**Міністерство освіти і науки України**
**Тернопільський національний технічний університет імені Івана Пулюя**
**Факультет комп'ютерно-інформаційних систем і програмної інженерії**
**Кафедра програмної інженерії**

# КВАЛІФІКАЦІЙНА РОБОТА

**на здобуття освітнього ступеня**

**бакалавр**

**на тему: Проєктування та розробка вебзастосунку для електронної комерції «Krydix» з використанням Node.js**

Виконав(ла): студент(ка) IV курсу, групи СПс-41 спеціальності 121 Інженерія програмного забезпечення

| | | |
|---|---|---|
| | (підпис) | Петришин Я. А. (прізвище та ініціали) |
| Керівник | (підпис) | Мудрик І. Я. (прізвище та ініціали) |
| Нормоконтроль | (підпис) | Стоянов Ю.М. (прізвище та ініціали) |
| Завідувач кафедри | (підпис) | Петрик М.Р. (прізвище та ініціали) |
| Рецензент | (підпис) | (прізвище та ініціали) |

Тернопіль
2026

Міністерство освіти і науки України
**Тернопільський національний технічний університет імені Івана Пулюя**

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри

(підпис) (прізвище та ініціали)
« » 20___ р.

# З А В Д А Н Н Я
## НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавра
(назва освітнього ступеня)
за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
студенту Петришину Ярославу Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Проєктування та розробка вебзастосунку для електронної комерції «Krydix» з використанням Node.js

Керівник роботи Мудрик Іван Ярославович, доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом ректора від «___» _____ 20___ року № _____
2. Термін подання студентом завершеної роботи
3. Вихідні дані до роботи Предметна область, завдання, вимоги та специфікація, програмне рішення, методичні вказівки

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступна частина
Аналіз предметної області та теоретичних основ
Визначення методики реалізації моделі
Реалізація моделі
Визначення основних аспектів охорони праці та безпеки життєдіяльності
Висновки роботи
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Слайди презентації та діаграми процесів

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Безпека життєдіяльності, | Мариненко С. Ю. канд. техн. наук, | | |
| основи охорони праці | доцент | | |
| Нормоконтроль | Стоянов Ю.М. к.т.н., доц. каф. ПІ | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання _____

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Отримання завдання | | виконано |
| 2 | Аналіз предметної галузі електронної комерції та огляд існуючих маркетплейс-платформ | | виконано |
| 3 | Формування функціональних і нефункціональних вимог | | виконано |
| 4 | Розробка UML-діаграми прецедентів та опис ключових сценаріїв використання | | виконано |
| 5 | Проєктування архітектури клієнт-серверного застосунку та вибір технологічного стеку | | виконано |
| 6 | Проєктування реляційної схеми бази даних PostgreSQL та побудова ER-діаграми | | виконано |
| 7 | Розробка модульної GraphQL-схеми (Apollo Server) та шару репозиторіїв над Prisma | | виконано |
| 8 | Реалізація серверних модулів: автентифікація, каталог, checkout, fee snapshots, виплати, модерація | | виконано |
| 9 | Реалізація клієнтського SPA на React + MUI: маршрутизація, базова UI-бібліотека, ключові сторінки | | виконано |
| 10 | Інтеграція real-time комунікації (Socket.IO) та підсистеми сповіщень | | виконано |
| 11 | Підтримка двомовності інтерфейсу (i18next, EN + UK) та інтеграція Cloudinary | | виконано |
| 12 | Розробка Jest unit/integration тестів і компонентів | | виконано |
| 13 | Розробка Cypress E2E тестів критичних сценаріїв | | виконано |
| 14 | Налаштування CI/CD у GitHub Actions та розгортання | | виконано |
| 15 | Верифікація відповідності функціональним і нефункціональним вимогам | | виконано |
| 16 | Оформлення пояснювальної записки та підготовка демонстраційних матеріалів | | виконано |
| | | | |
| | | | |

Студент _____ Петришин Ярослав
(підпис) (прізвище та ініціали)

Керівник роботи _____ Мудрик І.Я.
(підпис) (прізвище та ініціали)

# АНОТАЦІЯ

Проєктування та розробка багатомовного маркетплейсу «Krydix» з використанням технологій Node.js та React // Кваліфікаційна робота освітнього рівня «Бакалавр» // Петришин Ярослав // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СПс-41 // Тернопіль, 2026 // С. 71, рис. – 22, табл. – 4, кресл. – 0, додат. – 6, бібліогр. – 36.

*Ключові слова:* маркетплейс, електронна комерція, Node.js, React, GraphQL, PostgreSQL, Prisma ORM, JWT, Cloudinary, Socket.IO, верифікація продавця, fee snapshots.

Кваліфікаційна робота присвячена проєктуванню та розробці багатомовного маркетплейсу «Krydix».

У першому розділі проведено аналіз предметної області, сформульовано мету і завдання, визначено п'ять ролей акторів системи та описано ключові варіанти використання.

У другому розділі обрано інкрементальну модель розробки, спроєктовано клієнт-серверну архітектуру та схему бази даних з понад 30 моделей, обґрунтовано технологічний стек і описано реалізацію модулів автентифікації, каталогу, checkout з fee snapshots, модерації, виплат продавцям та real-time чату.

У третьому розділі описано тестування модулів (Jest та Cypress), визначено вимоги до розгортання на Render і Vercel та проведено верифікацію відповідності встановленим вимогам.

У четвертому розділі розглянуто питання безпеки життєдіяльності та основи охорони праці.

Об'єкт дослідження: процеси проєктування багатомовних маркетплейс-платформ з управлінням ролями та фінансовими потоками.

Предмет дослідження: методи та технології розробки маркетплейс-платформи «Krydix» з використанням Node.js, React, GraphQL та PostgreSQL.

# ABSTRACT

Design and Development of the Multilingual Marketplace «Krydix» Using Node.js and React Technologies // Qualification work of the educational level «Bachelor» // Petryshyn Yaroslav // Ternopil Ivan Puluj National Technical University, Computer and Information Systems and Software Engineering Faculty, Software Engineering Department, group SPs-41 // Ternopil, 2026 // P. 71, fig. – 22, tabl. – 4, draw. – 0, annexes. – 6, references. – 36.

*Keywords:* marketplace, e-commerce, Node.js, React, GraphQL, PostgreSQL, Prisma ORM, JWT, Cloudinary, Socket.IO, seller verification, fee snapshots.

The qualification work is devoted to the design and development of the multilingual marketplace «Krydix».

The first chapter analyses the subject area, formulates the purpose and objectives of the development, defines five system actor roles and describes the key use cases.

The second chapter presents the selected incremental development model, designs the client-server architecture and the database schema with more than 30 models, justifies the technology stack and describes the implementation of the authentication, catalog, checkout with fee snapshots, moderation, seller payouts and real-time chat modules.

The third chapter describes the testing of system modules (Jest and Cypress), defines the deployment requirements on Render and Vercel, and verifies the compliance of the platform with the established requirements.

The fourth chapter examines the issues of life safety and fundamentals of labor protection.

Object of research: the processes of designing multilingual marketplace platforms with role management and financial flows.

Subject of research: methods and technologies for developing the marketplace platform «Krydix» using Node.js, React, GraphQL and PostgreSQL.

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

MVP – Minimum Viable Product

GraphQL – Graph Query Language

SDL – Schema Definition Language

ORM – Object-Relational Mapper

JWT – JSON Web Token

SMTP – Simple Mail Transfer Protocol

HTTP – HyperText Transfer Protocol

HTTPS – HyperText Transfer Protocol Secure

CORS – Cross-Origin Resource Sharing

TLS – Transport Layer Security

WS – WebSocket

DNS – Domain Name System

API – Application Programming Interface

URL – Uniform Resource Locator

MIME – Multipurpose Internet Mail Extensions

JSON – JavaScript Object Notation

ESM – ECMAScript Modules

HMR – Hot Module Replacement

SPA – Single Page Application

UI – User Interface

UX – User Experience

MUI – Material UI

SDK – Software Development Kit

npm – Node Package Manager

CI/CD – Continuous Integration/Continuous Deployment

DevOps – Development and Operations

E2E – End-to-End

**ЗМІСТ**

# ВСТУП

Стрімкий розвиток цифрової економіки та поширення інтернет-технологій призвели до фундаментальних змін у способах здійснення комерційних операцій. Традиційна роздрібна торгівля поступово поступається місцем електронним торговим платформам, які забезпечують безпосередній зв'язок між продавцями та покупцями незалежно від їх географічного розташування. Відповідно до звітів OECD та Statista, обсяг світового ринку електронної комерції у 2024 році перевищив 6 трильйонів доларів США і продовжує зростати [1,2]. За даними Cases Media, ринок e-commerce в Україні демонструє стійке щорічне зростання, а кількість онлайн-покупців збільшується [3]. Водночас існуючі рішення часто не відповідають вимогам щодо безпеки транзакцій, довіри між учасниками ринку та структурованого управління каталогом в умовах українського законодавства.

Саме в цьому контексті постала потреба в розробці сучасного багатомовного маркетплейсу Krydix – платформи, орієнтованої на B2C-сегмент із архітектурою, що допускає масштабування в напрямку B2B. Актуальність теми визначається кількома чинниками. По-перше, попит на локальні платформи, адаптовані до вимог українського ринку: підтримка гривні, ЄДРПОУ, місцевих способів доставки та двомовного інтерфейсу, яких міжнародні платформи (Amazon, eBay, Etsy) не забезпечують у повній мірі. По-друге, зростання вимог до верифікації продавців та модерації контенту: покупці довіряють платформам, де кожен продавець перевірений, а товари та відгуки проходять контроль. По-третє, технічна складність побудови маркетплейсів із GraphQL-архітектурою, real-time комунікацією та прозорими фінансовими потоками робить цю тему актуальною з науково-практичної точки зору.

Метою кваліфікаційної роботи є розробка багатомовного маркетплейсу Krydix – веб-застосунку, що забезпечує безпечну структуровану онлайн-торгівлю між покупцями та верифікованими бізнес-продавцями.

Для досягнення мети вирішено такі завдання: проведено аналіз предметної галузі електронної комерції та сформульовано вимоги до системи; визначено ролі

учасників та ключові сценарії взаємодії; спроєктовано архітектуру монорепозиторію з поділом на клієнтську та серверну частини; розроблено схему бази даних PostgreSQL з підтримкою багатомовності через таблиці перекладів; реалізовано модульний GraphQL API з 20 доменних модулів; розроблено клієнтський SPA-застосунок на базі React з Material UI; реалізовано систему автентифікації та авторизації на основі JWT; реалізовано повний цикл замовлень із платежами, виплатами продавцям та поверненнями; розроблено підсистеми модерації та верифікації продавців; забезпечено тестування за допомогою Jest та Cypress.

Об’єктом дослідження є процеси проєктування та розробки багатомовних маркетплейс-платформ з розширеним управлінням ролями та фінансовими потоками. Предметом дослідження є архітектура, технологічні рішення та методи реалізації веб-застосунку маркетплейсу Krydix з використанням стеку Node.js, React, GraphQL, PostgreSQL.

Розроблений застосунок є готовим до розгортання програмним продуктом. Архітектурні рішення, прийняті в ході розробки, – зокрема механізм fee snapshots для незмінності фінансових записів, модульна GraphQL-схема та підхід до E2E-тестування з фікстурами – можуть слугувати референсними рішеннями для аналогічних систем.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Перший розділ присвячений аналізу вимог: огляду предметної галузі, визначенню учасників системи та ключових сценаріїв використання. Другий розділ охоплює архітектуру та реалізацію: технологічний стек, базу даних, GraphQL API та інтерфейс користувача. Третій розділ описує тестування, розгортання та перевірку відповідності вимогам. Четвертий розділ присвячений охороні праці та безпеці життєдіяльності. У висновках підбиваються підсумки роботи та окреслюються напрямки подальшого розвитку.

# РОЗДІЛ 1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

У цьому розділі проведено аналіз предметної галузі електронної комерції, визначено учасників системи та їхні ролі, побудовано діаграму прецедентів і описано ключові сценарії взаємодії. Результати аналізу є підставою для архітектурних рішень, викладених у другому розділі.

## 1.1 Огляд предметної галузі

Для коректного формулювання вимог до системи необхідно спочатку дослідити предметну галузь, у якій вона функціонуватиме. У цьому підрозділі розглядається сутність маркетплейсу як типу електронної торгової платформи, аналізуються основні моделі взаємодії між учасниками ринку, визначаються технологічні тенденції, що є актуальними для сучасних e-commerce рішень, а також проводиться порівняльний аналіз існуючих рішень з метою обґрунтування необхідності розробки платформи Krydix.

### 1.1.1 Поняття маркетплейсу та його роль у цифровій економіці

Маркетплейс (від англ. marketplace – торговий майданчик) – це тип електронної торгової платформи, на якій кілька продавців пропонують товари або послуги покупцям через єдиний інтерфейс. На відміну від традиційного інтернет-магазину, що належить одному продавцю, маркетплейс є посередником між безліччю постачальників та споживачами, надаючи інфраструктуру, системи оплати, логістичні інструменти та механізми довіри [4]. Платформа не є власником товарів – вона лише створює середовище для здійснення угод і отримує дохід у вигляді комісійного збору.

Ключова відмінність маркетплейсу від класичного e-commerce магазину полягає у множинності продавців та необхідності механізмів довіри – системи рейтингів, верифікації, модерації контенту та захисту покупців. Серед найбільших світових платформ – Amazon, eBay, Etsy, AliExpress, а на українському ринку –

Rozetka. Кожна з них має свою специфіку: Amazon поєднує власні товари зі сторонніми продавцями, Etsy орієнтований на ручну роботу та унікальні вироби, Rozetka займає лідируючу позицію на вітчизняному ринку. Проте жодна з міжнародних платформ не є в повній мірі адаптованою до специфіки українського ринку: підтримка гривні, ЄДРПОУ, місцевих способів доставки та двомовного інтерфейсу.

