рейтингу та відгуків Rozetka

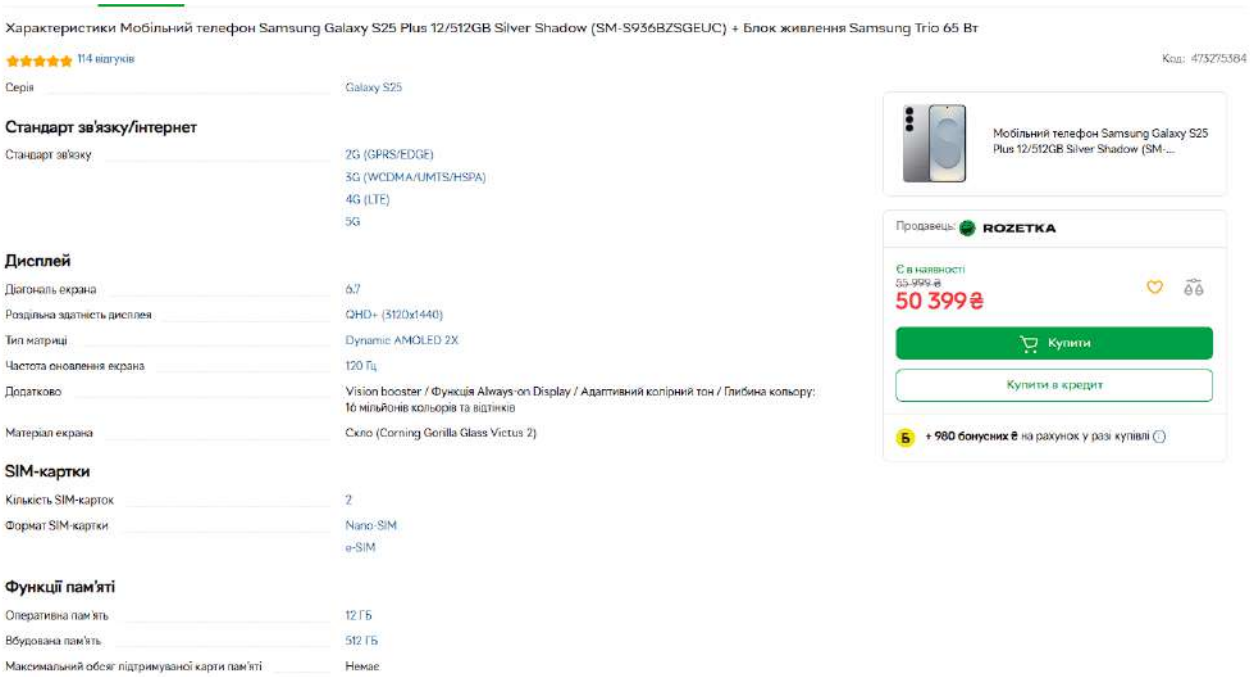

Рис. 1.4. Характеристики товару Rozetka

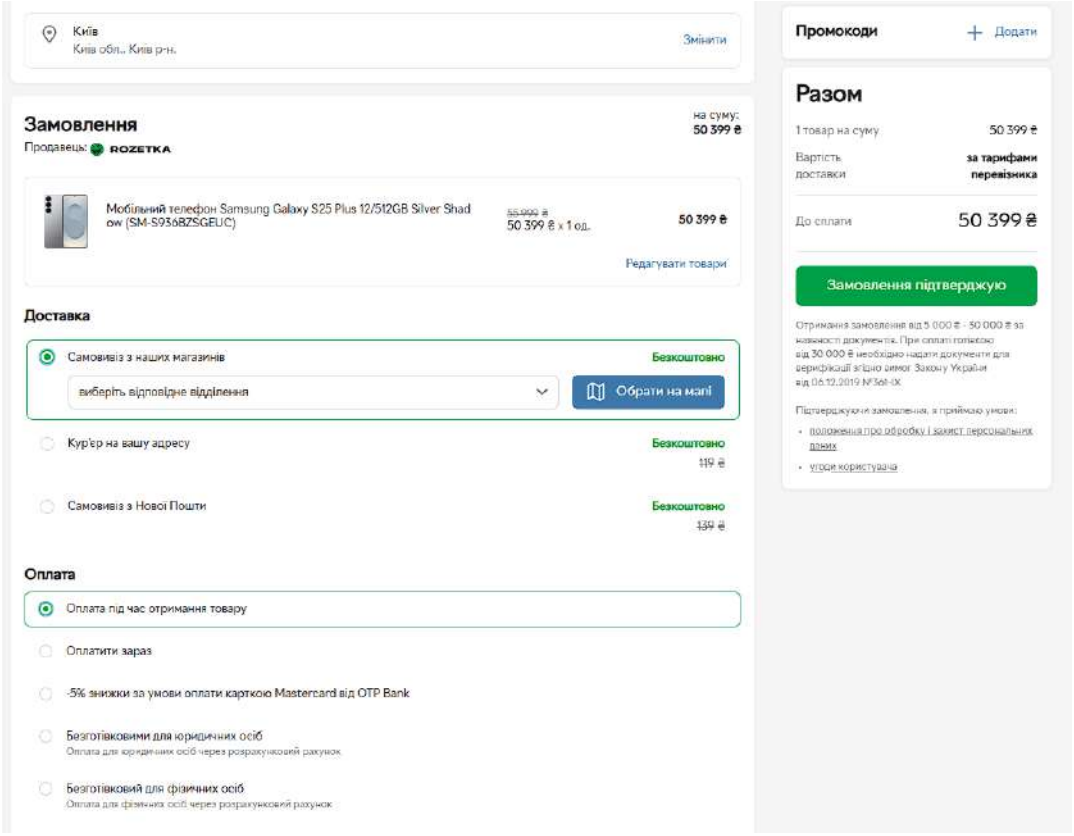

Рис. 1.5. Оформлення замовлення Rozetka

Comfy (comfy.ua) — ще один відомий ритейлер техніки. Як і Rozetka, Comfy має адаптивний вебінтерфейс (рис. 1.6.), сторінки товарів із зображеннями та детальною інформацією про смартфон (рис. 1.7.), характеристиками, оглядами та відео. Навігація по сайту зручна (рис. 1.8.), реалізовано систему сортування та підбору за параметрами (рис. 1.9.). Проте, як і сайт Rozetka, Comfy не реалізує підтримку інтерактивного 3D-огляду продукції, тобто взаємодії з візуальною частиною товару немає.

Цитрус (citrus.ua) також популярна мережа на ринку електронної торгівлі смартфонами в Україні. Окрім типового для таких платформ функціоналу (каталог товарів, фільтри, фото, описи), Цитрус акцентує увагу на «лайфстайл»-аспекті товарів (рис. 1.10.). На сайті часто представлено оглядові відео та статті, а інтерфейс орієнтований на молодіжну аудиторію. Однак, як і попередні приклади, Цитрус обмежується лише двовимірним відображенням товару. Навіть попри більший акцент на візуальні елементи, інтерактивної графіки чи 3D-моделей у каталозі не представлено.

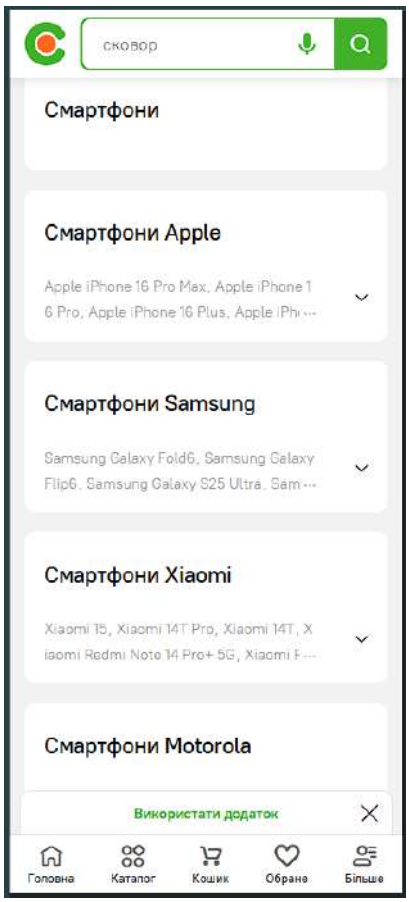

Рис. 1.6. Адаптивність сайту Comfy

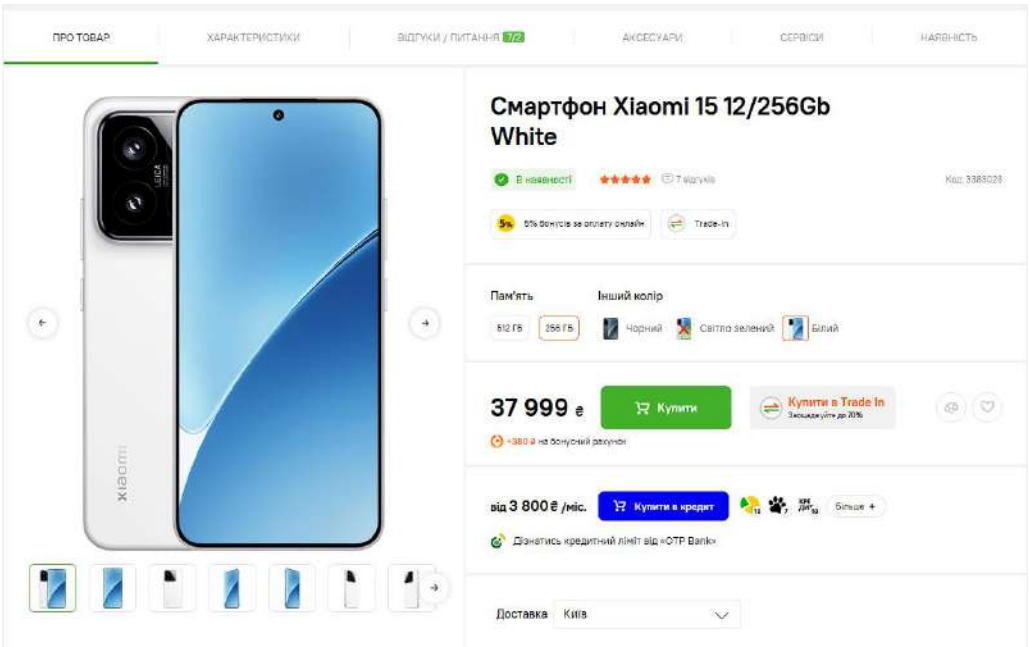

Рис. 1.7. Сторінка смартфону Comfy

Рис. 1.8. Навігація по каталогу Comfy

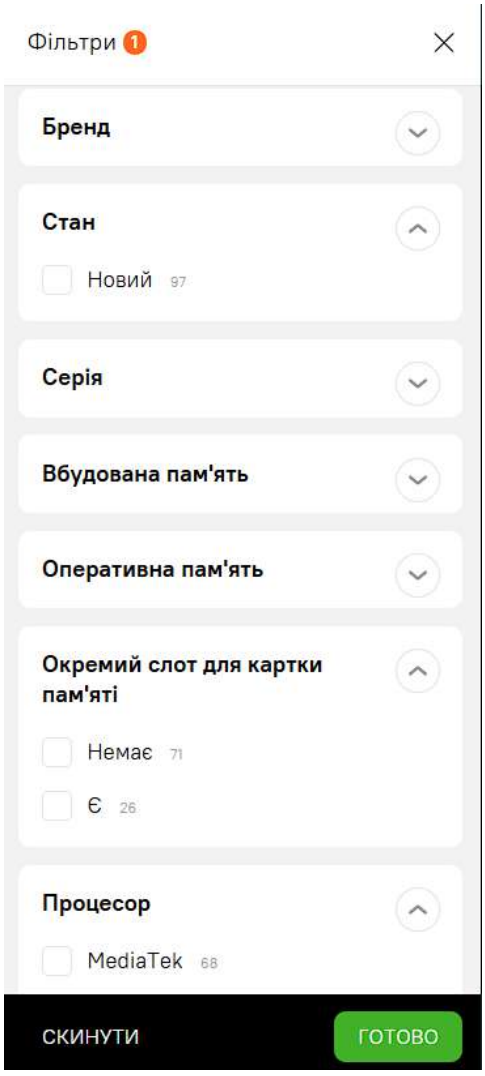

Рис. 1.9. Фільтрація смартфонів Comfy

Рис. 1.10. Сторінка відео-огляду смартфону Цитрус

Проведений огляд свідчить про те, що існуючі українські онлайн-магазини смартфонів мають добре розвинену структуру, широкий функціонал і привабливий дизайн. Усі вони зосереджені на спрощенні процесу вибору, оформлення покупки та отримання інформації про товар. Водночас, спільним недоліком є відсутність можливості взаємодіяти з товаром у режимі реального часу — у форматі тривимірної візуалізації або інтерактивного перегляду.

Цей недолік створює можливості для впровадження інноваційного підходу до представлення товарів. Реалізація функціоналу перегляду 3D-моделей смартфонів у спеціалізованому десктопному застосунку дозволить значно розширити враження користувача про товар. Такий підхід вирізнятиме платформу серед конкурентів, формуючи нову планку взаємодії в онлайн-торгівлі.

Результати проведеного аналізу було зібрано у таблицю, щоб наочно порівняти переваги та недоліки кожної з розглянутих платформ (табл. 1.1).

Таблиця 1.1

Порівняння переваг та недоліків існуючих магазинів

Магазин	Переваги	Недоліки
Rozetka	– Широкий асортимент смартфонів– Потужна система фільтрації та сортування– Наявність відгуків, рейтингів, відеооглядів– Зручний процес оформлення замовлення– Адаптивний дизайн	– Відсутність 3D візуалізації товарів– Статичні зображення не дають повного уявлення про зовнішній вигляд смартфона
Comfy	– Інформативний опис товарів– Зручна навігація по сайту– Інтегровані відео та огляди– Адаптивний інтерфейс для мобільних пристроїв	– Немає підтримки 3D моделей– Інтерфейс менш персоналізований порівняно з конкурентами
Цитрус	– Сучасний молодіжний інтерфейс– Акцент на «стилі життя» у подачі товарів– Багато медіаматеріалів (відео, огляди)– Часто актуальні акції та бонуси	– Відсутність інтерактивної 3D-візуалізації– Менш гнучка система фільтрації, ніж у Rozetka– Можливе перевантаження інтерфейсу візуальними елементами

Українські онлайн-магазини смартфонів Rozetka, Comfy та Цитрус зручні, мають хорошу подачу товарів і відеоогляди. Проте жоден з них не дає змоги побачити пристрій у 3D, тож користувач змушений орієнтуватися лише на фото.

1.3. Вибір та обґрунтування інструментів розробки

Проєкт передбачає створення двох частин: веб-застосунку для онлайн-продажу смартфонів і десктопного застосунку з підтримкою тривимірної графіки. Для реалізації обох компонентів було обрано ті інструменти, які добре підходять до завдань, мають достатню документацію та підтримуються спільнотою.

Для клієнтської частини веб-застосунку використовується зв’язка HTML, CSS та JavaScript. Ці технології дають змогу створити інтерфейс, який коректно працює у браузері та адаптується до різних пристроїв. JavaScript також забезпечує наступні можливості для реалізації: обробку фільтрів, форму замовлення та взаємодію з API.

Як фреймворк для створення інтерфейсу користувача розглядається React.js. Його структура дає змогу зручно будувати компоненти, повторно використовувати код і краще організувати логіку взаємодії між елементами інтерфейсу.

На сервері буде використовуватись Node.js. Це також технологія, яка використовує мову програмування JavaScript. Тому такий вибір дозволяє писати бекенд на тій самій мові, що й фронтенд. Таким чином спрощується обмін даними між частинами системи. Залежно від завдань, сервер може реалізовувати API для обробки запитів клієнта, взаємодії з базою даних та авторизації [1].

Для десктопного застосунку, який відповідає за відображення 3D-моделей смартфонів, обрано C# у середовищі Windows Forms. Такий вибір пояснюється наступними аргументами: Windows Forms — це зручний спосіб створення GUI-додатків під Windows, а C# — об’єктно орієнтована мова для цієї платформи. Візуалізація моделей реалізується з використанням OpenGL — бібліотеки, яка дозволяє відображати тривимірні об’єкти, обертати їх, тощо. OpenGL інтегрується з C# через обгортку OpenTK.

