НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
комп'ютерних наук
Голуб Б.Л., доц., к.т.н.

(підпис) (ПІБ, вчене звання і ступінь)

«___»_____2025 р

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Програмне забезпечення мобільного додатку для інформаційної системи інтернет магазину іграшок»

Спеціальність 121 «Інженерія програмного забезпечення»

Гарант освітньої програми

к.т.н. доцент _____ Вайганг Г.О.
(Науковий ступень та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи

_____ _____ Бородкін Г.О.
(Науковий ступень та вчене звання) (підпис) (ПІБ)

Виконав _____ Глухов Д.В.
(підпис) (ПІБ)

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп'ютерних наук
_____/ Голуб Б.Л., доцент, к.т.н /
підпис
"___" _____ 2025 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

студент Глухов Дмитро Володимирович

Спеціальність 121 «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи: **Програмне забезпечення мобільного додатку для інформаційної системи інтернет магазину іграшок**

Затверджена наказом ректора НУБіП України від 16.12.2024 № 2248 "С"

Термін подання завершеної роботи на кафедру _____
(рік, місяць, число)

Вихідні дані до роботи: опис програмного забезпечення

Перелік питань , які потрібно розробити:

Аналіз проблемної області, вибір та обґрунтування засобів для розробки системи, проектування інформаційної системи.

Дата видачі завдання "16" 12 2024р.

Керівник бакалаврської кваліфікаційної роботи

_____ _____ Бородкін Г.О
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ _____ Глухов Д.В.
(підпис) (ПІБ студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – (Application Programming Interface) – інтерфейс прикладного програмування, набір засобів для взаємодії між програмним забезпеченням.

JWT – (JSON Web Token) – відкритий стандарт для безпечної передачі даних між сторонами у вигляді JSON-об’єкта.

ORM – (Object-Relational Mapping) – технологія, яка дозволяє працювати з базою даних через об'єктно-орієнтовану модель.

REST – (Representational State Transfer) – архітектурний стиль взаємодії між клієнтом і сервером за допомогою стандартних HTTP-запитів.

UI – (User Interface) – інтерфейс користувача, через який користувач взаємодіє із системою.

UX – (User Experience) – користувацький досвід, враження користувача від роботи із системою.

SQL – (Structured Query Language) – мова структурованих запитів для роботи з реляційними базами даних.

NoSQL – тип баз даних, які не використовують традиційну реляційну модель.

DB – (Database) – база даних, сховище даних для застосунку.

Flutter – фреймворк для розробки кросплатформних мобільних застосунків мовою Dart.

Dart – мова програмування, яку використовує Flutter.

Go – мова програмування, що використовується для розробки високопродуктивного бекенду.

Gin – веб-фреймворк для Go, що використовується для побудови REST API.

PostgreSQL – об’єктно-реляційна система керування базами даних з відкритим кодом.

Firebase Storage – сервіс Google для зберігання файлів (зображень, відео тощо) у хмарі.

Stripe – платіжна система, що забезпечує онлайн-оплату в мобільному застосунку.

Docker – інструмент для створення, розгортання і запуску програм у ізольованих середовищах – контейнерах.

.env – файл середовищних змінних, у якому зберігаються конфіденційні налаштування, такі як паролі, ключі API тощо.

ID – унікальний ідентифікатор сутності в базі даних.

UML – (Unified Modeling Language) – уніфікована мова моделювання для візуалізації архітектури системи.

ERD – (Entity-Relationship Diagram) – діаграма зв’язків сутностей у базі даних.

CRUD – (Create, Read, Update, Delete) – базові операції над даними в інформаційній системі.

ВСТУП

У сучасних умовах розвитку цифрових технологій наявність онлайн-платформи є критично важливою для зростання й успішного функціонування будь-якого бізнесу. Це особливо актуально для малих виробників, зокрема майстрів, які займаються виготовленням іграшок ручної роботи. Оскільки такі майстри, як правило, не мають великих обсягів продукції чи фінансових можливостей для розміщення товарів на великих маркетплейсах, виникає потреба у створенні доступного інструменту для продажу своїх виробів через інтернет.

Мобільний застосунок для інтернет-магазину іграшок дає змогу вирішити одразу кілька проблем: автоматизувати процес продажу, зменшити витрати на обслуговування, розширити аудиторію покупців, підвищити впізнаваність власного бренду, а також забезпечити зручність як для покупців, так і для продавців. Завдяки цьому користувач може переглядати товари, замовляти їх у будь-який зручний час, не виходячи з дому.

У сучасному конкурентному середовищі відставання від технологічних тенденцій може коштувати бізнесу втрати клієнтів і ринкових позицій. Якщо підприємство не забезпечує зручний цифровий інтерфейс для взаємодії з клієнтом, його швидко замінить конкурент, який це зробив раніше. Тому впровадження цифрових рішень є необхідністю.

Саме тому в цій бакалаврській кваліфікаційній роботі на тему **«Програмне забезпечення мобільного додатку для інформаційної системи інтернет-магазину іграшок»** було реалізовано повноцінний мобільний застосунок, який дозволяє handmade-майстрам легко публікувати товари, керувати замовленнями та спілкуватися з клієнтами.

Для досягнення мети роботи були виконані такі завдання:

- проаналізовано предметну область та визначено вимоги до системи;
- побудовано моделі даних і UML-діаграми;
- обрано інструменти для реалізації мобільного застосунку;

- реалізовано серверну частину (бекенд) та логіку бізнес-процесів;
- створено інтерфейс користувача (фронтенд) для мобільних пристроїв;
- Підключення стороніх API сервісів;
- спроєктовано та підключено базу даних;
- протестовано функціональність системи;
- оформлено пояснювальну записку до роботи.

Основний функціонал мобільного додатку включає: реєстрацію та авторизацію користувачів, додавання та редагування товарів, управління замовленнями та статусами, чат з продавцем-покупцем, перегляд профілю, створених оголошень, додавання зображень до товарів, пошук, сортування та фільтрацію іграшок, поповненям балансу для оплати онлайн замовлень, створення товарно-транспортної накладної.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Інтернет-магазини є важливим інструментом для представлення та реалізації продукції у сучасному цифровому середовищі. Особливо це стосується малих виробників — зокрема майстрів, які виготовляють іграшки власноруч (handmade). Через обмежені ресурси їм складно виходити на великі маркетплейси або створювати складні сайти. Тому мобільний додаток, створений спеціально під їх потреби, може суттєво спростити процес продажу, забезпечити зручність користування та розширити коло потенційних покупців.

Розроблена система дає змогу створювати, переглядати, фільтрувати та купувати товари, а також обробляти замовлення. Окрім покупців, у системі передбачені ролі продавця (автора товару) і адміністратора (модератора).

Основні ролі користувачів:

- **Покупець**: має змогу переглядати каталог, шукати та фільтрувати товари, оформлювати замовлення, і взаємодіяти з продавцем.
- **Автор товару**: може створювати товари, редагувати або видаляти їх, обробляти вхідні замовлення, підтверджувати доставку, відповідати на повідомлення від покупців.
- **Адміністратор**: відповідає за модерацію товарів, зміну їхнього статусу (активний, неактивний), контроль контенту та дотримання політик платформи.

1.2 Аналіз існуючих рішень

Під час розробки даного мобільного застосунку для онлайн-магазину іграшок було проаналізовано ряд інформаційних джерел, серед яких:

- офіційна документація фреймворків **Flutter**, **Gin (Go)**, **PostgreSQL**, **Firebase** та **Stripe**;
- навчальні курси на платформах Udemy, YouTube, Coursera та офіційні туторіали (Flutter.dev, go.dev);
- статті на Medium, Dev.to, Habr та DOU щодо архітектури мобільних застосунків і створення маркетплейсів;
- приклади open-source рішень зі схожою функціональністю на GitHub;
- обговорення та вирішення помилок на технічних форумах, зокрема Stack Overflow.
- Крім того, було проведено порівняльний аналіз існуючих рішень для продажу handmade-товарів. Розглянуто три найбільш популярні платформи: Etsy, Tedsby і Crafta (Shafa.ua).

  Зупинимось на аналізі останніх платформ більш детально.

Etsy — міжнародний маркетплейс для продажу унікальних, хендмейд, вінтажних товарів та матеріалів для творчості.

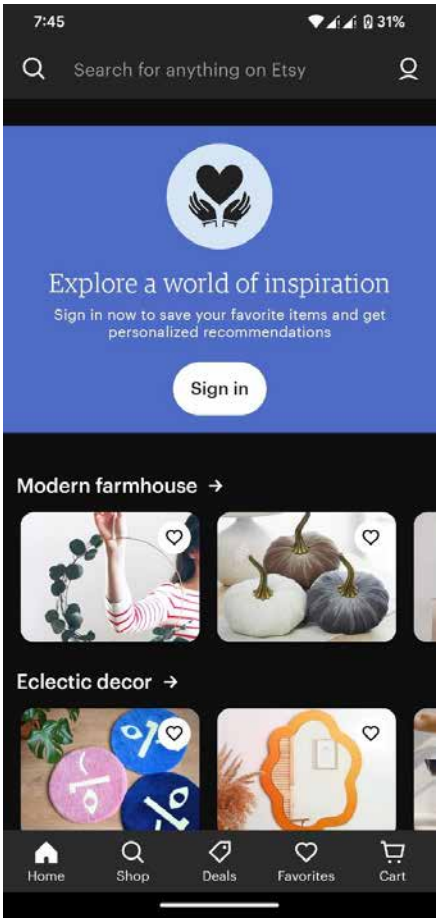


Рис. 1.1 **Головна сторінка мобільного додатку Etsy**

Переваги:

- велика аудиторія покупців по всьому світу;
- зручні інструменти для аналітики продажів;
- вбудовані рекламні функції (просування товарів);
- підтримка захищених оплат через платіжні шлюзи.

Недоліки:

- висока конкуренція, важко вийти в топ без бюджету на рекламу;
- регулярна комісія за кожен товар і транзакцію (5% + платіжні збори);
- необхідність дотримання політик платформи;
- орієнтація більше на англомовну аудиторію.

Tedsby — нішевий маркетплейс, що спеціалізується на продажу авторських плюшевих іграшок (зокрема Teddy bears).

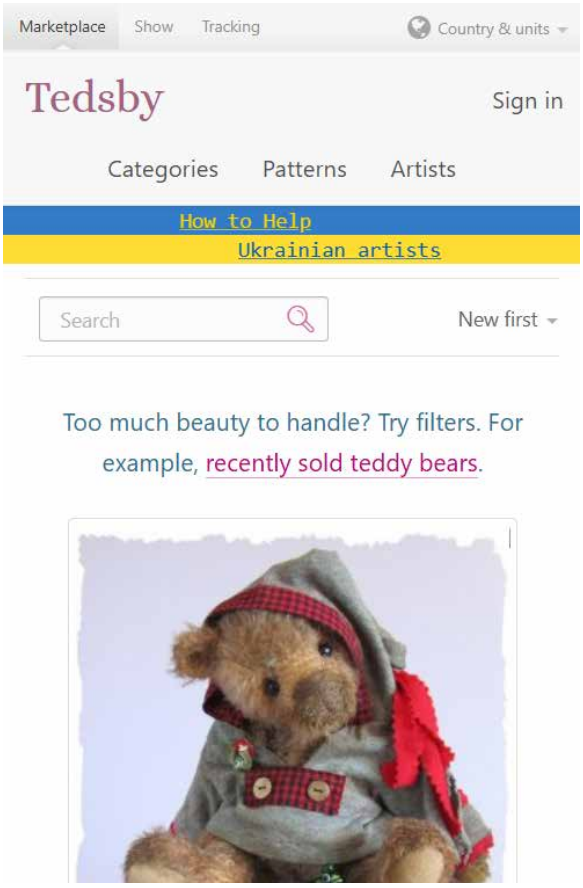


Рис. 1.2 **Головна сторінка Tedsby**

Переваги:

- чітко сфокусована аудиторія — покупці та колекціонери авторських іграшок;
- низька конкуренція порівняно з Etsy;
- можливість створення власного міні-магазину з каталогом товарів;
- простий процес реєстрації та розміщення товарів.

Недоліки:

- обмежений ринок збуту;
- відсутність мобільного застосунку;
- слабка SEO-оптимізація;
- не підтримується інтеграція з сучасними платіжними системами (наприклад, Stripe).

Crafta (Shafa.ua) — український маркетплейс для handmade-товарів, створений для локального ринку.

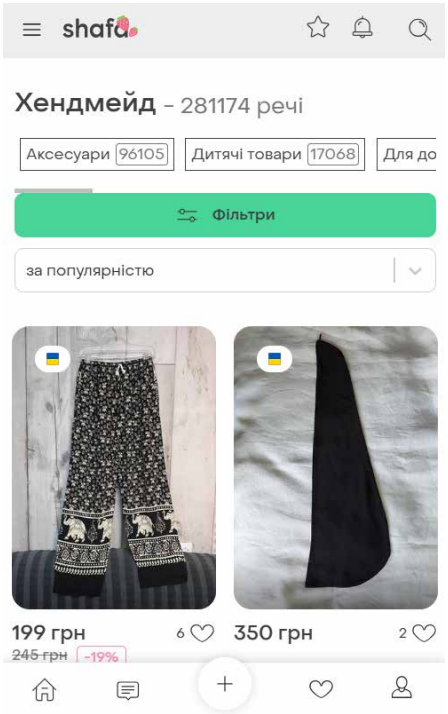


Рис. 1.3 **Головна сторінка Crafta (Shafa.ua)**

Переваги:

- орієнтація на українську аудиторію;
- відсутність мовного бар’єру;
- простий інтерфейс для розміщення товарів;
- можливість доставки через Нову Пошту, Укрпошту тощо;
- низькі або відсутні комісії за розміщення товарів.

Недоліки:

- слабка технічна інфраструктура (немає мобільного застосунку);
- обмежена функціональність (немає кастомної аналітики, розширеного пошуку, адаптивних фільтрів);
- малий обсяг аудиторії порівняно з глобальними платформами;
- підтримка користувачів та модерація можуть бути повільними.
- Закінчила свою підтримку у 2024 році як Crafta залишивши лише розділ hand-made товарів на Shafa.ua

Порівняно з цими платформами, запропонована система:

- не вимагає плати за розміщення оголошення чи продаж;
- має низький поріг входу для нових користувачів;
- дозволяє повністю адаптувати функціонал під потреби українського ринку;
- має зручний мобільний інтерфейс, авторизацію, модерування оголошень, вбудовану оплату та доставку;
- не потребує значних фінансових вкладень з боку продавця.

Таким чином, дана система поєднує гнучкість та локалізацію з сучасними технологіями, що робить її конкурентоспроможною альтернативою існуючим рішенням для продажу handmade-іграшок.

1.3 Аналіз вимог до програмної системи

Функціональні вимоги

Для покупця:

- Реєстрація та авторизація
- Перегляд каталогу товарів
- Пошук і фільтрація за категоріями
- Управління замовленнями
- Оформлення замовлення
- Перегляд замовлень і статусів
- Взаємодія з продавцем (чат або повідомлення)
- Управління даними профілю

Для автора товару:

- Створення та редагування товарів
- Додавання фото та опису
- Зміна цін, кількості, категорій
- Обробка вхідних замовлень (підтвердження, відхилення)

Для адміністратора:

- Перегляд усіх товарів у системі
- Модерація та зміна статусу товару
- Перегляд змін у товарах (до/після)
- Блокування або схвалення контенту

Нефункціональні вимоги

- Адаптивність інтерфейсу (під різні екрани смартфонів)
- Зрозумілий, інтуїтивний дизайн
- Підтримка кількох мов (наприклад, українська та англійська)
- Безпечна авторизація та зберігання даних (наприклад, токени, хешування паролів)
- Надійність і масштабованість системи

Очікуваний результат

Результатом є повнофункціональна мобільна система, яка дозволяє майстрам управляти своїми товарами, а покупцям — зручно шукати, купувати та отримувати іграшки з доставкою. Додаток має зручний інтерфейс, підтримує сповіщення, фільтри, оплату та багатомовність.

1.4 Моделювання предметної області

Моделювання предметної області є важливою частиною розробки програмного забезпечення, яка дозволяє формалізувати вимоги, уникнути помилок і краще спланувати архітектуру системи.

Мета моделювання:

1. Чітке розуміння задачі та потреб користувачів
2. Формалізація вимог і бізнес-логіки
3. Зменшення ризику помилок у процесі реалізації
4. Єдина мова спілкування між членами команди
5. Основа для проєктування бази даних, API та UI

Для моделювання використано мову UML — універсальний стандарт візуалізації об’єктно-орієнтованих систем.

Розроблені діаграми UML:

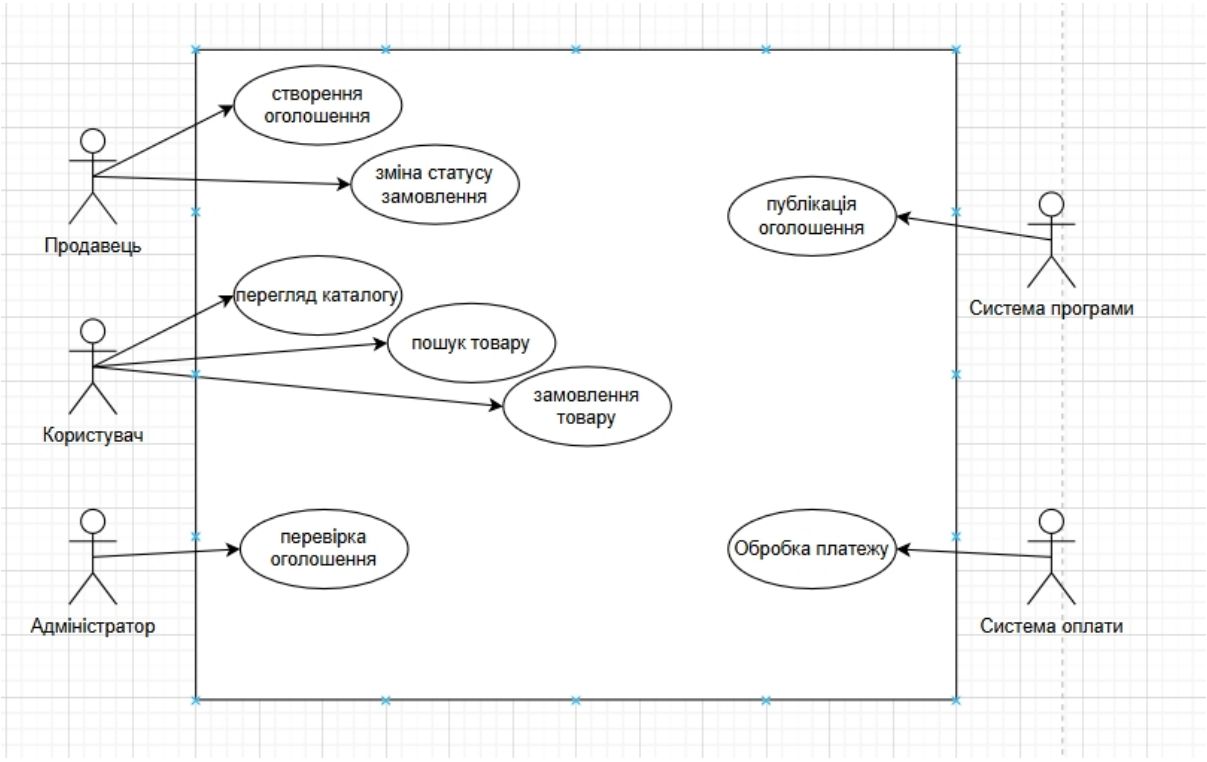


Рис. 1.1 Діаграма прецедентів (Use Case) — ілюструє взаємодію користувачів із системою.

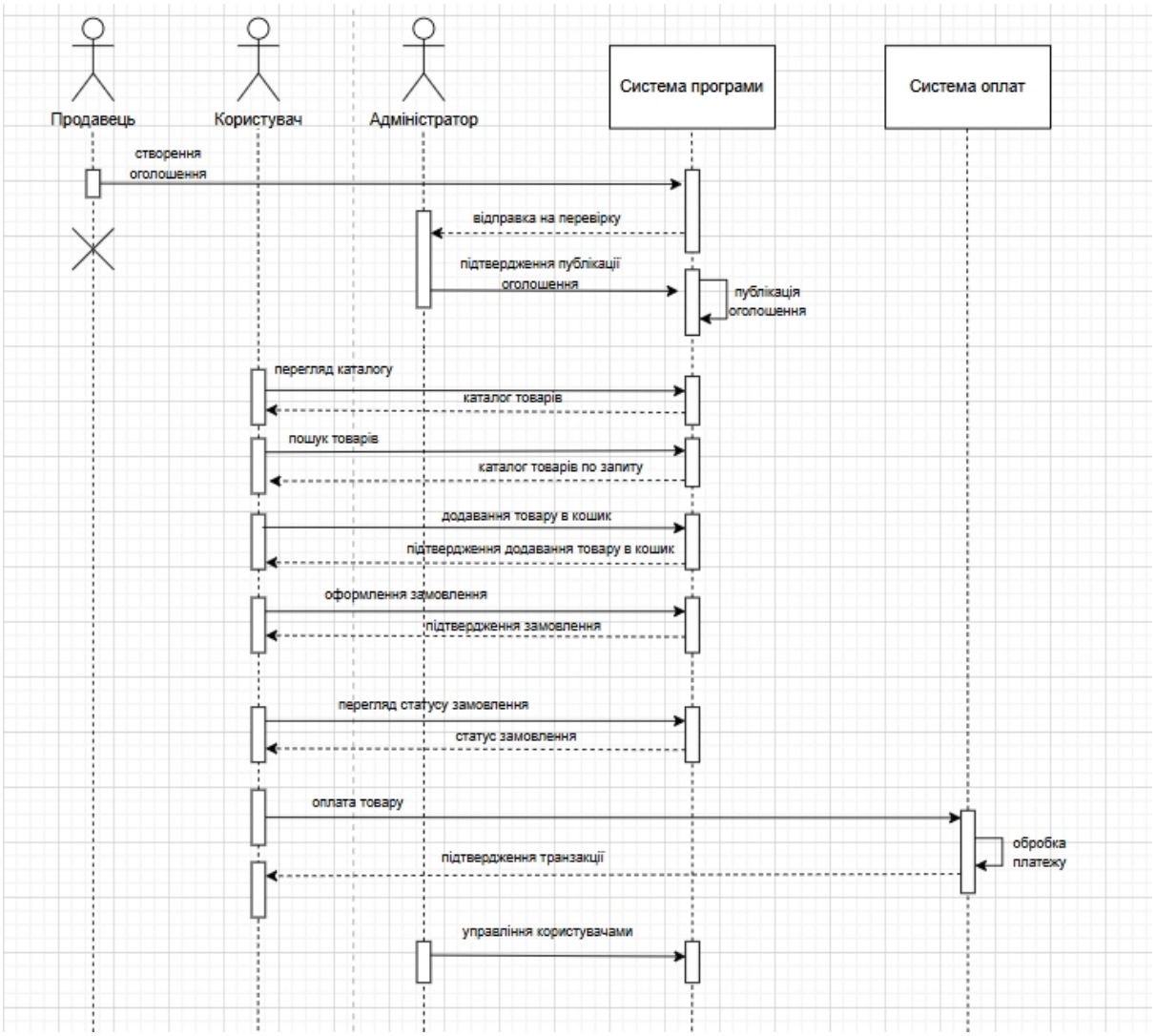


Рис.1.2 Діаграма послідовності (Sequence) — демонструє порядок обміну повідомленнями між об’єктами.