### 1.1.2 Моделі взаємодії та технологічні тренди

Основні моделі взаємодії на маркетплейсах визначаються складом учасників угоди. B2C (Business-to-Consumer) є найпоширенішою моделлю, де продавцями виступають юридичні особи або підприємці, а покупцями – кінцеві споживачі. Саме на цій моделі зосереджена перша версія платформи Krydix. B2B (Business-to-Business) передбачає взаємодію між підприємствами, часто включає гуртові ціни та мінімальні обсяги замовлення; архітектура Krydix спроєктована з урахуванням потенційного розширення в цьому напрямку.

З технологічного боку сучасні маркетплейс-платформи характеризуються кількома ключовими трендами. Підхід API-first та використання GraphQL замість традиційних REST API дозволяє клієнту точно вказувати, які дані йому потрібні, зменшуючи надлишковість передачі даних [5]. Real-time комунікація через WebSocket є обов’язковою для чату між покупцями та продавцями, а також для сповіщень про зміни статусу замовлення. Багатомовність досягається через зберігання перекладів у реляційних базах даних окремими таблицями, що дозволяє масштабувати кількість мов без зміни основної схеми. SPA-підхід (Single Page Application) на базі React, Vue або Angular домінує в клієнтській частині завдяки швидкості реагування інтерфейсу [6]. Зображення та відео товарів зберігаються на CDN-платформах (Cloudinary, AWS S3) для оптимальної доставки медіаконтенту.

### 1.1.3 Моделі взаємодії та технологічні тренди

Перед формулюванням вимог до системи необхідно проаналізувати наявні ринкові рішення та визначити їхні обмеження, що є прямим обґрунтуванням для

розробки Krydix. Для порівняння обрано платформи, що є найбільш релевантними для українського ринку або широко використовуються у схожому сегменті: Amazon, eBay, OLX, Rozetka та Prom.ua.

Таблиця 1.1 – Порівняльний аналіз існуючих маркетплейс-платформ

| **Платформа** | **Тип** | **Верифікація продавців** | **Адаптація до укр. ринку** | **Захист коштів покупця** | **Модерація товарів** | **Вбудований чат** |
|---|---|---|---|---|---|---|
| Amazon | B2C/B2B | Часткова | Відсутня | Так | Автоматична | Ні |
| eBay | B2C/C2C | Мінімальна | Відсутня | Частково | Ручна | Ні |
| OLX | C2C | Відсутня | Так | Так | Ручна, обмежена | Так |
| Rozetka | B2C | Часткова | Так | Ні | Є | Ні |
| Prom.ua | B2B/B2C | Базова | Так | Ні | Є | Обмежений |
| Krydix | B2C→B2B | Повна (ЄДРПОУ, ІПН, документи) | Так | Escrow-подібний механізм | Автоматична + ручна | Вбудований |

Аналіз виявив кілька системних недоліків наявних рішень. По-перше, відсутність документарної верифікації: OLX дозволяє продавати без підтвердження юридичної особи, що створює умови для шахрайства; Rozetka та Prom.ua вимагають лише базової реєстрації без перевірки ЄДРПОУ чи ІПН. По-друге, відсутність захисту коштів: не всі з вітчизняних платформ реалізують механізму утримання платежу до підтвердження отримання товару покупцем. По-третє, обмежена комунікація: Rozetka та Prom.ua не надають прямої комунікації покупець↔продавець у межах платформи. По-четверте, відсутність адаптації до вимог українського законодавства в міжнародних платформах: Amazon та eBay не підтримують ЄДРПОУ, ІПН та гривню як основну валюту.

Таким чином, Krydix заповнює нішу українського B2C-маркетплейсу, пропонуючи документарну верифікацію продавців, escrow-подібний захист коштів покупця, незмінні фінансові записи на рівні архітектури, вбудований чат та повну адаптацію до вимог українського ринку – комплекс характеристик, якого не надає жодна з розглянутих платформ у сукупності.

## 1.2 Формування мети та завдань розробки

На основі проведеного аналізу предметної галузі сформульовано мету розробки та конкретний перелік завдань, що необхідно вирішити для її досягнення. У цьому підрозділі визначено постановку задачі, окреслено функціональні та нефункціональні вимоги до системи, що є вихідними умовами для проєктування архітектури у другому розділі.

### 1.2.1 Постановка задачі

Аналіз предметної галузі показав, що існує потреба в маркетплейс-платформі, орієнтованій на український ринок, яка б поєднувала зручний інтерфейс для покупців зі структурованим каталогом, прозору верифікацію продавців із перевіркою юридичних документів (ЄДРПОУ, ІПН, банківські реквізити), систему модерації товарів і відгуків, повний цикл замовлень із підтримкою повернень і виплат продавцям, а також підтримку двох мов.

Основна ціль розробки – реалізація MVP (Minimum Viable Product) маркетплейсу Krydix, функціонально придатного для реального використання з архітектурою, здатною до масштабування. Вторинними цілями є забезпечення архітектурної гнучкості для подальшого розширення (B2B, зовнішні PSP-інтеграції, мобільні клієнти), побудова системи тестування з покриттям критичних бізнес-потоків та використання технологічного стеку, що відповідає сучасним стандартам індустрії.

### 1.2.2 Функціональні вимоги

Функціональні вимоги розділені за доменами системи. Каталог і пошук передбачають перегляд товарів з підтримкою до 4 рівнів категорій, пошук за назвою, описом та SKU, фільтрацію за ціною, категорією, брендом, рейтингом та наявністю, а також сортування за новизною, популярністю та ціною.

Вимоги до автентифікації та профілів охоплюють реєстрацію з підтвердженням email, вхід через email/пароль з JWT-токенами та управління

профілем. Покупець повинен мати можливість додавати товари до кошика та списку бажань, оформлювати замовлення, переглядати їх статуси, підтверджувати отримання, запитувати повернення, писати відгуки та отримувати сповіщення.

Для продавця необхідна можливість подати заявку на верифікацію з документами, керувати товарами (CRUD, варіанти, медіа), завантажувати товари масово (xlsx/csv), управляти замовленнями та переглядати аналітику і фінансові виплати. Модератор повинен мати можливість переглядати та модерувати товари й відгуки, розглядати скарги та заявки на верифікацію продавців. Адміністратор отримує повний контроль над платформою: управління користувачами та ролями, категоріями, platform config, аудит-логом та жорстке видалення.

### 1.2.3 Нефункціональні вимоги

З точки зору безпеки система повинна забезпечувати хешування паролів (bcrypt), JWT із refresh rotation, rate limiting для чутливих ендпоінтів та CORS-обмеження. Продуктивність забезпечується через repository layer для батчованих запитів до БД та кешування на клієнті через Apollo Client. Надійність досягається аудит-логом для всіх фінансових і модераційних дій та fee snapshots для незмінності фінансових записів. Масштабованість підтримується модульною GraphQL-схемою, підготовленою до додавання нових доменів. Всі рядки інтерфейсу повинні бути реалізовані через i18n-ключі. Система повинна бути покрита Jest-тестами для unit/integration та Cypress-тестами для E2E.

## 1.3 Визначення учасників системи та сценаріїв взаємодії

Розробка маркетплейс-платформи передбачає роботу з кількома категоріями користувачів, кожна з яких має власний набір прав та обов'язків. У цьому підрозділі ідентифіковано всіх учасників системи – акторів у термінах UML – та побудовано діаграму прецедентів, яка відображає взаємодію кожного актора із функціями платформи.

### 1.3.1 Учасники системи

У системі Krydix визначено п’ять ролей, що відображені безпосередньо в enum Role бази даних та GraphQL-схемі.

Гість (GUEST) – незареєстрований відвідувач. Може переглядати каталог, сторінки товарів та профілі продавців. Для будь-яких комерційних дій потрібна реєстрація.

Покупець (BUYER) – зареєстрований користувач за замовчуванням. Може здійснювати покупки, писати відгуки, подавати скарги, користуватися чатом та списком бажань. Подає заявку на статус продавця, після чого, у разі схвалення, отримує роль SELLER.

Продавець (SELLER) – верифікований бізнес-продавець. Управляє своїми товарами, обробляє замовлення, має доступ до аналітики та фінансів. Верифікація є обов’язковою умовою для отримання цієї ролі.

Модератор (MODERATOR) – співробітник платформи. Модерує товари, відгуки та скарги, переглядає заявки продавців, може змінювати ролі для користувачів не вище рівня MODERATOR. Має доступ до чату підтримки.

Адміністратор (ADMIN) – найвищий рівень доступу. Повний контроль над платформою: управління ролями (включно з ADMIN), категоріями, налаштуваннями платформи, жорстке видалення записів, аудит всіх дій.

### 1.3.2 Діаграма прецедентів

Діаграма прецедентів охоплює три основні групи сценаріїв. Публічний доступ включає перегляд каталогу, товарів та профілів продавців. Аутентифікаційні сценарії охоплюють реєстрацію, вхід, верифікацію email та оновлення токена. Захищені сценарії розподілені за ролями: покупець – кошик, checkout, замовлення, відгуки; продавець – управління товарами, замовленнями, аналітика; модератор – модерація, скарги, підтримка; адміністратор – управління платформою та аудит. Розширена діаграма прецедентів для всіх п’яти ролей системи представлена в додатку Б.

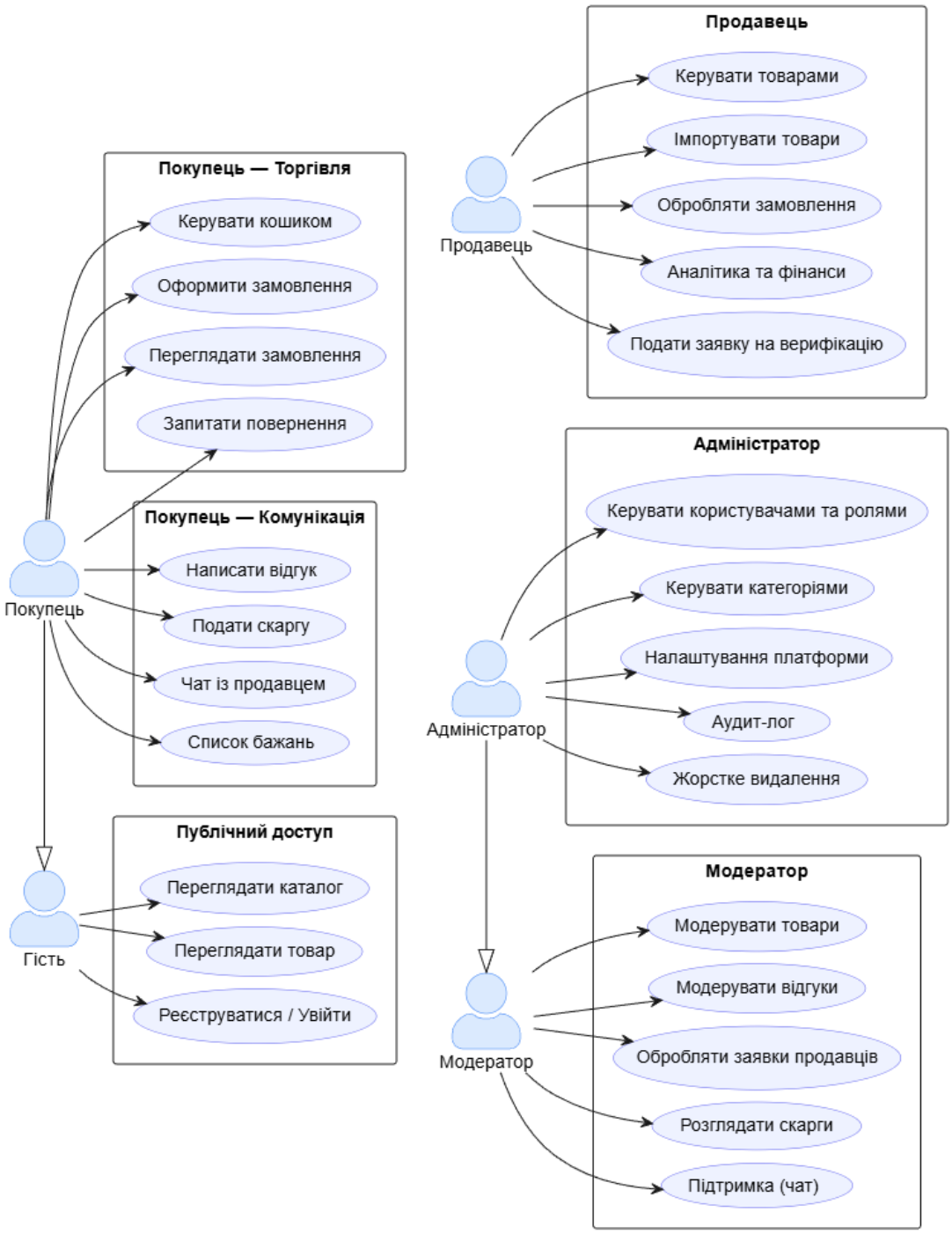



Рисунок 1.1 – Діаграма прецедентів системи Krydix

Подана діаграма прецедентів формує концептуальну карту функціональних можливостей системи та слугує підставою для детального опису окремих сценаріїв у наступному підрозділі. Чітке відображення спадковості ролей дозволяє уникати дублювання прав у моделі доступу та узгоджується з матрицею ролей.

## 1.4 Опис ключових сценаріїв використання

Після визначення переліку прецедентів необхідно детально описати найважливіші з них, зосереджуючись на критичних бізнес-потоках платформи. Нижче розглянуто вісім сценаріїв, що охоплюють реєстрацію, перегляд каталогу, оформлення замовлення, верифікацію продавця, модерацію товарів, виплати продавцям, відгуки та real-time чат. Для кожного сценарію визначено акторів, передумови, основний потік та виняткові ситуації.