Для зберігання даних використовується PostgreSQL. Це реляційна СУБД, яка добре працює з великою кількістю записів, підтримує складні запити та зберігає структуровану інформацію у вигляді таблиць. Це добре підходить під задачу зберіганння характеристики смартфонів, дані користувачів та замовлення.

Prism.js — це клієнтська JavaScript-бібліотека для підсвічування синтаксису коду на вебсторінках. Вона часто використовується в інформаційних порталах, технічних блогах, документаційних системах або інтерфейсах, що містять фрагменти коду [2]. Її головна мета — зробити програмний код в інтерфейсі читабельним, структурованим і візуально чітким. У межах сучасних вебтехнологій, де часто публікуються динамічно згенеровані дані або інструкції, ця бібліотека виконує роль стилістичного оформлення коду без втручання в логіку.

Бібліотека має модульну структуру: основна частина підтримує базові мови та функції, а додаткові модулі, як-от підтримка інших мов програмування, нумерація рядків або кнопка «копіювати». Вони можуть підключатися за потреби. Завдяки цій бібліотеці, сайт буде менше навантажуватися графічними елементами.

Базовий принцип роботи Prism.js полягає в обробці елементів HTML, в яких міститься текст програми. На основі класів, які вказують мову, бібліотека виконує синтаксичний аналіз і обгортає відповідні частини коду у структуровані HTML-елементи з CSS-класами, після чого ці елементи стилізуються таблицями стилів. Усе це відбувається на стороні клієнта, одразу після завантаження сторінки.

До основних функцій Prism.js належать:

- підтримка великої кількості мов програмування;
- можливість кастомізації тем оформлення;
- плагіни для додаткових функцій;
- мінімальний обсяг коду ядра;
- робота без серверної обробки;

- підтримка асинхронного завантаження мов.

Prism.js підтримує понад двісті мов програмування, в тому числв JavaScript, Python, C#, HTML, SQL, Java. Завдяки цьому вона підходить для широкого спектру веб-ресурсів, які публікують фрагменти коду. Користувачі мають змогу вибрати одну з багатьох готових тем оформлення або створити власну за допомогою CSS.

Додаткові плагіни розширюють функціональність. Можна додати нумерацію рядків, згортання блоків коду, автоматичне копіювання в буфер обміну, підсвічування активного рядка або підтримку вкладених фрагментів. Ці плагіни можна підключати окремо — лише ті, які потрібні для конкретного проекту.

Prism.js не залежить від сторонніх бібліотек на кшталт jQuery, тому легко інтегрується у будь-який код. Її використання не впливає на швидкість роботи веб-застосунку, оскільки бібліотека має оптимізований код і не потребує складних обчислень. Підсвічування відбувається лише після повного завантаження HTML, що дозволяє уникнути конфліктів із динамічним вмістом.

У межах розробки вебзастосунку для продажу смартфонів Prism.js буде застосована на інформаційних сторінках, у розділах довідки, інструкцій або налаштувань, де публікуються конфігурації, API-дані або технічна інформація. Таким чином покращується сприйняття інформації, спрощується навігація в коді та інтерфейс робиться зручнішим для користувача, особливо якщо сайт орієнтований на технічно обізнану аудиторію.

Висновки до розділу 1

У межах розділу було сформульовано загальну характеристику задачі — створення інтегрованого програмного рішення, яке об’єднує веб-застосунок для онлайн-продажу смартфонів та десктопну програму з можливістю тривимірної візуалізації товару. Проведено функціональний аналіз складових

частин проєкту: сайт має забезпечити пошук, перегляд, фільтрацію та оформлення замовлень, тоді як десктопна частина виконує роль візуалізації смартфона за допомогою 3D-моделі.

Було розглянуто існуючі рішення на прикладі популярних платформ Rozetka, Comfy та Zitrus. Їхній функціонал охоплює базові можливості електронної комерції, але в жодному з випадків не реалізовано інтерактивну тривимірну взаємодію з моделлю товару. Після цього було сформульовано унікальну ідею та перевагу розроблюваного ПЗ — включення 3D-компонента до структури користувацького досвіду.

Також виконано обґрунтований вибір інструментів реалізації. Для веб-розробки застосову.ться мови програмування JavaScript з HTML/CSS. Серверна частина базується на Node.js у зв’язці з PostgreSQL. Десктопна програма створюється з використанням C# у середовищі Windows Forms, а за тривимірну візуалізацію відповідає OpenGL.

Окремо досліджена була бібліотека Prism.js. Було визначено, що вона виконує допоміжну функцію на сайті, забезпечуючи підсвічування синтаксису у випадках, коли потрібно показати технічний код чи конфігурацію. Такий інструмент буде корисним для сайту, задля запобігання надмірного навантаження сайту.

РОЗДІЛ ІІ
ПРОЕКТУВАННЯ ЗАДАЧІ

2.1. Проектування 3D моделі смартфона

Етап проектування 3D-моделі смартфона виконує підготовчу функцію в контексті розробки десктопного застосунку. Його мета — сформувати технічне уявлення про те, як саме має бути реалізована модель у вигляді програмного коду з використанням засобів OpenGL. На відміну від підходів, де 3D-об’єкти створюються у зовнішніх графічних редакторах і імпортуються у програму як готові файли.

Початковим кроком був аналіз об’єкта моделювання — сучасного смартфона. Визначені геометричні характеристики, співвідношення сторін, розміщення елементів, таких як кнопки, блок камери, рамки дисплея. Ця інформація служить основою для формулювання математичної моделі, яка описує об’єкт за допомогою вершин, граней і примітивів. З технічного боку це означає, що кожен елемент моделі — корпус, кнопки, виступи — буде описано вручну шляхом задання координат, кольорів і нормалей у коді на C# через відповідні виклики OpenGL API.

Особливістю такого підходу є повний контроль над кожним елементом геометрії, а також можливість оптимізувати модель відповідно до потреб застосунку. Створення об’єкта в реальному часі передбачає визначення масивів вершин, буферів кольорів і побудову сцени на основі трикутників та прямокутників. Також треба буде встановити положення камери та джерел освітлення, щоб досягти базової просторової виразності без складних ресурсозатратних обчислень.

Процес проектування моделі на цьому етапі не обмежується лише графічною частиною. Визначається також спосіб зв’язку візуальної складової з функціональними елементами інтерфейсу. Модель розглядається не як

самодостатній об'єкт, а як елемент загальної структури застосунку, який буде малюватися в одному вікні з характеристиками смартфону.

В табл. 2.1 нижче подано узагальнені характеристики елементів моделі, які підлягають реалізації безпосередньо в коді.

Таблиця 2.1

Характеристики елементів моделі

Елемент моделі	**Геометрична форма**	**Спосіб побудови**	**Особливості реалізації**
Корпус смартфона	Прямокутний паралелепіпед із заокругленнями	Побудова з шестигранника через вершини	Враховується співвідношення сторін і загальні габарити
Дисплей	Плоский прямокутник	Поверхня на передній частині корпусу	Візуально виділяється кольором або текстурою
Камера	Комбінація циліндра та кільця	Побудова вручну через полігони	Локалізація на задній панелі
Спалах	Комбінація циліндра та кільця	Побудова вручну через полігони	Локалізація на задній панелі

У ході аналітичного етапу було визначено основні параметри майбутньої 3D-моделі, які враховуються під час програмної побудови об'єкта у середовищі OpenGL, які описує таблиця.

2.2. Розробка алгоритму програми

Алгоритм роботи клієнтської частини програми побудований на поетапній взаємодії з базою даних PostgreSQL та подальшому відображенні даних у графічному інтерфейсі Windows Forms. Після запуску додатку ініціалізується головна форма, в якій відразу ж здійснюється спроба

підключення до бази даних за допомогою бібліотеки Npgsql. У разі успішного підключення у статусному рядку відображається відповідне повідомлення. Якщо ж підключення не вдалося, користувач отримує повідомлення про помилку, а програма повідомляє про неможливість доступу до бази.

Після встановлення з’єднання програма виконує запит до таблиці категорій, де обираються лише ті, що позначені як активні для відображення. Назви цих категорій динамічно додаються до випадаючого меню головної форми, і кожній з них автоматично призначається обробник події, що дозволяє завантажити відповідні товари при виборі.

За замовчуванням, при першому запуску інтерфейсу користувачеві відображаються останні 10 смартфонів, які позначені як доступні до показу. Дані про товари вибираються з бази та зберігаються у внутрішньому списку. Для кожного елемента створюється окремий графічний блок із зображенням смартфона, його назвою, ціною та кнопкою «Докладніше». Якщо зображення відсутнє або шлях до нього неправильний, система може використати стандартну заглушку.

При виборі певної категорії з меню здійснюється фільтрація товарів. На панелі після цього відображаються лише ті смартфони, що належать до відповідної групи. Усі товари при цьому виводяться в тому самому форматі, що й за замовчуванням: зображення, назва, ціна, кнопка перегляду деталей.

Натискання кнопки «Докладніше» відкриває нову форму, яка отримує ідентифікатор вибраного товару і встановлене з’єднання з базою даних. Завантажується та відображається інформація про конкретний смартфон, включаючи його характеристики, опис та інші деталі, збережені в базі даних.

При завершенні роботи додатку, тобто при закритті головної форми, програма перевіряє стан з’єднання з базою даних. Якщо з’єднання активне, воно закривається, щоб запобігти витоку ресурсів та забезпечити коректне завершення роботи програми.

2.3. Проєктування алгоритму функціонування веб-застосунку

У процесі розробки веб-застосунку особлива увага приділятиметься побудові логіки клієнтсько-серверної взаємодії, яка забезпечуватиме ефективний обмін даними між інтерфейсом користувача та серверною частиною системи. Передбачається, що структура веб-застосунку ґрунтуватиметься на поділі на клієнтську частину, реалізовану засобами сучасних інтерфейсних бібліотек, та серверну, яка відповідатиме за обробку запитів і взаємодію з базою даних.

На початковому етапі, після завантаження сторінки, інтерфейс ініціюватиме звернення до серверної частини з метою отримання актуального переліку об’єктів, що мають бути представлені користувачеві. Очікується, що це буде набір візуально уніфікованих елементів, кожен з яких міститиме коротку інформацію — зображення, назву, ціну — та кнопку для перегляду розширених відомостей. У випадку, якщо графічні матеріали відсутні або недоступні, система автоматично підставлятиме альтернативне зображення-заглушку.

На окремому рівні буде реалізовано динамічне меню або панель навігації, яка дозволятиме здійснювати фільтрацію відображуваного вмісту за відповідними категоріями. Ці категорії передбачено отримувати у вигляді окремого запиту на сервер, після чого вони формуватимуть інтерактивний список у клієнтському інтерфейсі. При виборі певного пункту здійснюватиметься новий запит до бази даних, що повертатиме тільки ті елементи, які відповідають заданому критерію.

У випадку взаємодії користувача з окремим елементом, зокрема при натисканні кнопки розширеного перегляду, передбачено відкриття спеціального інформаційного блоку або сторінки, що завантажуватиме детальні дані про обраний об'єкт. Ці дані включатимуть текстовий опис, технічні характеристики та, за наявності, додаткові ілюстрації. Для

забезпечення швидкодії передбачатиметься механізм завантаження лише необхідної інформації за допомогою параметризованих запитів до серверу.

Окрему роль у логіці функціонування відіграватиме механізм оформлення замовлення. Після заповнення інтерактивної форми з контактною інформацією користувача, дані будуть надіслані на сервер, де відбудеться створення відповідних записів у структурі бази даних. Внутрішні зв’язки між сутностями дозволятимуть зафіксувати, які саме об’єкти були замовлені, та прив’язати їх до конкретного користувача або замовлення.

На завершальному етапі передбачено обробку результатів взаємодії — як позитивних, так і помилкових. У разі успішного виконання дії користувач отримуватиме візуальне підтвердження, тоді як у випадку технічних збоїв система повідомлятиме про невдачу у зручному для сприйняття вигляді. Таким чином, алгоритм забезпечуватиме не лише функціональність, але й високий рівень зворотного зв’язку.

У табл. 2.2 наведено узагальнені етапи функціональної логіки майбутнього веб-застосунку, що підлягають реалізації.

Таблиця 2.2

Етапи функціонування веб-застосунку

Дія користувача або системи	**Мета дії**	**Реалізація на рівні системи**
1	2	3
Відкриття головної сторінки	Відображення вступного екрану з основним контентом	Відправляється запит до сервера; сторінка формується з анімацією при першому рендері
Завантаження переліку елементів	Формування списку актуального контенту	Клієнт надсилає GraphQL-запит; відповідь інтегрується у візуальні компоненти
Застосування фільтру за категорією	Виведення лише релевантних користувачу записів	Фільтрація через параметри запиту або локально після отримання повного масиву даних

Продовження таблиці 2.2

1	2	3
Виведення повідомлення після дії	Інформування про результат взаємодії	Відповідь сервера обробляється клієнтом; формується візуальне підтвердження
Надсилання форми замовлення/створення	Фіксація взаємодії користувача із сайтом	Валідація даних; передача запиту на сервер з відповідним типом операції (create/update)
Перехід на сторінку деталізації	Отримання розширеної інформації про вибраний об’єкт	При переході передається ID; виконується запит до бази, результат відображається у компоненті
Навігація між сторінками	Забезпечення переходу між різними розділами сайту	Задіяний клієнтський роутинг; URL змінюється без перезавантаження сторінки
Завантаження додаткового контенту	Розширення поточного списку записів	Динамічне підвантаження з offset/page параметром; дані додаються до вже наявного списку

Завдяки поетапній логіці та урахуванню поведінкових сценаріїв користувача, майбутній веб-застосунок забезпечуватиме плавну та послідовну взаємодію з цифровим середовищем. Передбачені механізми фільтрації, перегляду, формування замовлень і обробки повідомлень створюватимуть умови для ефективного користування навіть при зростанні обсягів даних або кількості запитів. Проєктована структура дозволить масштабувати систему в подальшому та інтегрувати нові функціональні можливості без потреби у радикальних змінах архітектури.

Висновки до розділу 2

У цьому розділі було детально розглянуто процес розробки графічного інтерфейсу програми для перегляду смартфонів, включаючи структуру головного вікна, відображення товарів у каталозі, функцію фільтрації за категоріями та механізм переходу до детального перегляду.

Окрему увагу приділено інтерфейсу вебсайту інтернет-каталогу, де реалізовано схожий функціонал для користувачів, що переглядають товари через браузер. Сайт відображає ту саму структуру товарів, підтримує фільтрацію та пошук, а також дозволяє переглядати детальну інформацію про смартфони, що забезпечує цілісність досвіду користувача незалежно від платформи доступу.

РОЗДІЛ ІІІ
ПРОЕКТУВАННЯ БАЗИ ДАНИХ

3.1. Структура бази даних

Проєктування бази даних є фундаментальним кроком у створенні будь-якої сучасної інформаційної системи, зокрема інтернет-магазину мобільних телефонів. База даних виступає як основний елемент, що забезпечує надійне збереження, організацію та обмін інформацією між усіма компонентами системи. Вона є ключем до реалізації бізнес-логіки, управління товарами, категоріями, характеристиками та обробки замовлень, а також аналізу користувацької активності. Правильно спроєктована база даних гарантує стабільність роботи додатку, швидкість доступу до інформації, а також дає змогу масштабувати систему та додавати новий функціонал у майбутньому.

У цьому проєкті використовується реляційна база даних на базі системи PostgreSQL, яка є однією з сильних сучасних систем управління базами даних з відкритим кодом. Вибір PostgreSQL продиктований її широкими можливостями, високою надійністю, а також гнучкістю, що важливо для складних комерційних систем з великими обсягами даних і високою одночасною активністю користувачів. Однією з головних переваг PostgreSQL є її повна відповідність стандартам SQL, що дозволяє виконувати складні запити, включно з віконними функціями, рекурсивними запитами та загальними табличними виразами. Крім того, ця СУБД має розширювану архітектуру: можна створювати власні типи даних, оператори, функції, що забезпечує гнучкість і адаптивність до специфічних вимог бізнесу.