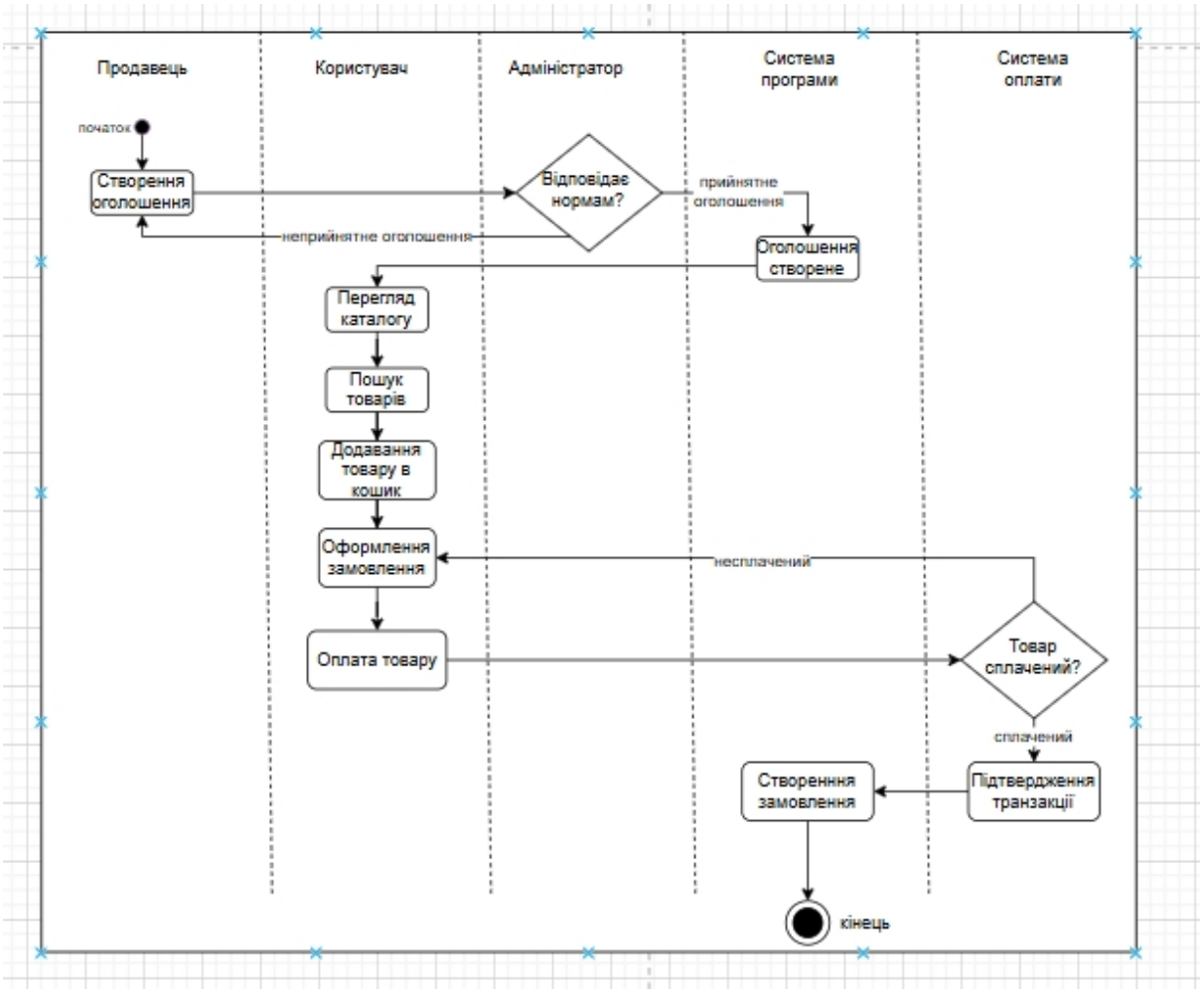


Рис.1.3 Діаграма активності (Activity) — показує логіку виконання бізнес-процесів.

1.5 Постановка завдання

Метою є створення мобільного додатку для продажу handmade-іграшок, який:

- дає змогу легко керувати товарами та замовленнями;
- має зручний інтерфейс для покупців і продавців;
- підтримує адміністрування та модерацію;
- дозволяє розширення функціоналу в майбутньому.

Розробка реалізується з використанням сучасних підходів до клієнт-серверної архітектури, об'єктно-реляційного моделювання, авторизації через токени, а також підтримки зберігання зображень і оплати товарів.

1.6 Висновки розділу

Порівняльна характеристика платформ для продажу handmade-товарів

Таблиця 1.1 – Порівняння існуючих аналогів інформаційної системи

Критерій	Etsy	Tedsby	Shafa (Crafta)	Мобільний додаток (проєкт)
Аудиторія	Глобальна, мільйони користувачів	Нішева (Teddy Bears)	Українська аудиторія	Локальна (орієнтація на українських майстрів)
Комісії	5% + додаткові збори	Комісії не фіксовані	Мінімальні або відсутні	Відсутні
Мобільний застосунок	Є для Android та iOS	Відсутній	Відсутній	Є (Flutter: Android/iOS)
Інтерфейс українською	Відсутній	Відсутній	Є	Є
Інтеграція з доставкою	Частково (через сторонні сервіси)	Відсутня	Є (Укрпошта, Нова Пошта)	Є (Нова Пошта API)
Онлайн-оплата	Stripe, PayPal	Відсутня	Частково (наложений платіж)	Є (Stripe Checkout, баланс користувача)
Панель модерації	Автоматизована/частково ручна	Відсутня	Часткова	Є (ручна модерація адміністратором)
Можливість чату з продавцем	Відсутня / через сторонні засоби	Відсутня	Відсутня	Є (система повідомлень)
Вартість публікації товару	Платна ($0.20 за кожну публікацію)	Переважно безкоштовно	Безкоштовно	Безкоштовно
Підтримка кастомізації	Обмежена шаблонами Etsy	Мінімальна	Відсутня	Повна (редагування профілю, оголошень)

У цьому розділі проаналізовано предметну область, визначено основні ролі користувачів, сформульовано функціональні та нефункціональні вимоги до системи. Для кращого розуміння та подальшого проєктування побудовано UML-діаграми. Це дозволить ефективно реалізувати систему, яка відповідатиме потребам handmade-майстрів і забезпечить якісний досвід для кінцевих користувачів.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

2.1 Логічна модель даних у вигляді ER-діаграми

Для побудови ефективної бази даних важливо попередньо спроектувати логічну модель системи. Найпоширенішим способом відображення структури даних є ER-діаграма (Entity-Relationship Diagram). ER-діаграми дозволяють візуалізувати сутності, атрибути та зв'язки між ними.

Основні цілі використання ER-діаграм:

- Проєктування бази даних;
- Візуалізація структури даних;
- Полегшення комунікації між розробниками;
- Документування структури системи;
- Аналіз бізнес-процесів;
- Виявлення потенційних помилок у структурі даних.

У процесі розробки було створено ER-діаграму в інструменті draw.io (Рис. 4), яка демонструє логічну модель даних, що використовується в системі. Вона буде використана для реалізації бази даних на етапі програмування.

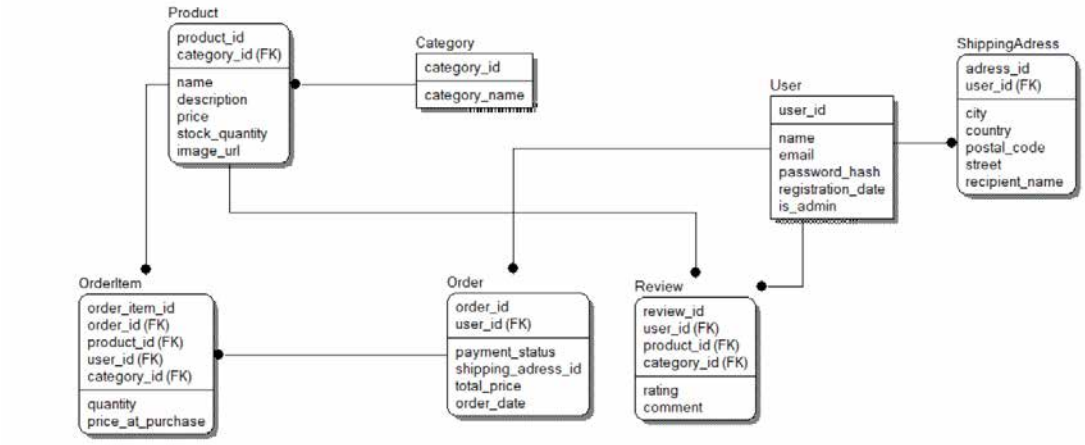


Рис.3 ER Діаграма

2.2 Діаграма класів та кооперацій

2.2.1 Діаграма класів

Діаграма класів є одним із ключових інструментів моделювання в UML і відображає об'єкти (класи), їх атрибути, методи та взаємозв'язки між ними. Це дозволяє розробнику краще розуміти структуру системи до початку програмування.

Призначення діаграми класів:

- Проєктування об'єктно-орієнтованого ПЗ;
- Візуалізація структури коду;
- Аналіз системи;
- Документування системи.

Діаграма класів (Рис. 5) описує, які класи, атрибути та методи будуть використовуватись у системі.

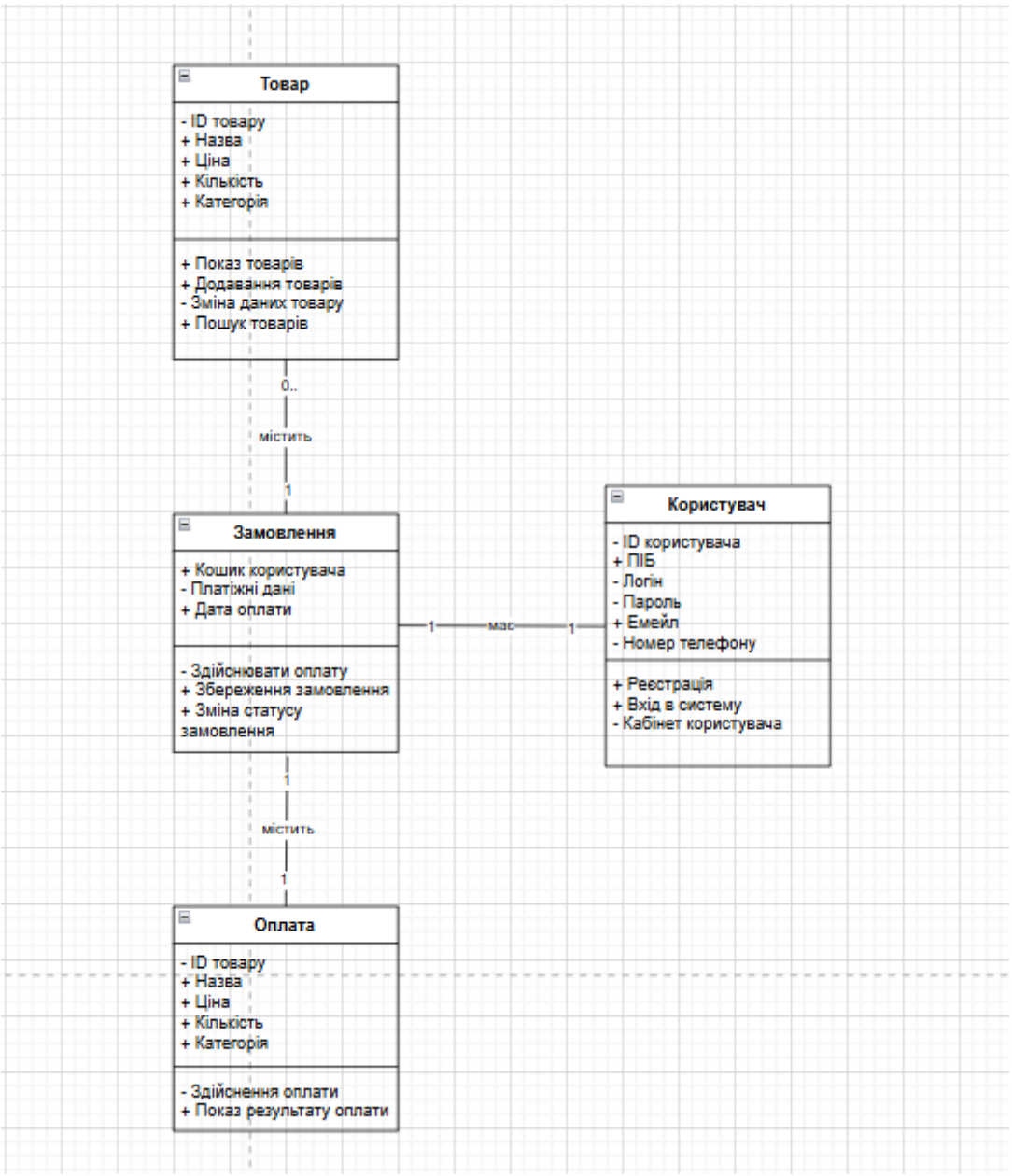


Рис.4 Діаграма класів

Описує класи їх атрибути і методи які будуть задіяні під час розробки системи

2.2.2 Діаграма кооперацій

Діаграма кооперацій (або діаграма зв'язків) показує взаємодію об'єктів у межах одного сценарію використання. На відміну від діаграми послідовності, вона акцентує увагу на структурі зв'язків, а не на часі виконання.

Призначення:

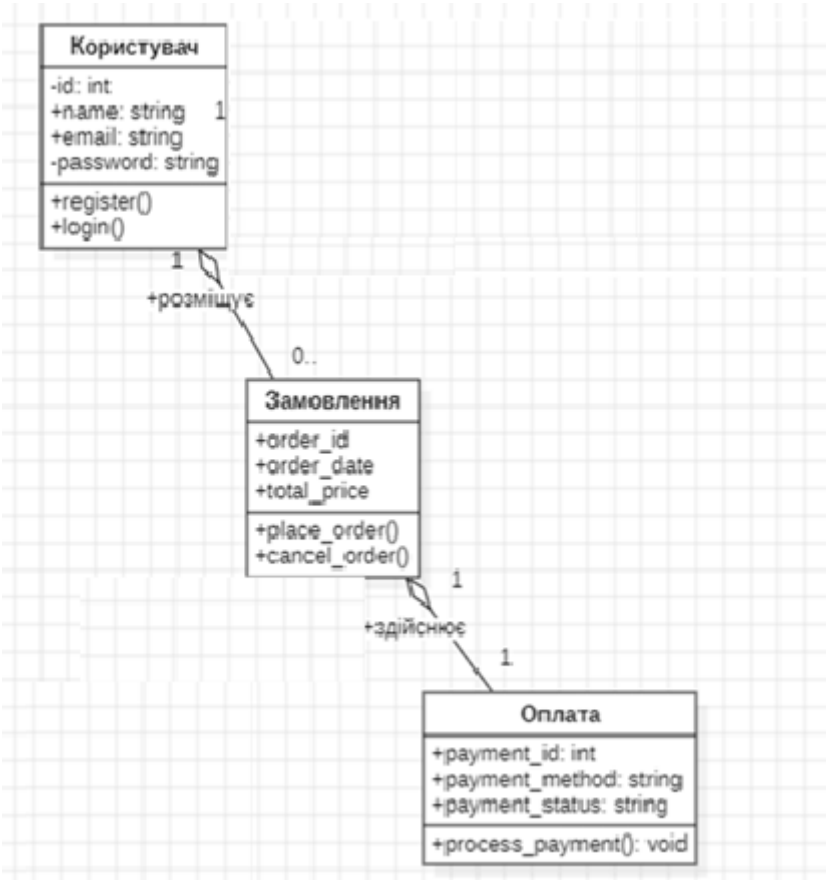
- Відображення взаємодій між об'єктами;
- Показ викликів методів;
- Деталізація логіки взаємодії в рамках одного процесу.

Оскільки система не містить класичного кошика, сценарій взаємодії передбачає оформлення замовлення безпосередньо з картки товару. Користувач натискає кнопку "Замовити", переходить на екран введення деталей замовлення (кількість, спосіб оплати, адреса), після чого підтверджує замовлення. Якщо обрано онлайн-оплату — користувач одразу може здійснити оплату. Автор товару (продавець) отримує замовлення та приймає рішення — підтвердити чи відхилити його.

Були створені кілька діаграм кооперацій, зокрема:

- Рис. 5 —Діаграма кооперацій створення замовлення;

Описує взаємодію управління даними для замовлення в системі.



- Рис. 6 — Діаграма кооперацій оформлення та оплата замовлення.

Показує взаємодію управління даними для оплати замовлення

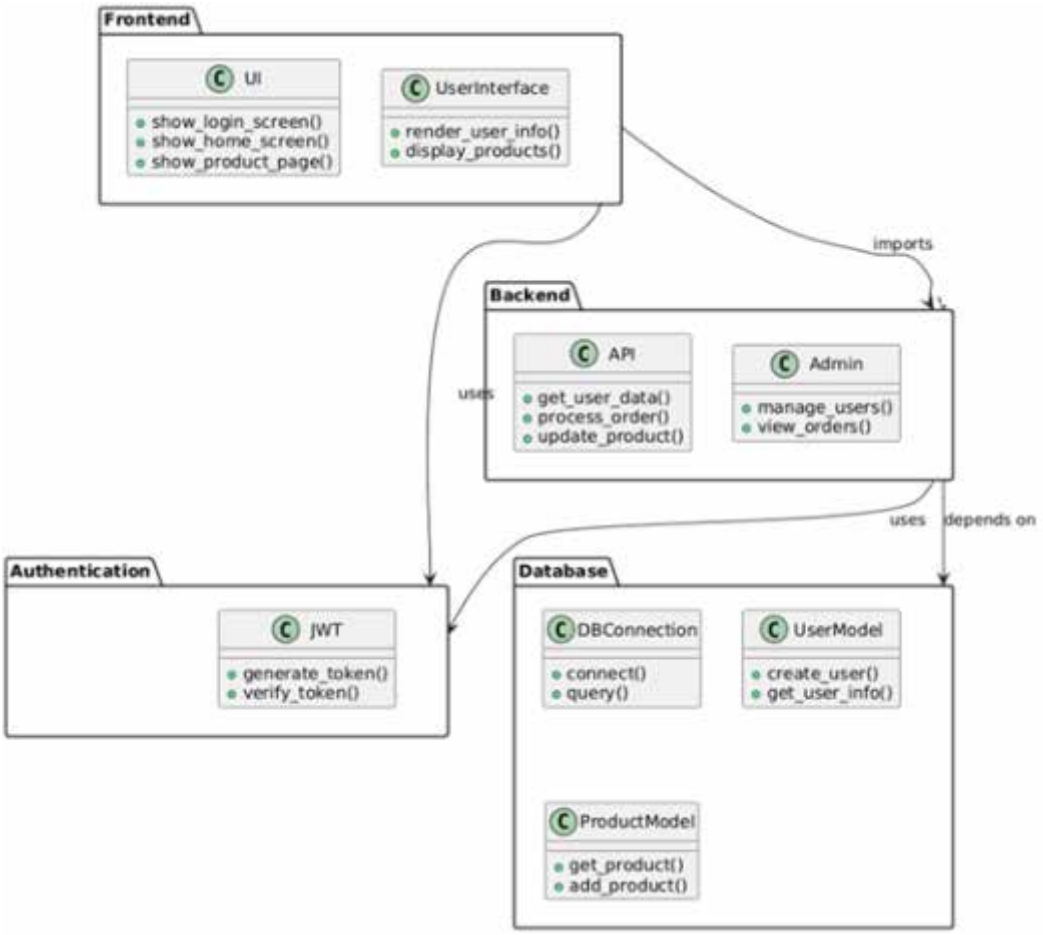
Повні версії діаграм наведено у Додатку В.

2.3 Діаграма пакетів

Діаграма пакетів використовується для структурування великих систем у логічні модулі. Вона дозволяє бачити залежності між частинами системи та знижує зв’язність компонентів.

Призначення:

- Упорядкування проєкту;
- Візуалізація залежностей між модулями;
- Зменшення зв’язності та підвищення модульності;
- Документація структури системи.



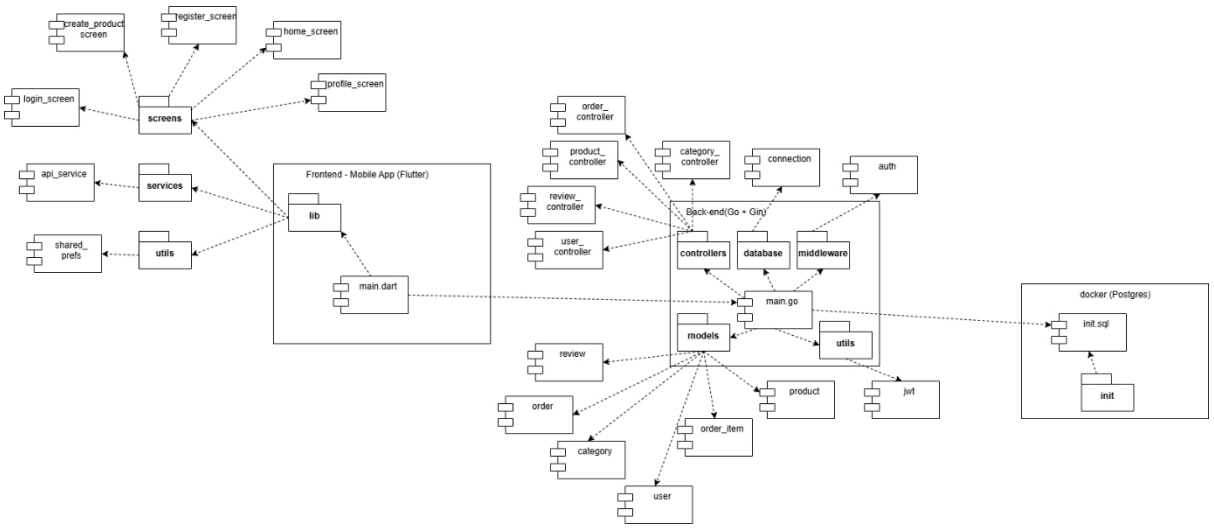
Діаграма пакетів (Рис. 7) Демонструє, як згруповано компоненти у проєкті.

2.4 Діаграма компонентів

Діаграма компонентів — це UML-діаграма, яка відображає фізичні частини системи (модулі, бібліотеки, пакети) та їхню взаємодію. Вона є важливою частиною опису архітектури.

Призначення:

- Візуалізація компонентів і залежностей;
- Планування архітектури;
- Комунікація між командами;
- Документування проєкту.



Діаграма компонентів (Рис. 8)

Показує, з яких частин складається система та які залежності між ними існують.

2.5 Висновки розділу

У результаті аналізу було побудовано низку UML-діаграм, які демонструють структуру, логіку та архітектуру системи. Це дозволяє перейти до етапу реалізації з чітким розумінням функціональних зв’язків, взаємодій між модулями та структури бази даних.

3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Система управління інформаційною базою

Одним із ключових етапів побудови прикладного програмного забезпечення є розробка ефективної системи управління інформаційною базою. Інформаційна база — це сукупність структурованих даних, що використовуються для збереження, обробки, аналізу та передачі інформації, необхідної для функціонування інформаційної системи. Її роль є фундаментальною в процесі забезпечення коректної, узгодженої, своєчасної та надійної обробки запитів користувачів.