### 1.4.1 Реєстрація та верифікація email

Актором цього сценарію є гість або майбутній покупець, який має дійсну електронну адресу. Користувач відкриває сторінку реєстрації, вводить email, пароль, ім’я та прізвище. Форма валідується на клієнті через React Hook Form + Zod, а потім на сервері тими самими Zod-схемами. Сервер хешує пароль через bcrypt і створює запис User зі статусом isEmailVerified: false. На email надсилається лист із токеном верифікації через Nodemailer + Gmail. Користувач переходить за посиланням; сервер позначає isEmailVerified: true, після чого вхід стає доступним.

Виняткові ситуації включають спробу реєстрації з вже зайнятим email, прострочений або вже використаний токен верифікації. Реалізація охоплює authService.ts, authResolvers.ts, модель EmailVerification у Prisma та email.ts у utils.

### 1.4.2 Перегляд каталогу та пошук товарів

Актором є гість або покупець. Сторінка каталогу (CatalogPage) завантажує товари зі статусом APPROVED. Користувач може відфільтрувати результати за категорією через дерево категорій, ціною через range slider, брендом, рейтингом та наявністю на складі. Пошуковий запит передається в GraphQL-запит catalog, який шукає за назвою, описом та SKU через catalogService в обох мовах одночасно. Результати відображаються у вигляді карток із можливістю сортування.

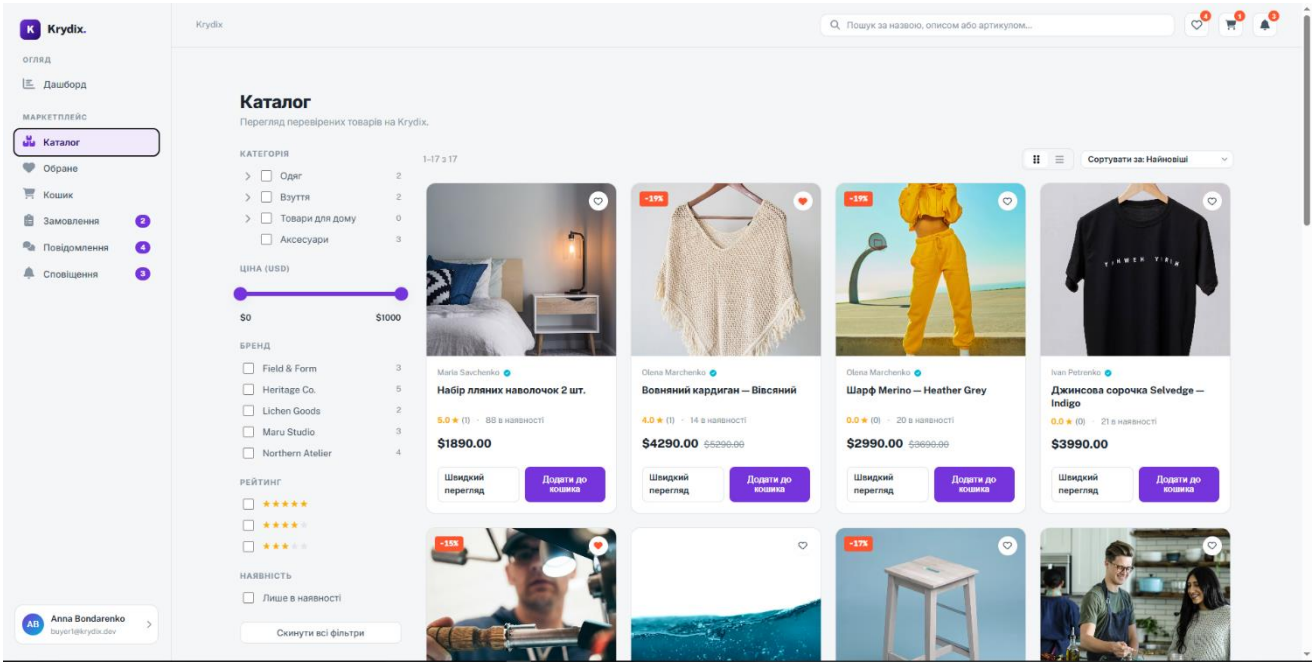



Рисунок 1.2 – Сторінка каталогу з деревом категорій та фільтрами

Наведений інтерфейс каталогу демонструє узгодженість дерева категорій, цінового фільтра та сітки карток, що відповідає функціональним вимогам до пошуку та фільтрації, сформульованим у підрозділі 1.2.2.

### 1.4.3 Оформлення замовлення (Checkout)

Покупець, маючи непорожній кошик та автентифікований стан, переходить на сторінку checkout. Він вводить контактні дані, спосіб доставки та оплати, опціонально застосовує промокод. Сервер перевіряє наявність усіх товарів на складі. Мутація createOrder у рамках однієї транзакції створює записи Order, OrderItem з fee snapshots, PaymentRecord (mock), DeliveryRecord та SellerPayout для кожного продавця зі статусом ON_HOLD. Fee Calculation Service обчислює комісії за даними CommissionRule та PlatformConfig і записує їх як незмінні поля у OrderItem. Кошик очищається, покупець перенаправляється на сторінку деталей замовлення.

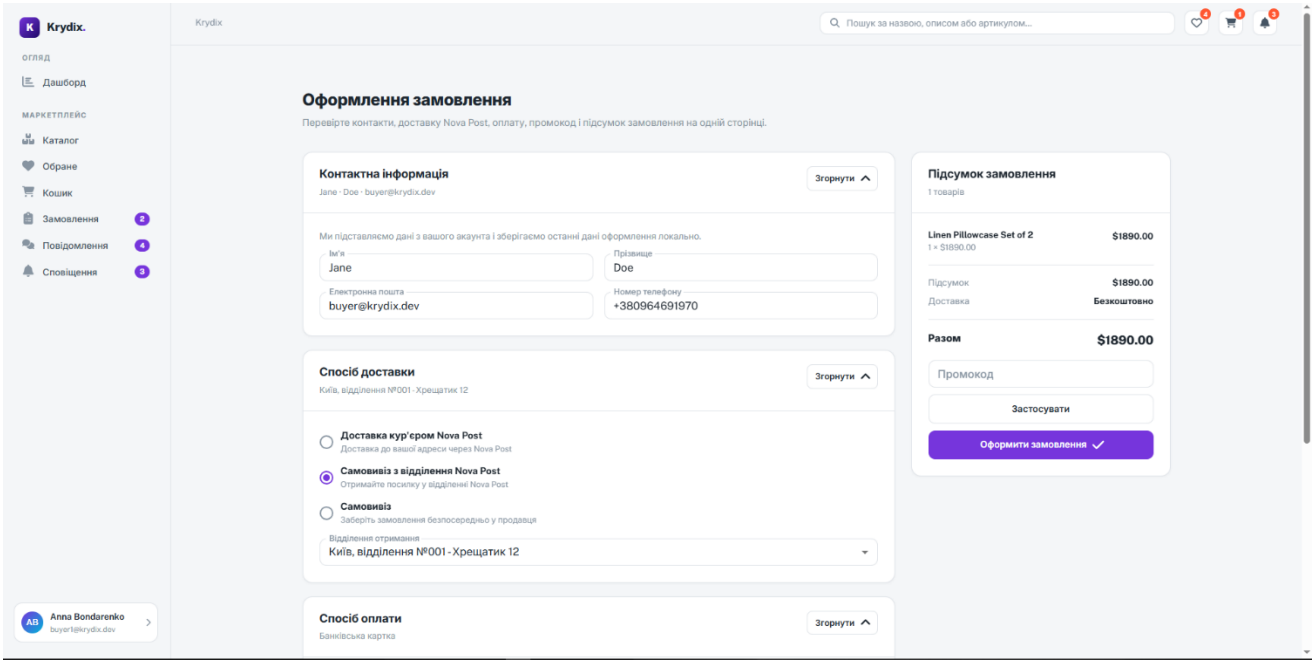



Рисунок 1.3 – Форма оформлення замовлення з вибором доставки та оплати

Виняткові ситуації: товар недоступний або закінчився – сервер блокує оформлення та повертає конкретну помилку. Невалідний промокод повертає повідомлення з причиною.

### 1.4.4 Верифікація продавця

Покупець, що бажає стати продавцем, подає заявку через форму верифікації. Заявка містить повне ім’я, назву компанії, ЄДРПОУ, ІПН, адресу, телефон, email, банківські реквізити та підтверджуючі документи, що завантажуються через Cloudinary у вигляді моделі SellerDocument. Заявка отримує статус PENDING і потрапляє до черги модератора.

Модератор у панелі верифікації переглядає документи та інформацію заявника, змінює статус на UNDER_REVIEW, а потім на APPROVED або REJECTED із обов’язковою причиною. При APPROVED роль користувача автоматично змінюється на SELLER. Кожна зміна статусу записується в AuditLog з action: SELLER_VERIFICATION_CHANGE.

### 1.4.5 Модерація товару

Після створення або редагування товару продавцем статус автоматично переходить у PENDING_MODERATION. Модератор бачить товар у черзі модерації в ProductModerationPage, перевіряє контент, фотографії, категорію та опис. Він встановлює статус APPROVED або REJECTED із причиною. При APPROVED товар стає видимим покупцям. Будь-яке подальше редагування повертає товар у PENDING_MODERATION.

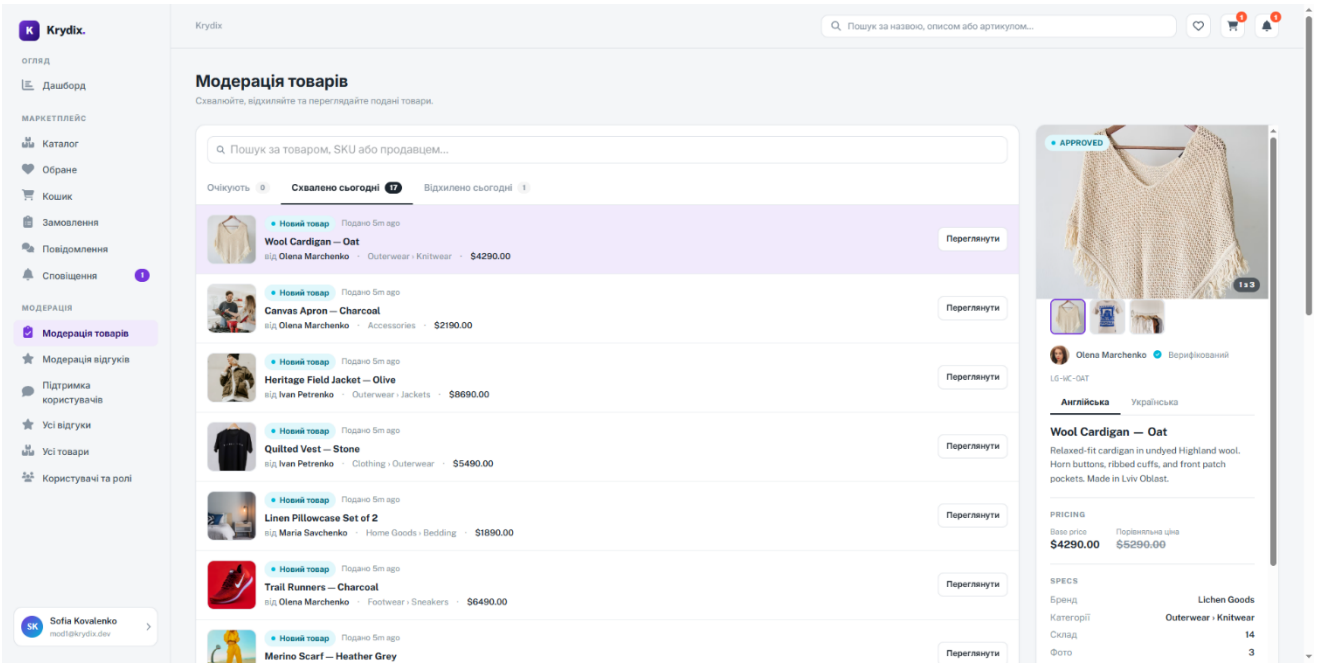



Рисунок 1.4 – Панель модерації товарів

Подана панель модерації забезпечує модератору повний контекст рішення – заявлений контент, фото та категорію – в одному екрані, що скорочує середній час обробки заявки.

### 1.4.6 Цикл виплат продавцю

Покупець підтверджує отримання замовлення або це відбувається автоматично через autoConfirmDays. При підтвердженні для кожного OrderItem фоновий процес в orderJobService чекає закінчення payoutHoldDays (за замовчуванням 3 дні з PlatformConfig), після чого виплата переходить з ON_HOLD

у ELIGIBLE_FOR_RELEASE та RELEASED. Якщо відкрите повернення – виплата переходить у BLOCKED. Продавець може ініціювати виведення коштів, що змінює статус на WITHDRAWN.

### 1.4.7 Відгуки та рейтинги

Після завершення замовлення (статус DELIVERED) покупець може залишити відгук на товар (ProductReview) та на продавця (SellerReview). За замовчуванням відгук має isApproved: false і не відображається покупцям до рішення модератора. Продавець може відповісти на відгук через поле sellerReply. Адміністратор або модератор може видалити відгук через soft delete (deletedAt). Унікальна комбінація (productId, reviewerId, orderId) запобігає дублюванню відгуків.