Особливістю PostgreSQL є механізм багатоверсійної конкурентності (MVCC), який дозволяє одночасно виконувати велику кількість транзакцій без блокування читання даних. Це забезпечує високу продуктивність і стабільність роботи інтернет-магазину навіть під час пікових навантажень, коли одночасно обробляються замовлення, оновлюється асортимент і

формуються звіти. Перевагою є також розширений набір типів даних, серед яких є підтримка JSON і JSONB — форматів, які ідеально підходять для зберігання напівструктурованих і динамічних даних. Це дозволяє гнучко зберігати різноманітні характеристики товарів, не змінюючи структуру таблиць, що значно спрощує роботу з каталого мобільних телефонів.

PostgreSQL підтримує повноцінні ACID-транзакції, що гарантує цілісність і надійність збережених даних. Це особливо важливо в умовах комерційної діяльності, де втрата або дублювання інформації про замовлення неприпустимі. Завдяки цим властивостям система надійно контролює паралельні операції, знижуючи ризики помилок при одночасних змінах даних. СУБД також пропонує можливості роботи з географічними даними завдяки розширенню PostGIS, що може бути корисним при розвитку мережі доставки або фізичних магазинів.

Масштабованість PostgreSQL забезпечується за рахунок підтримки різних типів реплікації, що дозволяє створювати надійні резервні копії і організовувати балансування навантаження. Це дозволяє зберегти стабільність роботи магазину навіть під час великих навантажень, наприклад, під час сезонних акцій або розпродажів. Одним з факторів швидкості пошуку і вибірки даних є підтримка різноманітних типів індексів, що дозволяє ефективно опрацьовувати запити за різними параметрами, у тому числі за текстовими описами товарів або їх характеристиками.

Безпека даних в PostgreSQL забезпечується за допомогою розвинених механізмів контролю доступу, шифрування з’єднань і журналювання дій користувачів, що дуже необхідно для захисту комерційної інформації та персональних даних клієнтів. Крім того, серверна логіка бази даних дозволяє реалізувати складні бізнес-процеси безпосередньо на рівні СУБД за допомогою тригерів, процедур і правил, що оптимізує роботу додатку і підвищує узгодженість даних.

Логічна структура бази даних спроєктована відповідно до принципів нормалізації, що дозволяє уникнути дублювання інформації і зберігати

цілісність даних. Всі основні сутності — товари, категорії, характеристики — розподілені по окремих таблицях із встановленими зв’язками через зовнішні ключі. Така організація надає гнучкість для швидкого внесення змін у каталог товарів, а також полегшує підтримку і розвиток системи. Зокрема, характеристики товарів винесені в окрему таблицю, що дає змогу додавати нові параметри без зміни основної структури.

Важливо також відзначити, що безпека і масштабованість бази даних — це основа надійної роботи інтернет-магазину. Система забезпечує резервне копіювання, багаторівневий контроль доступу, аудит і моніторинг, що знижує ризики втрати або несанкціонованого доступу до даних. Реплікація і кластеризація дозволяють підтримувати високу доступність сервісу, зменшуючи час простою.

Інтернет-магазин, що працює з PostgreSQL, має змогу ефективно інтегруватися з клієнтськими додатками — як веб-інтерфейсом, так і застосунком для перегляду 3D-моделей телефонів. Централізоване сховище і стандартизовані API забезпечують синхронність і актуальність інформації у всіх компонентах системи.

Отже, вибір PostgreSQL для проєкту бази даних інтернет-магазину мобільних телефонів є оптимальним рішенням, що поєднує функціонал, гнучкість, надійність і масштабованість, необхідні для реалізації складних і вимогливих бізнес-процесів сучасної електронної комерції.

3.2. Розробка бази даних застосунку

Таблиця Goods призначена для зберігання детальної інформації про товари, які доступні у каталозі. Вона має унікальний ідентифікатор id, що автоматично збільшується при додаванні нових записів, що забезпечує однозначну ідентифікацію кожного товару. Поле categoryID встановлює зв’язок з таблицею категорій, що дозволяє логічно впорядкувати товари за певними групами, спрощуючи навігацію та фільтрацію в системі.

Назви товарів та їх описи представлені двома мовами — англійською та українською — для підтримки мультимовного інтерфейсу, що є важливим у багатомовних проєктах та розширює потенційну аудиторію користувачів. Аналогічно, ціни зберігаються у текстовому форматі, що надає можливість гнучко відображати різні валюти, формати запису або додаткові символи (наприклад, валютні знаки чи позначки акцій).

Поле sale використовується для зберігання відсотка знижки, що застосовується до товару, що дозволяє ефективно реалізовувати акційні пропозиції. Вміст поля photo — це шлях або посилання на зображення, яке ілюструє товар, що є важливим для візуального сприйняття і підвищення привабливості товарів у каталозі.

Автоматичне встановлення дати і часу створення запису у полі createDateTime дає змогу відслідковувати хронологію додавання товарів, що може бути корисним для аналітики або керування контентом. Поле count відображає кількість одиниць товару, доступних на складі, що дозволяє контролювати запаси і планувати поповнення. Значення isShown визначає, чи має товар відображатися у публічній частині сайту, даючи змогу приховувати тимчасово недоступні або зняті з продажу позиції без видалення їх з бази.

Поле goodsUrlName містить SEO-оптимізовану назву товару, яка використовується для формування веб-адрес, що покращує індексацію та позиціювання сторінок у пошукових системах.

Зв’язок таблиці Goods з таблицею GoodsCharacteristic дозволяє додавати та зберігати набір характеристик для кожного товару, що розширює інформаційне наповнення і дозволяє деталізувати опис товару відповідно до специфіки.

Сутність Goods описана в табл. 3.1

Таблиця 3.1

Опис сутності Goods

Поле	Тип даних	Опис
id	integer	Унікальний ідентифікатор товару
categoryID	integer	Ідентифікатор категорії, до якої належить товар
name_En	text	Назва товару англійською мовою
name_Ua	text	Назва товару українською мовою
description_En	text	Опис товару англійською мовою
description_Ua	text	Опис товару українською мовою
price_En	text	Ціна у текстовому форматі англійською
price_Ua	text	Ціна у текстовому форматі українською
sale	integer	Відсоток знижки
photo	text	Посилання на зображення товару
createDateTime	timestamp	Дата і час створення запису
count	integer	Кількість товару в наявності
isShown	boolean	Визначає, чи відображати товар у публічній частині
goodsUrlName	text	SEO-оптимізована назва для URL

Таблиця Category зберігає інформацію про категорії товарів, що дозволяє структурувати каталог за логічними групами. Кожна категорія має унікальний ідентифікатор id, який служить первинним ключем і забезпечує однозначність записів. Поля name_En та name_Ua містять назви категорій

двома мовами, що відповідає загальній мультимовній концепції системи та дозволяє коректно відображати інформацію залежно від мови інтерфейсу.

Поле isOnBar використовується для позначення певної категорії як такої, що може відображатися в специфічних розділах або на панелі, залежно від бізнес-логіки. Це може бути корисним для швидкого доступу до популярних або акцентованих груп товарів. Поле url містить текстове значення для формування URL-адреси категорії, що оптимізує структуру посилань з урахуванням SEO та покращує навігацію користувачів.

Загалом, таблиця виконує функцію організації товарів у зручні та зрозумілі блоки, що спрощує як адміністрування каталогу, так і пошук товарів для кінцевих користувачів. Зв’язки з таблицею Goods реалізують множинність: одна категорія може містити багато товарів.

Опис сутності Category знаходиться в табл. 3.2

Таблиця 3.2

Опис сутності Category

Поле	**Тип даних**	**Опис**
id	integer	Унікальний ідентифікатор категорії
name_En	text	Назва категорії англійською мовою
name_Ua	text	Назва категорії українською мовою
isOnBar	boolean	Позначка для відображення категорії у певних розділах
url	text	Текст для формування URL-адреси категорії

Таблиця Characteristic призначена для зберігання інформації про характеристики, які можна застосовувати до товарів у каталозі. Кожен запис у цій таблиці має унікальний ідентифікатор id, що є первинним ключем, та поля

name_En і name_Ua, які містять назви характеристики англійською та українською мовами відповідно. Такий підхід забезпечує підтримку мультимовності і гнучкість у відображенні даних на різних мовах.

Ця таблиця не містить жодної додаткової інформації про те, як саме характеристика використовується, але служить основою для зв’язку з таблицею GoodsCharacteristic, де конкретні значення цих характеристик прив’язуються до конкретних товарів. Завдяки цьому можна детально описувати товари, надаючи їм різноманітні атрибути, які можуть бути корисними для користувачів при виборі або порівнянні товарів.

Структура таблиці є досить простою, оскільки вона виконує роль довідника, у якому зберігаються типи характеристик, а не конкретні значення.

Опис сутності Characteristic знаходиться в табл. 3.3

Таблиця 3.3

Опис сутності Characteristic

Поле	Тип даних	Опис
id	integer	Унікальний ідентифікатор характеристики
name_En	text	Назва характеристики англійською мовою
name_Ua	text	Назва характеристики українською мовою

Таблиця GoodsCharacteristic виконує функцію зв’язкової таблиці між товарами та їх характеристиками. Вона містить унікальний ідентифікатор id для кожного запису, а також поля characteristicId і goodsId, які встановлюють зовнішні ключі на відповідні записи у таблицях Characteristic та Goods. Такий зв’язок реалізує відношення «багато до багатьох», де один товар може мати кілька характеристик, а одна характеристика — належати кільком товарам.

Крім цього, таблиця зберігає конкретні значення характеристики для кожного товару у двомовному форматі: value_En і value_Ua. Це дає змогу показувати користувачам детальну інформацію про товар із врахуванням

мовних особливостей інтерфейсу. Наприклад, для характеристики «колір» значення можуть бути «red» і «червоний» відповідно.

За допомогою цієї таблиці можна гнучко розширювати опис товарів, додаючи різні параметри, що підвищує інформативність каталогу і допомагає користувачам робити усвідомлений вибір.

Опис сутності GoodsCharacteristic знаходиться в табл. 3.4

Таблиця 3.4

Опис сутності GoodsCharacteristic

Поле	Тип даних	Опис
id	integer	Унікальний ідентифікатор запису
characteristicId	integer	Ідентифікатор характеристики (зовнішній ключ)
goodsId	integer	Ідентифікатор товару (зовнішній ключ)
value_En	text	Значення характеристики англійською мовою
value_Ua	text	Значення характеристики українською мовою

Таблиця Order призначена для зберігання інформації про замовлення, які здійснюють користувачі. Кожне замовлення має унікальний ідентифікатор id, що автоматично генерується. Поле customer містить дані про замовника — це може бути ім’я або інша інформація для ідентифікації клієнта. Поле adress зберігає адресу доставки або місце отримання замовлення, що важливо для логістики.

У полі description може зберігатися додаткова інформація або коментарі, пов’язані із замовленням, наприклад, особливі побажання клієнта або деталі, які необхідно врахувати при виконанні. Загальна сума замовлення зберігається

у полі total_price і представлена цілим числом, що спрощує подальші обчислення та звіти.

Ця таблиця формує основу для обробки замовлень, дозволяючи зберігати важливі атрибути, що потрібні для організації роботи з клієнтами та виконання замовлень.

Опис сутності Order знаходиться в табл. 3.5

Таблиця 3.5

Опис сутності Order

Поле	Тип даних	Опис
id	integer	Унікальний ідентифікатор замовлення
customer	text	Інформація про замовника
adress	text	Адреса доставки або отримання замовлення
description	text	Додаткова інформація чи коментарі
total_price	integer	Загальна сума замовлення

Таблиця OrderedGoods відображає зв’язок між замовленнями та товарами, які до них входять. Кожен запис цієї таблиці містить унікальний ідентифікатор id, а також поля Goods і Order, які є зовнішніми ключами до відповідних таблиць товарів і замовлень. Таким чином, вона реалізує багато-до-багатьох зв’язок між товарами та замовленнями.

Поле Count зберігає кількість конкретного товару в замовленні, що дозволяє відслідковувати точні обсяги продажів кожного товару. Ця таблиця є необхідною для коректного обліку замовлених позицій, оскільки одне замовлення може містити кілька різних товарів у різних кількостях.

Також використання зовнішніх ключів забезпечує цілісність даних, гарантує зв’язок лише з існуючими замовленнями та товарами і дозволяє автоматично підтримувати оновлення та видалення.

Опис сутності OrderedGoods знаходиться в табл. 3.6

Таблиця 3.6

Опис сутності OrderedGoods

Поле	**Тип даних**	**Опис**
id	integer	Унікальний ідентифікатор запису
Goods	integer	Ідентифікатор товару (зовнішній ключ)
Order	integer	Ідентифікатор замовлення (зовнішній ключ)
Count	integer	Кількість товару у відповідному замовленні

Усі описані таблиці утворюють цілісну структуру, що дозволяє ефективно зберігати інформацію про товари, їх характеристики, категорії, а також замовлення та асоційовані з ними позиції. Взаємозв’язки між таблицями забезпечують цілісність даних і підтримують масштабованість системи.

Висновки до розділу 3

У цьому розділі було розглянуто структуру бази даних, що використовується для обліку товарів та замовлень у системі інтернет-каталогу. Основна увага приділена послідовному опису кожної таблиці, їхніх полів і взаємозв’язків, що відповідають принципам реляційної моделі та нормалізовані для уникнення надмірності й забезпечення цілісності даних. Для кожного поля наведено тип даних та функціональне призначення, що дає повне уявлення про організацію інформації. Опис структури бази даних дозволяє зрозуміти логіку збереження, обробки та взаємодії об’єктів у межах системи.

Окрім опису структури, розділ охоплює особливості розробки бази даних, починаючи з визначення її основних складових. Увага зосереджена на забезпеченні цілісності інформації та уникненні дублювання шляхом правильного моделювання таблиць і зв’язків. Підхід до побудови бази даних базується на послідовному формуванні логічної структури, що відповідає вимогам системи інтернет-каталогу, з урахуванням функціональних ролей кожного елемента. Такий підхід дозволяє ефективно реалізувати зберігання й облік товарів та замовлень.

РОЗДІЛ IV
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Розробка каталогу

Головним інтерфейсом програми є каталог смартфонів, який реалізований у вигляді вікна MainForm [Додаток А]. Тут користувач може переглядати перелік товарів, сортувати їх за категоріями, а також отримувати детальну інформацію про кожен пристрій (рис. 4.1.).

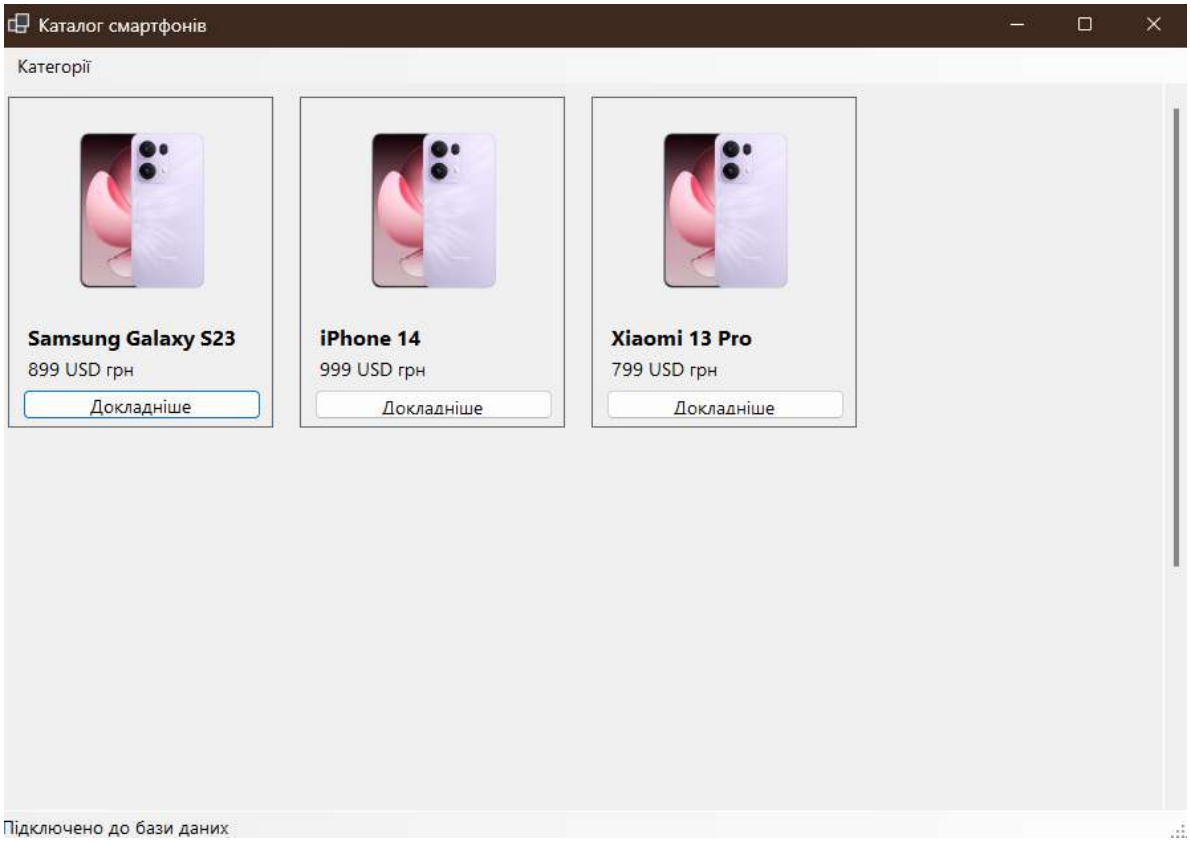

Рис. 4.1. Головне вікно програми з каталогом

При запуску програми одразу встановлюється з'єднання з базою даних PostgreSQL, де зберігаються всі дані про смартфони, категорії та інші параметри. Якщо підключення відбувається успішно, у статусному рядку з'являється повідомлення про успішне з'єднання (рис. 4.2.).