В межах реалізації програмного забезпечення мобільного додатку інтернет-магазину іграшок постало питання вибору оптимального засобу для організації системи зберігання та опрацювання даних. Враховуючи вимоги до цілісності даних, масштабованості, високої швидкодії та підтримки транзакцій, було прийнято рішення використати реляційну об’єктно-орієнтовану базу даних PostgreSQL як основне сховище інформації.

Серед важливих характеристик PostgreSQL, які визначили її вибір для даного проєкту, варто виокремити наступні: повна підтримка стандарту SQL, відкритий вихідний код, можливість роботи з JSON та іншими напівструктурованими типами даних, наявність потужної ORM-підтримки через сторонні бібліотеки, а також активна міжнародна спільнота, яка забезпечує регулярні оновлення та покращення системи. Крім того, PostgreSQL дозволяє гнучко налаштовувати схеми без шкоди для загальної продуктивності.

Для розгортання бази даних PostgreSQL у проєкті використано технологію контейнеризації Docker.

Це дозволяє запускати інстанси СУБД в ізольованому середовищі, конфігурувати параметри доступу, керувати портами та змінними середовища, а також забезпечити відтворюваність результатів у різних середовищах — від розробки до продакшну. Використання Docker сприяє стандартизації процесів і підвищенню безпеки, оскільки кожен контейнер працює незалежно від основної системи.

Файл `docker-compose.yml` використовується як головний інструмент для конфігурації сервісу бази даних. У ньому описано такі параметри як ім’я контейнера, образ PostgreSQL, порти (зокрема, переадресація 5432 → 5433 на локальній машині), а також об’єми (volumes), які зберігають дані на жорсткому диску навіть після перезапуску контейнера. Це забезпечує збереження інформації та гнучке адміністрування структури БД.

Таким чином, підхід до створення інформаційної бази у рамках даного проєкту базувався на використанні сучасних відкритих технологій, що довели свою ефективність у промислових умовах. Комбінація PostgreSQL, Docker та грамотної моделі сутностей забезпечує надійну основу для побудови стабільної, продуктивної та масштабованої інформаційної

3.2 Розробка інформаційної бази

Розробка інформаційної бази є критично важливою складовою процесу створення інформаційної системи, адже саме на її основі відбувається збереження, обробка та обмін даними між усіма компонентами. У випадку мобільного додатку інтернет-магазину іграшок інформаційна база забезпечує взаємодію користувачів із товарами, замовленнями, відгуками, доставкою, авторизацією, повідомленнями та іншими функціональними блоками.

В основі реалізації інформаційної бази проєкту використано реляційну модель даних. Це означає, що всі об'єкти системи представлені у вигляді таблиць, між якими встановлено логічні зв’язки (один-до-багатьох, багато-до-багатьох тощо). Завдяки цьому забезпечується нормалізована структура даних, яка знижує надмірність та підвищує цілісність інформації.

Процес моделювання таблиць було здійснено на основі попередньо створених UML-діаграм, які описували предметну область, включаючи діаграму класів, прецедентів та діаграму кооперацій. Згідно з цими моделями, було сформовано таблиці: користувачів, товарів, категорій, замовлень, повідомлень, відгуків, тощо. Для кожної таблиці було передбачено первинні та зовнішні ключі, типи даних, обмеження унікальності та інші правила цілісності.

Створення таблиць реалізовано безпосередньо у коді серверної частини на мові Go. Для цього використовувались структури типу `struct`, які описують відповідні поля таблиць, а також анотації (`gorm` або `sql` теги), що задають імена колонок, обмеження та зв'язки. Після запуску застосунку відбувається автоматична міграція (створення або оновлення таблиць у базі даних), що дозволяє уникнути ручного написання SQL-запитів для кожної зміни структури БД.

Наприклад, модель товару (Product) включає такі поля: ID, назва, опис, ціна, категорія, кількість, зображення (URL до Firebase), ідентифікатор автора, статус (active/pending/inactive) тощо. Також передбачено зв'язки з іншими таблицями, зокрема з таблицею категорій (один до багатьох).

Крім того, були передбачені таблиці для зберігання: 1) інформації про користувачів (email, ім’я, роль, хешований пароль, баланс); 2) замовлень (тип оплати, кількість, адреса доставки, статус, покупець, продавець); 3) відгуків до товарів; 4) чатів і повідомлень (діалоги, відправник, отримувач, зміст, час створення); 5) статусів модерації з полем попередніх даних (`previous_data`) для збереження змінених значень при редагуванні товару.

Таким чином, у результаті було створено логічно пов’язану систему таблиць, яка відповідає структурі та логіці мобільного застосунку. Інформаційна база стала основою для реалізації всіх бізнес-процесів — від взаємодії з користувачами до формування замовлень та обробки оплат.

Таблиця залежностей між сутностями інформаційної бази

Таблиця 1.2 – залежності між сутностями інформаційної бази

Сутність	Залежна від	Тип зв’язку / Коментар
Users	-	Основна таблиця користувачів
Products	Users, Categories	Продукти належать користувачам і мають категорії
Categories	-	Список категорій товарів
Orders	Users, Products	Замовлення прив’язане до покупця і товару
Messages	Users	Повідомлення мають відправника і одержувача
Reviews	Users, Products	Відгуки від користувачів на товари
ShippingAddresses	Orders	Кожне замовлення має адресу доставки
PaymentStatus	Orders	Статус оплати належить до замовлення
ModerationLogs	Products, Users	Історія змін товарів модератором

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Вибір інструментарію є невід'ємною частиною процесу розробки прикладного програмного забезпечення, оскільки саме від обраних технологій, мов програмування, середовищ розробки та супровідних бібліотек залежить не лише швидкість і якість реалізації функціоналу, а й можливість масштабування, підтримки та інтеграції із зовнішніми сервісами. В даному розділі розглянуто обґрунтування використаного інструментарію для створення мобільного додатку інтернет-магазину іграшок.

У якості мови програмування для розробки серверної частини (бекенду) було обрано мову Go (Golang), яка відзначається високою продуктивністю, простотою синтаксису та вбудованою підтримкою паралелізму. Для побудови REST API використовувався фреймворк Gin, який забезпечує зручний механізм маршрутизації, підключення middleware, обробку запитів та інтеграцію з JWT, CORS і шаблонами.

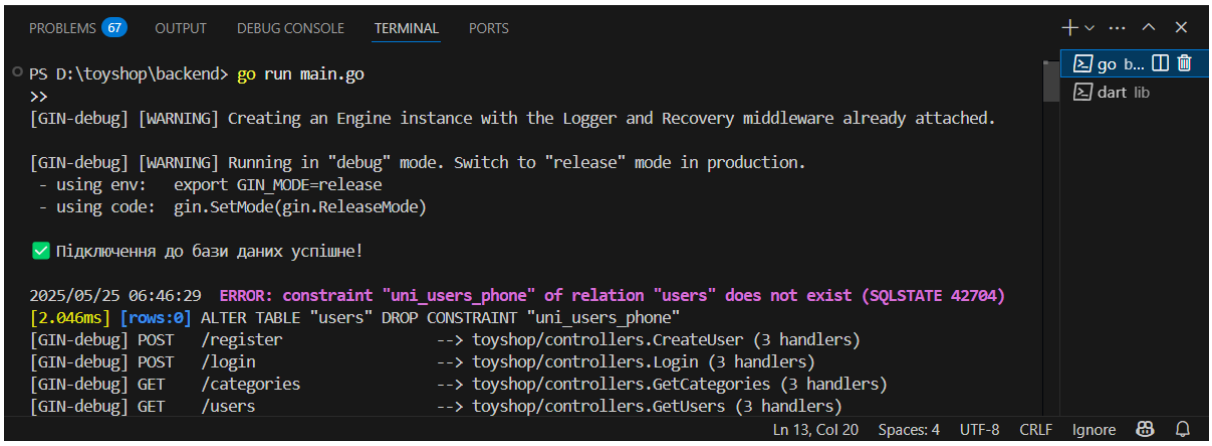


Рис. 9.1 Частина терміналу запущеного фреймворку Gin

Фронтенд-частина реалізована як кросплатформений мобільний додаток за допомогою фреймворку Flutter на мові програмування Dart. Flutter забезпечує єдину кодову базу для Android і iOS, сучасний інтерфейс користувача, підтримку state management (через Provider, Riverpod або інші підходи) та легку інтеграцію з RESTful API. Інструментальна підтримка включає hot reload, інтерфейсні редактори, і вбудовану систему навігації.

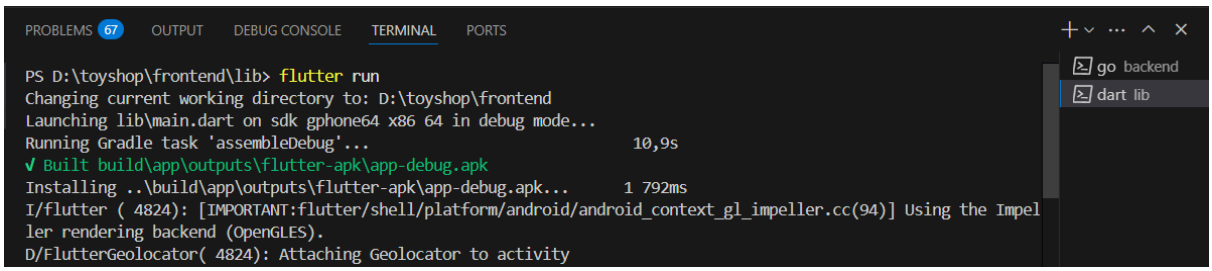


Рис. 9.2 Частина терміналу запущеного фреймворку Flutter

В якості бази даних для зберігання усіх критично важливих даних (користувачі, товари, замовлення, повідомлення, відгуки, оплати) використано об’єктно-реляційну СУБД PostgreSQL. База даних розгортається в ізольованому середовищі за допомогою Docker-контейнера. Це дозволяє уникнути проблем із сумісністю середовищ та забезпечує гнучкість у налаштуванні параметрів.

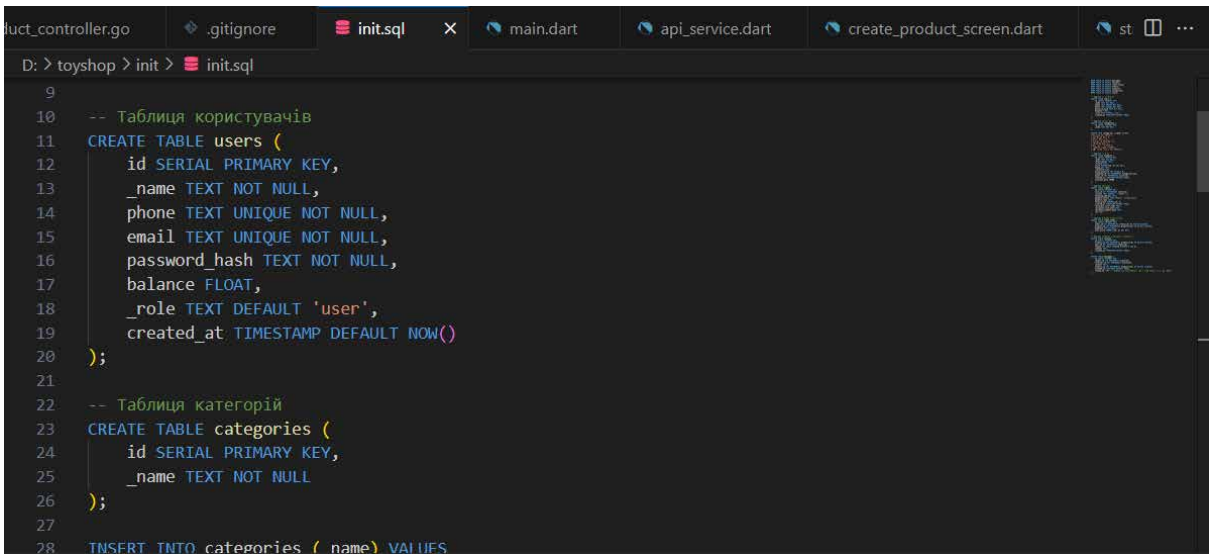


Рис. 9.3 Частина коду створення таблиць для PostgreSQL

Зображення товарів зберігаються у хмарному середовищі Firebase Storage, що дозволяє розвантажити основну базу даних і зберігати медіа-файли у спеціалізованому сховищі. Замість файлів у БД зберігаються лише URL-посилання. Це покращує продуктивність і дозволяє зменшити обсяг локального зберігання.

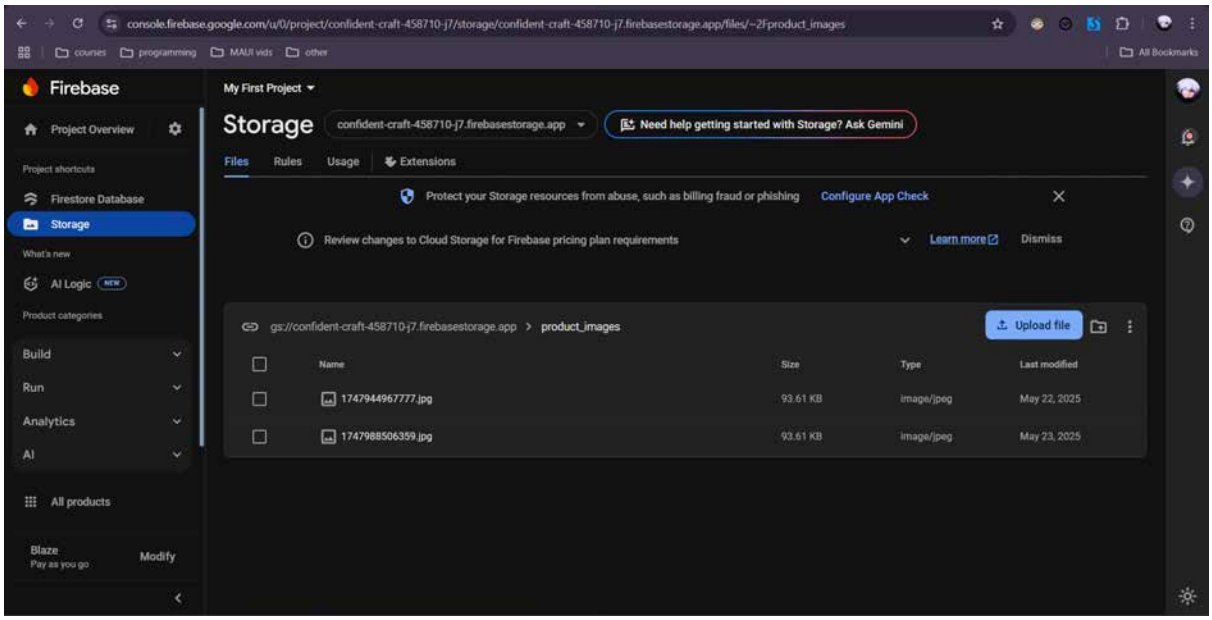


Рис. 9.4 Панель керування Firebase Storage

Для реалізації онлайн-платежів у мобільному додатку інтегровано Stripe API. Це рішення забезпечує повну підтримку обробки платежів з використанням карток, дотримання вимог PCI DSS, а також безпеку транзакцій. Зокрема, підтримується оплата після підтвердження замовлення, перевірка балансу, списання коштів, а також перенаправлення користувача на сторінку поповнення у разі недостатньої суми.

```dart
await Stripe.instance.initPaymentSheet(
  paymentSheetParameters: SetupPaymentSheetParameters(
    paymentIntentClientSecret: clientSecret,
    merchantDisplayName: 'ToyShop',
    style: ThemeMode.light,
  ),
);

await Stripe.instance.presentPaymentSheet();

final prefs = await SharedPreferences.getInstance();
final token = prefs.getString('token');
```

Рис. 9.5 Частина коду Stripe Api

Для розробки використовувалося середовище Visual Studio Code. Його було обрано через широку підтримку Flutter, Go, Docker, PostgreSQL та безлічі плагінів (Go, Flutter, Dart, Docker, PostgreSQL, Firebase, GitLens тощо). Більшість функцій, таких як автодоповнення, підсвічування синтаксису, форматування, дебаг, запуск емульованих пристроїв, були доступні без потреби в сторонніх IDE.

Версіонування та колективну розробку забезпечував Git у зв’язці з GitHub.

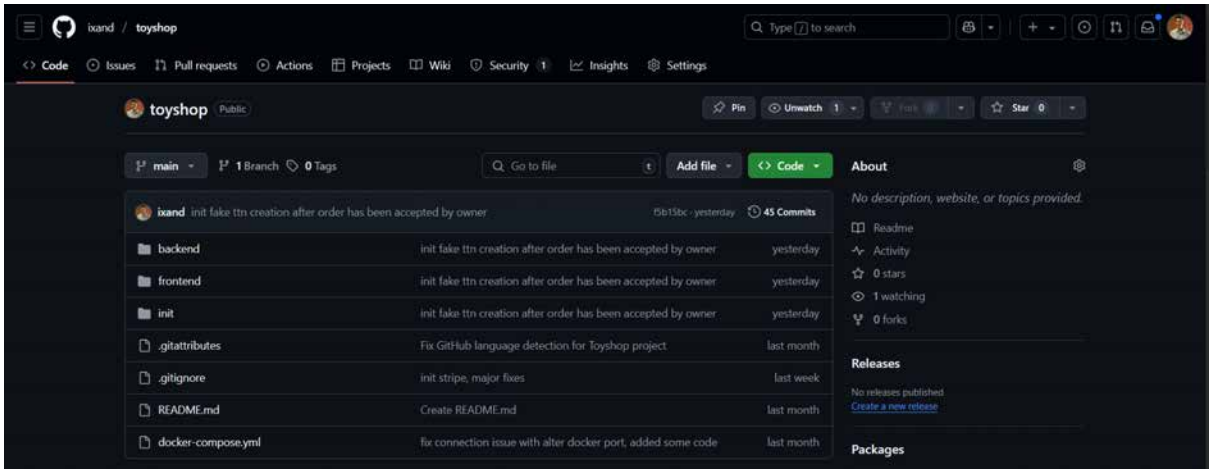


Рис. 9.6 Скріншот репозиторію

Репозиторій містить історію змін, окремі гілки для тестування, пул-реквести, рев'ю коду та інтеграцію через GitHub Actions.

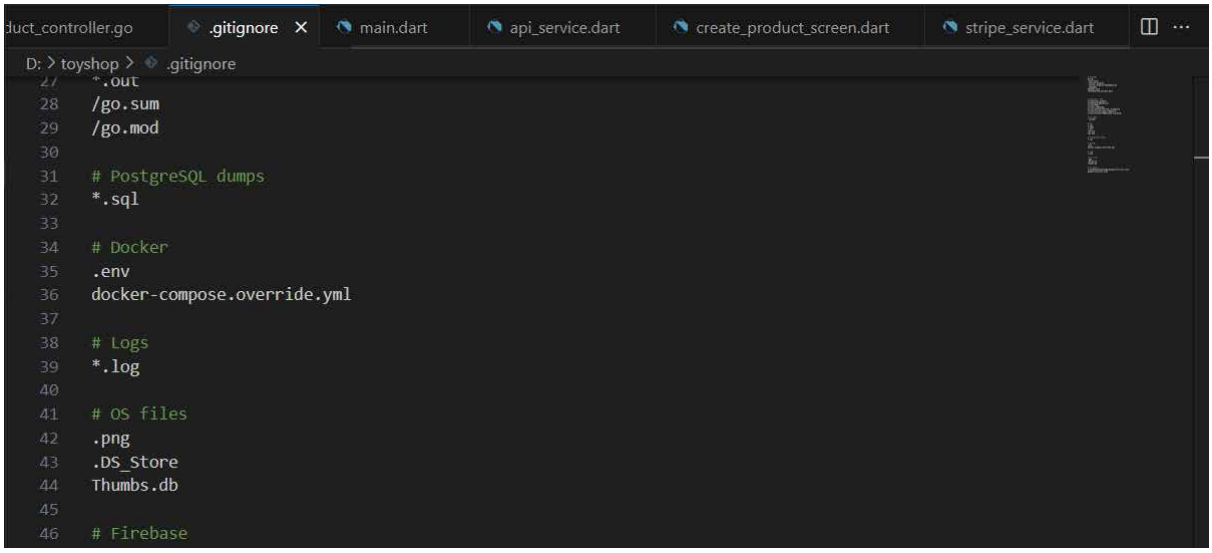


Рис. 9.7 Частина коду файлу .gitignore

Окремо конфігуровано `.gitignore` для виключення середовищних файлів, секретів, медіа та кешів з репозиторію. Для зберігання секретів використовувався `.env` файл (ключі Firebase, Stripe, DB доступ тощо).

3.4 Алгоритмізація та програмування програмних модулів

На етапі реалізації прикладного програмного забезпечення особливу увагу було приділено процесу алгоритмізації — формалізації бізнес-логіки у вигляді структурованих логічних послідовностей дій, які повинна виконувати система у відповідь на певні дії користувача або системні події. Алгоритмізація є основою ефективного програмування, адже дозволяє заздалегідь передбачити послідовність обробки даних, виявити гілки логіки, передбачити виняткові ситуації та забезпечити відповідність поведінки системи очікуванням користувачів.

Процес програмування функціональних модулів мобільного застосунку розпочався з побудови блок-схем, які описували логіку основних сценаріїв:

реєстрація та авторизація користувача, створення оголошення, модерація товарів, оформлення замовлення, оплата, доставка та комунікація між користувачами. Ці схеми дозволили побачити повну картину обробки даних — від моменту введення до відображення результату.

Програмна реалізація бекенд-логіки здійснювалася на мові Go з використанням фреймворку Gin. Було створено RESTful API з чітко структурованими маршрутами та контролерами, які відповідали за обробку запитів і повернення відповідей у форматі JSON. Зокрема, було реалізовано окремі групи ендпоінтів для реєстрації, логіну, товарів, замовлень, повідомлень, відгуків, категорій та адміністративних функцій. JWT-токени забезпечували автентифікацію, а middleware-функції контролювали доступ користувача до дозволених ресурсів.

Кожен запит, який надсилався з мобільного додатку, оброблявся відповідним маршрутом на сервері, проходив аутентифікацію, після чого відбувалась взаємодія з базою даних через SQL-драйвер.