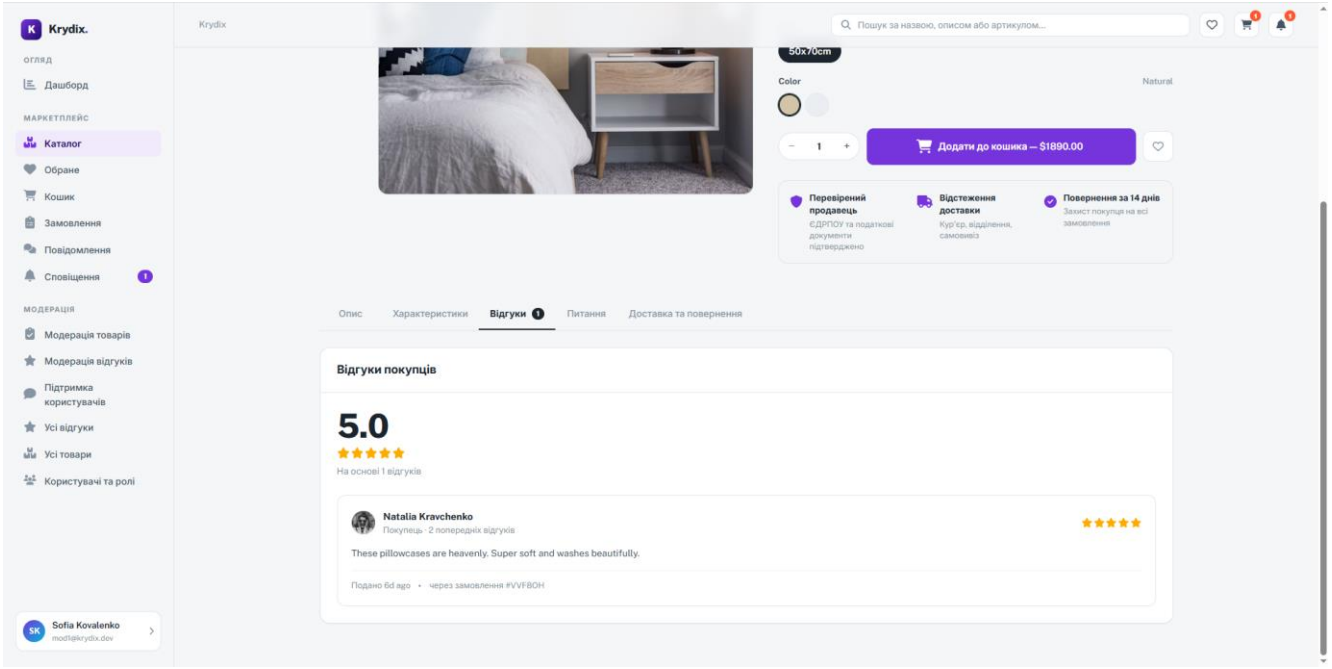



Рисунок 1.5 – Секція відгуків та рейтингу на сторінці товару

Інтерфейс секції відгуків поєднує середній рейтинг, текстові відгуки та відповідь продавця в одному компоненті, що формує цілісну репутаційну картку товару для покупця.

### 1.4.8 Real-time чат

Покупець ініціює чат зі сторінки товару (тип MARKETPLACE). На сервері через Socket.IO встановлюється з’єднання; кімнати ідентифікуються за chatId. Повідомлення зберігаються в ChatMessage з прапорами isDelivered та isRead. Для підтримки між продавцем і модератором використовується чат типу SUPPORT. Presence-сервіс оновлює User.lastSeenAt при активних запитах, відображаючи статус онлайн.

## 1.5 Висновок до розділу

У першому розділі проведено аналіз предметної галузі електронної комерції та обґрунтовано актуальність розробки платформи Krydix. Визначено п’ять ролей учасників системи з описом їхніх прав та відповідальностей. Описано вісім ключових сценаріїв використання, що охоплюють критичні бізнес-потоки: реєстрацію, перегляд каталогу, оформлення замовлення, верифікацію продавця, модерацію товарів, виплати продавцям, відгуки та чат. Проведений аналіз є підставою для архітектурних рішень, описаних у другому розділі.

# РОЗДІЛ 2 АРХІТЕКТУРА ТА РЕАЛІЗАЦІЙНЕ ПРОЄКТУВАННЯ

У цьому розділі описано архітектурні рішення та технічну реалізацію платформи Krydix. Розглянуто обґрунтування методології розробки, ключові архітектурні патерни, схему бази даних, GraphQL API, технологічний стек, реалізацію серверних і клієнтських модулів, а також організацію системи контролю версій.

## 2.1 Обґрунтування підходу до розробки

Для розробки системи Krydix обрано інкрементальну модель розробки програмного забезпечення, орієнтовану на MVP (Minimum Viable Product). Цей підхід передбачає послідовне нарощування функціональності: спочатку реалізується базовий набір можливостей, достатній для перевірки гіпотез і реального використання, а потім система розширюється в наступних ітераціях.

Інкрементальний підхід обраний з кількох причин. Складність повного маркетплейсу є занадто великою для розробки «водоспадним» методом. Наявність чітких доменних меж – auth, catalog, orders, payments, reviews – дозволяє розробляти й тестувати кожен модуль незалежно. Бізнес-вимоги до деяких підсистем, зокрема escrow та B2B, можуть уточнюватися в процесі розробки MVP.

Планування роботи ведеться за допомогою GitHub Issues та Pull Requests. Кожна функціональна можливість оформлюється як окрема задача (issue), реалізується у feature-гілці та проходить code review перед злиттям у основну гілку develop.

При проєктуванні системи дотримано кількох ключових архітектурних принципів. Принцип розподілу відповідальностей (Separation of Concerns) визначає, що кожен шар системи виконує чітко визначену функцію: GraphQL resolver приймає запит і делегує сервісу, сервіс реалізує бізнес-логіку, репозиторій виконує запит до бази даних. Принцип DRY забезпечується тим, що спільні константи, enum-и та типи визначені в одному місці та використовуються в обох

частинах монорепозиторію – файл constants/enums.ts з ідентичним вмістом присутній і на backend, і на frontend. Принцип незмінності фінансових даних є критичним для маркетплейсу: при створенні замовлення всі комісії зберігаються як знімки в полях OrderItem, тому жодна подальша зміна PlatformConfig не вплине на вже створені замовлення. Принцип аудитованості означає, що всі модераційні та фінансові дії записуються в AuditLog з посиланням на актора, дію та ціль.

## 2.2 Архітектурні рішення та патерни проєктування

Архітектура системи Krydix будується навколо набору перевірених патернів, кожен з яких розв'язує конкретну проблему проєктування багатокомпонентного веб-застосунку. У цьому підрозділі формалізовано ключові патерни та обґрунтовано їх застосування саме до контексту маркетплейс-платформи.

Клієнт-серверна архітектура зі SPA-клієнтом. Система реалізована як SPA (Single Page Application) на React, що взаємодіє з єдиним серверним вузлом через HTTP та WebSocket. Перевагою такого підходу є чітке розмежування зон відповідальності: сервер відповідає лише за бізнес-логіку, доступ до даних та авторизацію, а клієнт – виключно за презентацію та user experience. SPA забезпечує миттєву реакцію інтерфейсу на дії користувача без перезавантаження сторінки і дозволяє повторно використовувати компоненти між різними сторінками. Для маркетплейсу з різноманітними ролями (покупець, продавець, модератор, адміністратор) SPA є особливо доречною, оскільки дозволяє динамічно змінювати доступне меню та маршрути без переходу на окремі субдомени.

Resolver-Service-Repository як серверний патерн. Замість традиційного для REST-API патерна Routes-Controller-Service у проєкті застосовано трирівневу структуру Resolver-Service-Repository, адаптовану для GraphQL. Шар resolver'ів виконує роль, аналогічну контролерам у MVC: приймає GraphQL-запит, виконує перевірку авторизації через context.user.role та делегує виконання сервісному шару. Сервісний шар реалізує бізнес-логіку конкретного домену (checkoutService, feeCalculationService, moderationService) і не знає деталей зберігання даних. Шар

репозиторіїв (orderRepository, productRepository) інкапсулює всі звернення до Prisma і є єдиною точкою контакту з базою даних. Така структура підтримує принципи Single Responsibility (кожен шар вирішує одну задачу) та Dependency Inversion (сервіси залежать від абстракцій репозиторіїв, а не від ORM безпосередньо). Порівняно з класичним MVC, де view відсутній (його роль виконує клієнтський SPA), Resolver-Service-Repository забезпечує ту саму розв'язку компонентів, але адаптовану під GraphQL-парадигму.

GraphQL як протокол взаємодії. На відміну від REST API, де клієнт отримує фіксовану структуру відповіді з заданого ендпоінту, GraphQL дозволяє клієнту точно описати, які саме поля він потребує, в одному запиті. Для маркетплейсу з великою кількістю взаємопов'язаних сутностей (товар, варіанти, медіа, відгуки, продавець, категорія) це знижує мережевий overhead та дозволяє уникнути «over-fetching» і «under-fetching». Єдиний ендпоінт POST /graphql обробляє всі запити та мутації, що спрощує налаштування мережевої інфраструктури та CORS-політик. Real-time комунікація (чат, сповіщення) винесена в окремий канал – Socket.IO над WebSocket, оскільки GraphQL Subscriptions у MVP не використовуються.

Stateless JWT-аутентифікація. Замість sessions-based підходу зі збереженням стану на сервері застосовано stateless-автентифікацію через пару JWT-токенів: короткоживучий access token та довгоживучий refresh token. Stateless-модель означає, що сервер не зберігає інформацію про активні сесії – кожен запит несе всю необхідну для авторизації інформацію в Authorization–заголовку. Це забезпечує горизонтальну масштабованість (будь–який екземпляр сервера може обробити будь–який запит без необхідності спільного session storage) та спрощує розгортання на безсерверних платформах. Refresh–токени зберігаються у базі даних з можливістю анулювання через поле revokedAt, що дозволяє вийти з сесії або примусово припинити доступ при компрометації.

ORM як абстракція над базою даних. Замість прямої роботи з SQL–запитами застосовано Prisma ORM як рівень абстракції. Prisma забезпечує type–safe доступ до бази даних з автоматичною генерацією TypeScript–типів безпосередньо зі schema.prisma. Це усуває цілий клас помилок під час компіляції: неможливо

звернутися до неіснуючого поля, помилитися з типом параметра або забути обов'язкове поле при створенні запису. Prisma також керує міграціями схеми через prisma migrate, забезпечуючи відтворюваність структури БД між середовищами.

Сукупність описаних патернів формує цілісне архітектурне рішення, у якому кожен компонент має чітко окреслену відповідальність, а взаємодії між компонентами стандартизовані. Така архітектура є підставою для подальшого опису конкретної реалізації модулів у наступних підрозділах розділу 2.

## 2.3 Побудова архітектури системи

Визначивши підхід до розробки та архітектурні принципи, перейдемо до конкретного проєктування системи. Цей підрозділ описує загальну клієнт–серверну архітектуру, організацію GraphQL–схеми за доменним принципом, шар репозиторіїв між бізнес–логікою та базою даних, а також механізм фонових задач для автоматизованих процесів платформи.

### 2.3.1 Загальна архітектура клієнт–сервер

Система побудована за архітектурою клієнт–сервер зі SPA (Single Page Application) на клієнті. Клієнтська частина побудована на React 18 з Vite та Apollo Client, де GraphQL–клієнт забезпечує кешування та синхронізацію даних, а Socket.IO Client забезпечує real–time комунікацію. Zustand зберігає глобальний стан автентифікації та кошика; i18next забезпечує інтернаціоналізацію; React Router v6 реалізує клієнтський роутинг. Серверна частина побудована на Node.js з Express v5 та Apollo Server 4. Prisma ORM виконує всі запити до PostgreSQL; Socket.IO–сервер обробляє WebSocket–з’єднання для чату; Pino забезпечує структуроване логування; express–rate–limit – обмеження частоти запитів. Cloudinary слугує хмарним сховищем медіафайлів, Nodemailer – транспортом для відправлення email через Gmail.

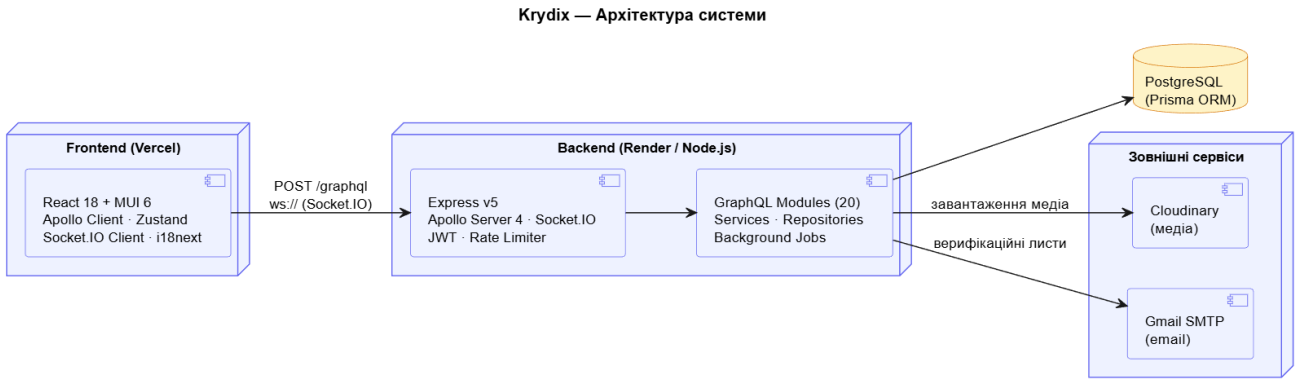



Рисунок 2.1 – Загальна архітектура системи Krydix

Розгортання системи організоване таким чином: frontend–SPA розміщується на Vercel (статичний хостинг з CDN), backend розміщується на Render (Node.js Web Service), а PostgreSQL використовується як managed–сервіс із підключенням через pgBouncer.

### 2.3.2 GraphQL–архітектура

GraphQL–схема зібрана з 20 доменних модулів за допомогою @graphql–tools/merge. Кожен модуль містить два файли: typeDefs.ts з визначенням типів і операцій та resolvers.ts з їх реалізацією. Такий поділ забезпечує чітку ізоляцію доменів та можливість незалежного розвитку кожного з них. Точкою збирання схеми є файл backend/src/graphql/schema.ts, де всі typeDefs та resolvers об’єднуються функціями mergeTypeDefs та mergeResolvers. Єдиний GraphQL–ендпоінт – POST /graphql – обробляє всі запити і мутації, а для чату окремо функціонує Socket.IO–сервер.