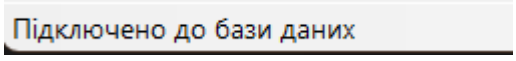

Рис. 4.2. Статус підключення до бд

Інтерфейс каталогу складається з кількох основних елементів. У верхній частині вікна розташоване меню категорій (рис. 4.3.),. Воно автоматично заповнюється даними з таблиці категорій бази даних. Категорії вибираються за ознакою isOnBar, що дозволяє легко керувати тим, які розділи будуть доступні користувачу. Кожна категорія у меню представлена як елемент ToolStripMenuItem, і при виборі категорії відбувається оновлення списку смартфонів, які належать до цієї категорії.

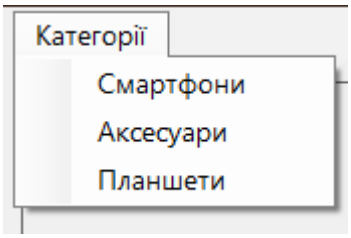

Рис. 4.3. Категорії

Основна частина вікна займає FlowLayoutPanel, у якому кожен смартфон представлений у вигляді окремої панелі (рис. 4.4.). Ця панель містить зображення пристрою, його назву, ціну та кнопку «Докладніше».

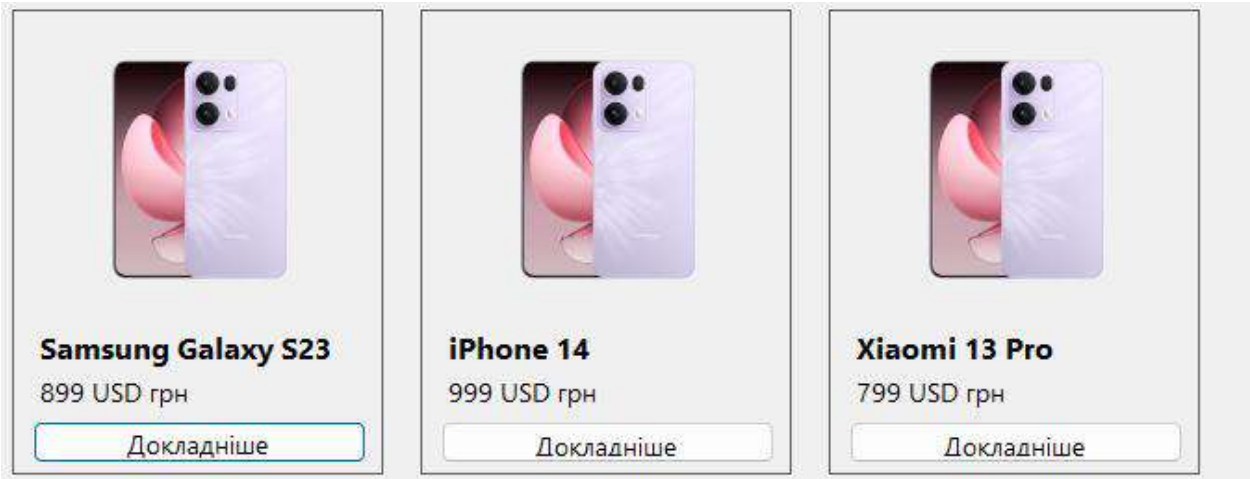

Рис. 4.4. Панель товарів

Зображення смартфонів завантажуються з файлів, шлях до яких зберігається у базі даних. Програма передбачає перевірку існування файлу, і якщо фотографія відсутня, замість неї показується стандартне зображення-заглушка.

Після кліку по кнопці «Докладніше» для певного смартфона відкривається нове вікно SmartphoneDetailsForm [Додаток Б]. Це вікно містить детальну інформацію про пристрій, отриману з бази даних, включно з описом, технічними характеристиками та 3д-моделю смартфону (рис. 4.5.).

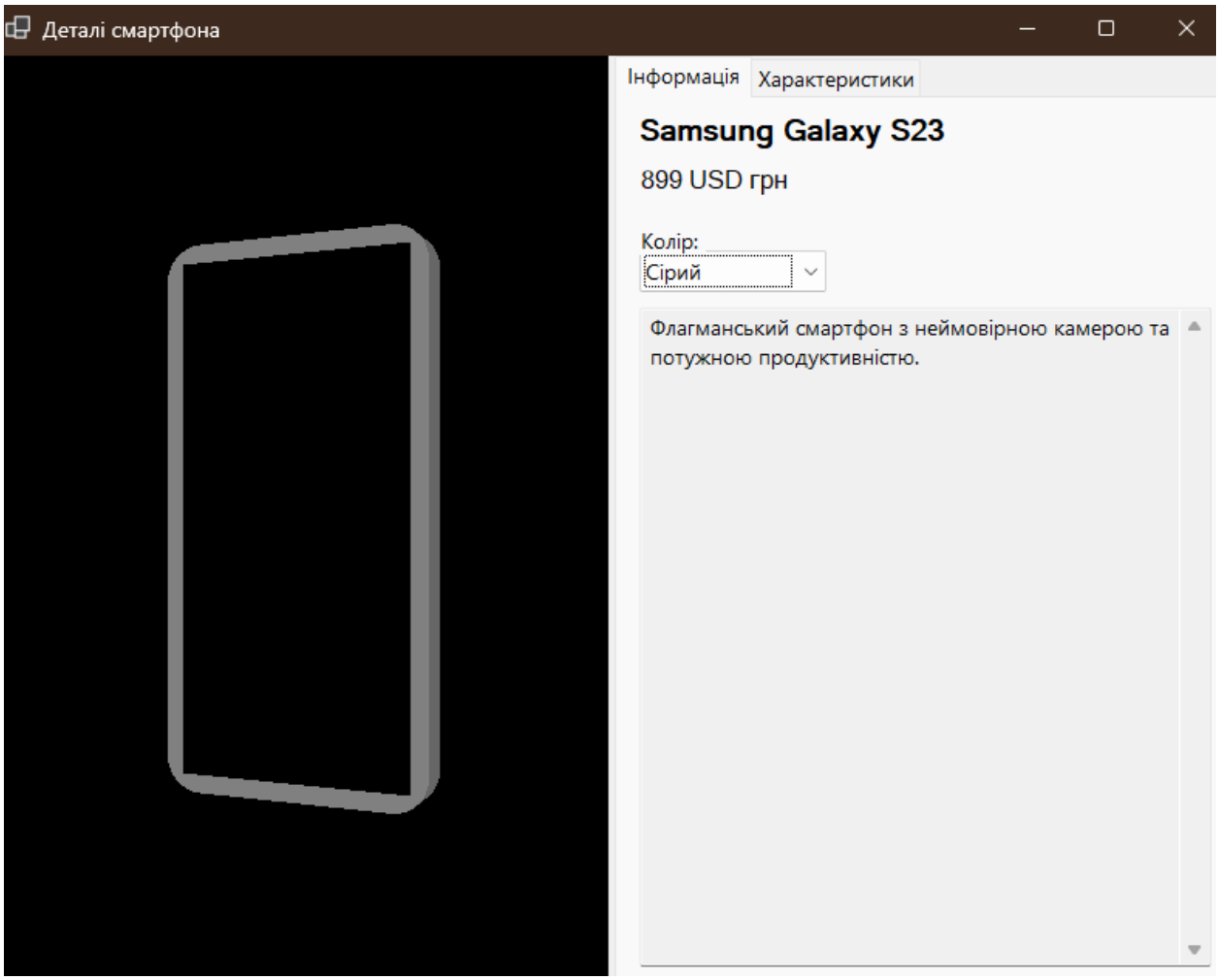

Рис. 4.5. Деталі про смартфон

Вікно деталей налаштоване на забезпечення зручного перегляду інформації: текст розбитий на блоки, модель розташована зліва, інформація розташована у двох вкладках - детальних характеристик та основної важливої інформації про смартфон.

Вкладка характеристик реалізована у вигляді окремої панелі, де структуровано представлені ключові технічні характеристики смартфона — такі як процесор, обсяг оперативної пам’яті, тип і розмір екрану, ємність батареї, камера та інші важливі параметри. Інформація завантажується з бази

даних і відображається у вигляді таблиці з назвами характеристик та відповідними значеннями (рис. 4.6.).

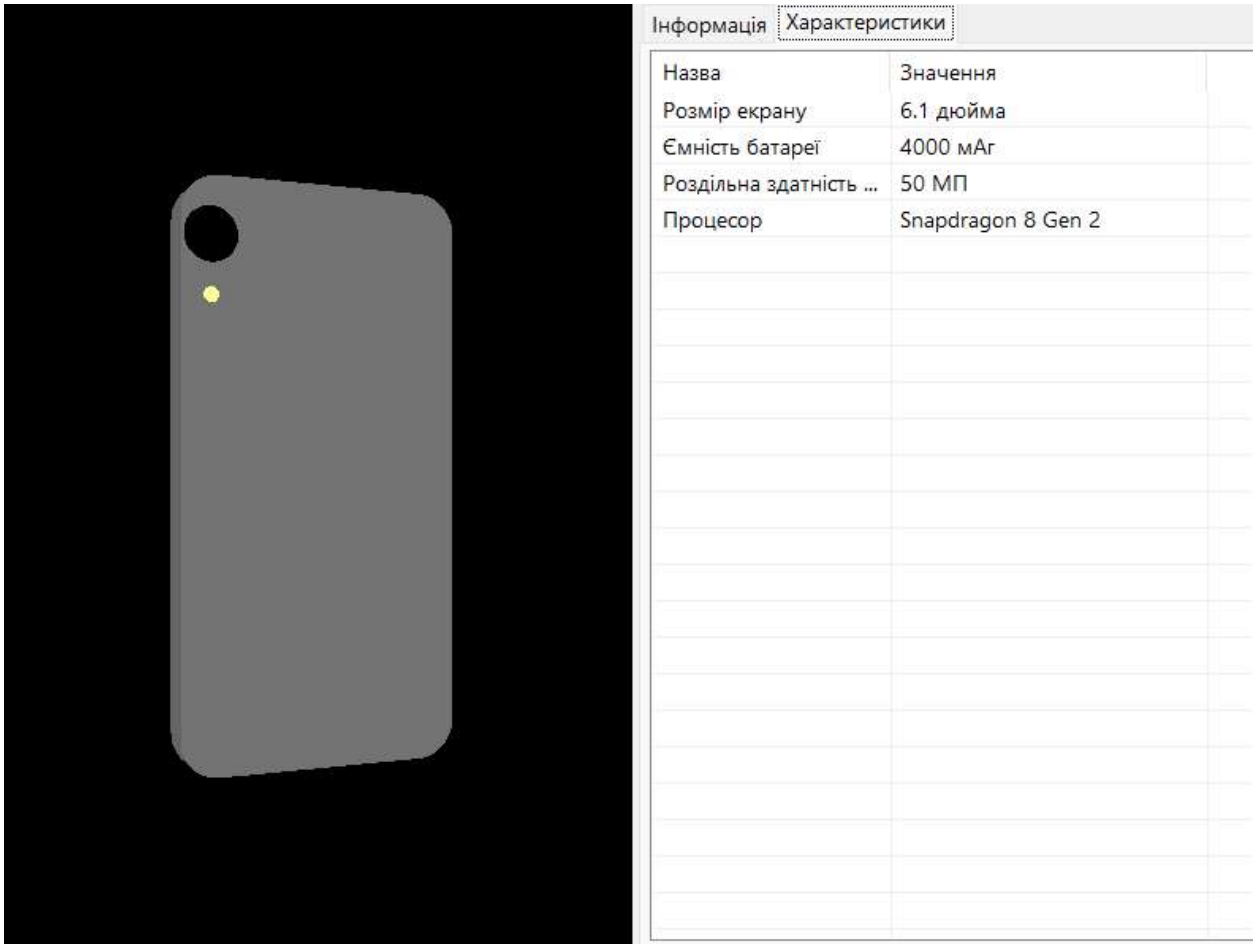

Рис. 4.6. Характеристики

Однією з головних особливостей додатку є наявність інтегрованої 3D-моделі пристрою, яка розташована у лівій частині вікна перегляду деталей смартфона (рис. 4.7). Ця модель дозволяє користувачу розглядати смартфон у тривимірному форматі без потреби відкривати сторонні вікна чи ресурси. Вікно побудоване таким чином, що зліва виводиться віртуальне зображення пристрою, яке обертається. В інтерфейсі реалізовано можливість зміни кольору корпусу смартфона - сірий, білий, червоний (рис. 4.8.), синій (рис. 4.9.).

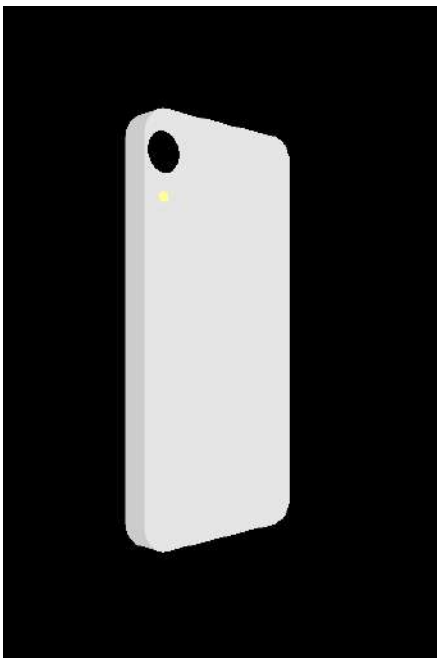

Рис. 4.7. 3D-моделі пристрою

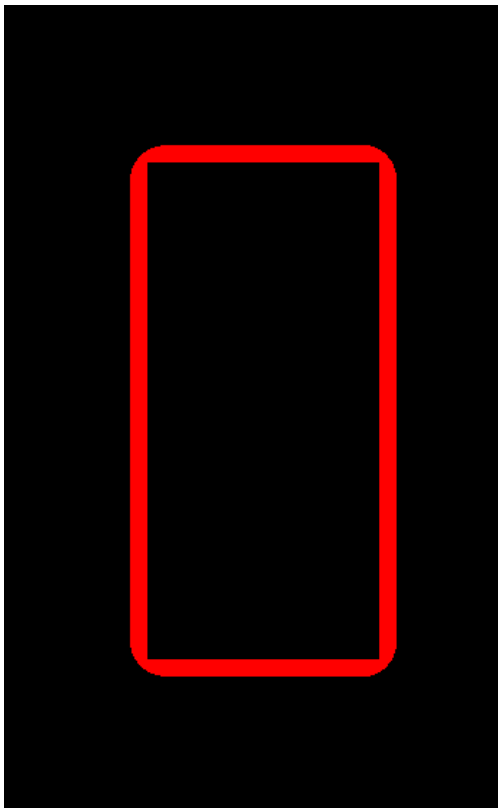

Рис. 4.8. Червоний колір

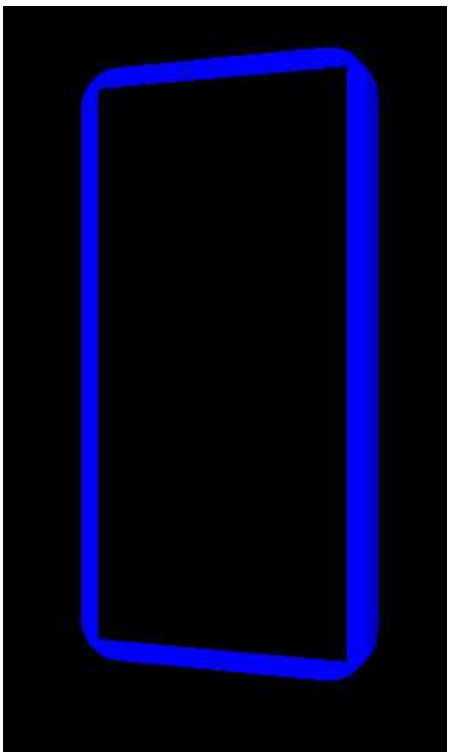

Рис. 4.9. Синій колір

Користувач може вибрати бажаний колір із запропонованого списку, після чого модель автоматично змінює відтінок відповідно до вибору. Ця функція передбачає сценарій, коли один і той самий смартфон представлено у кількох кольорах.