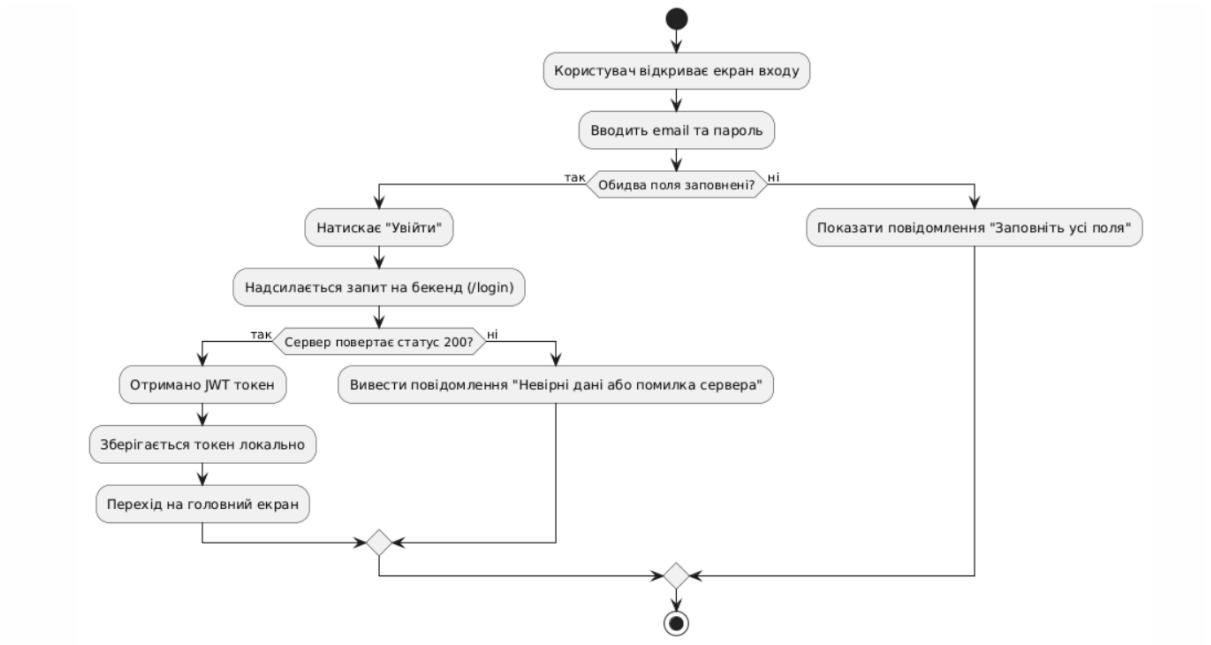


Рис. 10.1 блок-схема аутентифікації

Наприклад, при створенні замовлення система перевіряла валідність товару, його наявність, баланс користувача (у випадку онлайн-оплати) та створювала новий запис у таблиці замовлень, прив'язуючи його до покупця і продавця.

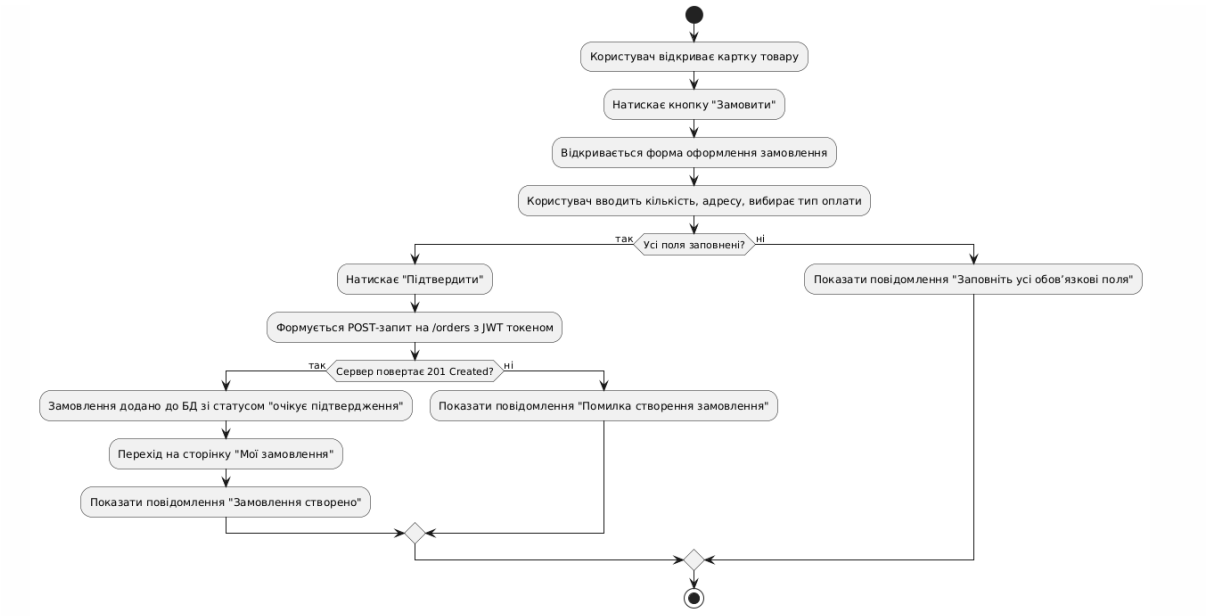


Рис. 10.2 блок-схема створення замовлення

Фронтенд-логіка реалізовувалась у Flutter із використанням структурованого підходу до навігації, управління станом та побудови інтерфейсу. Кожен екран застосунку був окремим віджетом (widget), що містив власний інтерфейс, бізнес-логіку та зв’язок із сервером. Було розроблено окремі екрани для логіну/реєстрації, каталогу товарів, сторінки товару, оформлення замовлення, підтвердження оплати, профілю користувача, модерації, доставки, перегляду чатів та повідомлень. Інформація оновлювалась динамічно за допомогою HTTP-запитів з обробкою статусів відповіді.

Для збереження сесії використовувалась локальна пам’ять пристрою — зокрема, SharedPreferences для збереження токенів та стану авторизації.

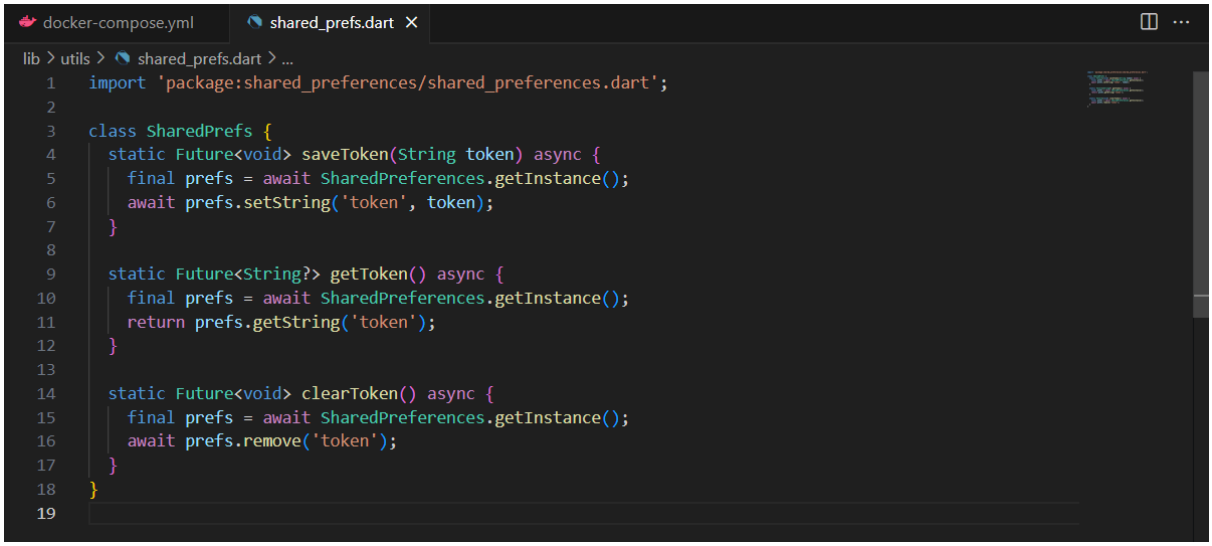


Рис. 11.1 Код для збереження токенів та стану авторизації.

Для інтеграції із Firebase Storage застосовувались офіційні пакети Flutter Firebase, які дозволяли користувачу завантажити фото, отримати URL і передати його у серверний запит для збереження в базі даних. Це забезпечило розділення медіа-контенту та даних.

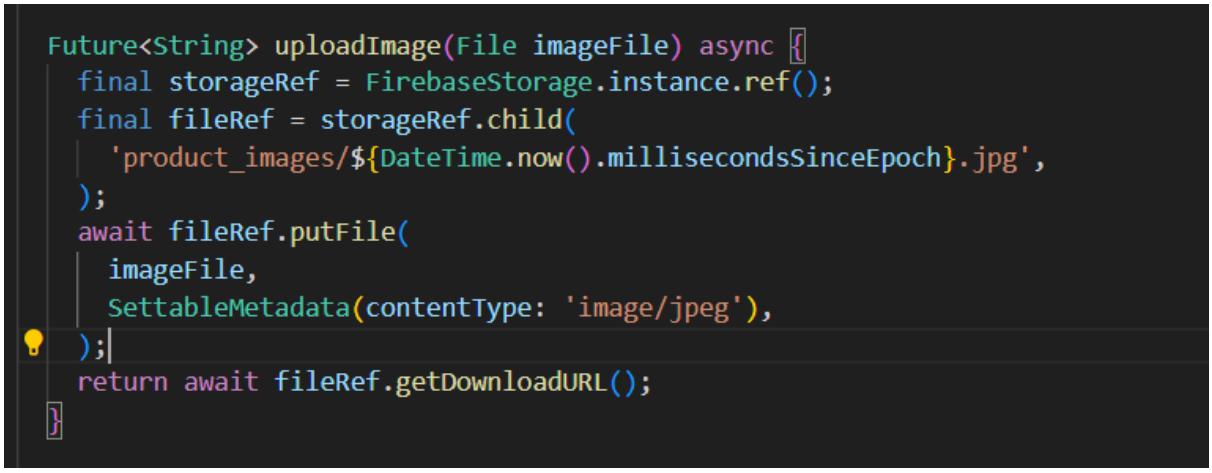


Рис. 11.2 Частина коду завантаження фото у Firebase Storage

Онлайн-платежі реалізовані через Stripe Checkout. Користувач при підтвердженні замовлення з типом оплати “онлайн” отримує можливість здійснити платіж безпосередньо у застосунку.

У разі недостатнього балансу — система сповіщає про це та пропонує поповнити рахунок.

Після успішної оплати баланс зменшується, статус замовлення змінюється, а продавець отримує повідомлення з кнопкою «прийняти» або «відхилити». У випадку прийняття — генерується номер ТТН і зберігається у базі даних для відображення у вкладці доставки.

Крім того, реалізована система модерації товарів: після створення чи редагування оголошення воно отримує статус 'очікується'. Адміністратор отримує список усіх товарів, що потребують перевірки, та може підтвердити або відхилити публікацію, при цьому зміни у вмісті відображаються кольоровим порівнянням (старе — червоним, нове — зеленим).

Таким чином, модулі системи були реалізовані згідно із заздалегідь спроєктованими алгоритмами, що дозволило забезпечити надійність логіки, простоту супроводу, відповідність вимогам користувача і технічному завданню. Результатом є цілісна система, що дозволяє реалізовувати повний життєвий цикл товару — від створення до доставки, із підтримкою повідомлень, рейтингу та управління власними оголошеннями.

3.5 Висновки розділу

У результаті виконання третього розділу дипломної роботи було безпосередньо реалізовано прикладне програмне забезпечення, що є центральною складовою мобільного додатку для інформаційної системи

інтернет-магазину іграшок. На основі вимог, сформованих у попередніх розділах, було здійснено вибір архітектури, технологій і засобів реалізації, що дозволило створити функціональну, сучасну та масштабовану систему.

Під час розробки було обґрунтовано вибір відповідного інструментарію. Як мову програмування для серверної частини використано **Go** у поєднанні з веб-фреймворком **Gin**, що забезпечує високу продуктивність та зручну маршрутизацію REST-запитів.

Для зберігання даних обрано **PostgreSQL** — надійну об’єктно-реляційну базу даних, яка забезпечує підтримку складних структур, транзакцій та індексів. З метою контейнеризації сервісів та спрощення їхнього розгортання було використано **Docker**, що дозволило відокремити середовище розробки від середовища розгортання та забезпечити стабільність.

Клієнтська частина реалізована з використанням **Flutter** — кросплатформного фреймворку, що забезпечує єдиний кодовий базис для Android, iOS та Web з нативною швидкодією та сучасним інтерфейсом. Для зберігання зображень товарів було інтегровано **Firebase Storage**, що дозволяє надійно зберігати мультимедійний контент та отримувати до нього URL-доступ. А для реалізації процесу **тестової онлайн-оплати** — впроваджено інтеграцію з **Stripe**, який забезпечує гнучку обробку транзакцій через web або мобільні інтерфейси.

У ході алгоритмізації були визначені ключові модулі системи: створення оголошень, підтвердження замовлень, обробка платежів, модерація товарів, інтеграція з API Нової пошти, перегляд повідомлень та управління профілем. Було створено зручну структуру маршрутів API, реалізовано відповідні запити у Flutter-додатку, а також забезпечено повний цикл взаємодії між користувачами (покупцем та автором товару).

Виконана розробка показала, що обраний стек технологій дозволяє ефективно реалізувати мобільний додаток із розширеною бізнес-логікою, підтримкою транзакцій, доставкою та модерацією. Особливу увагу приділено цільовій аудиторії — локальним виробникам хендмейд-іграшок, для яких система спрощує процеси публікації, продажу та обробки замовлень.

Таким чином, прикладне програмне забезпечення, реалізоване в межах дипломного проєкту, повністю відповідає поставленим завданням та демонструє готовність до реального використання з перспективою масштабування.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇСИСТЕМИ

4.1 Тестування системи

Для того щоб програмне забезпечення працювало стабільно, без збоїв і відповідало функціональним та нефункціональним вимогам, критично важливо проводити всебічне тестування на всіх етапах розробки. Це дозволяє виявити помилки на ранніх стадіях життєвого циклу продукту, мінімізувати витрати на виправлення багів та забезпечити високу якість кінцевого результату.

Особливу увагу в рамках цього проєкту було приділено покриттю як серверної частини, так і мобільного клієнта. Система включає кілька взаємодіючих компонентів (бекенд на Go, мобільний клієнт на Flutter, база даних

PostgreSQL, зовнішні API Firebase і Stripe), тому тестування мало на меті перевірку як кожного модуля окремо, так і взаємодії між ними.

Основні типи тестування, які були застосовані:

- **Модульне тестування (unit tests)**:
  Для перевірки окремих функцій та обробників HTTP-запитів у серверному API (реалізованому з використанням фреймворку Gin) було створено серію модульних тестів на Go. Вони перевіряють логіку реєстрації, логіну, обробки замовлень, валідації введених даних, створення та зміну статусу товарів тощо.
- **Інтеграційне тестування**:
  Було протестовано взаємодію бекенду з базою даних PostgreSQL — через попередньо підготовлені Docker-контейнери, які дозволяють створювати окреме тестове середовище з фіксованими параметрами. Також було перевірено коректну інтеграцію з Firebase Storage (завантаження/отримання зображень) та Stripe API (створення сесій оплати та обробка успішних транзакцій).
- **Тестування користувацького інтерфейсу (UI/UX testing)**:
  З боку клієнта, побудованого у Flutter, проводилось ручне функціональне тестування. Воно охоплювало ключові сценарії — логін, реєстрація, перегляд каталогу, замовлення товару, зміна профілю, поповнення балансу, перегляд замовлень, модерація, підтвердження або відхилення замовлень автором товару тощо. Основну увагу приділено перевірці зручності інтерфейсу, валідності форм, зворотного зв'язку після дій користувача (наприклад, діалогові вікна підтвердження, повідомлення про успіх/помилку).
- **Тестування мобільної адаптивності**:
  Особливої уваги надано коректному відображенню інтерфейсу на різних розмірах екранів. Flutter дозволяє забезпечити адаптивну верстку, однак при тестуванні на фізичних пристроях були виявлені і виправлені дрібні недоліки щодо падінь верстки при вузьких екранах.
- **E2E (end-to-end) тестування користувацьких сценаріїв** (обмежено):
  Було симульовано повний цикл: реєстрація → створення оголошення → модерація → перегляд товару іншим користувачем → створення замовлення → оплата → підтвердження автором → створення ТТН → перегляд у вкладці доставки.

Висновки за результатами тестування:

Тестування підтвердило стабільність роботи більшості модулів. У процесі були виявлені та усунені помилки в роботі з невалідними вхідними даними, неправильними статусами замовлень та обробкою недоступних товарів. Особливо ефективним виявилось попереднє тестування інтеграції зі сторонніми

сервісами (Firebase, Stripe), що дозволило уникнути критичних помилок на продакшн-етапі.

В цілому, проведене тестування забезпечило впевненість у коректній роботі системи та її готовності до впровадження в реальних умовах використання.

4.2 Вимоги до апаратного та програмного забезпечення

Для повноцінного розгортання системи мобільного інтернет-магазину іграшок необхідне наявність принаймні одного сервера, на якому буде розміщено основну логіку бекенду, базу даних і забезпечено взаємодію з клієнтськими мобільними пристроями. Мінімальна конфігурація такого сервера:

Процесор: Intel Core i3 або еквівалент з архітектурою x64

Оперативна пам’ять: не менше 8 ГБ

Жорсткий диск (HDD/SSD): від 120 ГБ

Операційна система: Linux (Ubuntu 20.04+ / Debian 11+) або Windows Server

Інтернет-з’єднання: стабільне з низькою затримкою (ping < 100 мс)

Однак для підвищення надійності та масштабованості проєкту рекомендується впровадити архітектуру на основі трьох серверів, де кожен відповідає за окремий функціональний модуль:

Web-сервер: виконує бізнес-логіку застосунку, обробляє REST-запити, відповідає за маршрутизацію запитів через фреймворк Gin (Go)

Сервер бази даних: окремий сервер з PostgreSQL, що зберігає інформацію про користувачів, товари, замовлення та платежі

Сервер кешу / сховища медіа: використовується для обробки запитів до Firebase Storage (через API) і зберігання тимчасових даних у майбутньому (опційно через Redis)

Усі сервіси обгорнуті у Docker-контейнери, що забезпечує легкість розгортання, незалежність від конкретної ОС, масштабованість і швидке відновлення. Для запуску системи на сервері достатньо мати встановлену Docker Engine та docker-compose. При потребі також використовується .env файл для безпечного зберігання ключів доступу (Stripe, Firebase, JWT).

Мінімальні вимоги для користувача (мобільного клієнта)

Система орієнтована на мобільних користувачів. Flutter-застосунок працює на більшості сучасних смартфонів, які підтримують Android 7.0+ або iOS 13+.

Android (мінімальні вимоги):

Операційна система: Android 7.0 (Nougat) або новіше

Оперативна пам’ять: від 1.5 ГБ

Вільне місце на пристрої: щонайменше 100 МБ

Інтернет: необхідний для роботи з сервером та Stripe Checkout

iOS (мінімальні вимоги):

Операційна система: iOS 13 або вище

Пристрої: iPhone 7 або новіше

Вільне місце на диску: не менше 100 МБ

Вимоги до адміністратора системи

Для керування бекендом (наприклад, модерація товарів, обробка замовлень, оновлення категорій) адміністратору необхідний доступ до веб-інтерфейсу або адміністративного модуля через браузер або термінал. Мінімальні вимоги:

Процесор: будь-який сучасний з підтримкою SSE2

Операційна система: Windows 10, macOS 10.14+ або сучасний дистрибутив Linux

Браузер: Chrome, Firefox, Edge, Safari — останні версії

Доступ до портів: 443 (HTTPS), 80 (HTTP), 5432 (PostgreSQL — для локального налаштування)

Програмні залежності на сервері

Docker (версія 20.10+) та docker-compose

Go (версія 1.20+) — для локальної розробки (опційно)

PostgreSQL (версія 13+)

curl, git, make — базові утиліти для CI/CD

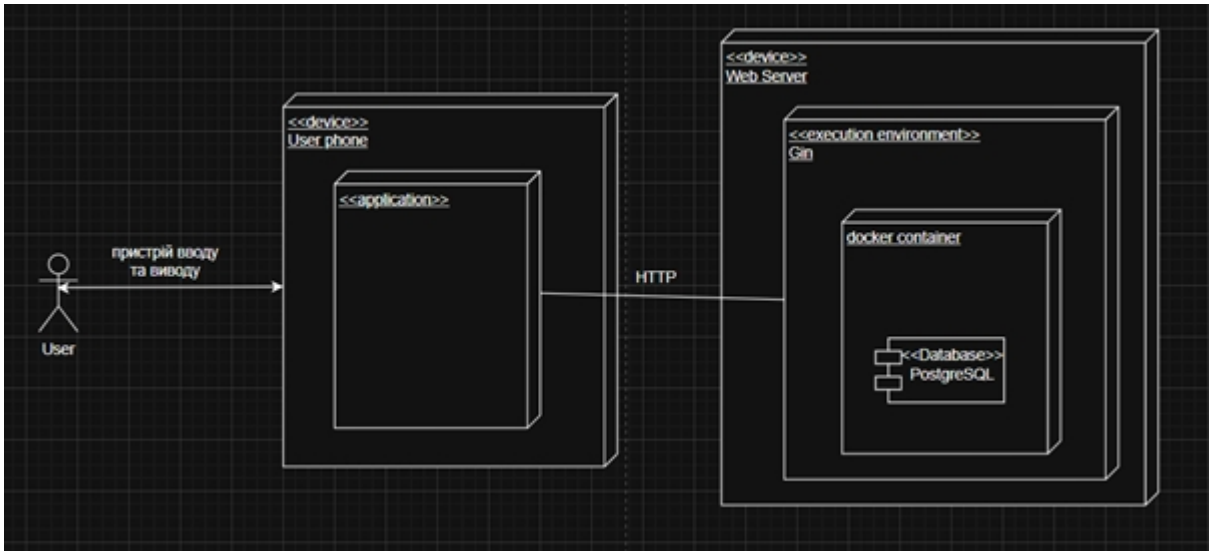


Рис. 12 Схема розгортання системи

Схематичне представлення розгортання системи подано на Рис. 12. У ній зазначено основні компоненти, точки взаємодії (Flutter App ↔ API ↔ PostgreSQL ↔ Firebase ↔ Stripe) та контейнери, у яких виконуються окремі модулі.

4.3 Склад інсталяційного пакету

У зв’язку з використанням **Docker-контейнеризації**, система є зручною у розгортанні та не потребує ручного встановлення залежностей. Усі необхідні модулі згруповано у вигляді ізольованих контейнерів, які автоматично взаємодіють між собою на основі файлу docker-compose.yml.

Інсталяційний пакет включає:

- **Docker** — обов’язковий компонент для запуску контейнеризованої системи. Потрібно попередньо встановити Docker Engine.
- **Docker-compose** — інструмент для одночасного запуску кількох пов’язаних контейнерів (бекенд, база даних).
- **Контейнер з бекенд-сервером**:
  - Побудований на базі мови **Go** з використанням фреймворку **Gin**;
  - Містить HTTP REST API, логіку обробки запитів, взаємодію з Firebase Storage та Stripe API;
- **Контейнер з PostgreSQL**:
  - Офіційний образ PostgreSQL (версія 13 або новіше);
  - Попередньо налаштований на створення бази даних, користувача, пароля;
- **Файл init.sql** (опціонально): SQL-інструкції для початкового створення структури таблиць;
- **.env файл** — файл конфігурації, в якому зберігаються глобальні змінні середовища, необхідні для запуску системи;
- **Інструкція з розгортання** — включає покрокове налаштування Docker та запуск команд docker-compose up.

.env файл системи

Файл .env використовується для конфігурації під час розгортання. Він містить змінні середовища, які вказують на:

- POSTGRES_HOST — адресу контейнера з базою даних PostgreSQL;

- POSTGRES_PORT, POSTGRES_USER, POSTGRES_PASSWORD — стандартні параметри доступу до БД;
- JWT_SECRET, STRIPE_API_KEY, FIREBASE_API_KEY — секрети, що використовуються для аутентифікації, оплати та зберігання файлів;
- BACKEND_URL — URL-адреса бекенду, що використовуватиметься для запитів з мобільного застосунку;
- DEBUG — логічна змінна, що активує/деактивує режим розробника.