Шар авторизації реалізований у кожному resolver: перевірка наявності та ролі користувача відбувається через об’єкт контексту GraphQLContext, що містить дані автентифікованого користувача, отримані через JWT–middleware extractAuthUser. Серверна авторизація завжди звертається до бази даних для перевірки актуальної ролі, не покладаючись виключно на дані токена – це забезпечує захист від компрометованих або застарілих токенів. Лістинг GraphQL–схеми checkout–модуля наведено в додатку Ж.

### 2.3.3 Шар репозиторіїв

Між сервісами та Prisma–клієнтом реалізовано шар репозиторіїв у директорії repositories/. Кожен репозиторій інкапсулює запити до бази даних для конкретного домену. Такий підхід забезпечує можливість заміни сховища без зміни сервісного шару, централізовану оптимізацію запитів через include та select, а також зручне мокування в тестах. Прикладами є orderRepository.ts, productRepository.ts, reviewRepository.ts, payoutRepository.ts, returnRequestRepository.ts.

### 2.3.4 Модуль фонових задач

orderJobService.ts запускає фонові процеси через setInterval одразу після старту сервера. Реалізовано дві ключові задачі. Перша – auto–confirm receipt: якщо покупець не підтвердив отримання замовлення протягом autoConfirmDays (за замовчуванням 7 днів, параметр з PlatformConfig) після відправлення, система автоматично переводить замовлення в статус DELIVERED. Друга – payout release: після закінчення payoutHoldDays виплата продавцю переходить з ON_HOLD у ELIGIBLE_FOR_RELEASE. Параметри задач зберігаються в PlatformConfig та можуть змінюватись адміністратором без перезапуску сервера.

## 2.4 Модель даних і структура бази

Проєктування схеми бази даних є одним із центральних завдань розробки маркетплейсу, оскільки якість моделі даних безпосередньо впливає на надійність фінансових розрахунків, гнучкість каталогу та масштабованість системи в цілому. У цьому підрозділі описано повну схему PostgreSQL з більш ніж 30 моделями, детально розглянуто сутності товарів і варіантів, фінансовий блок із механізмом fee snapshots, а також модель категорій із динамічними атрибутами.

### 2.4.1 Загальна характеристика схеми бази даних

База даних PostgreSQL містить понад 30 моделей, описаних у файлі backend/prisma/schema.prisma. Схема розроблена з дотриманням принципів

нормалізації (3NF), за винятком спеціально денормалізованих полів fee snapshots в OrderItem для забезпечення незмінності фінансових записів. Повна ER–діаграма зі схемою всіх 30+ моделей бази даних подана в додатку В.

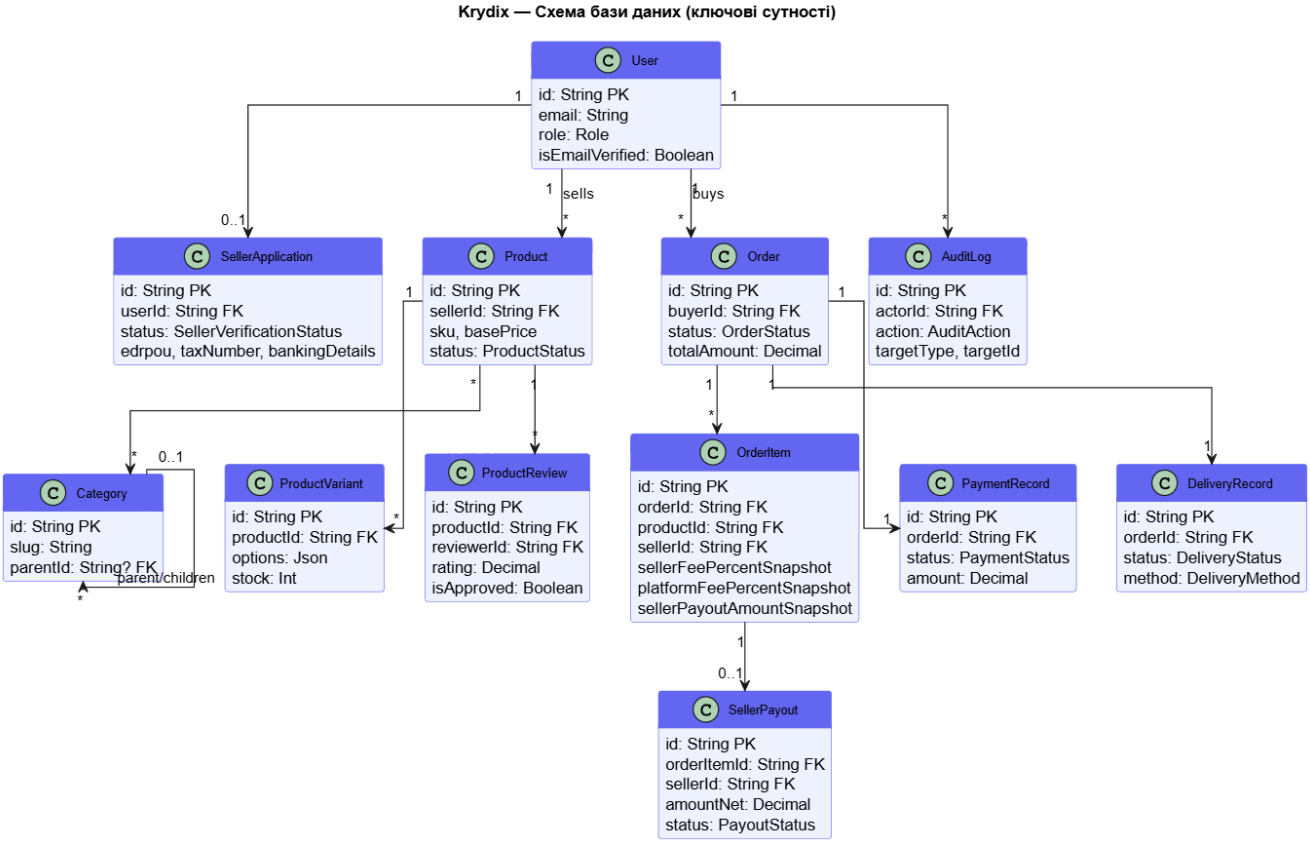



Рисунок 2.2 – ER–діаграма бази даних системи Krydix

Центральною сутністю є модель User, пов’язана з усіма доменними сутностями через зовнішні ключі. UserProfile зберігає розширені дані профілю (ім’я, фото, телефон, адреса) окремо від облікового запису, що дозволяє в майбутньому підтримувати кілька профілів на один акаунт. RefreshToken зберігає refresh–токени з полем revokedAt для анулювання при виході або ротації. EmailVerification зберігає токен з полем usedAt для запобігання повторному використанню.

Таблиця 2.1 – Основні сутності бази даних

| **Модель** | **Призначення** |
|---|---|
| User | Облікові записи всіх ролей |
| UserProfile | Розширений профіль (ім’я, фото, телефон, адреса) |
| RefreshToken | Refresh–токени для ротації JWT |
| EmailVerification | Токени підтвердження email |
| SellerApplication | Заявка на верифікацію продавця |
| SellerDocument | Файли підтверджуючих документів продавця |
| Category | Ієрархія категорій (self–referencing, parentId) |
| CategoryTranslation | Переклади назв категорій (EN + UK) |
| CategoryAttribute | Специфічні атрибути категорії для порівняння |
| Product | Товар продавця |
| ProductTranslation | Переклади назви та опису товару |
| ProductCategory | Зв’язок M:N між товарами та категоріями |
| ProductVariant | Варіанти товару (колір, розмір) з окремим stock |
| ProductAttributeValue | Значення атрибутів конкретного товару |
| Media | Медіафайли товару (зображення, відео) |
| CartItem | Елемент кошика |
| WishlistItem | Елемент списку бажань |
| Order | Замовлення покупця |
| OrderItem | Позиція замовлення (з fee snapshots) |
| ReturnRequest | Запит на повернення |
| SellerPayout | Виплата продавцю (payout lifecycle) |
| PaymentRecord | Запис про платіж |
| DeliveryRecord | Запис про доставку |
| PromoCode | Промокод |
| CommissionRule | Правило комісії для категорії або за замовчуванням |
| PlatformConfig | Глобальні налаштування платформи |
| ProductReview | Відгук покупця на товар |
| SellerReview | Відгук покупця на продавця |
| BuyerReview | Відгук продавця на покупця |
| Complaint | Скарга на товар, продавця, покупця або відгук |
| UserFeedback | Зворотний зв’язок від користувача до платформи |
| ReleaseNote | Нотатки про оновлення платформи |
| AuditLog | Журнал адміністративних та фінансових дій |
| Chat | Чат–кімната (marketplace або support) |
| ChatMessage | Повідомлення чату |
| Notification | In–app сповіщення |
| ImportJob | Задача масового імпорту товарів |

### 2.4.2 Модель товару та варіантів

Товар (Product) містить лише числові та статусні поля; текстовий контент зберігається в ProductTranslation із прив’язкою до мови через enum Language (EN, UK). Це дозволяє масштабувати кількість підтримуваних мов без зміни основної моделі. ProductVariant зберігає поле options типу JSON – набір характеристик

варіанта (наприклад, {"color": "red", "size": "XL"}). Запас (stock) відстежується на рівні варіанта, що є обов'язковою умовою для коректного обліку залишків. Media підтримує типи IMAGE та VIDEO; поле isMain позначає головне зображення; sortOrder керує порядком відображення.

ProductCategory реалізує зв'язок «багато до багатьох» між товарами та категоріями, що дозволяє одному товару належати до кількох категорій одночасно. CategoryAttribute та ProductAttributeValue забезпечують динамічні атрибути для кожної категорії, що є основою для функції порівняння товарів у межах однієї категорії.

### 2.4.3 Модель замовлення та фінансовий блок

Модель Order є покупецьким записом і не містить sellerId, оскільки одне замовлення може включати товари від різних продавців. Вся продавець–специфічна логіка реалізована через OrderItem. Кожен OrderItem містить набір знімків (snapshots) фінансових даних: currencySnapshot, buyerFeePercentSnapshot та buyerFeeAmountSnapshot, sellerFeePercentSnapshot та sellerFeeAmountSnapshot, platformFeePercentSnapshot та platformFeeAmountSnapshot, sellerPayoutAmountSnapshot. Ці поля незмінні після створення замовлення – будь–яка зміна PlatformConfig або CommissionRule не впливає на їх значення.

SellerPayout – окрема модель, що реалізує повний lifecycle виплати від ON_HOLD через ELIGIBLE_FOR_RELEASE та RELEASED до WITHDRAWN. Поля availableAt, releasedAt, withdrawnAt фіксують часові мітки кожного переходу, забезпечуючи повну аудитованість фінансових операцій.

PaymentRecord відстежує статус платежу від PENDING до AUTHORIZED, PAID та IN_ESCROW. Поле escrowAt фіксує момент переходу коштів у режим утримання. Для MVP платіжна логіка є mock–реалізацією, але структура даних підготовлена для інтеграції з реальними PSP без зміни схеми. DeliveryRecord зберігає статус доставки, метод (courier, branch pickup, self pickup), адресу та трекінг–код. Діаграма фінансового блоку з повним складом полів fee snapshots та зв'язками Order → OrderItem → SellerPayout подана в додатку А.

### 2.4.4 Модель категорій та атрибутів

Category реалізує самореференційну структуру через поле parentId, дозволяючи будувати ієрархію до 4 рівнів вкладеності. CategoryTranslation зберігає локалізовані назви, описи та SEO–метадані окремо для кожної мови. Зв’язок CommissionRule з Category через optional categoryId дозволяє встановлювати різні відсотки комісії для різних категорій, де за відсутності категорійного правила застосовується глобальне правило з isDefault: true.

## 2.5 UML – та поведінкові діаграми

Для унаочнення ключових взаємодій між компонентами системи розроблено набір UML–діаграм. У цьому підрозділі наведено три діаграми послідовностей для критичних бізнес–потоків, а також описано lifecycle статусів усіх основних сутностей системи, що є обов'язковою умовою коректної реалізації бізнес–правил.

### 2.5.1 Діаграми послідовностей

Для унаочнення взаємодії між компонентами системи в межах критичних бізнес-потоків розроблено три діаграми послідовностей: для процесу реєстрації та верифікації електронної пошти, для оформлення замовлення та для модерації товару. Діаграми побудовано відповідно до нотації UML 2.5 і відображають хронологічну послідовність повідомлень між акторами та компонентами системи – клієнтом, GraphQL-сервером, сервісним шаром, репозиторієм та базою даних. Кожна діаграма охоплює як основний (успішний) потік виконання, так і альтернативні гілки для виняткових ситуацій, що забезпечує повноту опису поведінки системи в реальних умовах.