4.2. Розробка серверної частини сайту

Серверна частина системи реалізована за допомогою мови JavaScript із використанням середовища виконання Node.js, фреймворку Express та бібліотеки Apollo Server, що дозволяє організувати взаємодію між клієнтською частиною та базою даних через GraphQL API. В основі структури серверного застосунку лежить поділ на логічні блоки, де окремо реалізовані запити, мутації, утиліти, а також механізми генерації типів GraphQL. Усі запити та мутації винесено у відповідні папки, кожна з яких містить окремі файли для конкретних операцій, що значно спрощує навігацію, модифікацію та масштабування системи. Початковим файлом для запуску є index.js, у якому

виконується ініціалізація сервера, конфігурація підключення до бази даних PostgreSQL через Prisma ORM та запуск GraphQL-сервера.

GraphQL-схема проєкту описана у файлі schema.graphql і містить типи, що відповідають сутностям бази даних: товари, категорії, полиці та характеристики. Кожен тип містить як основні текстові поля (назви, описи, ціни кількома мовами), так і додаткові – наприклад, дата створення або URL-представлення. Для товарів також реалізована можливість зберігати масив характеристик, що пов’язані через окрему таблицю, де для кожного значення зберігається його значення українською та англійською мовами. Типи мутацій дозволяють додавати товари, оновлювати їх, створювати нові категорії та полиці, а також завантажувати пов’язані з товарами файли зображень.

У моделі бази даних, визначеній у файлі schema.prisma, реалізовано всі необхідні зв’язки між сутностями, зокрема зовнішні ключі, що визначають належність товару до певної категорії або характеристики. Модель Goods містить поле isShown, яке використовується для фільтрації актуальних товарів на сайті, а також поле goodsUrlName, що служить для зручного створення посилань на конкретний товар. Особливістю реалізації є збереження дати створення товару автоматично, що забезпечується атрибутом @default(now()). Для взаємодії з базою даних використовуються можливості Prisma, які дозволяють виконувати як стандартні запити, так і складні фільтрації або пошуки.

На етапі запуску сервер автоматично зчитує змінні середовища з файлу .env, де міститься URL підключення до бази. Після цього виконується створення контексту GraphQL, який дозволяє передавати до кожного запиту дані про підключення та інші глобальні параметри. Для кожного запиту реалізовано окрему функцію, яка обробляє параметри, отримані від клієнта, виконує запит до бази даних через Prisma і повертає результат у відповідності до схеми. Аналогічно побудовано мутації, де, окрім збереження даних, передбачено обробку файлів зображень за допомогою бібліотеки graphql-upload. Це дає змогу завантажувати фотографії товарів разом із даними та

зберігати їх у відповідному форматі на сервері. Усі завантажені файли обробляються через окремі утиліти, що дозволяє легко змінювати формат або оптимізацію зображень при потребі.

Для взаємодії клієнтської частини з базою даних у застосунку реалізовано базовий набір GraphQL-запитів і мутацій, які покривають основні сценарії роботи з товарами. Зокрема, реалізовано можливість отримання списку пристроїв за обраною категорією, що дає змогу формувати динамічний каталог на сайті. У межах цього запиту дані фільтруються за наявністю та видимістю, а також сортуються за датою, що дозволяє виводити найактуальніші позиції першими.

Мутації охоплюють стандартні операції — додавання нового товару, оновлення його даних та видалення з каталогу. При створенні нового пристрою сервер очікує передані дані у структурованому вигляді, включно з назвою, описом, ціною, зображенням та характеристиками. Під час оновлення можна змінити будь-яке з цих полів, зберігаючи гнучкість у керуванні товарним асортиментом. Видалення реалізовано м’яким способом: товари позначаються як неактивні, що дозволяє зберігати їх у базі для подальшого аналізу або відновлення.

Також у схемі передбачено додаткові запити для отримання категорій, детальної інформації про товар за символьним ім’ям, а також списку властивостей, які використовуються для формування технічних характеристик у інтерфейсі. Такий набір запитів і мутацій дозволяє побудувати повноцінний функціонал магазину з можливістю легкої масштабованості та адаптації під майбутні потреби.

Для зберігання та обробки даних на сервері використовується PostgreSQL [Додаток Г], з яким взаємодія здійснюється за допомогою ORM Prisma. Усі структури таблиць описані в окремому файлі schema.prisma, що дозволяє централізовано керувати моделями й автоматично генерувати клієнтські методи для запитів до бази даних. Міграції реалізовано через Prisma Migrate — кожна зміна структури бази даних фіксується в окремій версії з

відповідним SQL-файлом, що забезпечує контроль і простежуваність усіх змін у процесі розробки.

Обробка запитів і мутацій реалізована в окремих модулях для кращої організації коду. Кожна функція має чітко визначене призначення та розміщується у відповідній папці (queries або mutations). Це дозволяє легко розширювати функціонал та підтримувати чисту архітектуру проєкту. Усі запити проходять через центральну точку входу, де ініціалізується контекст, який включає підключення до бази даних, роботу з файлами та інші утиліти.

Окрім основної логіки, у серверній частині передбачено обробку завантаження зображень. Завдяки підтримці типу Upload, користувач може передавати фото товару, яке зберігається у файловій системі, а шлях до нього записується в базу даних. Додатково передбачено обробку запитів на генерацію типів для зручної розробки клієнтської частини. Цей процес автоматизовано через graphql-codegen, що дозволяє підтримувати актуальність типів відповідно до поточного стану схеми GraphQL.

Таким чином, серверна частина проєкту забезпечує надійне зберігання, обробку та надання даних, а також легко масштабується завдяки модульній структурі та сучасним засобам роботи з базами даних і API.

4.3. Розробка клієнтської частини сайту

Клієнтська частина веб-застосунку реалізована за допомогою React, що забезпечує швидку реакцію інтерфейсу на дії користувача, підтримує компонентну структуру та спрощує повторне використання елементів інтерфейсу. Основна логіка розміщена в компоненті MainPage, що є точкою входу для головного контенту сайту.

Компонент MainPage використовує хук useQuery з бібліотеки Apollo Client для отримання даних з GraphQL-серверу. Запит GET_ALLGOODS надсилається з параметром take: 10, що обмежує кількість отриманих товарів.

Поки триває завантаження або сталася помилка, користувач бачить компонент-заглушку LoadingPage.

У разі успішного отримання даних, головна сторінка рендерить такі компоненти:

- Banner — слайдер банерів;
- GoodsSection — секція з товарами, що приймає масив товарів з сервера;
- RecentlyViewedSection — секція з нещодавно переглянутими товарами, яка використовує глобальний стан (AppContext) для зберігання історії переглядів.

На рис. 4.10 зображено зовнішній вигляд головної сторінки після завантаження даних з сервера. У верхній частині розміщено навігаційне меню, яке містить логотип, поле для пошуку та іконку кошика. Ліворуч відображено бокову панель з категоріями товарів, а також перемикачі мови (UA / EN). Центральна частина зайнята банером, який реалізовано через компонент Banner.

Рис. 4.10. Головна сторінка

Компонент Banner використовує модуль ThemedSwiper, побудований на бібліотеці Swiper. До нього підключені модулі Autoplay, Navigation та Pagination, що дозволяє створити інтерактивний слайдер з керуванням та

автоматичним перемиканням зображень. Зображення для банера зберігаються локально та підключаються через require.

Компонент GoodsSection приймає два пропси: заголовок секції (title) та масив товарів (goods). Для відображення товарів також використовується ThemedSwiper, але вже з підтримкою горизонтального скролу (slidesPerView='auto') та стрілок навігації. Кожен товар обгортається у компонент GoodsCard. Це дозволяє створити зручний, адаптивний перегляд великої кількості товарів без перевантаження інтерфейсу.

Остання секція — RecentlyViewedSection — виводиться лише в тому разі, якщо є щонайменше один нещодавно переглянутий товар. Вона використовує дані з глобального контексту AppContext.

На рис. 4.11 показано вигляд сторінки товару, яка відкривається при кліку на будь-яку картку. Вона містить повне зображення товару, його назву, ціну, кнопку "Add to cart", а також статус доступності (наприклад, "Available").

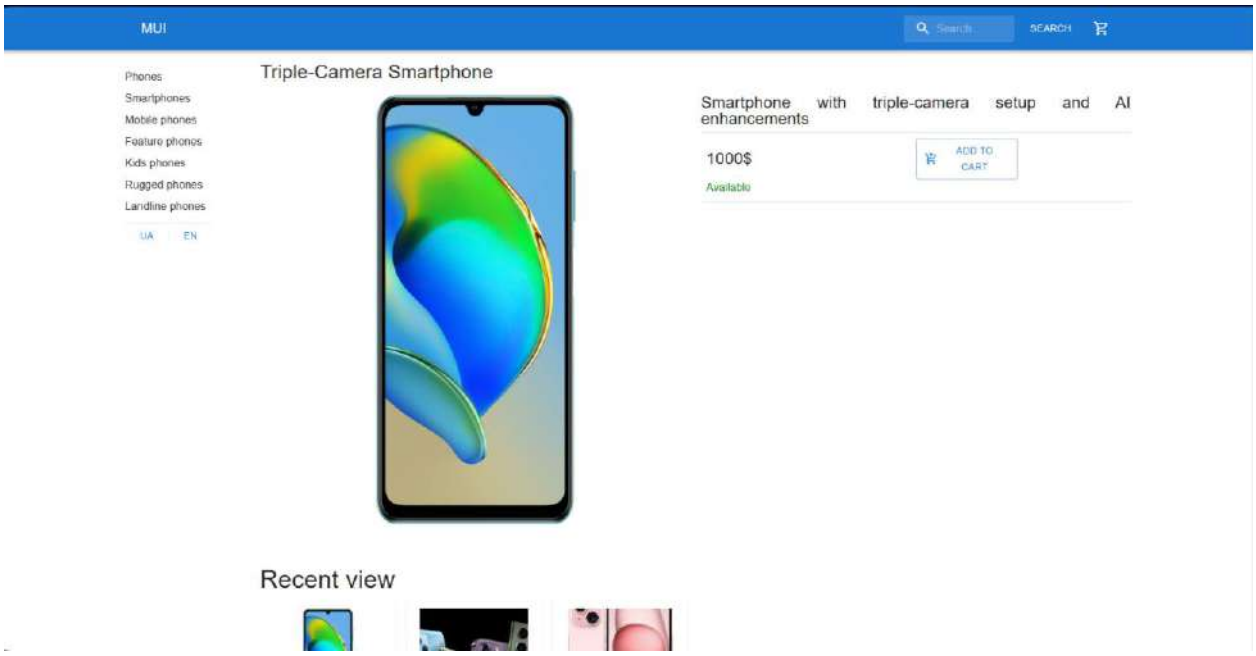

Рис. 4.11. Сторінка товару

Нижче на сторінці товару розташована таблиця з характеристиками пристрою (рис. 4.12.), де відображено параметри, такі як вага (наприклад,

"120g") та колір (наприклад, "White"). Структура цієї секції дозволяє швидко орієнтуватися в основних властивостях товару.

Рис. 4.12. Секція характеристик товару у вигляді таблиці

В самому низу сторінки реалізовано блок "Recent view" (рис. 4.13.), який допомагає користувачеві легко повернутись до раніше переглянутих позицій. Ця функціональність особливо корисна при ознайомленні з кількома варіантами товарів.

Рис. 4.13. Блок "Recent view"

Сторінка категорій реалізована як окремий React-компонент CategoryPage, який відповідає за динамічне завантаження та відображення товарів, що належать до певної категорії. Вона є ключовою частиною сайту, яка дозволяє користувачеві швидко орієнтуватися в асортименті, переглядати потрібні групи товарів і змінювати сторінки результатів.

Запит до бази даних здійснюється за допомогою GraphQL та бібліотеки Apollo Client. У запиті GET_GOODS передаються змінні category, skip та take. Вони дозволяють отримати лише ту частину товарів, яка відповідає поточній

сторінці пагінації. Кількість товарів на одну сторінку обмежена константою MAX_GOODS_ON_PAGE, яка зазвичай дорівнює 12.

Реалізовано підтримку зміни сторінки за допомогою компонента Pagination з MUI. Поточна сторінка зберігається у стані currentPage, а при переході на іншу категорію (зміна параметра category в URL) лічильник сторінки скидається на початок.

На рис. 4.14 зображено сторінку, що відображає товари обраної категорії — наприклад, "Phones". Товари представлені у вигляді сітки (Grid), де кожен елемент є карткою товару, реалізованою у вигляді окремого компонента GoodsCard.

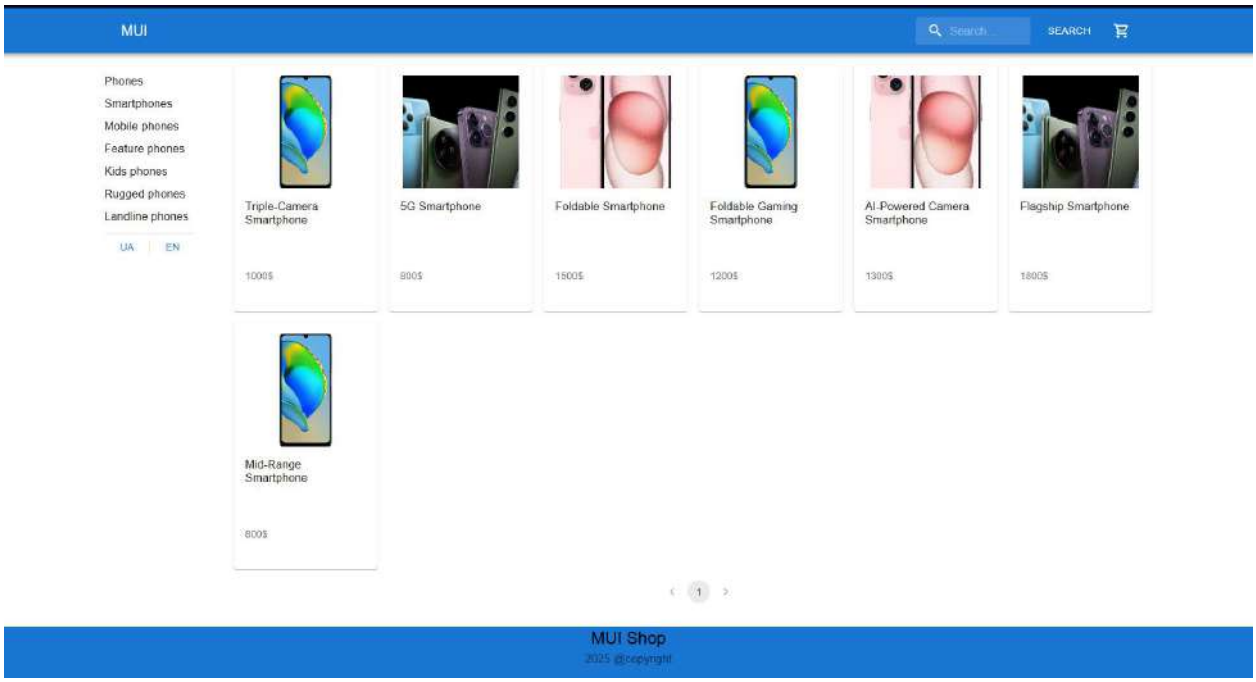

Рис. 4.14. Сторінка категорії товарів

Компонент GoodsCard відображає основну інформацію про товар: зображення, назву та ціну. Зображення автоматично масштабується у межах контейнера (objectFit: 'cover'), а при кліку користувача на зображення або назву здійснюється навігація на сторінку товару за допомогою функції navigate. Посилання формується на основі властивості goods.goodsUrlName, що дозволяє уникати дублювання назв товарів і створює унікальні, зручні для SEO URL-адреси (рис. 4.15.).