4.4 Опис роботи програми

Робота програми починається з головного екрана мобільного застосунку, де користувач бачить каталог товарів та може взаємодіяти з основними функціональними елементами інтерфейсу. Застосунок має інтуїтивно зрозумілу навігацію, адаптовану для використання на мобільних пристроях.

Каталог товарів

Після запуску програми користувач потрапляє до головного каталогу, в якому відображаються всі активні товари. Користувач має змогу здійснювати пошук за назвою, сортувати товари за ціною або датою додавання, а також фільтрувати за категоріями (рис. 13). При натисканні на картку товару відкривається докладна інформація про нього: зображення, опис, ціна, кількість, локація продавця та кнопка “Замовити”.

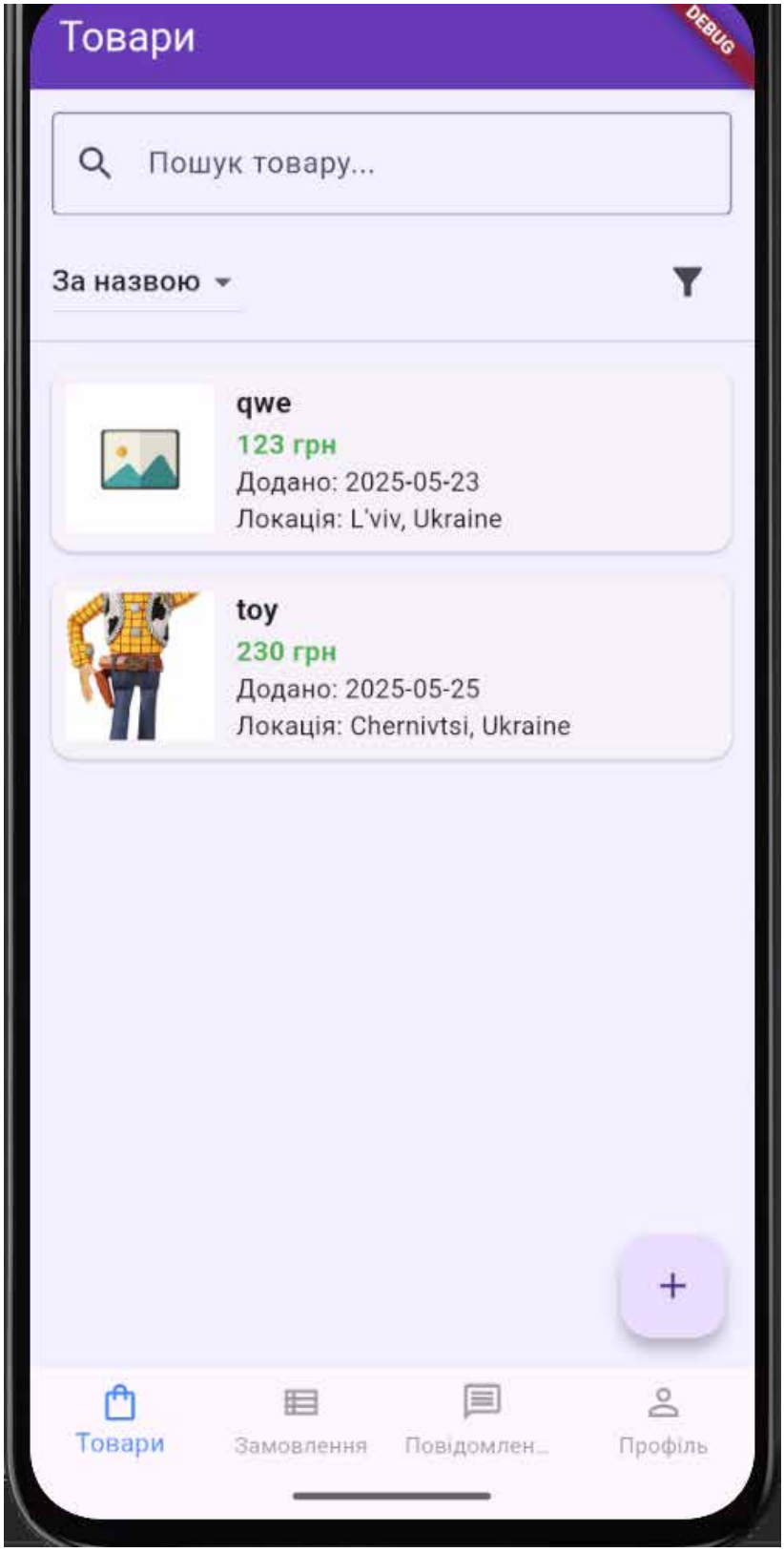


Рис. 13. Сторінка каталогу товарів з пошуком і фільтрами

Замовлення товару

Користувач, який бажає придбати товар, натискає кнопку “Замовити” без потреби створювати “кошик”. Після цього відкривається екран оформлення замовлення (рис. 13), де користувач вводить кількість товару, адресу доставки, обирає спосіб оплати (готівка або онлайн). У разі вибору онлайн-оплати, користувач перенаправляється на екран оплати через Stripe Checkout, а після успішної транзакції — повертається до програми.

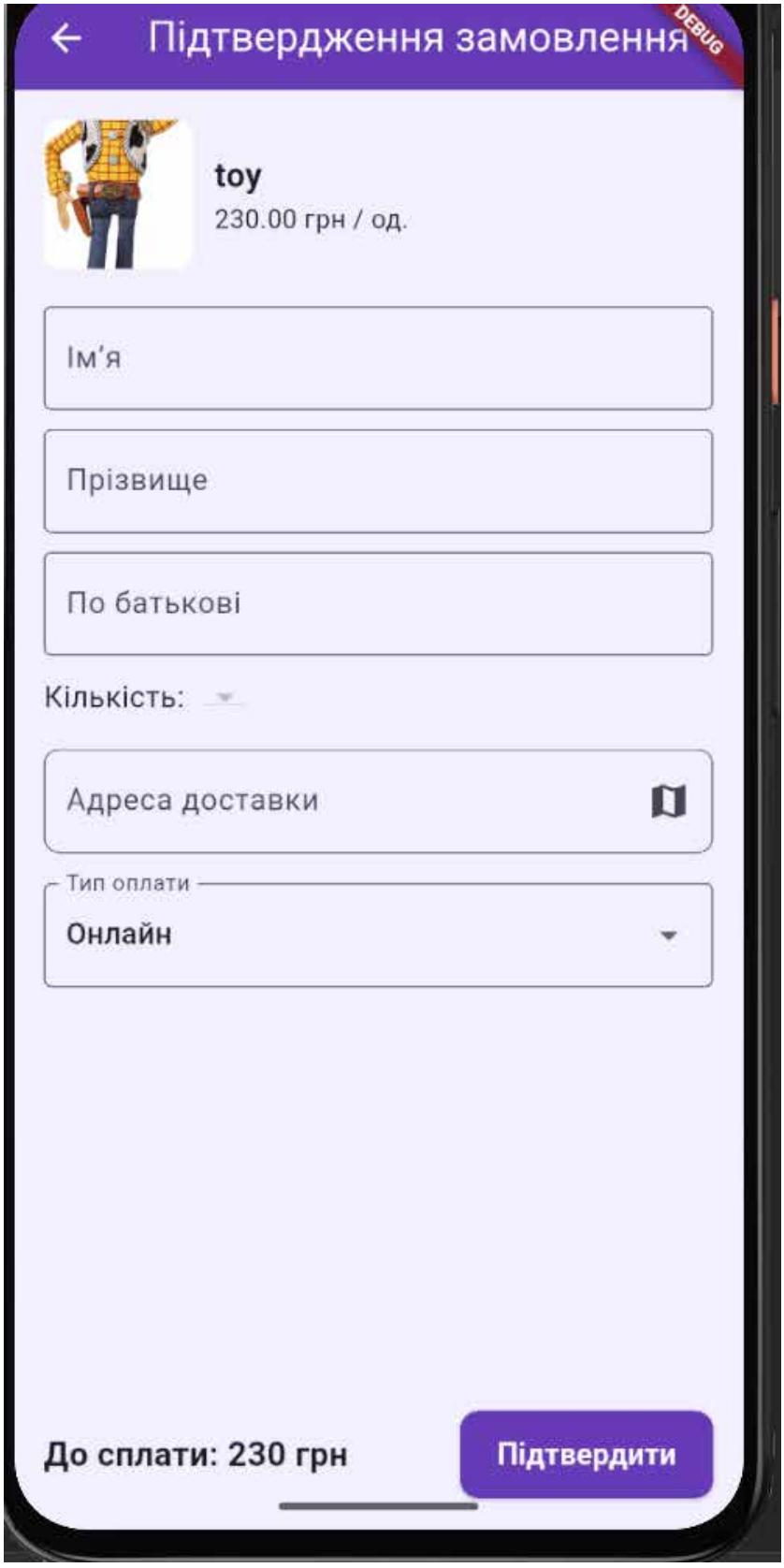


Рис. 13. Форма для оформлення замовлення

Перегляд замовлень

Після підтвердження замовлення користувач може переглядати історію своїх замовлень у вкладці “Мої замовлення” (рис. 14). Тут відображається список усіх замовлень користувача із зазначенням їх статусу: “Очікує підтвердження”, “Прийнято”, “Відхилено” або “Оплачено”.

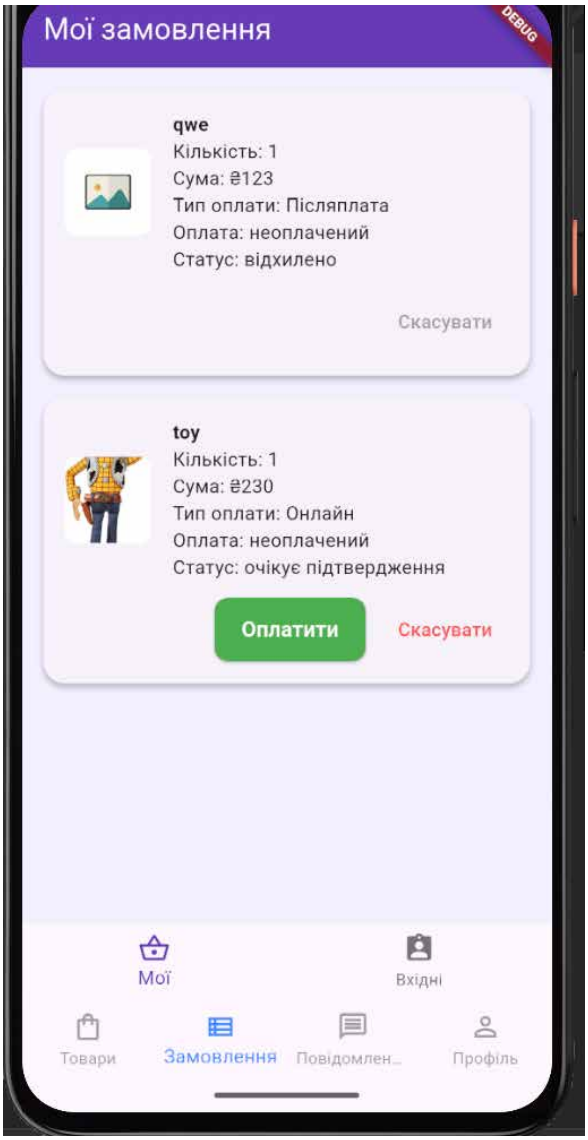


Рис. 14. Екран замовлень користувача

Робота з профілем

У вкладці “Профіль” (рис. 15) користувач може побачити інформацію про свій акаунт, дату реєстрації, email, а також переглянути свої оголошення, змінити аватар та вийти з облікового запису. Крім того, з цього екрана можна перейти до налаштувань, де доступна зміна мови, теми та вихід із системи.

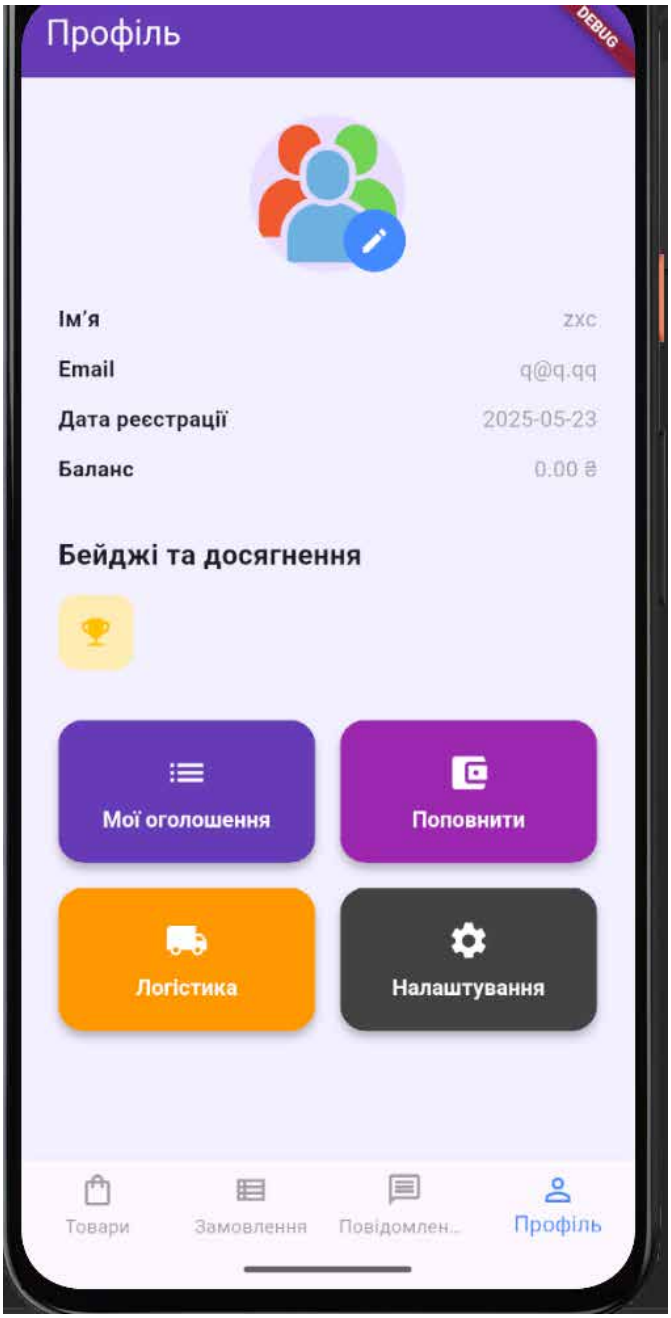


Рис. 15. Екран профілю користувача

Робота адміністратора

Користувачі з адміністративними правами мають доступ до спеціальної вкладки “Модерація”, де можуть переглядати всі оголошення, які мають статус “Очікує модерації” (рис. 16). Адміністратор має змогу переглядати деталі товару, порівнювати зміни (зеленим — нові значення, червоним — попередні) та встановлювати фінальний статус: “Активний” або “Неактивний”.

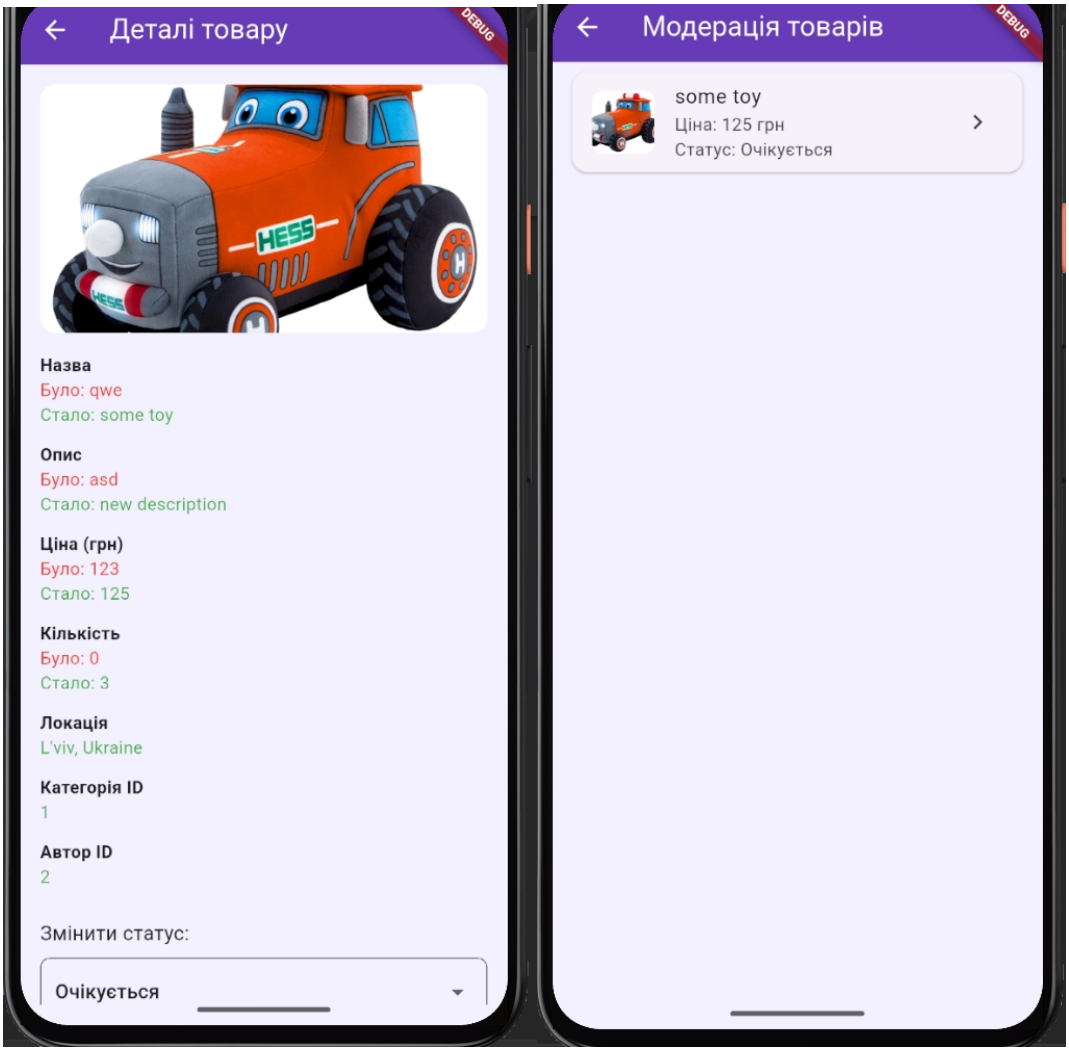


Рис. 16. Екран модерації товарів (адміністративна панель)

Система повідомлень

Користувачі мають змогу надіслати повідомлення автору товару безпосередньо зі сторінки товару або замовлення. Усі повідомлення групуються у вигляді чатів, де кожен діалог — це потік листування з конкретним користувачем (рис. 17).

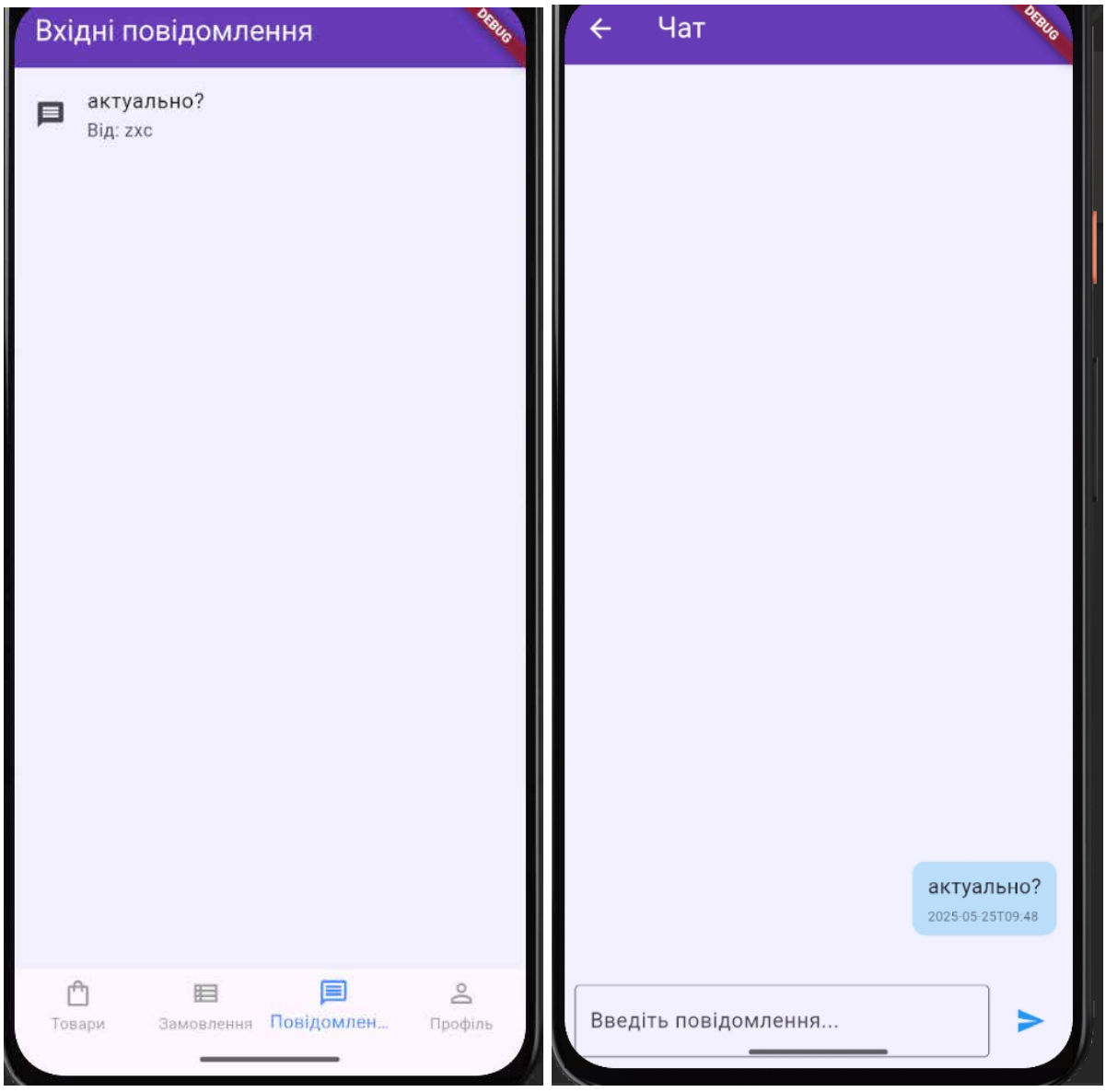


Рис. 17. Екран повідомлень (чат)

Доставка та логістика

Після підтвердження замовлення продавцем автоматично створюється накладна (ТТН) через інтеграцію з API Нової Пошти, дані з якої відображаються в розділі “Доставка”. Вкладка “Відправка” призначена для продавців, а “Отримання” — для покупців (рис. 18).