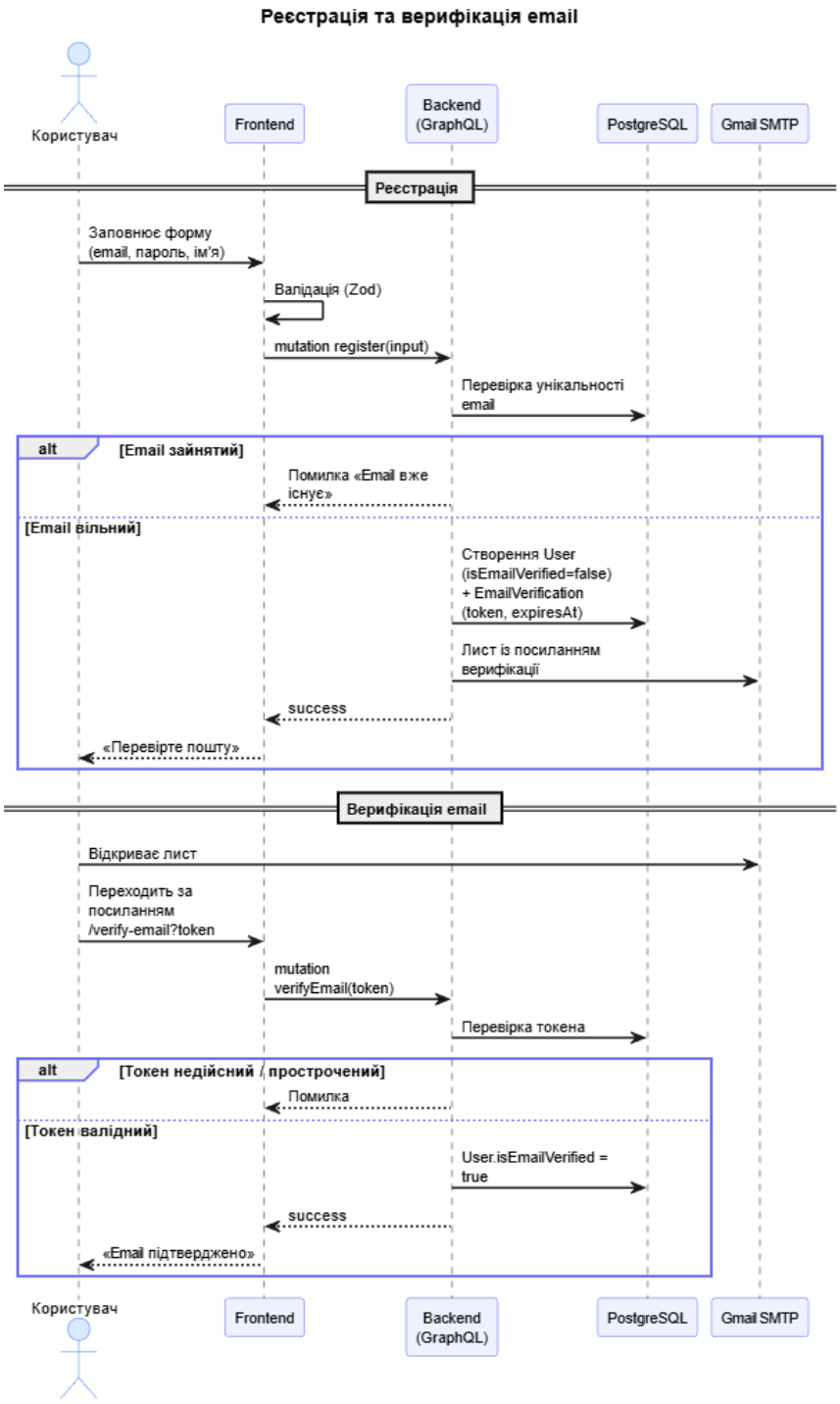



Рисунок 2.3 – Діаграма послідовності: реєстрація та верифікація email

Діаграма послідовностей для реєстрації ілюструє два основних потоки: заповнення форми з подвійною валідацією (клієнт та сервер), хешування пароля та відправлення верифікаційного листа; а потім – перехід за посиланням, перевірку токена та активацію облікового запису. Обидва потоки включають обробку виняткових ситуацій.

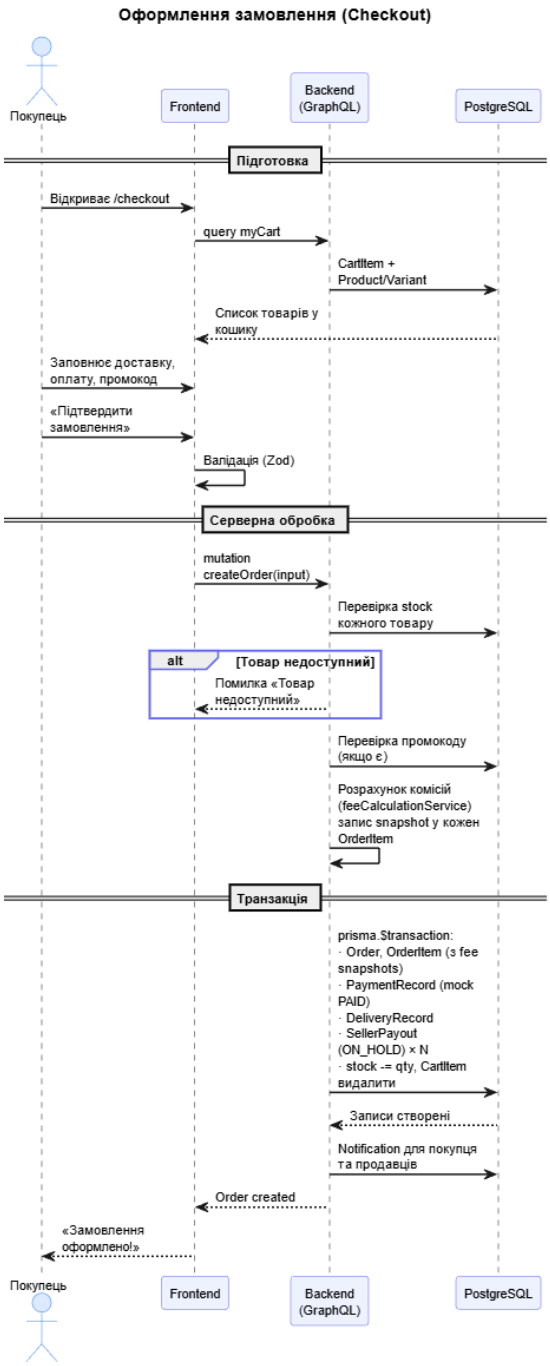



Рисунок 2.4 – Діаграма послідовності: оформлення замовлення

Діаграма послідовностей для оформлення замовлення відображає шість основних фаз: завантаження кошика, валідацію форми, перевірку залишку на складі кожного товару, застосування промокоду, розрахунок fee через feeCalculationService, і нарешті – атомарну транзакцію Prisma, що одночасно створює Order, кілька OrderItem зі snapshots, PaymentRecord, DeliveryRecord, SellerPayout та очищає кошик.

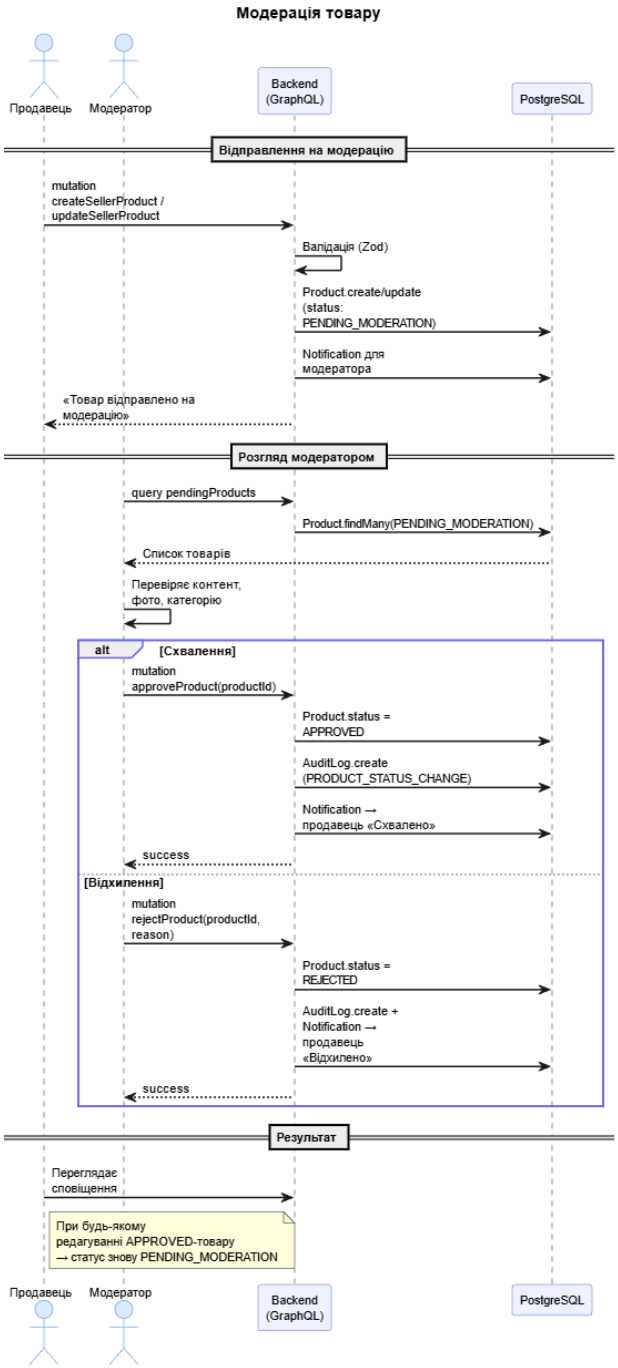



Рисунок 2.5 – Діаграма послідовності: модерація товару

Діаграма послідовностей для модерації товару ілюструє три фази: збереження товару продавцем із автоматичним переходом у PENDING_MODERATION, розгляд модератором та встановлення статусу APPROVED або REJECTED, і отримання продавцем результату через сповіщення. Кожна зміна статусу супроводжується записом в AuditLog та відправленням Notification.

### 2.5.2 Lifecycle статусів

Lifecycle товару: DRAFT → PENDING_MODERATION → APPROVED / REJECTED. Після схвалення продавець може перевести товар у ENABLED, DISABLED або ARCHIVED. Модератор може в будь–який момент встановити BLOCKED. Будь–яке редагування схваленого товару повертає його в PENDING_MODERATION.

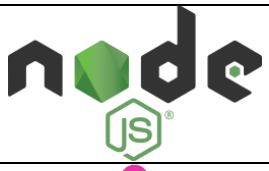
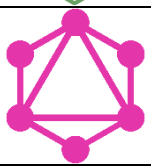
Lifecycle замовлення: PENDING → CONFIRMED → SHIPPED → DELIVERED, з можливістю CANCELLED на стадіях PENDING або CONFIRMED, та REFUNDED після завершення ReturnRequest.

Lifecycle верифікації продавця: DRAFT → PENDING → UNDER_REVIEW → APPROVED / REJECTED. Після відхилення продавець може повторно подати заявку. Після схвалення роль автоматично змінюється на SELLER.

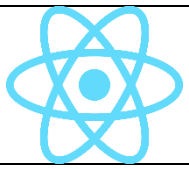
## 2.6 Вибір технологічного стеку та інструментів

Вибір технологій є стратегічним рішенням, що визначає можливості системи на роки вперед. При формуванні стеку для платформи Krydix керувалися такими критеріями: зрілість бібліотеки та активна підтримка спільнотою, типобезпечність через TypeScript, спільне використання інструментів на backend і frontend де можливо, а також мінімальний overhead при максимальній виразності коду. Нижче обґрунтовано вибір кожної технологій окремо для серверної частини, клієнтського застосунку та тестового інструментарію.

Таблиця 2.2 – Реалізовані базові UI–компоненти

| Логотип | Технологія | Призначення |
|---|---|---|
|  | Node.js 20 LTS | Серверна платформа (backend runtime) |
|  | GraphQL / Apollo | API-шар між клієнтом і сервером |

Продовження таблиці 2.2

| Логотип | Технологія | Призначення |
|---|---|---|
|  | React 18 | Клієнтський SPA-застосунок |
|  | Zustand 5 | Глобальний стан (auth, кошик) |
| Prisma | Prisma 5 | ORM та міграції PostgreSQL |

### 2.6.1 Backend–стек

Node.js обраний як платформа завдяки ефективній обробці I/O–інтенсивних операцій, широкій екосистемі npm та зручності спільного використання TypeScript з frontend [7]. Express v5 забезпечує HTTP–маршрутизацію та обробку middleware. Apollo Server 4 реалізує GraphQL–шар із вбудованою підтримкою introspection та sandbox для розробки. Prisma 5 забезпечує type–safe доступ до бази даних з автоматичною генерацією TypeScript–типів із схеми та підтримкою міграцій без boilerplate. PostgreSQL обраний як основна СУБД завдяки підтримці складних запитів, транзакцій, індексів і JSON–полів, що використовуються для options у ProductVariant та metadata в AuditLog. Socket.IO 4 забезпечує WebSocket–з'єднання з підтримкою кімнат та автоматичним відновленням з'єднання.

Для безпеки використовуються jsonwebtoken для підпису JWT, bcrypt для хешування паролів із регульованим cost factor та express–rate–limit для захисту від brute–force атак. Zod – TypeScript–first бібліотека схем, що дозволяє використовувати одні й ті ж схеми валідації як на backend, так і на frontend, формуючи спільний контракт між рівнями. Pino – структурований логер із JSON–виводом та мінімальним overhead. Для роботи з медіа – Cloudinary SDK; для email – Nodemailer; для xlsx – бібліотека xlsx.

### 2.6.2 Frontend–стек

React 18 є основою клієнтської частини, версія 18 додає Concurrent Features, що покращує продуктивність при рендерингу складних компонентів [8]. Vite 5 забезпечує надшвидкий HMR у розробці та оптимізований production build через ESM–нативний підхід. Material UI 6 надає комплексну бібліотеку компонентів із системою тем через createTheme(), що дозволяє повністю кастомізувати стилі, зберігаючи доступність та відповідність Material Design 3.

Apollo Client 3 із нормалізованим кешем мінімізує зайві запити до API – оновлення одного запису в кеші автоматично оновлює всі компоненти, що його відображають. React Router v6 реалізує декларативний клієнтський роутинг із вкладеними маршрутами та захищеними роутами через ProtectedRoute. Zustand 5 – легковісний менеджер стану, що зберігає глобальний стан автентифікації та кошика з мінімальним API. i18next з react–i18next забезпечує інтернаціоналізацію через JSON–файли перекладів; мова перемикається через UI–перемикач без зміни URL. React Hook Form у поєднанні з Zod забезпечує типобезпечну валідацію форм, де схема є спільним контрактом між клієнтом і сервером. Recharts використовується для відображення аналітики в seller dashboard. FontAwesome React є єдиною іконковою бібліотекою проєкту – всі іконки централізовані у constants/icons.ts.