Рис. 4.15. Приклад картки товару

У нижній частині сторінки категорії розміщується блок з пагінацією. Компонент Pagination автоматично визначає кількість сторінок на основі загальної кількості товарів (count), отриманої з GraphQL-запиту. Це дозволяє підтримувати ефективну навігацію навіть при великій кількості результатів.

Коли товари певної категорії відсутні, користувач бачить повідомлення про відсутність результатів, яке локалізовано через власний хук useEasyTranslation.

Таким чином, сторінка категорії реалізована з урахуванням адаптивності, динамічності та масштабованості. Вона підтримує інтеграцію з сервером через GraphQL, використовує пагінацію для оптимізації відображення великої кількості товарів і забезпечує зручний інтерфейс для переходу до сторінки кожного товару.

Інтерфейс сайдбару реалізується за допомогою компонента Sidebar. Він містить вертикальний список категорій товарів, що динамічно завантажуються з сервера. Дані для відображення категорій отримуються через GraphQL-запит у файлі GET_CATEGORIES. Для виведення кожної категорії використовується компонент ListItemText, загорнутий у стилізоване посилання StyledLink. Внизу панелі розміщено перемикач мови — компонент

LanguageSwitcher, що дозволяє змінювати локалізацію інтерфейсу. Зовнішній вигляд сайдбару показано на рис. 4.16.

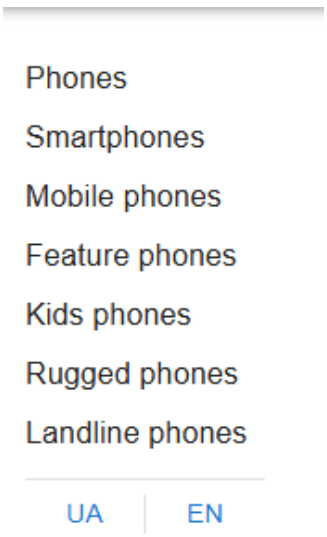

Рис. 4.16. Сайдбар

Сайдбар виводиться лише на екранах середнього розміру і більше. Це забезпечується компонентом SidebarRoutes, який ділить інтерфейс на дві колонки: ліву з компонентом Sidebar, праву — з контентом сторінки, що виводиться через компонент Outlet. На малих екранах панель приховується, а контент розтягується на всю ширину, а сам сайдбар відкривається при натисканні на спеціальну кнопку. Вигляд сайдбару на мобільних пристроях зображений на рис. 4.17.

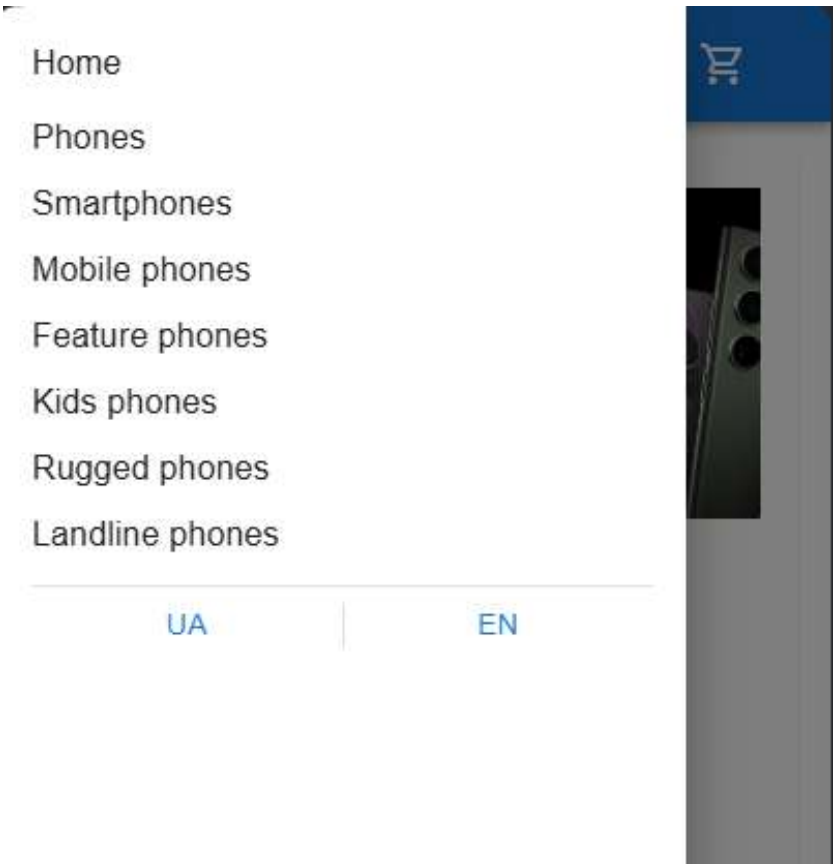

Рис. 4.17. Сайдбар на мобільному пристрої

Маршрутизація реалізована за допомогою компонента Routes [Додаток В], який містить повну структуру шляхів до сторінок сайту. В основі маршрутизації використовується BrowserRouter, що дозволяє перемикатись між сторінками без перезавантаження. Всі маршрути обгорнуті компонентом NavigateRoutes, що відповідає за логіку перенаправлень. Основні сторінки виводяться через вкладений компонент SidebarRoutes.

Для кожної сторінки визначено окремі компоненти: MainPage — головна сторінка, CategoryPage — сторінка категорії товарів, GoodsPage — сторінка конкретного товару, SearchPage — сторінка пошуку, NotFoundPage — повідомлення про помилку навігації. Усі вони визначаються як окремі маршрути в компоненті Routes.

Коли сторінка відкривається в межах структури з сайдбаром, вона виводиться всередині компонента SidebarRoutes, який завжди містить компонент Sidebar та Outlet, через який рендериться відповідний компонент сторінки.

Глобальний стан застосунку реалізований через контекст, оголошений у компоненті AppContext. Його значення формується за допомогою useReducer, який працює з логікою, описаною в окремому файлі reducer [Додаток Г]. Початковий стан визначається об'єктом defaultContext, що містить поля для переглянутих товарів, товарів у кошику та інших параметрів користувацької взаємодії.

Контекст огортає всі дочірні компоненти за допомогою обгортки AppContextProvider [Додаток Д]. Після монтування сторінки в контексті автоматично виконується перевірка локального сховища браузера. Ідентифікатори переглянутих товарів (VIEWS_ITEM) і товарів у кошику (CART_ITEM) зчитуються з localStorage, після чого виконується запит GET_GOODS_BY_IDS через useLazyQuery. Після отримання даних відповідні дії SET_CARD_ITEM і SET_VIEWS_ITEM оновлюють глобальний стан, зберігаючи повну інформацію про товари. До моменту завершення завантаження відображається компонент LoadingPage.

Ця реалізація дає змогу ефективно працювати з інформацією користувача навіть після оновлення сторінки або повторного входу на сайт. Вся логіка централізована, що спрощує керування станом у різних частинах застосунку.

Виведення окремих товарів реалізовано в компоненті GoodsCard, який використовується як на сторінках категорій, так і в кошику. Універсальність цього компонента дозволяє його багаторазове використання. Він відображає зображення товару, назву та ціну, а також містить логіку переходу на сторінку конкретного товару при кліку на назву чи зображення. Дані передаються через пропси у вигляді об’єкта goods. Зовнішній вигляд кошика можна побачити на рис. 4.18.

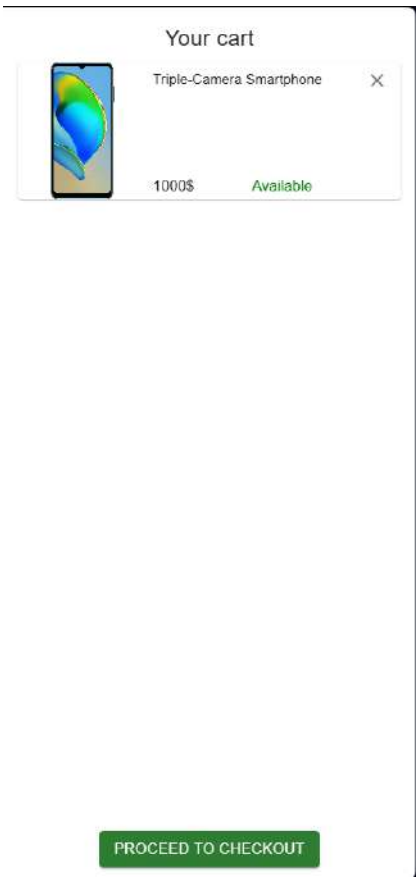

Рис. 4.18. Корзина

У застосунку реалізована підтримка багатомовного інтерфейсу за допомогою бібліотеки i18next. Для інтеграції з React використано обгортку I18nextProvider, яка надає доступ до функціоналу перекладу всім дочірнім компонентам. Провайдер міститься в компоненті I18nProvider, що загортає основну частину програми та забезпечує контекст перекладу.

Налаштування бібліотеки виконується в окремому файлі конфігурації. Використовуються плагіни i18next-browser-languagedetector для автоматичного визначення мови користувача та i18next-http-backend для завантаження мовних файлів із директорії /locales. За замовчуванням застосовується українська мова, якщо система не розпізнає іншу.

Мовні ресурси організовані у вигляді JSON-файлів, які поділені на тематичні блоки відповідно до назв компонентів. Наприклад, блок Header містить рядки, що використовуються в шапці сайту, Goods — у картках товарів. Це дозволяє швидко знаходити потрібні фрази та редагувати лише окремі частини, не зачіпаючи інші розділи інтерфейсу.

Завдяки такому підходу до локалізації, сайт може працювати двома мовами — українською та англійською — і легко розширюється новими мовами без необхідності змінювати основну логіку програми. Користувач може перемикати мову вручну, а система також автоматично враховує мовні налаштування браузера.

Клієнтська частина застосунку реалізована на React із використанням бібліотек Apollo Client для взаємодії з GraphQL, MUI — для візуального оформлення та React Router — для маршрутизації. Структура побудована так, щоб забезпечити масштабованість, повторне використання компонентів і збереження стану користувача. Завдяки контексту AppContext, сайт зберігає дані переглядів і кошика, дозволяючи користувачу легко орієнтуватися навіть після перезапуску. Також присутня локалізація на дві мови - українську та англійську. Загалом, клієнтська логіка є гнучкою, адаптивною і підтримує динамічну взаємодію без втрати даних.

Висновки до розділу 4

У розділі описано розробку програмного забезпечення для каталогу смартфонів та веб-сайту магазину. Основний інтерфейс — каталог із можливістю перегляду, сортування та детального огляду товарів, який взаємодіє з базою даних PostgreSQL. Особливістю є підтримка 3D-моделі смартфонів із вибором кольорів.

Серверна частина побудована на Node.js з Express і Apollo Server, використовуючи GraphQL API та ORM Prisma для ефективної роботи з базою даних. Архітектура серверу модульна, що полегшує масштабування й підтримку, забезпечуючи обробку запитів, мутацій та завантаження зображень.

Клієнтська частина реалізована на React з Apollo Client для взаємодії з сервером, що забезпечує динамічне і швидке оновлення інтерфейсу. Загалом

проєкт має сучасну, гнучку і масштабовану архітектуру, що підтримує комфортний користувацький досвід і простоту адміністрування.

ВИСНОВКИ

У результаті виконання даної кваліфікаційної роботи було досягнуто поставленої мети — створено програмне забезпечення для продажу смартфонів, яке поєднує веб-сайт для онлайн-покупок з функціоналом перегляду та оформлення замовлення, а також застосунок для візуалізації 3D моделі товару. Для реалізації використано сучасні засоби веб-розробки, а також технології тривимірної графіки на базі OpenGL.

У результаті виконаної роботи можна зробити такі висновки:

1. У процесі аналізу предметної області було досліджено сучасні технології створення веб-додатків та засоби побудови тривимірної графіки. На основі цього аналізу сформовано вимоги до програмного забезпечення, що дозволило визначити технічний стек і загальний підхід до реалізації.

2. Спроєктовано архітектуру системи, яка об'єднує веб-інтерфейс і застосунок у єдиний програмний комплекс. Було визначено логіку взаємодії між компонентами, схему роботи з базою даних та принципи передачі інформації між модулями.

3. Розроблено функціональний веб-сайт, який дозволяє користувачу здійснювати пошук смартфонів, переглядати характеристики товару та оформлювати замовлення. Реалізовано систему категоризації, фільтрації та базової інтерактивної взаємодії з користувачем.

4. Створено застосунок з інтеграцією OpenGL, що дозволяє користувачу ознайомитися з тривимірною моделлю смартфона. Візуалізація реалізована програмно, з використанням вершин, буферів та примітивів, без залучення сторонніх моделей. Забезпечено можливість масштабування, обертання та перегляду елементів корпусу смартфона.

Розроблене програмне забезпечення об’єднує переваги веб-технологій та засобів тривимірної візуалізації. Завдяки цьому користувач отримує не лише зручний інтерфейс для здійснення покупки, а й наближену до реальності можливість ознайомлення з товаром. Це підвищує довіру до платформи,

зменшує ризик помилки при виборі та сприяє загальному підвищенню ефективності онлайн-продажу смартфонів. Отримані результати демонструють актуальність запропонованого підходу та підтверджують доцільність впровадження подібних систем у сферу електронної комерції.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wikipedia [Электронный ресурс]. – Режим доступа : https://uk.wikipedia.org/

2. MSDN Documentation [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/

3. PostgreSQL Documentation [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/

4. Npgsql Official Site [Електронний ресурс]. – Режим доступу: https://www.npgsql.org/

5. Richter J. CLR via C# / J. Richter. – Microsoft Press, 2012.

6. Troelsen A., Japikse P. Pro C# 9 with .NET 5 / A. Troelsen, P. Japikse. – Apress, 2021.

7. OpenGL Official Documentation [Електронний ресурс]. – Режим доступу: https://www.khronos.org/opengl/

8. Albahari J., Albahari B. C# 10 in a Nutshell: The Definitive Reference / J. Albahari, B. Albahari. – O'Reilly Media, 2022.

9. Freeman A., Sharp M. Pro ASP.NET Core MVC 2 / A. Freeman, M. Sharp. – Apress, 2017.

10. Костильов С.І. Основи програмування на C# / С.І. Костильов. – Київ: КНУ, 2020.

11. Соловйов В.М. Технології розробки прикладного програмного забезпечення / В.М. Соловйов. – Львів: ЛНУ, 2019.