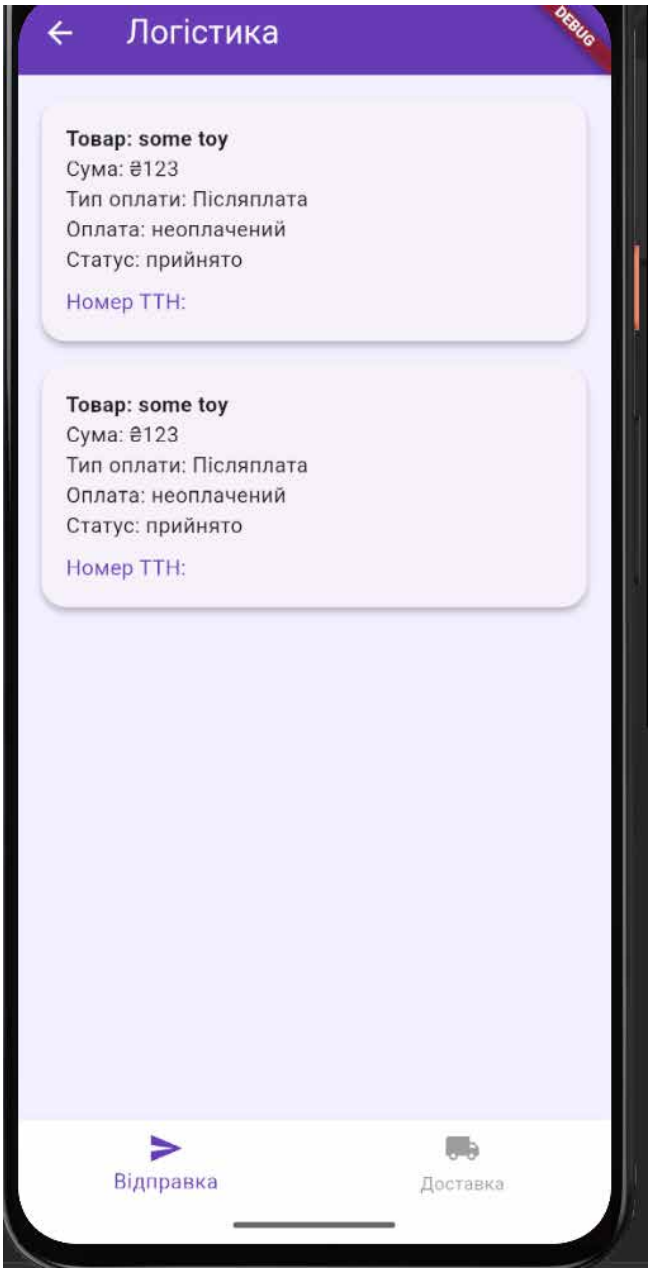


Рис. 18. Сторінка доставки з розділенням на вкладки “Відправка” / “Отримання”

4.5 Висновки розділу

У результаті виконання четвертого розділу було всебічно розглянуто ключові аспекти впровадження та експлуатації розробленої інформаційної системи мобільного додатку для інтернет-магазину іграшок.

Зокрема:

- Проведено базове функціональне тестування програмного забезпечення з метою виявлення логічних помилок у бізнес-логіці та перевірки стабільності роботи інтерфейсу користувача;
- Сформовано **інсталяційний пакет**, до якого включено усі необхідні елементи для швидкого та зручного розгортання системи за допомогою **Docker**;
- Визначено **вимоги до апаратного та програмного забезпечення**, як з боку серверної частини, так і з боку клієнта — користувача мобільного застосунку;
- Описано **послідовність взаємодії користувача із системою**, починаючи з моменту відкриття каталогу і закінчуючи створенням замовлення та переглядом історії у профілі;
- Розглянуто специфіку **роботи адміністратора**, включно з модерацією товарів, та додаткові функції системи, такі як повідомлення між

користувачами, створення ТТН через API Нової Пошти, автоматичне оновлення статусів тощо.

Таким чином, описані у розділі заходи забезпечують **надійне впровадження та стабільну експлуатацію** мобільного додатку в реальних умовах, включаючи масштабованість, простоту інсталяції, зручність користування та підтримку актуальних технічних вимог. Це створює основу для подальшого вдосконалення та можливого розширення функціоналу системи у майбутньому.

ВИСНОВКИ

У процесі виконання бакалаврської кваліфікаційної роботи на тему **«Програмне забезпечення мобільного додатку для інформаційної системи інтернет-магазину іграшок»** було реалізовано повноцінну інформаційну систему, що дозволяє користувачам зручно взаємодіяти з товарами handmade-майстрів, оформлювати замовлення, здійснювати оплату онлайн, а адміністраторам — модерацію контенту та керування оголошеннями.

У ході виконання роботи було досягнуто наступних результатів:

- **Розроблено архітектуру системи**, що складається з клієнтської частини на **Flutter**, серверної частини на **Go (Gin)**, та бази даних **PostgreSQL**, що розгортається у Docker-контейнерах;
- Реалізовано функціональність реєстрації, авторизації з JWT, профілю користувача з відображенням балансу та замовлень;
- **Створено каталог товарів** з можливістю пошуку, фільтрації та сортування, а також перегляду детальної інформації про товари;
- **Реалізовано систему замовлень без використання класичного кошика** — замовлення формується напряму з картки товару із заповненням адреси, кількості та типу оплати;
- Інтегровано **Stripe** для онлайн-платежів та **Firebase Storage** для зберігання фотографій товарів;
- Реалізовано двостороннє спілкування між користувачами у вигляді **чатів** та систему **повідомлень**;
- Впроваджено **систему модерації** оголошень з інтерфейсом адміністратора для підтвердження чи відхилення товарів та перегляду змінених полів;
- **Інтегровано API Нової Пошти** для автоматичної генерації ТТН після підтвердження замовлення продавцем;
- Проведено **функціональне тестування** окремих частин системи для забезпечення надійної роботи у реальних умовах;

- Отримано **практичні навички роботи з Docker**, організації баз даних у контейнерах, конфігурації .env-файлів та підключення сервісів у розподіленому середовищі;
- Застосовано на практиці знання з **проектування UML-діаграм**, серед яких: діаграми прецедентів, послідовностей, активностей, класів, компонентів та розгортання.

Загалом, у межах дипломної роботи було не лише розроблено сучасний мобільний застосунок, а й **закріплено теоретичні знання практикою**, що надало цінний досвід створення складного прикладного програмного забезпечення для електронної комерції з урахуванням потреб реального бізнесу.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. **Flutter Documentation**. Build apps for any screen. URL: https://docs.flutter.dev (дата звернення: 25.05.2025).
2. **Gin Web Framework**. A Go (Golang) web framework. URL: https://gin-gonic.com/docs (дата звернення: 25.05.2025).
3. **The Dart Programming Language**. Dart Language Tour. URL: https://dart.dev/guides/language/language-tour (дата звернення: 25.05.2025).
4. **PostgreSQL Documentation**. The world's most advanced open source database. URL: https://www.postgresql.org/docs/ (дата звернення: 25.05.2025).
5. **Firebase Documentation**. Store and retrieve user-generated content with Firebase Cloud Storage. URL: https://firebase.google.com/docs/storage

   (дата звернення: 25.05.2025).

6. **Stripe Documentation**. Accept payments online, in person, or through your platform. URL: https://stripe.com/docs (дата звернення: 25.05.2025).
7. **Docker Documentation**. What is Docker?. URL: https://docs.docker.com/get-started/overview/ (дата звернення: 25.05.2025).
8. **UML Practical Guide** – Unified Modeling Language Diagrams and Notations. Visual Paradigm. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-practical-guide/ (дата звернення: 25.05.2025).
9. **What is an ERD (Entity Relationship Diagram)?** Lucidchart. URL: https://www.lucidchart.com/pages/er-diagrams (дата звернення: 25.05.2025).
10. **Go by Example**. Practical Go code examples. URL: https://gobyexample.com/ (дата звернення: 25.05.2025).
11. **Stripe Flutter SDK GitHub Repository**. Example and integration guide. URL: https://github.com/stripe/stripe-flutter (дата звернення: 25.05.2025).

ДОДАТОК А

База даних

Київ-2025

Код створення таблиць:

```
DROP TABLE IF EXISTS messages;
DROP TABLE IF EXISTS reviews;
DROP TABLE IF EXISTS order_items;
DROP TABLE IF EXISTS orders;
DROP TABLE IF EXISTS products;
DROP TABLE IF EXISTS categories;
DROP TABLE IF EXISTS users;

-- Таблиця користувачів
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    _name TEXT NOT NULL,
    phone TEXT UNIQUE NOT NULL,
    email TEXT UNIQUE NOT NULL,
    password_hash TEXT NOT NULL,
    balance FLOAT,
    _role TEXT DEFAULT 'user',
    created_at TIMESTAMP DEFAULT NOW()
);

-- Таблиця категорій
CREATE TABLE categories (
    id SERIAL PRIMARY KEY,
    _name TEXT NOT NULL
);


-- Таблиця товарів
CREATE TABLE products (
    id SERIAL PRIMARY KEY,
    _name TEXT NOT NULL,
    _description TEXT,
```

```
    _status TEXT,
    price NUMERIC(10, 2) NOT NULL,
    image_url TEXT,
    _location TEXT,
    stock_quantity INT DEFAULT 0,
    category_id INT REFERENCES categories(id),
    owner_id INT REFERENCES users(id),
    created_at TIMESTAMP DEFAULT NOW(),
    previous_data JSONB
);

-- Таблиця замовлень
CREATE TABLE orders (
    id SERIAL PRIMARY KEY,
    user_id INT REFERENCES users(id),
    _status TEXT DEFAULT 'очікується',
    shipping_address TEXT,
    payment_status TEXT DEFAULT 'неоплачений',
    payment_type TEXT,
    total_price NUMERIC(10, 2),
    created_at TIMESTAMP DEFAULT NOW(),
    recipient_first_name TEXT,
    recipient_last_name TEXT,
    recipient_middle_name TEXT,
    ttn TEXT
);

-- Таблиця позицій у замовленні
CREATE TABLE order_items (
    id SERIAL PRIMARY KEY,
    order_id INT REFERENCES orders(id) ON DELETE CASCADE,
    product_id INT REFERENCES products(id) ON DELETE CASCADE,
    quantity INT NOT NULL,
    unit_price NUMERIC(10, 2) NOT NULL
);
```

```
-- Таблиця відгуків з каскадним видаленням
CREATE TABLE reviews (
    id SERIAL PRIMARY KEY,
    product_id INT REFERENCES products(id) ON DELETE CASCADE,
    user_id INT REFERENCES users(id),
    rating INT CHECK (rating BETWEEN 1 AND 5),
    comment TEXT,
    created_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE messages (
    id SERIAL PRIMARY KEY,
    sender_id INT REFERENCES users(id),
    receiver_id INT REFERENCES users(id),
    content TEXT,
    product_id INT REFERENCES products(id) ON DELETE CASCADE,
    created_at TIMESTAMP DEFAULT NOW(),
    thread_id TEXT  -- унікальний ідентифікатор пари (наприклад: "1_2" або UUID)
);
```

Код вставки даних у таблицю категорії :

```
INSERT INTO categories (_name) VALUES

('Іграшки для дітей'),

('Настільні ігри'),

('Конструктори'),

('Ляльки та аксесуари'),

('М’які іграшки'),

('Творчість та хобі'),

('Радіокеровані моделі'),

('Спортивні товари для дітей'),
```

('Розвиваючі іграшки'),

('Інтерактивні іграшки'),

('Дитячі музичні інструменти'),

('Іграшки для ванної'),

('Іграшки для вулиці'),

('Пісочниці та аксесуари'),

('Іграшкова зброя та арбалети'),

('Дитячі транспортні засоби'),

('Дитячі велосипеди'),

('Скейтборди та самокати'),

('Гіроборди та електротранспорт'),

('Розвиваючі книги та плакати'),

('Пазли та головоломки'),

('Кубики та сортери'),

('Моделі та збірні набори'),

('Фігурки героїв і тварин'),

('Машинки та техніка'),

('Іграшки з улюбленими персонажами'),

('Ігрові набори та сценки'),

('Намистини, бісер та браслети'),

('Набори для малювання'),

('3D ручки та пластик'),

('Ліплення: пластилін, глина, тісто'),

('Набори для наукових експериментів'),

('Дитячі телескопи та мікроскопи'),

('Дитячі намети, будиночки та тунелі'),

('Ігрові килимки та підлоги'),

('Дитячі кухні та побутова техніка'),

('Касові апарати та магазини'),

('Меблі для ляльок'),

('Набори для лікаря / перукаря'),

('Іграшки-антистреси (pop-it, squishy)'),

('STEM-набори'),

('Магнітні ігри та дошки');

ДОДАТОК Б

Код Програми

Київ-2025

Frontend

Папка lib(головна)

Файл main.dart

```
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import 'screens/login_screen.dart';
import 'screens/my_products_screen.dart';
import 'package:flutter_stripe/flutter_stripe.dart';
import 'screens/edit_product_screen.dart';
import 'package:flutter_dotenv/flutter_dotenv.dart';
import 'secrets.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  try {
  await dotenv.load();
  print('☑ .env loaded, GOOGLE_API_KEY: ${dotenv.env['GOOGLE_API_KEY']}');
  } catch (e) {
  print('❌ Failed to load .env: $e');
  }

  Stripe.publishableKey = Secrets.stripePublishableKey;
  await Firebase.initializeApp();
```

```
  await Stripe.instance.applySettings();
  print('🔑 API KEY: ${dotenv.env['GOOGLE_API_KEY']}');
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Toyshop App',
      theme: ThemeData(
        primarySwatch: Colors.deepPurple,
        scaffoldBackgroundColor: const Color(0xFFF3F0FF),
        appBarTheme: const AppBarTheme(
          backgroundColor: Colors.deepPurple,
          foregroundColor: Colors.white,
        ),
        elevatedButtonTheme: ElevatedButtonThemeData(
          style: ElevatedButton.styleFrom(
            backgroundColor: Colors.deepPurple,
            foregroundColor: Colors.white,
            shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(10)),
            padding: const EdgeInsets.symmetric(horizontal: 20, vertical: 14),
```

```
      textStyle: const TextStyle(fontSize: 16, fontWeight: FontWeight.bold),
    ),
  ),
  inputDecorationTheme: InputDecorationTheme(
    border: OutlineInputBorder(borderRadius: BorderRadius.circular(8)),
    focusedBorder: OutlineInputBorder(
      borderSide: const BorderSide(color: Colors.deepPurple),
      borderRadius: BorderRadius.circular(8),
    ),
  ),
),
home: const LoginScreen(),
routes: {
  '/my-products': (context) => const MyProductsScreen(),
},
onGenerateRoute: (settings) {
  if (settings.name == '/edit-product') {
    final args = settings.arguments as Map<String, dynamic>;
    return MaterialPageRoute(
      builder: (_) => const EditProductScreen(),
      settings: RouteSettings(arguments: args),
    );
  }
  return null;
},
```

```
    );
  }
}
```

Папка Screens (екрани)

Файл admin_product_detail_screen.dart

```
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';
import '../utils/shared_prefs.dart';

class AdminProductDetailScreen extends StatefulWidget {
  final Map<String, dynamic> product;

  const AdminProductDetailScreen({super.key, required this.product});

  @override
  State<AdminProductDetailScreen> createState() =>
      _AdminProductDetailScreenState();
}
```

```
class _AdminProductDetailScreenState extends State<AdminProductDetailScreen> {
  late String _status;
  late Map<String, dynamic> _previous;
  final List<String> _statusOptions = ['pending', 'active', 'inactive'];

  @override
  void initState() {
    super.initState();
    _status = widget.product['status'] ?? 'pending';

    try {
      _previous = jsonDecode(widget.product['previous_data'] ?? '{}');
    } catch (_) {
      _previous = {};
    }
  }

  Future<void> _updateStatus(String newStatus) async {
    final token = await SharedPrefs.getToken();
    if (token == null) return;

    final response = await http.put(
      Uri.parse(
        'http://10.0.2.2:8080/admin/products/${widget.product['id']}/status',
```

```
    ),
    headers: {
      'Authorization': 'Bearer $token',
      'Content-Type': 'application/json',
    },
    body: jsonEncode({'status': newStatus}),
  );

  if (response.statusCode == 200) {
    if (!mounted) return;
    ScaffoldMessenger.of(
      context,
    ).showSnackBar(const SnackBar(content: Text('Статус оновлено')));
    Navigator.pop(context, true); // ☑ Повертаємось до попереднього екрану
  } else {
    if (!mounted) return;
    ScaffoldMessenger.of(
      context,
    ).showSnackBar(SnackBar(content: Text('Помилка: ${response.body}')));
  }
}

Widget _buildField(String label, dynamic current, dynamic previous) {
  final hasChanged =
      previous != null && previous.toString() != current.toString();
```

```
  return Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Text(label, style: const TextStyle(fontWeight: FontWeight.bold)),
      if (hasChanged) ...[
        Text('Було: $previous', style: const TextStyle(color: Colors.red)),
        Text('Стало: $current', style: const TextStyle(color: Colors.green)),
      ] else
        Text(
          current?.toString() ?? '',
          style: const TextStyle(color: Colors.green),
        ),
      const SizedBox(height: 12),
    ],
  );
}

@override
Widget build(BuildContext context) {
  final product = widget.product;

  return Scaffold(
    appBar: AppBar(title: const Text('Деталі товару')),
    body: Padding(
```

```
padding: const EdgeInsets.all(16),
child: ListView(
  children: [
    if (product['image_url'] != null &&
        product['image_url'].toString().isNotEmpty)
      ClipRRect(
        borderRadius: BorderRadius.circular(12),
        child: Image.network(
          product['image_url'],
          height: 200,
          fit: BoxFit.cover,
        ),
      ),
    const SizedBox(height: 16),
    _buildField('Назва', product['name'], _previous['name']),
    _buildField(
      'Опис',
      product['description'],
      _previous['description'],
    ),
    _buildField('Ціна (грн)', product['price'], _previous['price']),
    _buildField(
      'Кількість',
      product['stock_quantity'],
      _previous['stock_quantity'],
```

```
),
_buildField('Локація', product['location'], _previous['location']),
_buildField(
  'Категорія ID',
  product['category_id'],
  _previous['category_id'],
),
_buildField('Автор ID', product['owner_id'], _previous['owner_id']),
const SizedBox(height: 12),
const Text('Змінити статус:', style: TextStyle(fontSize: 16)),
const SizedBox(height: 8),
DropdownButtonFormField<String>(
  value: _status,
  items:
      _statusOptions
          .map(
            (s) => DropdownMenuItem(
              value: s,
              child: Text(_mapStatusToUkr(s)),
            ),
          )
          .toList(),
  onChanged: (newStatus) {
    if (newStatus != null && newStatus != _status) {
      showDialog(
```

```
            context: context,
            builder:
                (ctx) => AlertDialog(
                  title: const Text('Підтвердження'),
                  content: Text(
                    'Змінити статус на "${_mapStatusToUkr(newStatus)}"?',
                  ),
                  actions: [
                    TextButton(
```

Файл admin_product_moderation_screen.dart

```
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';
import '../utils/shared_prefs.dart';
import 'admin_product_detail_screen.dart';

class AdminProductModerationScreen extends StatefulWidget {
  const AdminProductModerationScreen({super.key});

  @override
  State<AdminProductModerationScreen> createState() =>
      _AdminProductModerationScreenState();
}
```

```
class _AdminProductModerationScreenState
  extends State<AdminProductModerationScreen> {
 List<dynamic> _products = [];

 @override
 void initState() {
  super.initState();
  _loadProducts();
 }

 Future<void> _loadProducts() async {
  final token = await SharedPrefs.getToken();
  final response = await http.get(
   Uri.parse('http://10.0.2.2:8080/admin/products'),
   headers: {'Authorization': 'Bearer $token'},
  );

  if (response.statusCode == 200) {
   setState(() {
    _products = jsonDecode(response.body);
   });
  }
 }

 String _mapStatusToUkr(String status) {
```

```
  switch (status) {
    case 'active':
      return 'Активний';
    case 'inactive':
      return 'Неактивний';
    case 'pending':
    default:
      return 'Очікується';
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: const Text('Модерація товарів')),
    body:
        _products.isEmpty
            ? const Center(child: Text('Немає товарів для модерації'))
            : ListView.builder(
              itemCount: _products.length,
              itemBuilder: (context, index) {
                final product = _products[index];

                return GestureDetector(
                  onTap: () {
```

```
  Navigator.push(
    context,
    MaterialPageRoute(
      builder:
          (_) => AdminProductDetailScreen(product: product),
    ),
  ).then(
    (_) => _loadProducts(),
  ); // оновити після повернення
},
child: Card(
  margin: const EdgeInsets.symmetric(
    vertical: 8,
    horizontal: 16,
  ),
  child: ListTile(
    leading:
        product['image_url'] != null &&
                product['image_url'].toString().isNotEmpty
            ? ClipRRect(
                borderRadius: BorderRadius.circular(8),
                child: Image.network(
                  product['image_url'],
                  width: 50,
                  height: 50,
```

```
                    fit: BoxFit.cover,
                  ),
                )
                : const Icon(Icons.image_not_supported),
            title: Text(product['name'] ?? 'Без назви'),
            subtitle: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Text('Ціна: ${product['price']} грн'),
                Text(
                  'Статус: ${_mapStatusToUkr(product['status'] ?? '')}',
                ),
              ],
            ),
            trailing: const Icon(Icons.chevron_right),
          ),
        ),
      );
    },
  ),
 );
 }
}
```

Файл chat_screen.dart

```
import 'dart:convert';
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
import 'package:toyshop/utils/shared_prefs.dart';

class ChatScreen extends StatefulWidget {
  final int? receiverId;
  final int? productId;
  final String? threadId;

  const ChatScreen({
    super.key,
    required this.receiverId,
    required this.productId,
    required this.threadId,
  });

  @override
  State<ChatScreen> createState() => _ChatScreenState();
}

class _ChatScreenState extends State<ChatScreen> {
  List<dynamic> _messages = [];
```

```
final _controller = TextEditingController();
int? _currentUserId;
final _scrollController = ScrollController();

@override
void initState() {
 super.initState();
 _loadCurrentUser();
}

Future<void> _loadCurrentUser() async {
 final token = await SharedPrefs.getToken();
 final res = await http.get(
  Uri.parse('http://10.0.2.2:8080/me'),
  headers: {'Authorization': 'Bearer $token'},
 );
 if (res.statusCode == 200) {
  final user = jsonDecode(res.body);
  _currentUserId = user['id'];
  _loadThreadMessages(); // завантажуємо після отримання currentUserId
 }
}

Future<void> _loadThreadMessages() async {
 final token = await SharedPrefs.getToken();
```

```
  final res = await http.get(
    Uri.parse('http://10.0.2.2:8080/messages/thread/${widget.threadId}'),
    headers: {'Authorization': 'Bearer $token'},
  );

  if (res.statusCode == 200) {
    final data = jsonDecode(res.body);
    setState(() => _messages = data);
    _scrollDown();
  } else {
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(content: Text('Помилка при завантаженні чату')),
    );
  }
}

Future<void> _sendMessage() async {
  final token = await SharedPrefs.getToken();

  if (_currentUserId == widget.receiverId) {
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(
        content: Text('Ви не можете надіслати повідомлення самому собі'),
      ),
    );
```

```
    return;
  }

  final res = await http.post(
    Uri.parse('http://10.0.2.2:8080/messages'),
    headers: {
      'Authorization': 'Bearer $token',
      'Content-Type': 'application/json',
    },
    body: jsonEncode({
      'receiver_id': widget.receiverId,
      'content': _controller.text,
      'product_id': widget.productId,
    }),
  );

  if (res.statusCode == 201) {
    _controller.clear();
    _loadThreadMessages();
  } else {
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(content: Text('Помилка при надсиланні повідомлення')),
    );
  }
}
```

```
void _scrollDown() {
  WidgetsBinding.instance.addPostFrameCallback((_) {
    if (_scrollController.hasClients) {
      _scrollController.animateTo(
        0.0,
        duration: const Duration(milliseconds: 300),
        curve: Curves.easeOut,
      );
    }
  });
}

@override
void dispose() {
  _controller.dispose();
  _scrollController.dispose();
  super.dispose();
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: const Text('Чат')),
    body: Column(
```

```
children: [
  Expanded(
    child: ListView.builder(
      reverse: true, // Найновіші внизу
      controller: _scrollController,
      itemCount: _messages.length,
      itemBuilder: (context, index) {
        final msg = _messages[_messages.length - 1 - index];
        final isMine = msg['sender_id'] == _currentUserId;

        return Container(
          alignment:
              isMine ? Alignment.centerRight : Alignment.centerLeft,
          padding: const EdgeInsets.symmetric(
            horizontal: 12,
            vertical: 6,
          ),
          child: Container(
            padding: const EdgeInsets.symmetric(
              horizontal: 12,
              vertical: 8,
            ),
            decoration: BoxDecoration(
              color: isMine ? Colors.blue[100] : Colors.grey[300],
              borderRadius: BorderRadius.circular(10),
```

```
),
child: Column(
  crossAxisAlignment: CrossAxisAlignment.start,
  children: [
    Text(
      msg['content'],
      style: const TextStyle(fontSize: 16),
    ),
    const SizedBox(height: 4),
    Text(
      msg['created_at']?.substring(0, 16) ?? '',
      style: const TextStyle(
        fontSize: 10,
        color: Colors.black54,
      ),
    ),
  ],
),
```