### 2.6.3 Тестовий стек

Jest 29 із ts–jest забезпечує unit та integration тести для backend–сервісів і frontend–компонентів. @testing–library/react дозволяє тестувати React–компоненти через user–centric queries, що наближає тести до реального використання. Cypress реалізує E2E тестування з fixture–based мокуванням GraphQL через cy.intercept().

## 2.7 Реалізація серверних і клієнтських модулів

Визначивши архітектуру та стек, перейдемо до опису фактичної реалізації ключових доменних модулів. Кожен модуль розглядається з точки зору серверної

бізнес–логіки (service, repository) та клієнтської взаємодії (GraphQL–запити, стан, UI). У цьому підрозділі описано автентифікацію та авторизацію, каталог і пошук, checkout із розрахунком комісій, підсистему продавця з масовим імпортом, а також модуль чату та сповіщень.

### 2.7.1 Автентифікація та авторизація

Автентифікація реалізована через пару токенів: access token (JWT, короткий термін дії) та refresh token (довгий термін, зберігається в localStorage). При кожному запиті middleware extractAuthUser зчитує Authorization–заголовок, верифікує access token та встановлює об'єкт user у GraphQL–контексті. При виході або виявленні підозрілої активності refresh token анулюється через поле revokedAt. Ротація refresh–токенів відбувається при кожному оновленні – старий токен анулюється, новий видається.

Авторизація в resolver'ах перевіряє context.user.role. Критичне правило: роль перевіряється через актуальний запис у базі даних, а не лише через claims токена. На frontend автентифікований стан зберігається у authStore (Zustand). ProtectedRoute перевіряє роль і зону доступу (RouteZone) перед рендерингом сторінки, перенаправляючи неавторизованих на /login. Лістинг модуля контролю доступу за ролями подано в додатку Д.

### 2.7.2 Модуль каталогу та пошуку

Сервіс catalogService.ts реалізує пошук за назвою через ProductTranslation, фільтрацію за кількома параметрами та сортування. Запит до БД використовує Prisma where з OR–умовами для пошуку в обох мовах одночасно. Сортування підтримує enum ProductSort: NEWEST, POPULARITY, PRICE_ASC, PRICE_DESC, RATING. Пагінація реалізована через стандартний offset–підхід з параметрами page та pageSize. Лише товари зі статусом APPROVED повертаються в публічному каталозі.

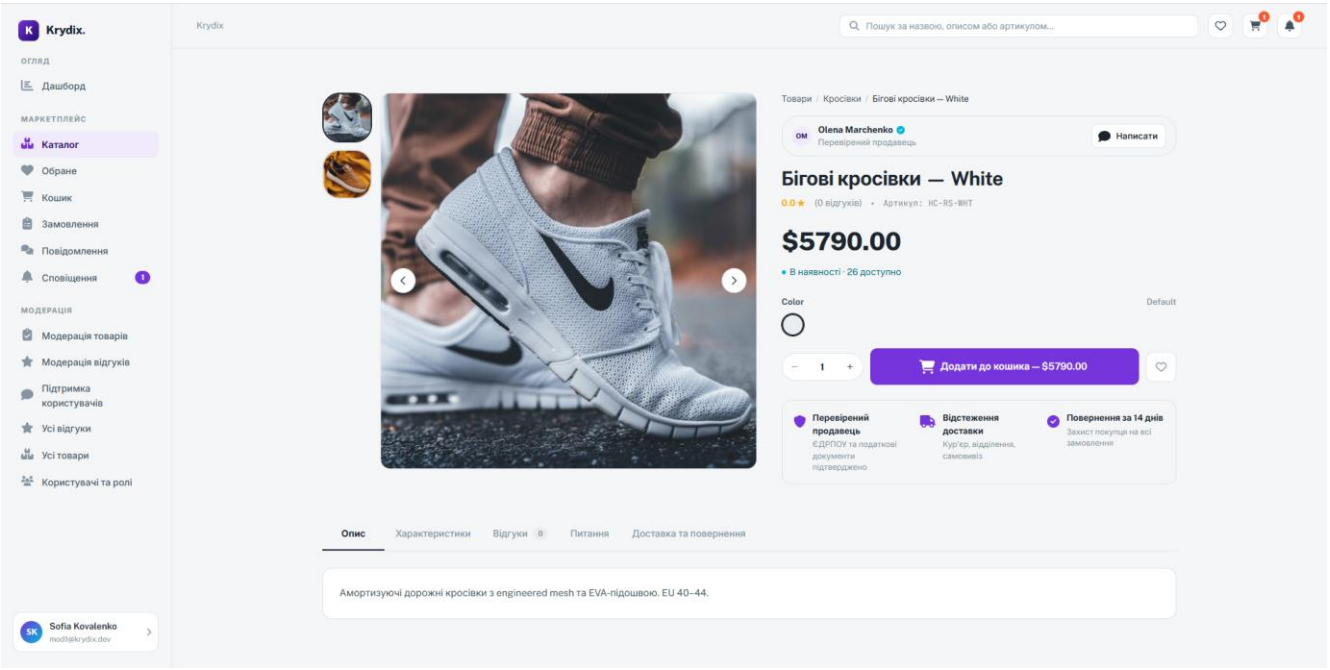



Рисунок 2.6 – Сторінка товару

Реалізація сторінки товару об'єднує всю інформацію, необхідну для прийняття рішення про покупку – від галереї до варіантів і кнопок дії – без переходу на додаткові екрани.

### 2.7.3 Модуль checkout та fee calculation

Сервіс checkoutService.ts реалізує повний потік у суворій послідовності: перевірка stock для кожного CartItem, застосування промокоду з перевіркою активності та ліміту використань, виклик feeCalculationService для обчислення комісій, атомарна транзакція Prisma для збереження всіх пов’язаних записів, зменшення stock, очищення кошика та відправлення сповіщень. Сервіс feeCalculationService.ts зчитує CommissionRule для кожної категорії товару (або загальне правило, якщо специфічного немає) і PlatformConfig для відсотків комісій, після чого повертає готовий масив snapshot–об’єктів для кожного OrderItem.

### 2.7.4 Модуль продавця

Сервіс sellerProductService.ts реалізує повний CRUD товарів, включно з управлінням варіантами ProductVariant, завантаженням медіа через Cloudinary

(upload, reorder, delete), дублюванням товару (дублікат завжди отримує статус DRAFT) та зміною статусу. Модуль bulkImportService.ts обробляє масовий імпорт у форматах .xlsx та .csv з валідацією рядків, детальним звітом про помилки та збереженням стану через модель ImportJob зі статусами PENDING → VALIDATED → PROCESSING → COMPLETED / FAILED.

Сервіс sellerDashboardService.ts агрегує аналітику: обсяг продажів за обраний період, топ–продукти за кількістю замовлень, кількість активних та завершених замовлень, сповіщення про низький залишок stock.

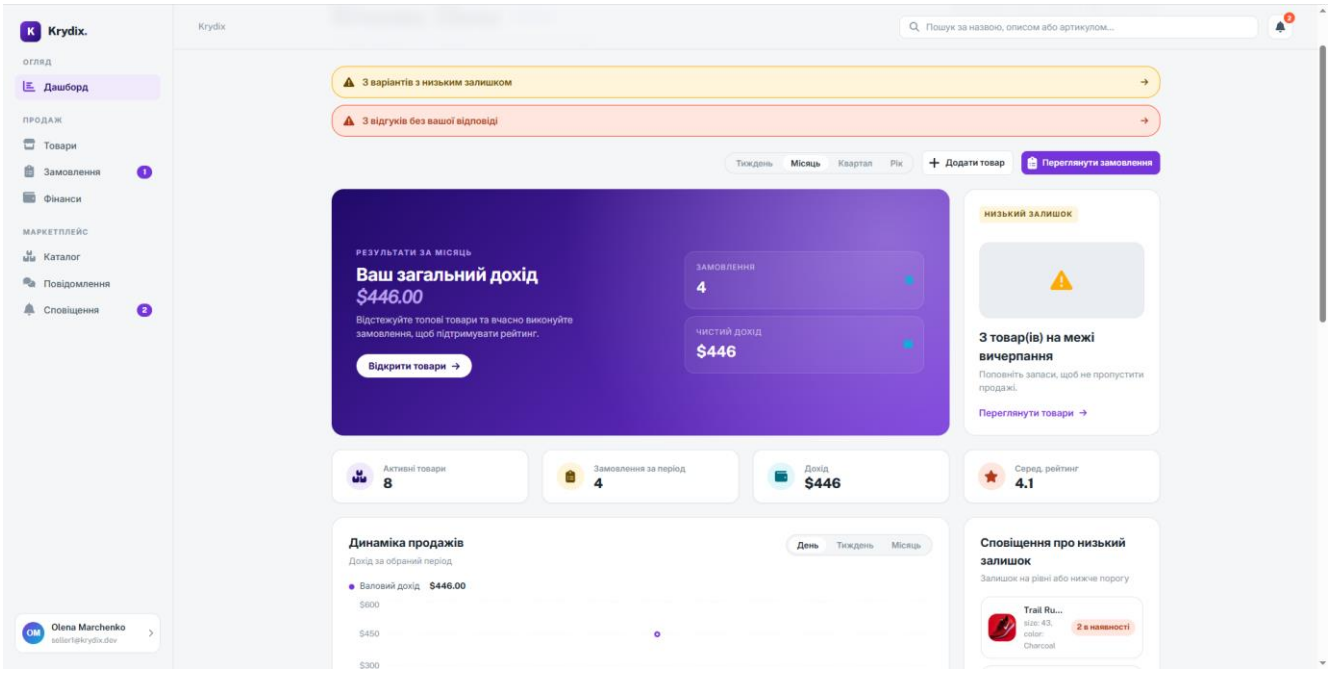



Рисунок 2.7 – Дашборд продавця

Дашборд продавця забезпечує оперативний моніторинг ключових метрик діяльності – обсяг продажів, динаміку, топ–товари – без необхідності переключатися між різними звітами.

### 2.7.5 Модуль сповіщень та чату

Сервіс notificationService.ts створює записи Notification при ключових подіях: нове замовлення, зміна статусу, нове повідомлення, результат модерації, результат верифікації, оновлення скарги. Socket.IO NotificationListener на frontend

підписується на real–time події через окремий namespace. NotificationPreference дозволяє кожному користувачу налаштувати, які з восьми типів подій він хоче отримувати.

Модуль chatSocket.ts управляє Socket.IO–підключеннями. Повідомлення зберігаються в ChatMessage з прапорами isDelivered та isRead. Система підтримує два типи чату: MARKETPLACE для комунікації покупець ↔ продавець та SUPPORT для комунікації продавець ↔ модератор. Сервіс presenceService.ts оновлює User.lastSeenAt при кожному автентифікованому запиті через GraphQL.

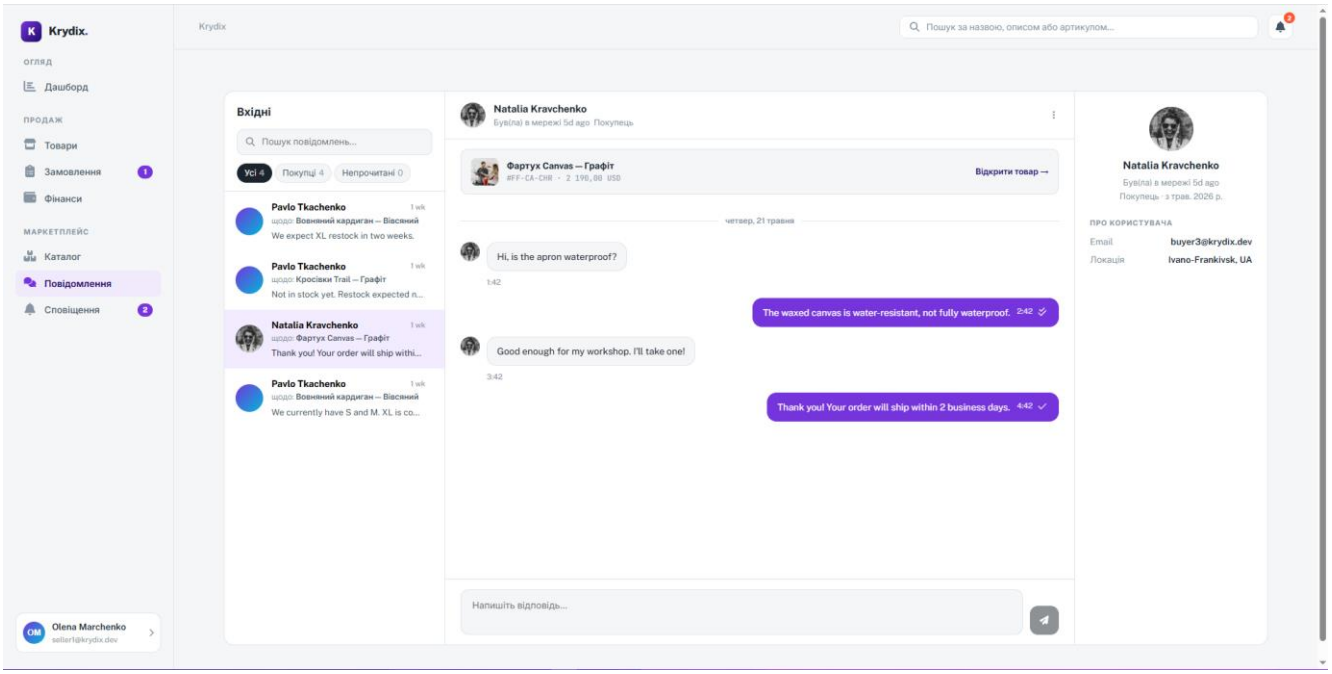



Рисунок 2.8 – Інтерфейс чату між покупцем та продавцем

Реалізований інтерфейс чату підтримує доставку повідомлень через Socket.IO та індикацію статусів прочитання, що відповідає сучасним стандартам месенджер–комунікації.