ДОДАТКИ

Додаток А

Форма каталогу

```
    public partial class MainForm : Form
    {

        private NpgsqlConnection connection;
        private List<SmartphoneItem> smartphoneItems = new
List<SmartphoneItem>();
        private string connectionString =
"Host=localhost;Username=postgres;Password=root;Database=technoSquirre
l";

        public MainForm()
        {
            InitializeComponent();
            ConnectToDatabase();
            LoadCategories();
            LoadSmartphones();
        }

        private void ConnectToDatabase()
        {
            try
            {
                connection = new NpgsqlConnection(connectionString);
                connection.Open();
                statusLabel.Text = "Підключено до бази даних";
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка підключення до бази даних: "
+ ex.Message);
                statusLabel.Text = "Помилка підключення";
            }
        }

        private void LoadCategories()
        {
            try
            {
                string query = "SELECT id, \"name_Ua\" FROM
public.\"Category\" WHERE \"isOnBar\" = true";
                using (NpgsqlCommand cmd = new NpgsqlCommand(query,
connection))
                {
                    using (NpgsqlDataReader reader =
cmd.ExecuteReader())
                    {
                        while (reader.Read())
                        {
                            int id = reader.GetInt32(0);
                            string name = reader.GetString(1);

                            ToolStripMenuItem categoryItem = new
ToolStripMenuItem(name);
                            categoryItem.Tag = id;
```

```
                            categoryItem.Click += CategoryItem_Click;

categoriesMenu.DropDownItems.Add(categoryItem);
                        }
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка завантаження категорій: " +
ex.Message);
            }
        }

        private void CategoryItem_Click(object sender, EventArgs e)
        {
            ToolStripMenuItem item = sender as ToolStripMenuItem;
            if (item != null)
            {
                int categoryId = (int)item.Tag;
                LoadSmartphonesByCategory(categoryId);
            }
        }

        private async Task LoadSmartphonesByCategory(int categoryId)
        {
            try
            {
                flowLayoutPanelCatalog.Controls.Clear();
                smartphoneItems.Clear();

                string query = "SELECT id, \"name_Ua\", \"price_Ua\",
\"photo\" FROM \"Goods\" " +
                               "WHERE \"categoryID\" = @categoryId
AND \"isShown\" = true";

                using (NpgsqlCommand cmd = new NpgsqlCommand(query,
connection))
                {
                    cmd.Parameters.AddWithValue("@categoryId",
categoryId);

                    using (NpgsqlDataReader reader =
cmd.ExecuteReader())
                    {
                        while (reader.Read())
                        {
                            SmartphoneItem item = new SmartphoneItem
                            {
                                Id = reader.GetInt32(0),
                                Name = reader.GetString(1),
                                Price = reader.GetString(2),
                                PhotoPath = reader.GetString(3)
                            };

                            smartphoneItems.Add(item);
```

```
                            await AddSmartphoneToPanel(item);
                        }
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка завантаження смартфонів: " +
ex.Message);
            }
        }

        private async Task LoadSmartphones()
        {
            try
            {
                flowLayoutPanelCatalog.Controls.Clear();
                smartphoneItems.Clear();

                string query = "SELECT id, \"name_Ua\", \"price_Ua\",
\"photo\" FROM \"Goods\" " +
                               "WHERE \"isShown\" = true ORDER BY
\"createDateTime\" DESC LIMIT 10";

                using (NpgsqlCommand cmd = new NpgsqlCommand(query,
connection))
                {
                    using (NpgsqlDataReader reader =
cmd.ExecuteReader())
                    {
                        while (reader.Read())
                        {
                            SmartphoneItem item = new SmartphoneItem
                            {
                                Id = reader.GetInt32(0),
                                Name = reader.GetString(1),
                                Price = reader.GetString(2),
                                PhotoPath = reader.GetString(3)
                            };

                            smartphoneItems.Add(item);
                            await AddSmartphoneToPanel(item);
                        }
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка завантаження смартфонів: " +
ex.Message);
            }
        }

        private async Task AddSmartphoneToPanel(SmartphoneItem item)
        {
            Panel panel = new Panel
```

```
            {
                Width = 200,
                Height = 250,
                BorderStyle = BorderStyle.FixedSingle,
                Margin = new Padding(10)
            };

            PictureBox pictureBox = new PictureBox
            {
                Width = 180,
                Height = 150,
                Location = new Point(10, 10),
                SizeMode = PictureBoxSizeMode.Zoom
            };

            Label nameLabel = new Label
            {
                Text = item.Name,
                Width = 180,
                Location = new Point(10, 170),
                Font = new Font(Font.FontFamily, 10, FontStyle.Bold)
            };

            Label priceLabel = new Label
            {
                Text = item.Price + " грн",
                Width = 180,
                Location = new Point(10, 195),
                Font = new Font(Font.FontFamily, 9)
            };

            Button detailsButton = new Button
            {
                Text = "Докладніше",
                Width = 180,
                Location = new Point(10, 220)
            };

            detailsButton.Tag = item.Id;
            detailsButton.Click += DetailsButton_Click;

            panel.Controls.Add(pictureBox);
            panel.Controls.Add(nameLabel);
            panel.Controls.Add(priceLabel);
            panel.Controls.Add(detailsButton);

            flowLayoutPanelCatalog.Controls.Add(panel);

            await LoadImageAsync(pictureBox, item.PhotoPath);
        }

        private async Task LoadImageAsync(PictureBox pictureBox,
string imagePath)
        {
            try
            {
```

```
                if (string.IsNullOrEmpty(imagePath))
                {
                    SetDefaultImage(pictureBox);
                    return;
                }

                if (imagePath.StartsWith("http",
StringComparison.OrdinalIgnoreCase))
                {
                    using (var httpClient = new
System.Net.Http.HttpClient())
                    {
                        httpClient.Timeout =
TimeSpan.FromSeconds(10);

                        byte[] imageBytes = await
httpClient.GetByteArrayAsync(imagePath);

                        using (var ms = new
System.IO.MemoryStream(imageBytes))
                        {
                            var originalImage = Image.FromStream(ms);
                            var imageCopy = new
Bitmap(originalImage);
                            originalImage.Dispose();

                            if (pictureBox.InvokeRequired)
                            {
                                pictureBox.Invoke(new Action(() =>
                                {
                                    if (pictureBox.Image != null)
                                        pictureBox.Image.Dispose();
                                    pictureBox.Image = imageCopy;
                                }));
                            }
                            else
                            {
                                if (pictureBox.Image != null)
                                    pictureBox.Image.Dispose();
                                pictureBox.Image = imageCopy;
                            }
                        }
                    }
                }
                else if (System.IO.File.Exists(imagePath))
                {
                    using (var fileStream = new
System.IO.FileStream(imagePath, System.IO.FileMode.Open,
System.IO.FileAccess.Read))
                    {
                        var originalImage =
Image.FromStream(fileStream);
                        var imageCopy = new Bitmap(originalImage);
                        originalImage.Dispose();

                        if (pictureBox.Image != null)
```

```
                        pictureBox.Image.Dispose();
                    pictureBox.Image = imageCopy;
                }
            }
            else
            {
                SetDefaultImage(pictureBox);
            }
        }
        catch (Exception ex)
        {
            System.Diagnostics.Debug.WriteLine($"Помилка
завантаження зображення {imagePath}: {ex.Message}");
            SetDefaultImage(pictureBox);
        }
    }

    private async Task SetDefaultImage(PictureBox pictureBox)
    {
        try
        {
            string defaultImageUrl = "https://opsg-img-cdn-
gl.heytapimg.com/epb/202412/19/AceLeaXtntKw1AZf.png";

            using (var httpClient = new
System.Net.Http.HttpClient())
            {
                httpClient.Timeout = TimeSpan.FromSeconds(5);

                byte[] imageBytes = await
httpClient.GetByteArrayAsync(defaultImageUrl);

                using (var ms = new
System.IO.MemoryStream(imageBytes))
                {
                    var originalImage = Image.FromStream(ms);
                    var imageCopy = new Bitmap(originalImage);
                    originalImage.Dispose();

                    if (pictureBox.Image != null)
                        pictureBox.Image.Dispose();
                    pictureBox.Image = imageCopy;
                }
            }
        }
        catch
        {
            if (pictureBox.Image != null)
            {
                pictureBox.Image.Dispose();
                pictureBox.Image = null;
            }
        }
    }
```

```
        private void DetailsButton_Click(object sender, EventArgs e)
        {
            Button button = sender as Button;
            if (button != null)
            {
                int smartphoneId = (int)button.Tag;
                ShowSmartphoneDetails(smartphoneId);
            }
        }

        private void ShowSmartphoneDetails(int smartphoneId)
        {
            SmartphoneDetailsForm detailsForm = new
SmartphoneDetailsForm(connection, smartphoneId);
            detailsForm.ShowDialog();
        }

        private void MainForm_FormClosing(object sender,
FormClosingEventArgs e)
        {
            if (connection != null && connection.State ==
ConnectionState.Open)
            {
                connection.Close();
            }
        }
    }
```

Додаток Б

Форма інформації про товар

```
        public partial class SmartphoneDetailsForm : Form
         {
             public SmartphoneDetailsForm()
             {
                 InitializeComponent();
             }

             private NpgsqlConnection connection;
             private int smartphoneId;
             private List<Color> availableColors = new List<Color>
             {
                 Color.Black,
                 Color.White,
                 Color.Gray,
                 Color.Blue,
                 Color.Red
             };
             private Color selectedColor = Color.Black;
             private float angle = 0.0f;
             private Timer rotationTimer;

             public SmartphoneDetailsForm(NpgsqlConnection connection,
int smartphoneId)
             {
                 this.connection = connection;
                 this.smartphoneId = smartphoneId;
                 InitializeComponent();

                 LoadSmartphoneDetails();
                 PopulateColorComboBox();

                 rotationTimer = new Timer();
                 rotationTimer.Interval = 16;
                 rotationTimer.Tick += (sender, e) =>
                 {
                     angle += 1.0f;
                     glControl.Invalidate();
                 };
                 rotationTimer.Start();
             }

             private void LoadSmartphoneDetails()
             {
                 try
                 {
                     string query = "SELECT \"name_Ua\",
\"description_Ua\", \"price_Ua\" FROM \"Goods\" WHERE id = @id";

                     using (NpgsqlCommand cmd = new
NpgsqlCommand(query, connection))
                     {
                         cmd.Parameters.AddWithValue("@id",
smartphoneId);
```

Продовження додатку Б

```
                        using (NpgsqlDataReader reader =
cmd.ExecuteReader())
                        {
                            if (reader.Read())
                            {
                                string name = reader.GetString(0);
                                string description =
reader.GetString(1);
                                string price = reader.GetString(2);

                                labelName.Text = name;
                                labelPrice.Text = price + " грн";
                                textBoxDescription.Text =
description;
                            }
                        }
                    }

                    query = "SELECT c.\"name_Ua\", gc.\"value_Ua\"
FROM \"GoodsCharacteristic\" gc " +
                            "JOIN \"Characteristic\" c ON
gc.\"characteristicId\" = c.id " +
                            "WHERE gc.\"goodsId\" = @id";

                    using (NpgsqlCommand cmd = new
NpgsqlCommand(query, connection))
                    {
                        cmd.Parameters.AddWithValue("@id",
smartphoneId);

                        using (NpgsqlDataReader reader =
cmd.ExecuteReader())
                        {
                            while (reader.Read())
                            {
                                string name = reader.GetString(0);
                                string value = reader.GetString(1);

                                ListViewItem item = new
ListViewItem(name);
                                item.SubItems.Add(value);

listViewCharacteristics.Items.Add(item);
                            }
                        }
                    }
                }
                catch (Exception ex)
                {
                    MessageBox.Show("Помилка завантаження даних
смартфона: " + ex.Message);
                }
            }

            private void PopulateColorComboBox()
            {
```

```
                foreach (Color color in availableColors)
                {
                    comboBoxColor.Items.Add(GetColorName(color));
                }
                comboBoxColor.SelectedIndex = 0;
            }

            private string GetColorName(Color color)
            {
                if (color == Color.Black) return "Чорний";
                if (color == Color.White) return "Білий";
                if (color == Color.Gray) return "Сірий";
                if (color == Color.Blue) return "Синій";
                if (color == Color.Red) return "Червоний";
                return color.Name;
            }

            private void comboBoxColor_SelectedIndexChanged(object
sender, EventArgs e)
            {
                selectedColor =
availableColors[comboBoxColor.SelectedIndex];
                glControl.Invalidate();
            }

            private void glControl_Load(object sender, EventArgs e)
            {
                glControl.MakeCurrent();
                GL.ClearColor(0.0f, 0.0f, 0.0f, 1.0f);
                GL.Enable(EnableCap.DepthTest);
            }

            private void glControl_Resize(object sender, EventArgs e)
            {
                glControl.MakeCurrent();
                GL.Viewport(0, 0, glControl.Width, glControl.Height);

                GL.MatrixMode(MatrixMode.Projection);
                GL.LoadIdentity();
                Matrix4 perspective =
Matrix4.CreatePerspectiveFieldOfView(
                    MathHelper.PiOver4,
                    glControl.Width / (float)glControl.Height,
                    0.1f, 10f);
                GL.LoadMatrix(ref perspective);

                GL.MatrixMode(MatrixMode.Modelview);
            }

            private void glControl_Paint(object sender,
PaintEventArgs e)
            {
                glControl.MakeCurrent();
                GL.Clear(ClearBufferMask.ColorBufferBit |
ClearBufferMask.DepthBufferBit);
                GL.ClearColor(0.0f, 0.9f, 0.9f, 1.0f);
```

```
                GL.Enable(EnableCap.DepthTest);

                GL.LoadIdentity();
                GL.Translate(0.0f, 0.0f, -3.0f);
                GL.Rotate(angle, 0.0f, 1.0f, 0.0f);

                DrawSmartphone();

                glControl.SwapBuffers();
            }

            private void DrawSmartphone()
            {
                float thickness = 0.05f;
                float length = 0.75f;
                float height = 1.5f;
                float cornerRadius = 0.1f;

                float r = selectedColor.R / 255.0f;
                float g = selectedColor.G / 255.0f;
                float b = selectedColor.B / 255.0f;

                GL.Color3(r, g, b);
                DrawRoundedRectangle(-length / 2, -height / 2,
length, height, thickness);

                DrawRoundedSideFaces(length, height, thickness,
cornerRadius);

                DrawCamera(0.3f, 0.6f, -thickness - 0.01f, 0.07f);
                DrawFlash(0.3f, 0.45f, -thickness - 0.01f, 0.02f);

                DrawScreen(-length / 2 + 0.05f, -height / 2 + 0.05f,
length - 0.1f, height - 0.1f, thickness + 0.001f);
            }

            private void DrawScreen(float x, float y, float width,
float height, float z)
            {
                GL.Color3(0.0f, 0.0f, 0.0f);
                GL.Begin(PrimitiveType.Quads);
                GL.Vertex3(x, y, z);
                GL.Vertex3(x + width, y, z);
                GL.Vertex3(x + width, y + height, z);
                GL.Vertex3(x, y + height, z);
                GL.End();
            }

            private void DrawRoundedSideFaces(float length, float
height, float thickness, float cornerRadius)
            {
                float r = selectedColor.R / 255.0f * 0.8f;
                float g = selectedColor.G / 255.0f * 0.8f;
                float b = selectedColor.B / 255.0f * 0.8f;
                GL.Color3(r, g, b);
```

```
                float l = length / 2;
                float h = height / 2;
                float t = thickness;
                int segments = 10;

                void DrawSide(float x1, float y1, float x2, float y2,
bool vertical)
                {
                    GL.Begin(PrimitiveType.Quads);
                    GL.Vertex3(x1, y1, -t);
                    GL.Vertex3(x1, y1, t);
                    GL.Vertex3(x2, y2, t);
                    GL.Vertex3(x2, y2, -t);
                    GL.End();
                }

                DrawSide(-l + cornerRadius, -h, l - cornerRadius, -h,
false);
                DrawSide(-l + cornerRadius, h, l - cornerRadius, h,
false);
                DrawSide(-l, -h + cornerRadius, -l, h - cornerRadius,
true);
                DrawSide(l, -h + cornerRadius, l, h - cornerRadius,
true);

                for (int i = 0; i < 4; i++)
                {
                    float cx = (i % 2 == 0) ? -l + cornerRadius : l -
cornerRadius;
                    float cy = (i < 2) ? -h + cornerRadius : h -
cornerRadius;
                    float startAngle = i * MathHelper.PiOver2;

                    for (int j = 0; j < segments; j++)
                    {
                        float theta1 = startAngle + j *
MathHelper.PiOver2 / segments;
                        float theta2 = startAngle + (j + 1) *
MathHelper.PiOver2 / segments;

                        float x1 = cx + (float)Math.Cos(theta1) *
cornerRadius;
                        float y1 = cy + (float)Math.Sin(theta1) *
cornerRadius;
                        float x2 = cx + (float)Math.Cos(theta2) *
cornerRadius;
                        float y2 = cy + (float)Math.Sin(theta2) *
cornerRadius;

                        DrawSide(x1, y1, x2, y2, false);
                    }
                }
            }

            private void DrawRoundedRectangle(float x, float y, float
width, float height, float depth)
```

```
            {
                int segments = 24;
                float cornerRadius = 0.1f;

                float x0 = x;
                float y0 = y;
                float x1 = x + width;
                float y1 = y + height;
                float zFront = depth;
                float zBack = -depth;

                void DrawFlatFace(float z)
                {
                    GL.Begin(PrimitiveType.Quads);
                    GL.Vertex3(x0 + cornerRadius, y0 + cornerRadius,
z);
                    GL.Vertex3(x1 - cornerRadius, y0 + cornerRadius,
z);
                    GL.Vertex3(x1 - cornerRadius, y1 - cornerRadius,
z);
                    GL.Vertex3(x0 + cornerRadius, y1 - cornerRadius,
z);

                    GL.Vertex3(x0 + cornerRadius, y1 - cornerRadius,
z); GL.Vertex3(x1 - cornerRadius, y1 - cornerRadius, z); GL.Vertex3(x1
- cornerRadius, y1, z); GL.Vertex3(x0 + cornerRadius, y1, z);
                    GL.Vertex3(x0 + cornerRadius, y0, z);
GL.Vertex3(x1 - cornerRadius, y0, z); GL.Vertex3(x1 - cornerRadius, y0
+ cornerRadius, z); GL.Vertex3(x0 + cornerRadius, y0 + cornerRadius,
z);
                    GL.Vertex3(x0, y0 + cornerRadius, z);
GL.Vertex3(x0 + cornerRadius, y0 + cornerRadius, z); GL.Vertex3(x0 +
cornerRadius, y1 - cornerRadius, z); GL.Vertex3(x0, y1 - cornerRadius,
z);
                    GL.Vertex3(x1 - cornerRadius, y0 + cornerRadius,
z); GL.Vertex3(x1, y0 + cornerRadius, z); GL.Vertex3(x1, y1 -
cornerRadius, z); GL.Vertex3(x1 - cornerRadius, y1 - cornerRadius, z);
                    GL.End();
                }

                DrawFlatFace(zFront);

                float colorR = selectedColor.R / 255.0f * 0.9f;
                float colorG = selectedColor.G / 255.0f * 0.9f;
                float colorB = selectedColor.B / 255.0f * 0.9f;
                GL.Color3(colorR, colorG, colorB);
                DrawFlatFace(zBack);
                void DrawCornerSurface(float cx, float cy, float z,
float startAngle)
                {
                    GL.Begin(PrimitiveType.TriangleFan);
                    GL.Vertex3(cx, cy, z);
                    for (int i = 0; i <= segments; i++)
                    {
                        float angle = startAngle + i *
(MathHelper.PiOver2 / segments);
```

```
                        float dx = (float)Math.Cos(angle) *
cornerRadius;
                        float dy = (float)Math.Sin(angle) *
cornerRadius;
                        GL.Vertex3(cx + dx, cy + dy, z);
                    }
                    GL.End();
                }

                void DrawCornerSide(float cx, float cy, float
startAngle)
                {
                    float sideColorR = selectedColor.R / 255.0f *
0.8f;
                    float sideColorG = selectedColor.G / 255.0f *
0.8f;
                    float sideColorB = selectedColor.B / 255.0f *
0.8f;
                    GL.Color3(sideColorR, sideColorG, sideColorB);

                    for (int i = 0; i < segments; i++)
                    {
                        float angle1 = startAngle + i *
(MathHelper.PiOver2 / segments);
                        float angle2 = startAngle + (i + 1) *
(MathHelper.PiOver2 / segments);

                        float x1 = cx + (float)Math.Cos(angle1) *
cornerRadius;
                        float y1 = cy + (float)Math.Sin(angle1) *
cornerRadius;

                        float x2 = cx + (float)Math.Cos(angle2) *
cornerRadius;
                        float y2 = cy + (float)Math.Sin(angle2) *
cornerRadius;

                        GL.Begin(PrimitiveType.Quads);
                        GL.Vertex3(x1, y1, zBack);
                        GL.Vertex3(x1, y1, zFront);
                        GL.Vertex3(x2, y2, zFront);
                        GL.Vertex3(x2, y2, zBack);
                        GL.End();
                    }
                }

                void DrawAllCorners()
                {
                    float frontColorR = selectedColor.R / 255.0f;
                    float frontColorG = selectedColor.G / 255.0f;
                    float frontColorB = selectedColor.B / 255.0f;
                    GL.Color3(frontColorR, frontColorG, frontColorB);

                    DrawCornerSurface(x0 + cornerRadius, y0 +
cornerRadius, zFront, MathHelper.Pi);
```