Файл create_product_screen.dart

```
// Імпорти залишаються незмінними
import 'dart:io';
import 'package:flutter/material.dart';
import 'package:image_picker/image_picker.dart';
import 'package:http/http.dart' as http;
import 'package:firebase_storage/firebase_storage.dart';
import 'package:toyshop/utils/shared_prefs.dart';
import 'package:toyshop/screens/location_picker_screen.dart';
import 'dart:convert';

class CreateProductScreen extends StatefulWidget {
  const CreateProductScreen({super.key});

  @override
  State<CreateProductScreen> createState() => _CreateProductScreenState();
}

class _CreateProductScreenState extends State<CreateProductScreen> {
  final _nameController = TextEditingController();
  final _descController = TextEditingController();
  final _priceController = TextEditingController();
```

```
final _locationController = TextEditingController();
final _quantityController = TextEditingController();
String? _selectedCategory;
List<Map<String, dynamic>> _categories = [];
File? _imageFile;

@override
void initState() {
 super.initState();
 _fetchCategories();
}

Future<void> _fetchCategories() async {
 final response = await http.get(
  Uri.parse('http://10.0.2.2:8080/categories'),
 );
 if (response.statusCode == 200) {
  final List<dynamic> data = jsonDecode(response.body);
  setState(() {
   _categories =
       data.map((e) => {'id': e['id'], 'name': e['name']}).toList();
   if (_categories.isNotEmpty) {
    _selectedCategory = _categories.first['id'].toString();
   }
  });
```

```
  }
}

Future<void> _pickImage() async {
  final picker = ImagePicker();
  final pickedFile = await picker.pickImage(source: ImageSource.gallery);
  if (pickedFile != null) {
    setState(() => _imageFile = File(pickedFile.path));
  }
}

Future<void> _selectLocation() async {
  final result = await Navigator.push(
    context,
    MaterialPageRoute(builder: (_) => const LocationPickerScreen()),
  );
  if (result != null && result['address'] != null) {
    _locationController.text = result['address'];
  }
}

Future<void> _submitProduct() async {
  if (_nameController.text.trim().isEmpty ||
      _descController.text.trim().isEmpty ||
      _locationController.text.trim().isEmpty ||
```

```
    double.tryParse(_priceController.text) == null ||
    int.tryParse(_quantityController.text) == null ||
    double.parse(_priceController.text) <= 0 ||
    int.parse(_quantityController.text) <= 0) {
  ScaffoldMessenger.of(context).showSnackBar(
    const SnackBar(content: Text('Заповніть всі поля правильно')),
  );
  return;
}

final token = await SharedPrefs.getToken();
if (token == null) {
  print('🟤 Токен відсутній');
  return;
}

String imageUrl = '';
if (_imageFile != null) {
  imageUrl = await uploadImage(_imageFile!);
}

final Map<String, dynamic> body = {
  'name': _nameController.text,
  'description': _descController.text,
  'price': double.tryParse(_priceController.text),
```

```
    'stock_quantity': int.parse(_quantityController.text),
    'location': _locationController.text,
    'category_id': int.tryParse(_selectedCategory ?? '') ?? 0,
    'image_url': imageUrl,
  };

  final response = await http.post(
    Uri.parse('http://10.0.2.2:8080/products'),
    headers: {
      'Authorization': 'Bearer $token',
      'Content-Type': 'application/json',
    },
    body: jsonEncode(body),
  );

  if (response.statusCode == 201) {
    Navigator.pop(context, true);
  } else {
    print('🔴 Помилка: ${response.statusCode}');
    print('🔴 Відповідь: ${response.body}');
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(content: Text('Помилка при створенні товару')),
    );
  }
}
```

```
Future<String> uploadImage(File imageFile) async {
  final storageRef = FirebaseStorage.instance.ref();
  final fileRef = storageRef.child(
    'product_images/${DateTime.now().millisecondsSinceEpoch}.jpg',
  );
  await fileRef.putFile(
    imageFile,
    SettableMetadata(contentType: 'image/jpeg'),
  );
  return await fileRef.getDownloadURL();
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: const Color.fromARGB(255, 248, 244, 255),
    appBar: AppBar(
      title: const Text('Новий товар'),
      backgroundColor: Colors.deepPurple,
    ),
```

Файл delivery_screen.dart

```
import 'package:flutter/material.dart';
import 'send_tab.dart';
import 'receive_tab.dart';

class DeliveryScreen extends StatefulWidget {
  const DeliveryScreen({super.key});

  @override
  State<DeliveryScreen> createState() => _DeliveryScreenState();
}

class _DeliveryScreenState extends State<DeliveryScreen> {
  int _selectedIndex = 0;

  final List<Widget> _pages = const [SendTab(), ReceiveTab()];

  void _onItemTapped(int index) {
    setState(() {
      _selectedIndex = index;
```

```
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Логістика')),
      body: _pages[_selectedIndex],
      bottomNavigationBar: BottomNavigationBar(
        currentIndex: _selectedIndex,
        selectedItemColor: Colors.deepPurple,
        unselectedItemColor: Colors.grey,
        backgroundColor: Colors.white,
        items: const [
          BottomNavigationBarItem(icon: Icon(Icons.send), label: 'Відправка'),
          BottomNavigationBarItem(
            icon: Icon(Icons.local_shipping),
            label: 'Доставка',
          ),
        ],
        onTap: _onItemTapped,
      ),
    );
  }
}
```

Файл edit_product_screen.dart

```
import 'dart:convert';

import 'dart:io';

import 'package:flutter/material.dart';

import 'package:image_picker/image_picker.dart';

import 'package:http/http.dart' as http;

import 'package:firebase_storage/firebase_storage.dart';

import 'package:toyshop/utils/shared_prefs.dart';

import 'package:toyshop/screens/location_picker_screen.dart';


class EditProductScreen extends StatefulWidget {

  const EditProductScreen({super.key});


  @override

  State<EditProductScreen> createState() => _EditProductScreenState();

}


class _EditProductScreenState extends State<EditProductScreen> {

  final _nameController = TextEditingController();

  final _descController = TextEditingController();

  final _priceController = TextEditingController();

  final _locationController = TextEditingController();
```

```
final _quantityController = TextEditingController();


bool _wasUpdated = false;


String? _selectedCategory;
String? _status;
List<Map<String, dynamic>> _categories = [];
File? _imageFile;
String? _initialImageUrl;
int? _productId;


late Map<String, dynamic> _initialValues;


@override
void initState() {
 super.initState();
 WidgetsBinding.instance.addPostFrameCallback((_) {
  if (!mounted) return;
  final args =
      ModalRoute.of(context)?.settings.arguments as Map<String, dynamic>;


  setState(() {
   _productId = args['id'];
   _nameController.text = args['name'];
   _descController.text = args['description'] ?? '';
```

```
      _priceController.text = args['price'].toString();
      _locationController.text = args['location'] ?? '';
      _quantityController.text = args['stock_quantity']?.toString() ?? '1';
      _selectedCategory = args['category_id'].toString();
      _status = args['status'];
      _initialImageUrl = args['image_url'];

      _initialValues = {
        'name': _nameController.text,
        'description': _descController.text,
        'price': _priceController.text,
        'location': _locationController.text,
        'stock_quantity': _quantityController.text,
        'category_id': _selectedCategory,
        'image_url': _initialImageUrl,
        'status': _status,
      };
    });

    _fetchCategories();
  });
}

Future<void> _fetchCategories() async {
  final response = await http.get(
```

```
    Uri.parse('http://10.0.2.2:8080/categories'),
  );
  if (response.statusCode == 200) {
    final List<dynamic> data = jsonDecode(response.body);
    setState(() {
      _categories =
          data.map((e) => {'id': e['id'], 'name': e['name']}).toList();
    });
  }
}

Future<String> uploadImage(File imageFile) async {
  final storageRef = FirebaseStorage.instance.ref();
  final imagesRef = storageRef.child(
    'product_images/${DateTime.now().millisecondsSinceEpoch}.jpg',
  );
  final metadata = SettableMetadata(contentType: 'image/jpeg');
  final uploadTask = imagesRef.putFile(imageFile, metadata);
  final snapshot = await uploadTask;
  return await snapshot.ref.getDownloadURL();
}

Future<void> _pickImage() async {
  final picker = ImagePicker();
  final pickedFile = await picker.pickImage(source: ImageSource.gallery);
```

```
  if (pickedFile != null) {
    setState(() => _imageFile = File(pickedFile.path));
  }
}

Future<void> _selectLocation() async {
  final result = await Navigator.push(
    context,
    MaterialPageRoute(builder: (_) => const LocationPickerScreen()),
  );
  if (result != null && result['address'] != null) {
    _locationController.text = result['address'];
  }
}

Future<void> _submitUpdate() async {
  if (_nameController.text.trim().isEmpty ||
      _descController.text.trim().isEmpty ||
      _locationController.text.trim().isEmpty ||
      double.tryParse(_priceController.text) == null ||
      int.tryParse(_quantityController.text) == null) {
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(content: Text('Будь ласка, заповніть всі поля')),
    );
    return;
```

```
}

final token = await SharedPrefs.getToken();
if (token == null || _productId == null) return;

String imageUrl = _initialImageUrl ?? '';
if (_imageFile != null) {
  imageUrl = await uploadImage(_imageFile!);
}

final Map<String, dynamic> newValues = {
  'name': _nameController.text,
  'description': _descController.text,
  'price': _priceController.text,
  'location': _locationController.text,
  'stock_quantity': _quantityController.text,
  'category_id': _selectedCategory,
  'image_url': imageUrl,
};

bool hasChanges = false;
for (final key in newValues.keys) {
  if (newValues[key]?.toString() != _initialValues[key]?.toString()) {
    hasChanges = true;
    break;
```

```
    }
  }

  String finalStatus = _status ?? 'active';
  if (_status == 'active' && hasChanges) {
    finalStatus = 'pending';
  }
```

Backend

Папка controllers

Файл category_controller.go

```
package controllers

import (
  "net/http"
  "toyshop/database"
  "toyshop/models"

  "github.com/gin-gonic/gin"
)

func GetCategories(c *gin.Context) {
  var categories []models.Category
  result := database.DB.Find(&categories)
```

```
    if result.Error != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": result.Error.Error()})
        return
    }

    c.JSON(http.StatusOK, categories)
}

func CreateCategory(c *gin.Context) {
    var category models.Category

    if err := c.ShouldBindJSON(&category); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
        return
    }

    result := database.DB.Create(&category)

    if result.Error != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": result.Error.Error()})
        return
    }

    c.JSON(http.StatusCreated, category)
}
```

Файл order_controller.go

```
package controllers

import (
```

```
    "fmt"
    "net/http"
    "toyshop/database"
    "toyshop/models"

    "github.com/gin-gonic/gin"
)

func GetMyOrders(c *gin.Context) {
    userIDRaw, exists := c.Get("user_id")
    if !exists {
        c.JSON(http.StatusUnauthorized, gin.H{"error": "Не авторизовано"})
        return
    }
    userID := userIDRaw.(uint)

    var orders []models.Order
    err := database.DB.
        Preload("Items.Product").
        Preload("User").
        Where("user_id = ?", userID).
        Find(&orders).Error

    if err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
```

```
        return
    }

    c.JSON(http.StatusOK, orders)
}

type OrderRequest struct {
    ShippingAddress     string `json:"shipping_address"`
    RecipientFirstName  string `json:"recipient_first_name"`
    RecipientLastName   string `json:"recipient_last_name"`
    RecipientMiddleName string `json:"recipient_middle_name"`
    PaymentType         string `json:"payment_type"`
    Items               []struct {
        ProductID uint `json:"product_id"`
        Quantity  int  `json:"quantity"`
    } `json:"items"`
}

func CreateOrder(c *gin.Context) {
    userIDRaw, exists := c.Get("user_id")
    if !exists {
        c.JSON(http.StatusUnauthorized, gin.H{"error": "Не авторизовано"})
        return
    }
    userID := userIDRaw.(uint)
```

```
var request OrderRequest
if err := c.ShouldBindJSON(&request); err != nil {
    c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
    return
}

var totalPrice float64
var orderItems []models.OrderItem

tx := database.DB.Begin()

for _, item := range request.Items {
    var product models.Product
    if err := tx.First(&product, item.ProductID).Error; err != nil {
        tx.Rollback()
        c.JSON(http.StatusNotFound, gin.H{"error": "Товар не знайдено"})
        return
    }

    if product.StockQuantity < item.Quantity {
        tx.Rollback()
        c.JSON(http.StatusBadRequest, gin.H{
            "error": fmt.Sprintf("Недостатньо товару '%s' на складі", product.Name),
        })
```

```
            return
        }

        product.StockQuantity -= item.Quantity
        if err := tx.Save(&product).Error; err != nil {
            tx.Rollback()
            c.JSON(http.StatusInternalServerError, gin.H{"error": "Помилка при оновленні кількості товару"})
            return
        }

        totalPrice += product.Price * float64(item.Quantity)
        orderItems = append(orderItems, models.OrderItem{
            ProductID: item.ProductID,
            Quantity:  item.Quantity,
            UnitPrice: product.Price,
        })
    }

    order := models.Order{
        UserID:             userID,
        ShippingAddress:    request.ShippingAddress,
        RecipientFirstName:  request.RecipientFirstName,
        RecipientLastName:   request.RecipientLastName,
        RecipientMiddleName: request.RecipientMiddleName,
```

```
        PaymentType:        request.PaymentType,
        PaymentStatus:      "неоплачений",
        Status:             "в обробці",
        TotalPrice:         totalPrice,
    }

    if err := tx.Create(&order).Error; err != nil {
        tx.Rollback()
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
        return
    }

    for i := range orderItems {
        orderItems[i].OrderID = order.ID
        if err := tx.Create(&orderItems[i]).Error; err != nil {
            tx.Rollback()
            c.JSON(http.StatusInternalServerError, gin.H{"error": "Помилка при створенні позицій
замовлення"})
            return
        }
    }
    order.Items = orderItems

    tx.Commit()
```

```
    c.JSON(http.StatusCreated, gin.H{
        "message":     "Замовлення створено",
        "order_id":    order.ID,
        "total_price": totalPrice,
    })
}

func CancelOrder(c *gin.Context) {
    orderID := c.Param("id")
    userID := c.MustGet("user_id").(uint)

    var order models.Order
    if err := database.DB.
        Preload("Items.Product").
        Preload("User").
        First(&order, orderID).Error; err != nil {

        c.JSON(http.StatusNotFound, gin.H{"error": "Замовлення не знайдено"})
        return
    }

    if order.Status == "скасований" {
        c.JSON(http.StatusBadRequest, gin.H{"error": "Замовлення вже скасовано"})
        return
    }
```

```
var user models.User
if err := database.DB.First(&user, userID).Error; err != nil {
    c.JSON(http.StatusInternalServerError, gin.H{"error": "Користувача не знайдено"})
    return
}

// Повернення коштів
if order.PaymentStatus == "оплачено" {
    user.Balance += order.TotalPrice
}

// Повернення кількості товару
for _, item := range order.Items {
    item.Product.StockQuantity += item.Quantity
    if err := database.DB.Save(&item.Product).Error; err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": "Помилка повернення товару"})
```

Файл product_controller.go

```
package controllers

import (
    "net/http"
    "toyshop/database"
    "toyshop/models"

    "github.com/gin-gonic/gin"
)

func GetProducts(c *gin.Context) {
    var products []models.Product
    result := database.DB.Find(&products)

    if result.Error != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": result.Error.Error()})
        return
    }

    c.JSON(http.StatusOK, products)
```

```
}


func GetActiveProducts(c *gin.Context) {
    var products []models.Product
    if err := database.DB.Where("_status = ?", "active").Find(&products).Error; err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": "Не вдалося завантажити активні товари"})
        return
    }
    c.JSON(http.StatusOK, products)
}


func GetMyProducts(c *gin.Context) {
    userID := c.MustGet("user_id").(uint)

    var products []models.Product
    if err := database.DB.Where("owner_id = ?", userID).Find(&products).Error; err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": "Не вдалося завантажити товари"})
        return
```

```
    }


    c.JSON(http.StatusOK, products)
}


func CreateProduct(c *gin.Context) {


    userID, exists := c.Get("user_id")
    if !exists {
        c.JSON(http.StatusUnauthorized, gin.H{"error": "Не авторизовано"})
        return
    }


    var product models.Product
    product.Status = "pending"
    product.PreviousData = "{}" // порожній об’єкт JSON — валідне значення


    if err := c.ShouldBindJSON(&product); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
```

```
        return
    }

    product.OwnerID = userID.(uint)

    if err := database.DB.Create(&product).Error; err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
        return
    }

    c.JSON(http.StatusCreated, product)
}

func UpdateProduct(c *gin.Context) {
    id := c.Param("id")

    var product models.Product
    if err := database.DB.First(&product, id).Error; err != nil {
        c.JSON(http.StatusNotFound, gin.H{"error": "Товар не знайдено"})
```

```
    return
}


// Створюємо мапу попередніх значень
prevMap := map[string]interface{}{
    "id":             product.ID,
    "name":           product.Name,
    "description":    product.Description,
    "price":          product.Price,
    "image_url":      product.ImageURL,
    "location":       product.Location,
    "stock_quantity": product.StockQuantity,
    "category_id":    product.CategoryID,
    "owner_id":       product.OwnerID,
    "created_at":     product.CreatedAt.Format("2006-01-02T15:04:05Z"),
    "status":         product.Status,
}


// Отримуємо нові дані
```

```
var input map[string]interface{}
if err := c.ShouldBindJSON(&input); err != nil {
    c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
    return
}

// Мапінг ключів на назви колонок у базі
if val, ok := input["name"]; ok {
    input["_name"] = val
    delete(input, "name")
}
if val, ok := input["description"]; ok {
    input["_description"] = val
    delete(input, "description")
}
if val, ok := input["location"]; ok {
    input["_location"] = val
    delete(input, "location")
}
```

```
    if val, ok := input["status"]; ok {
        input["_status"] = val
        delete(input, "status")
    }

    // Додаємо попередні дані як JSON
    input["previous_data"] = prevMap

    // Оновлюємо запис
    if err := database.DB.Model(&product).Updates(input).Error; err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
        return
    }

    c.JSON(http.StatusOK, product)
}

func DeleteProduct(c *gin.Context) {
    id := c.Param("id")
```

```
    var product models.Product
    if err := database.DB.First(&product, id).Error; err != nil {
        c.JSON(http.StatusNotFound, gin.H{"error": "Товар не знайдено"})
        return
    }

    if err := database.DB.Delete(&product).Error; err != nil {
        c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})
        return
    }

    c.JSON(http.StatusOK, gin.H{"message": "Товар видалено"})
}

func GetAllProducts(c *gin.Context) {
    var products []models.Product
    if err := database.DB.Find(&products).Error; err != nil {
        c.JSON(500, gin.H{"error": "Не вдалося завантажити продукти"})
```

```
        return
    }
    c.JSON(200, products)
}

func UpdateProductStatus(c *gin.Context) {
    id := c.Param("id")

    // Приймаємо JSON з ключем "status"
    var body struct {
        Status string `json:"status"`
    }
    if err := c.ShouldBindJSON(&body); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": "Некоректні дані"})
        return
    }

    // Валідація статусу (тільки дозволені значення)
    allowedStatuses := map[string]bool{
```

```
    "active":   true,
    "inactive": true,
    "pending":  true,
}
if !allowedStatuses[body.Status] {
    c.JSON(http.StatusBadRequest, gin.
```

ДОДАТОК В

Діаграми

Київ-2025

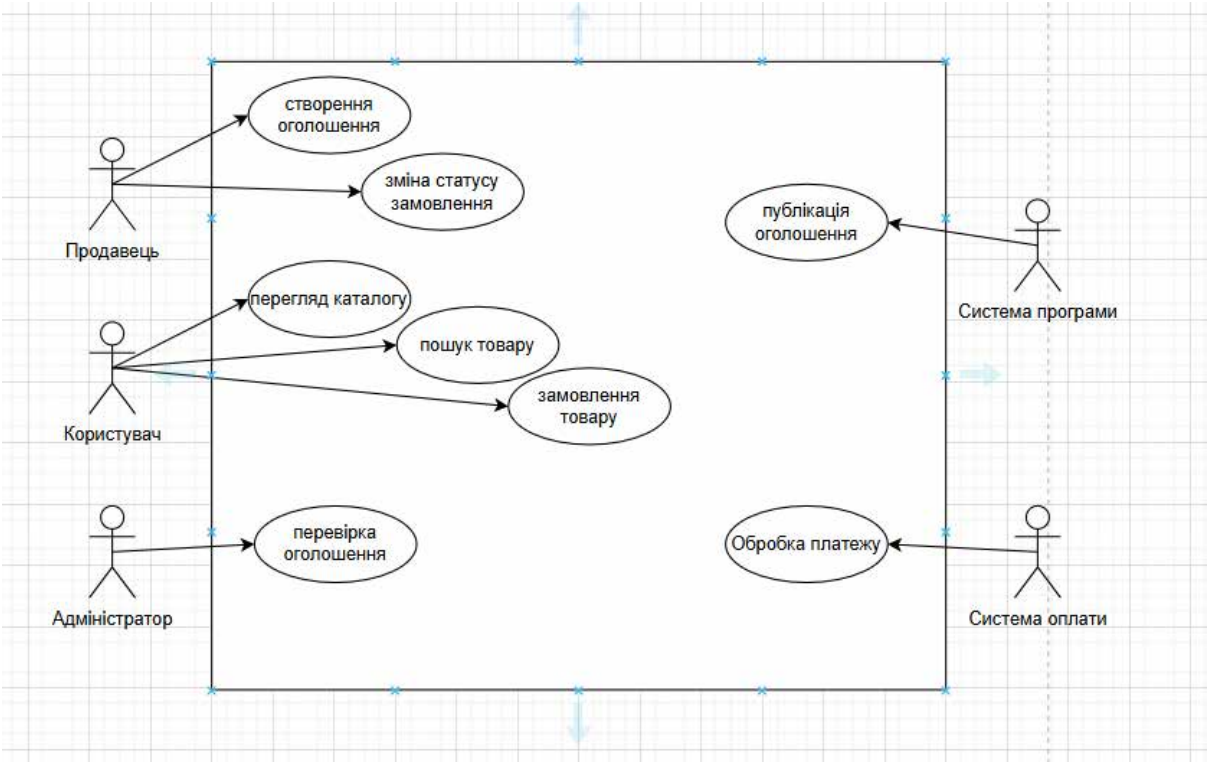


Рис. 1.1– Діаграма прецедентів (use case diagram)

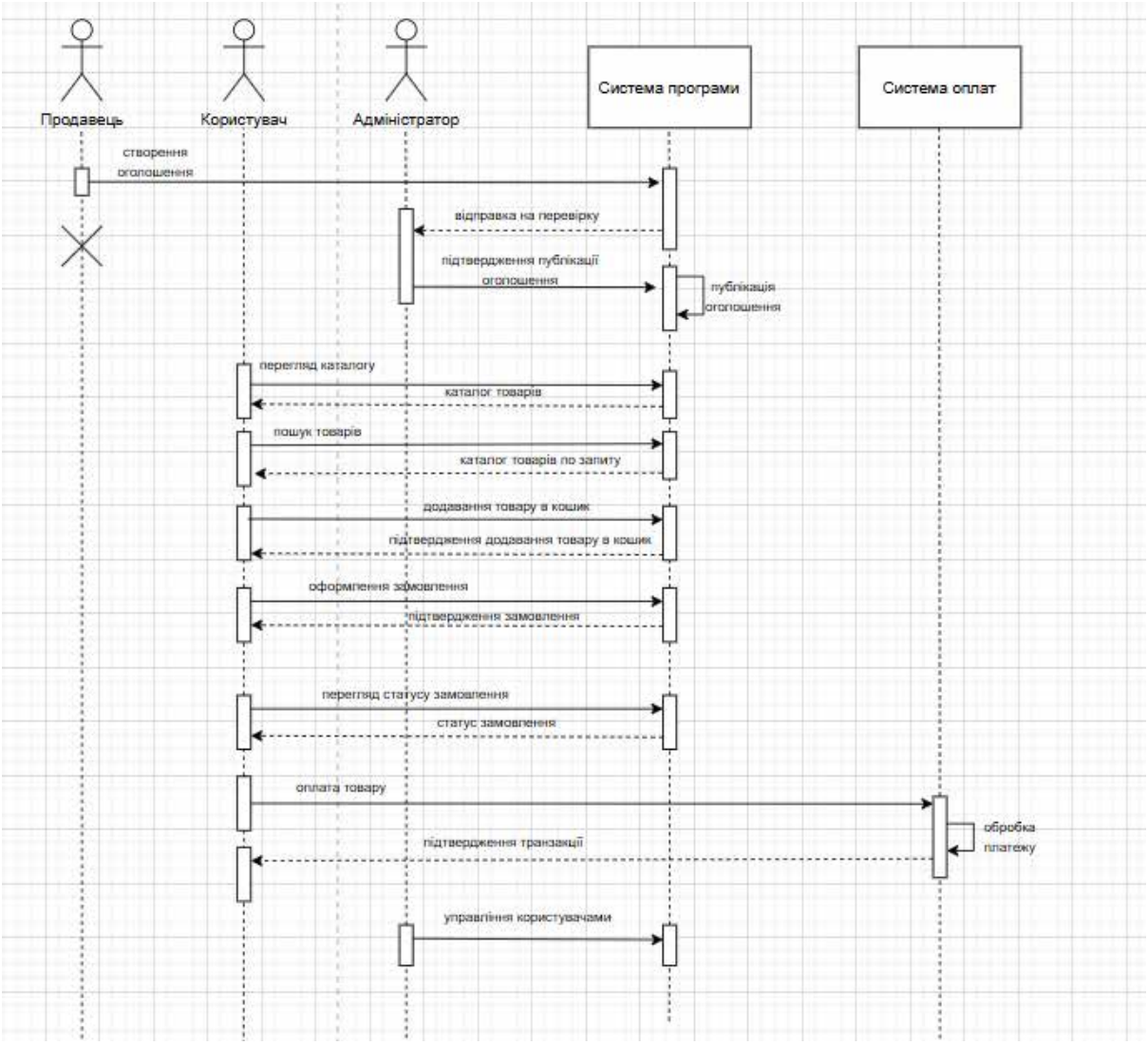


Рис. 1.2 – Діаграма послідовності (sequence diagram)

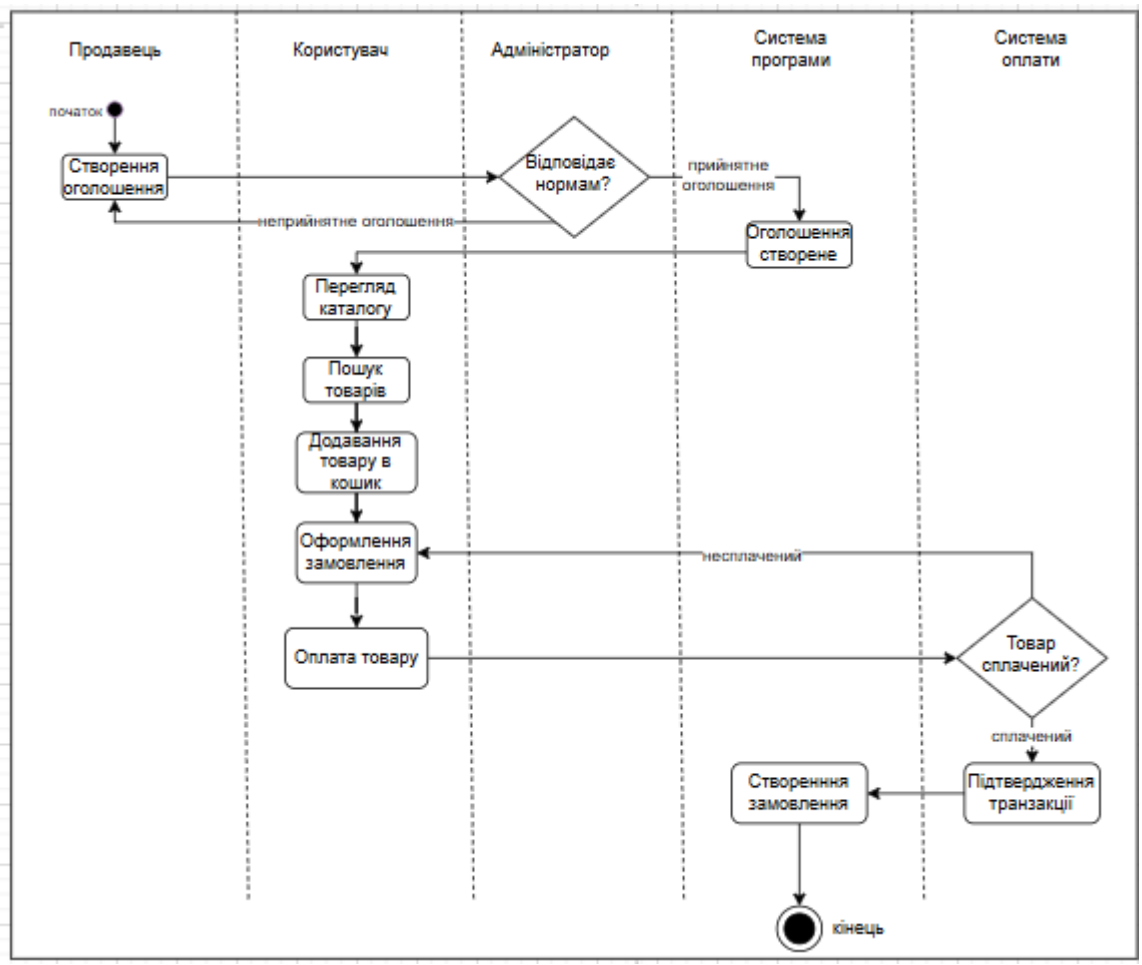


Рис. 1.3 – Діаграма активності (activity diagram)

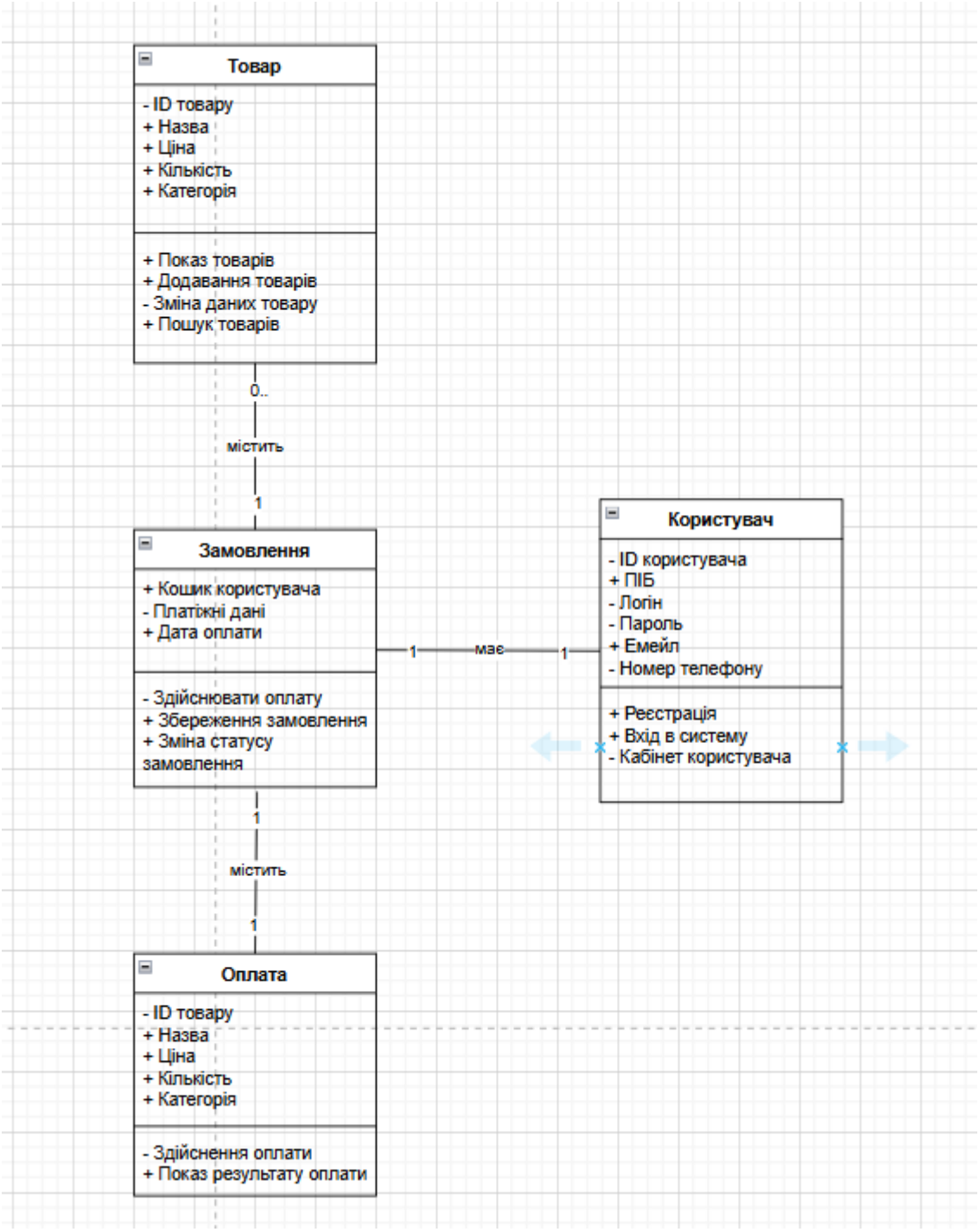
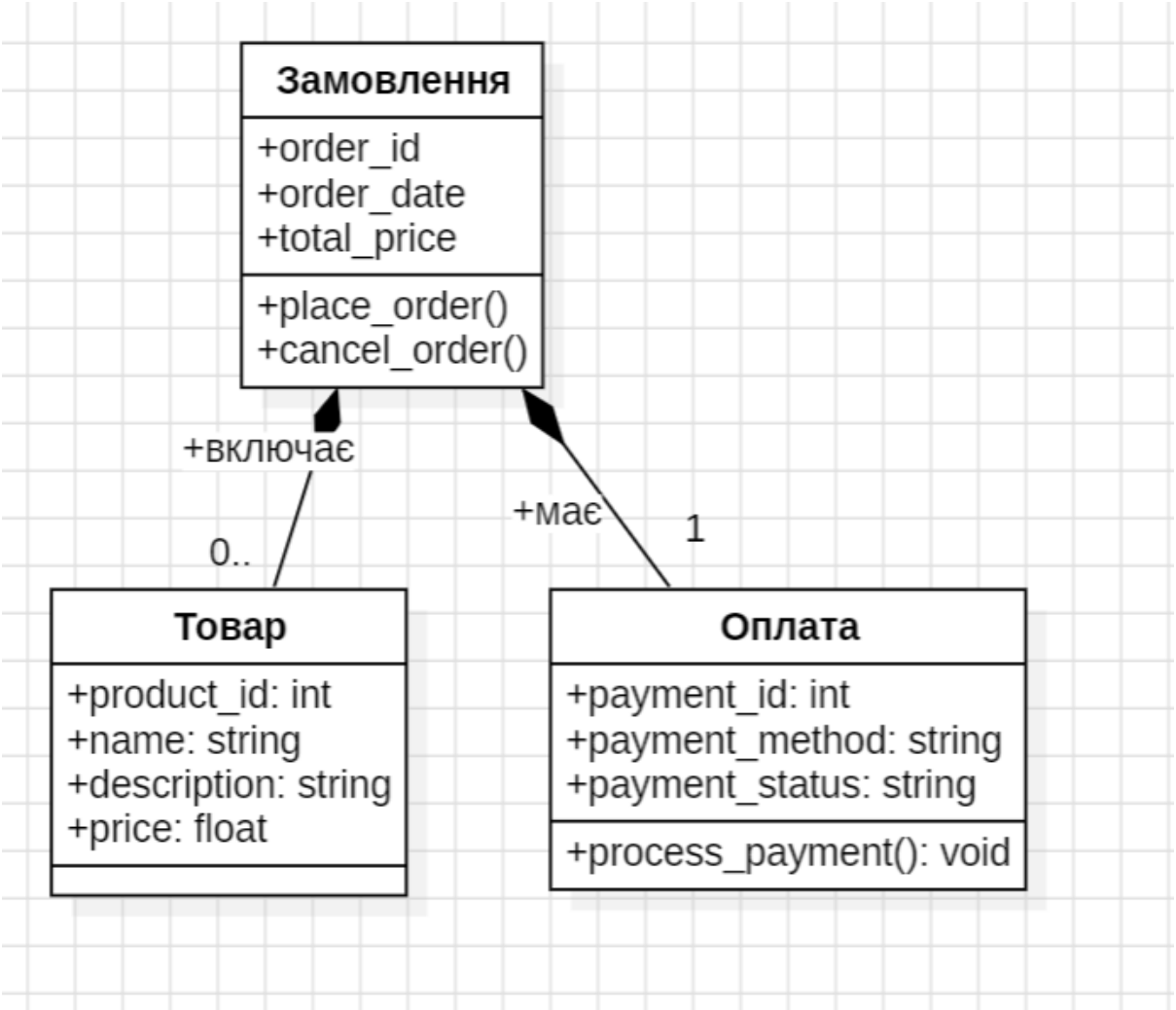


Рис. 4 – Діаграма класів (Class diagram)



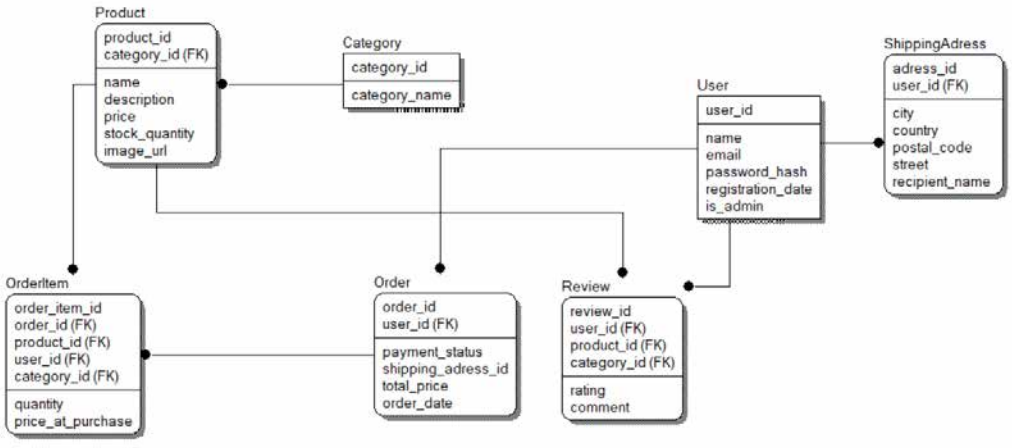
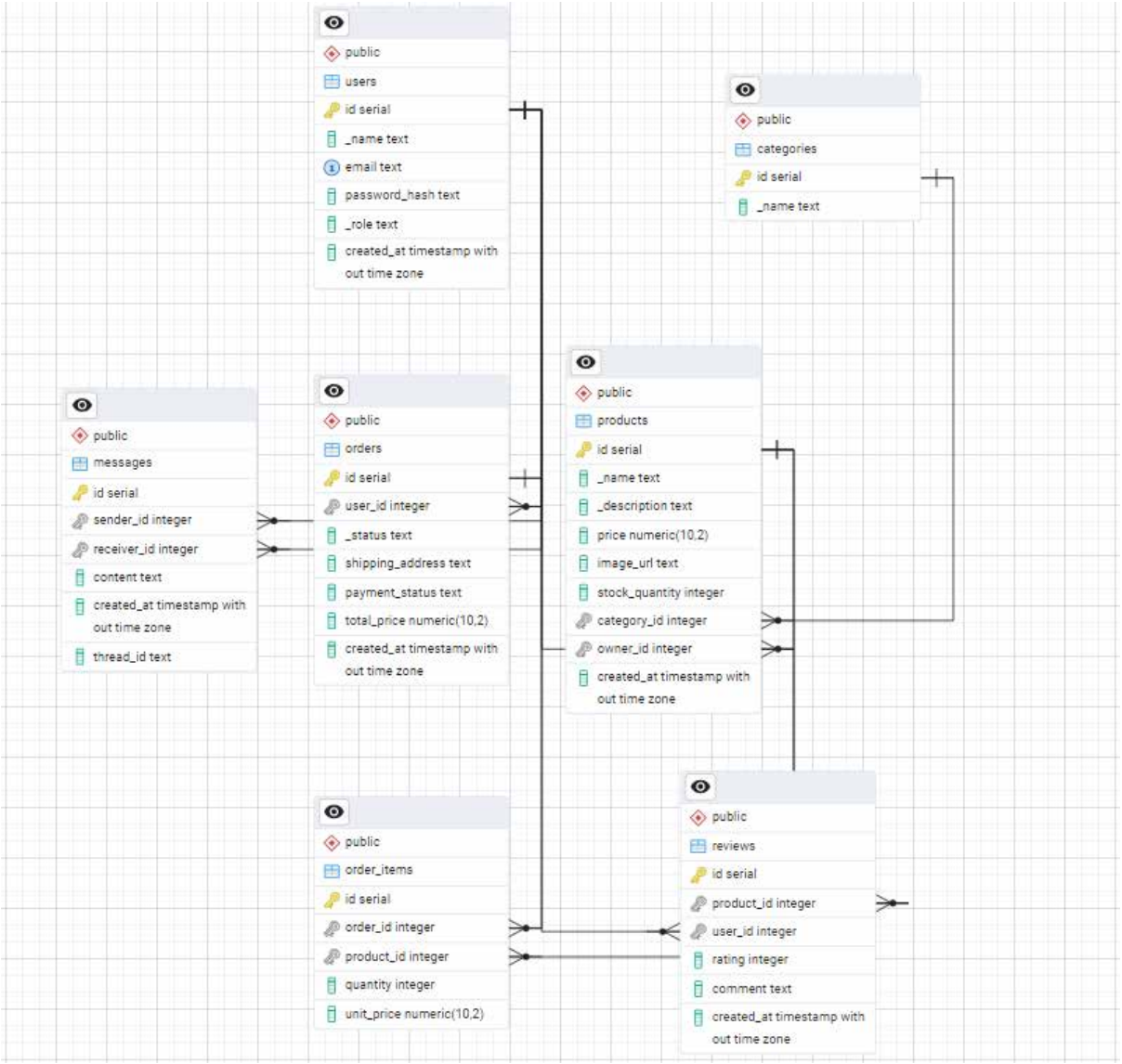


Рис. 3 – ER діаграма



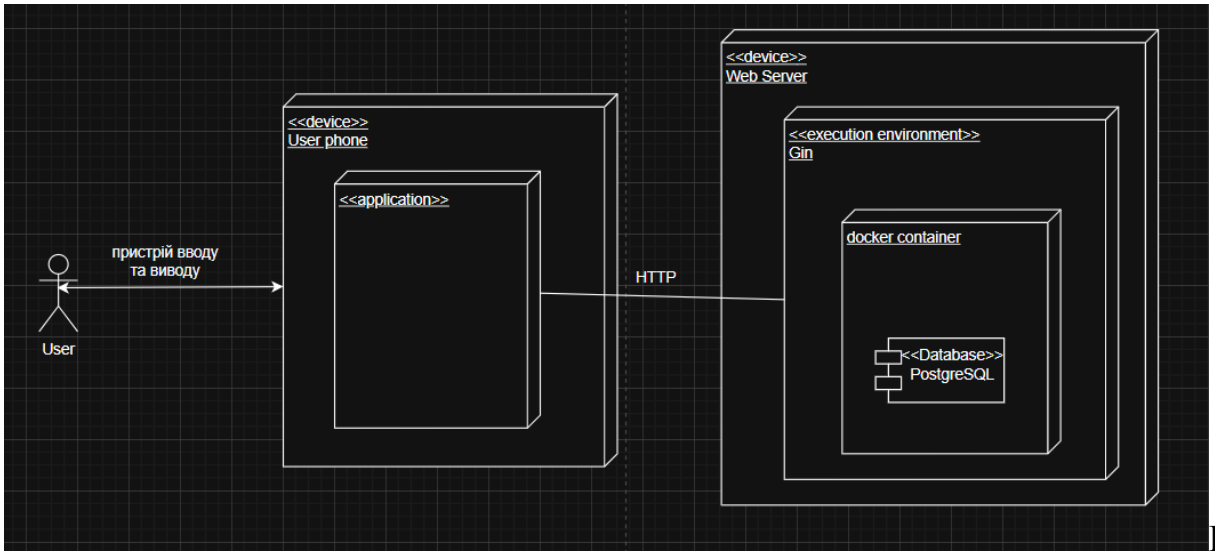


Рис. 12 – Діаграма розгортання (deployment diagram)