Продовження додатку Б

```
                    float backColorR = frontColorR * 0.9f;
                    float backColorG = frontColorG * 0.9f;
                    float backColorB = frontColorB * 0.9f;
                    GL.Color3(backColorR, backColorG, backColorB);

                    DrawCornerSurface(x0 + cornerRadius, y0 +
cornerRadius, zBack, MathHelper.Pi);
                    DrawCornerSide(x0 + cornerRadius, y0 +
cornerRadius, MathHelper.Pi);

                    GL.Color3(frontColorR, frontColorG, frontColorB);

                    DrawCornerSurface(x1 - cornerRadius, y0 +
cornerRadius, zFront, 1.5f * MathHelper.Pi);

                    GL.Color3(backColorR, backColorG, backColorB);

                    DrawCornerSurface(x1 - cornerRadius, y0 +
cornerRadius, zBack, 1.5f * MathHelper.Pi);
                    DrawCornerSide(x1 - cornerRadius, y0 +
cornerRadius, 1.5f * MathHelper.Pi);

                    GL.Color3(frontColorR, frontColorG, frontColorB);

                    DrawCornerSurface(x1 - cornerRadius, y1 -
cornerRadius, zFront, 0f);

                    GL.Color3(backColorR, backColorG, backColorB);

                    DrawCornerSurface(x1 - cornerRadius, y1 -
cornerRadius, zBack, 0f);
                    DrawCornerSide(x1 - cornerRadius, y1 -
cornerRadius, 0f);

                    GL.Color3(frontColorR, frontColorG, frontColorB);

                    DrawCornerSurface(x0 + cornerRadius, y1 -
cornerRadius, zFront, MathHelper.PiOver2);

                    GL.Color3(backColorR, backColorG, backColorB);

                    DrawCornerSurface(x0 + cornerRadius, y1 -
cornerRadius, zBack, MathHelper.PiOver2);
                    DrawCornerSide(x0 + cornerRadius, y1 -
cornerRadius, MathHelper.PiOver2);
                }

                DrawAllCorners();
            }

            private void DrawCamera(float x, float y, float z, float
radius)
            {
                int segments = 40;
                GL.Color3(0.0f, 0.0f, 0.0f);
```

```
                GL.Begin(PrimitiveType.TriangleFan);
                GL.Vertex3(x, y, z);

                for (int i = 0; i <= segments; i++)
                {
                    float angle = i * 2.0f * MathHelper.Pi /
segments;
                    float dx = (float)Math.Cos(angle) * radius;
                    float dy = (float)Math.Sin(angle) * radius;
                    GL.Vertex3(x + dx, y + dy, z);
                }
                GL.End();

                GL.Color3(0.2f, 0.2f, 0.4f);
                GL.Begin(PrimitiveType.TriangleFan);
                GL.Vertex3(x, y, z + 0.005f);
                for (int i = 0; i <= segments; i++)
                {
                    float angle = i * 2.0f * MathHelper.Pi /
segments;
                    float dx = (float)Math.Cos(angle) * radius *
0.6f;
                    float dy = (float)Math.Sin(angle) * radius *
0.6f;
                    GL.Vertex3(x + dx, y + dy, z + 0.005f);
                }
                GL.End();

                GL.Color3(1.0f, 1.0f, 1.0f);
                GL.Begin(PrimitiveType.TriangleFan);
                GL.Vertex3(x - radius * 0.2f, y - radius * 0.2f, z +
0.006f);
                for (int i = 0; i <= segments / 4; i++)
                {
                    float angle = i * 2.0f * MathHelper.Pi /
segments;
                    float dx = (float)Math.Cos(angle) * radius *
0.1f;
                    float dy = (float)Math.Sin(angle) * radius *
0.1f;
                    GL.Vertex3(x - radius * 0.2f + dx, y - radius *
0.2f + dy, z + 0.006f);
                }
                GL.End();
            }

            private void DrawFlash(float x, float y, float z, float
radius)
            {
                GL.Color3(1.0f, 1.0f, 0.6f);
                int segments = 20;

                GL.Begin(PrimitiveType.TriangleFan);
                GL.Vertex3(x, y, z);
                for (int i = 0; i <= segments; i++)
                {
```

Продовження додатку Б

```
                    float angle = i * 2.0f * MathHelper.Pi /
segments;
                    float dx = (float)Math.Cos(angle) * radius;
                    float dy = (float)Math.Sin(angle) * radius;
                    GL.Vertex3(x + dx, y + dy, z);
                }
                GL.End();

                GL.Color3(1.0f, 1.0f, 0.9f);
                GL.Begin(PrimitiveType.TriangleFan);
                GL.Vertex3(x, y, z + 0.001f);
                for (int i = 0; i <= segments; i++)
                {
                    float angle = i * 2.0f * MathHelper.Pi /
segments;
                    float dx = (float)Math.Cos(angle) * radius *
0.5f;
                    float dy = (float)Math.Sin(angle) * radius *
0.5f;
                    GL.Vertex3(x + dx, y + dy, z + 0.001f);
                }
                GL.End();
            }

            private void SmartphoneDetailsForm_FormClosing(object
sender, FormClosingEventArgs e)
            {
                rotationTimer.Stop();
            }

        }
```

Додаток В

Компонент Routes

```
     import { BrowserRouter, Route, Routes as RouteWrapper } from
'react-router-dom'
     import { Container } from '@mui/material'
     import Header from '../components/Header'
     import { Footer } from '../components'
     import MainPage from '../pages/Main'
     import CategoryPage from '../pages/Category'
     import GoodsPage from '../pages/Goods'
     import { AdminPage } from '../pages/Admin'
     import NavigateRoutes from './NavigateRoutes'
     import SidebarRoutes from './SidebarRoutes'
     import { NotFoundPage, SearchPage } from '../pages'


     const Routes = () => {


         return (
           <BrowserRouter >
                   <RouteWrapper>
                         <Route element={<NavigateRoutes/>}>
                             <Route element={<SidebarRoutes/>}>
                                     <Route path='/'
element={<MainPage/>}/>
                                     <Route path='category'>
                                         <Route path=':category'
element={<CategoryPage/>}/>
                                     </Route>
                                     <Route path='goods'>
                                         <Route path=':url'
element={<GoodsPage/>}/>
                                     </Route>
                                     <Route path='search'>
                                         <Route path=':findString'
element={<SearchPage/>}/>
                                     </Route>
                             </Route>
                             <Route path='admin'
element={<AdminPage/>}/>
                             <Route path='*'
element={<NotFoundPage/>}/>
                         </Route>
                   </RouteWrapper>
           </BrowserRouter>
         )
     }

     export default Routes
```

Додаток Г

Файл reducer

```
     import { State, AppAction } from "../models/context"
     import { CONTEXT_TYPES, LOCAL_STORAGE_TYPES,
MAX_RECENT_VIEWS_ELEMENTS } from "./consts"
     import { setLocalStorage } from "./localStorage"

     export const reducer = (state: State, action: AppAction): State
=> {
          switch (action.type) {

              case CONTEXT_TYPES.ADD_CART_ITEM:{
                  if (state.cartIds.includes(action.itemId)) return
state
                  const updatedIds = [...state.cartIds, action.itemId]
                  const updatedCart = [...state.cart, action.goods]
                  setLocalStorage(LOCAL_STORAGE_TYPES.CART_ITEM,
updatedIds)
                  return {...state, cartIds: updatedIds, cart:
updatedCart}
              }


              case CONTEXT_TYPES.REMOVE_CART_ITEM: {
                  const updatedIds = state.cartIds.filter(id => id !=
action.itemId)
                  const updatedCart = state.cart.filter(goods =>
goods.id != action.itemId)
                  setLocalStorage(LOCAL_STORAGE_TYPES.CART_ITEM,
updatedIds)
                  return {...state, cartIds: updatedIds, cart:
updatedCart}
              }


              case CONTEXT_TYPES.ADD_VIEWS_ITEM: {
                  if (state.recentViewsIds.includes(action.itemId))
return state
                  const updatedIds = [action.itemId,
...state.recentViewsIds]
                  const updatedCart = [action.goods,
...state.recentViews]
                  if (state.recentViews.length >=
MAX_RECENT_VIEWS_ELEMENTS){
                      updatedCart.pop()
                      updatedIds.pop()
                  }
                  setLocalStorage(LOCAL_STORAGE_TYPES.VIEWS_ITEM,
updatedIds)
                  return {...state, recentViewsIds: updatedIds,
recentViews: updatedCart}
              }

              case CONTEXT_TYPES.SET_CARD_ITEM: {
```

Продовження додатку Г

```
       return {...state, cart: action.goods, cartIds: action.itemIds}
              }

              case CONTEXT_TYPES.SET_VIEWS_ITEM: {
                  return {...state, recentViews: action.goods,
recentViewsIds: action.itemIds}
              }

              default:
                  throw new Error('Incorrect dispatch type')
              }

      }
```

Додаток Д

Компонент AppContextProvider

```
     import React, { useReducer, useEffect, createContext, useState }
from 'react'
     import { CONTEXT_TYPES, LOCAL_STORAGE_TYPES } from
'../utils/consts'
     import { getFromStorage } from '../utils/localStorage'
     import { defaultContext } from '../utils/defaultContext'
     import { useLazyQuery } from '@apollo/client'
     import { LoadingPage } from '../components'
     import { GlobalContext } from '../models/context'
     import { GET_GOODS_BY_IDS } from './queries'
     import { reducer } from '../utils/reducer'
     import { Goods } from '../models/good'

     interface Props {
         children: React.ReactNode
     }

     export const AppContext = createContext<GlobalContext>({
         state: defaultContext,
         dispatch: ()=>(defaultContext)
     })


     export const AppContextProvider = ({children}: Props) => {

         const [state, dispatch] = useReducer(reducer, defaultContext)
         const [getGoodsByIds] = useLazyQuery(GET_GOODS_BY_IDS)
         const [gettedData, setGettedData] = useState(false)
         const value = { state, dispatch };

         useEffect(()=>{

             const cartIds =
getFromStorage(LOCAL_STORAGE_TYPES.CART_ITEM) as number[]
             const viewsIds =
getFromStorage(LOCAL_STORAGE_TYPES.VIEWS_ITEM) as number[]

             let finalGoods: number[] = []

             if (cartIds) finalGoods = [...cartIds]
             if (viewsIds) finalGoods = [...finalGoods, ...viewsIds]


             if (finalGoods.length) {


                 getGoodsByIds({variables: {ids: finalGoods}})
                 .then((data) => {


                     if (cartIds) {

                         dispatch({
                             type: CONTEXT_TYPES.SET_CARD_ITEM,
```

Продовження додатку Д

```
                                itemIds: cartIds,
                                goods:
data.data.getGoodsByIds.filter((goods: Goods) =>
cartIds.includes(goods.id as number))
                            })
                        }

                        if (viewsIds) {

                            dispatch({
                                type: CONTEXT_TYPES.SET_VIEWS_ITEM,
                                itemIds: viewsIds,
                                goods:
data.data.getGoodsByIds.filter((goods: Goods) =>
viewsIds.includes(goods.id as number))
                            })
                        }

                        setGettedData(true)
                    })

                } else {
                    setGettedData(true)
                }
            }, [])

            if (!gettedData) return <LoadingPage/>


            return (
                <AppContext.Provider value={value}>
                    {children}
                </AppContext.Provider>
            )
        }
```