### 2.8 Інтерфейс користувача та основні екрани

Клієнтська частина системи спроєктована як SPA з єдиним деревом маршрутів, де видимість розділів визначається роллю автентифікованого

користувача. У цьому підрозділі описано систему макетів для різних ролей, бібліотеку базових UI–компонентів, що є фундаментом для всіх функціональних сторінок, підхід до теми та стилістики, а також реалізацію ключових екранів: каталогу, товару, checkout, панелей продавця, модератора та адміністратора.

### 2.8.1 Структура макету

Система використовує п’ять різних макетів, що визначаються роллю користувача. PublicLayout – мінімальний хедер без бічного меню, використовується для auth–flow сторінок. AppShell – головний макет для всіх ролей, містить верхній навбар (AppNavbar) та бічне меню (AppSidebar), що формується динамічно на основі ролі через navConfig.ts. BuyerLayout, SellerLayout, ModeratorLayout, AdminLayout – логічні контейнери всередині AppShell, що визначають доступні секції меню для кожної ролі.

### 2.8.2 Бібліотека базових UI–компонентів

Перед реалізацією будь–якої функціональної сторінки побудовано повну бібліотеку базових компонентів у frontend/src/components/ui/. Вся функціональна розробка використовує виключно ці компоненти, а не MUI–компоненти безпосередньо.

AppInput, AppTextarea, AppSelect, AppRadio, AppCheckbox – форм–компоненти з вбудованими лейблами, помилками та підказками. AppButton підтримує loading–стан, варіанти та розміри. AppModal надає модальне вікно зі слотами header/body/footer. ConfirmDialog є спеціалізованим діалогом підтвердження деструктивних дій. AppTable включає заголовок, стан завантаження та порожній стан. StatusBadge відображає кольорові позначки статусів. EmptyState відображає блок порожнього стану з іконкою та описом. AppPagination надає пагінацію з вибором розміру сторінки. StatCard використовується в dashboard для відображення числових метрик. AppToast обгортає систему toast–сповіщень. PageSectionWrapper є консистентним контейнером секції сторінки з заголовком та слотом для дій.

Таблиця 2.3 – Реалізовані базові UI–компоненти

| **Компонент** | **Призначення** |
|---|---|
| AppInput | Текстове поле з лейблом, помилкою, підказкою |
| AppButton | Кнопка з loading–станом, варіантами та розмірами |
| AppSelect | Випадаючий список з лейблом та помилкою |
| AppModal | Модальне вікно зі слотами header/body/footer |
| ConfirmDialog | Діалог підтвердження деструктивних дій |
| AppTable | Таблиця з заголовком, loading, empty state |
| StatusBadge | Кольорова позначка статусу |
| EmptyState | Блок порожнього стану з іконкою та описом |
| AppPagination | Пагінація з вибором розміру сторінки |
| StatCard | Картка статистики для dashboard |
| AppToast | Toast–сповіщення |
| PageSectionWrapper | Секція сторінки з заголовком та слотом дій |

### 2.8.3 Теми та стилістика

Тема MUI визначена в frontend/src/theme/index.ts через createTheme(). Стиль орієнтований на мінімалістичний дашборд – чисті шрифти, м’які тіні, нейтральна кольорова палітра, чіткий контраст. Глобальні змінні стилів визначені в styles/_variables.scss; міксини – в styles/_mixins.scss. Іконкова система побудована виключно на FontAwesome React; всі іконки централізовані в constants/icons.ts – будь–яка зміна іконки відбувається в одному місці.

### 2.8.4 Ключові сторінки

CatalogPage відображає список товарів у вигляді сітки карток, панель фільтрів CatalogFilters із деревом категорій CategoryFilterTree, слайдер ціни RangeSlider та пагінацію. Стан фільтрів синхронізується з URL–параметрами, що дозволяє ділитися посиланнями на відфільтровані результати.

ProductPage відображає галерею зображень з lightbox, опис, варіанти товару, вибір кількості, кнопки «Купити» та «До списку бажань», секцію відгуків ProductReviewsSection та ініціацію чату з продавцем.

CheckoutPage містить форму контактних даних, вибір способу доставки та оплати, поле промокоду та підсумок замовлення – всі поля валідуються через React Hook Form + Zod з миттєвим feedback.

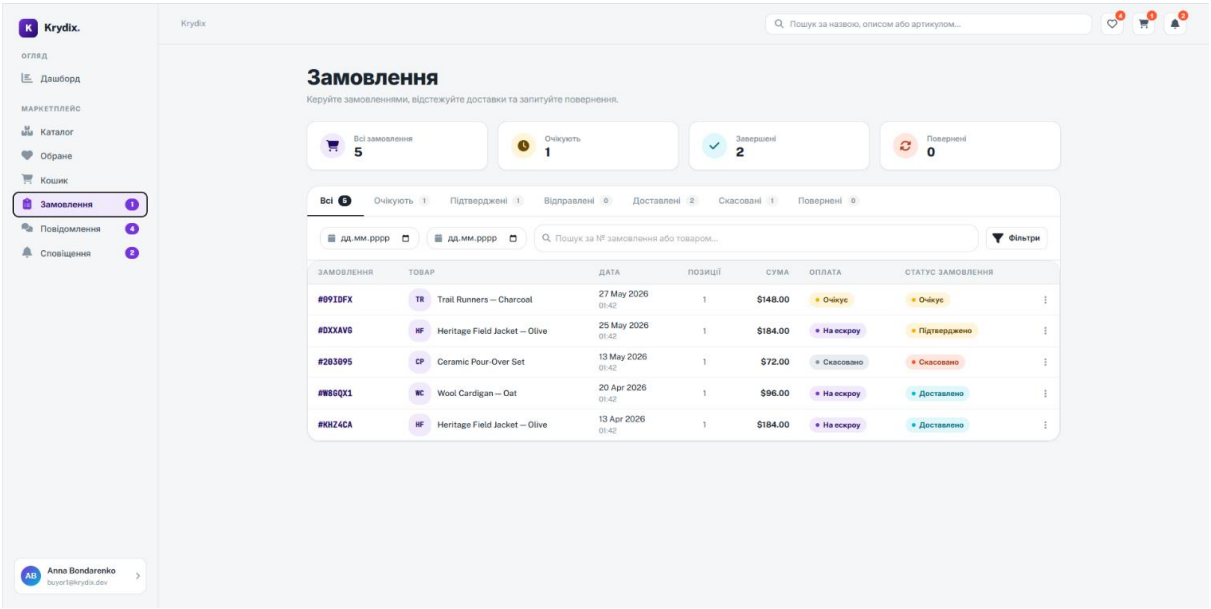



Рисунок 2.9 – Сторінка замовлень покупця з фільтрацією та статусами

SellerDashboardPage відображає метрики продажів через StatCard, графік динаміки через Recharts, топ–товари та швидкі дії. ProductModerationPage та ReviewModerationPage надають модераторам таблицю записів з фільтрацією за статусом та модальні вікна для дій. Панель адміністрування складається з кількох незалежних сторінок: UsersManagementPage, AdminCategoriesPage з деревом категорій, AdminAuditPage з журналом дій та AdminPlatformPage з конфігурацією платформи.

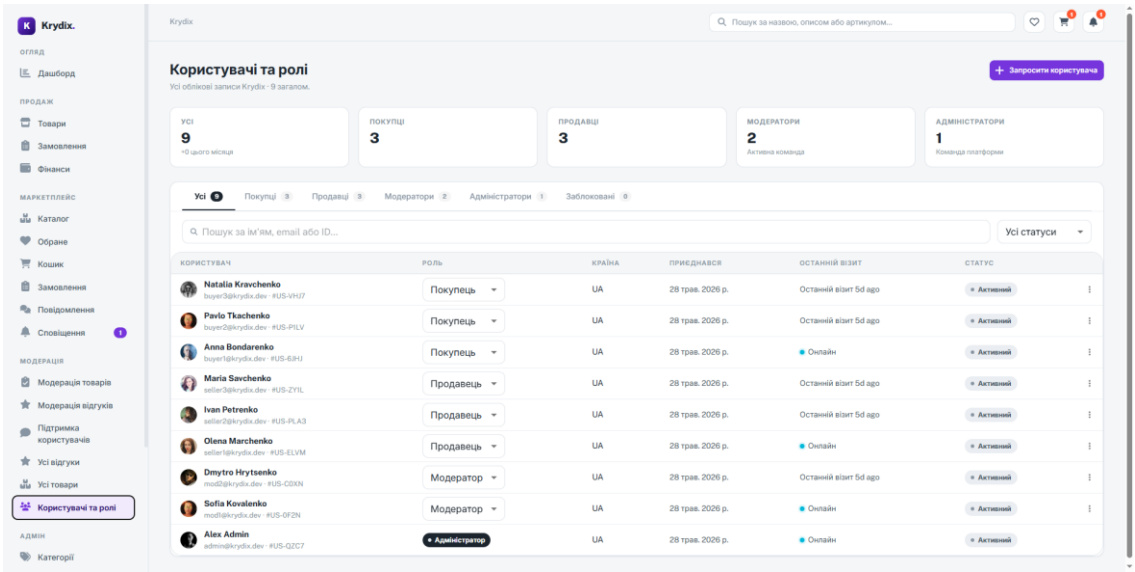



Рисунок 2.10 – Адміністративна панель: управління користувачами та їх ролями

Адміністративна панель управління користувачами демонструє реалізацію матриці прав, описаної в розділі 1, з можливістю зміни ролі та блокування облікового запису безпосередньо з таблиці.

### 2.8.5 Інтернаціоналізація UI

Всі рядки інтерфейсу реалізовані через i18n–ключі за допомогою react–i18next. Переклади зберігаються у frontend/src/i18n/locales/en/common.json та frontend/src/i18n/locales/uk/common.json. Перемикач мови розміщений у верхньому навбарі; мова зберігається в localStorage між сесіями. URL маршрути не змінюються при зміні мови – SPA–режим без locale–based routes.

## 2.9 Система контролю версій

Розробка системи Krydix велася з використанням розподіленої системи контролю версій Git та хостингу репозиторіїв GitHub. Така комбінація є де–факто стандартом у сучасній індустрії розробки веб–застосунків та забезпечує одночасну роботу кількох учасників, фіксацію кожної зміни в історії та можливість безпечного експериментування у feature–гілках без впливу на основну гілку.

Структура гілок проєкту дотримується спрощеної моделі GitHub Flow. Основна гілка main містить виключно стабільний код, готовий до розгортання у виробниче середовище. Для кожної функціональної можливості створювалася окрема feature–гілка з префіксом feature/ (наприклад, feature/fee–snapshots, feature/seller–payouts), у якій вівся розвиток конкретного домену. Після завершення роботи feature–гілка проходила code review через механізм Pull Request, після чого виконувалося злиття у develop. Кожен Pull Request обов'язково містив посилання на відповідний GitHub Issue, що забезпечувало прозоре трасування «вимога → реалізація → перевірка».

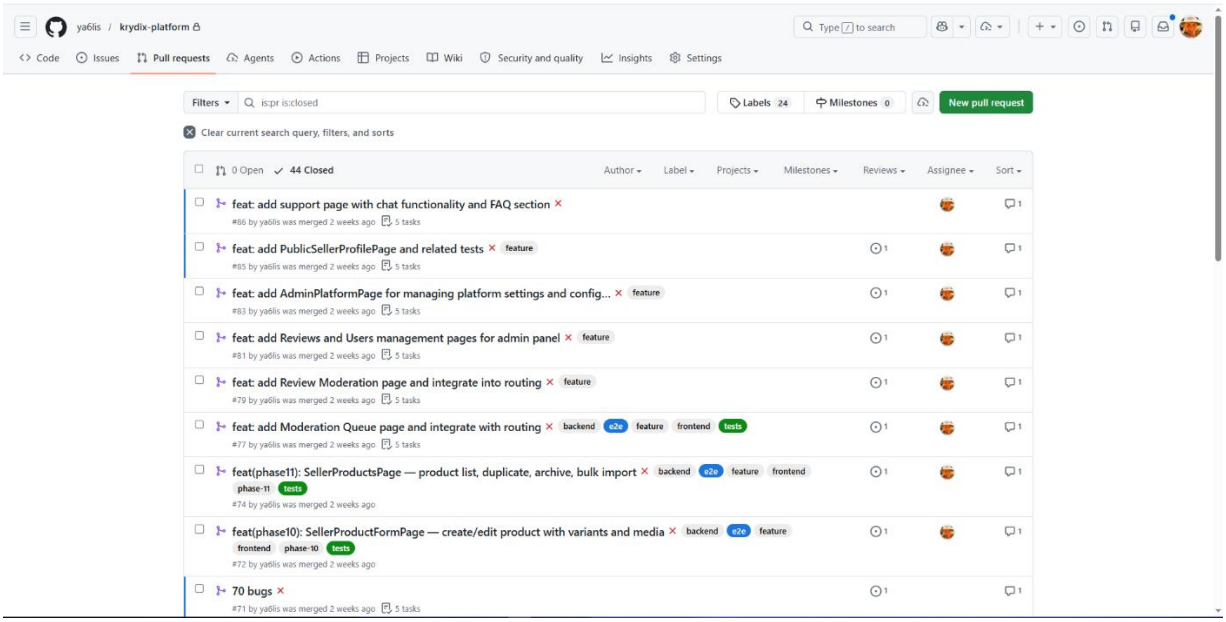



Рисунок 2.11 – Сторінка Pull Requests репозиторію Krydix у GitHub

Дерево комітів проєкту демонструє послідовний розвиток системи: початкові коміти присвячені налаштуванню монорепозиторію та конфігурації, далі – створенню Prisma–схеми та базових сутностей, потім – реалізації автентифікації, каталогу, кошика, checkout, модерації, виплат, чату та адміністративної панелі. Загальна кількість комітів у репозиторії перевищує кілька сотень; середній розмір коміту відповідає одній логічно завершеній зміні, що спрощує перегляд історії та відкат у разі регресій.

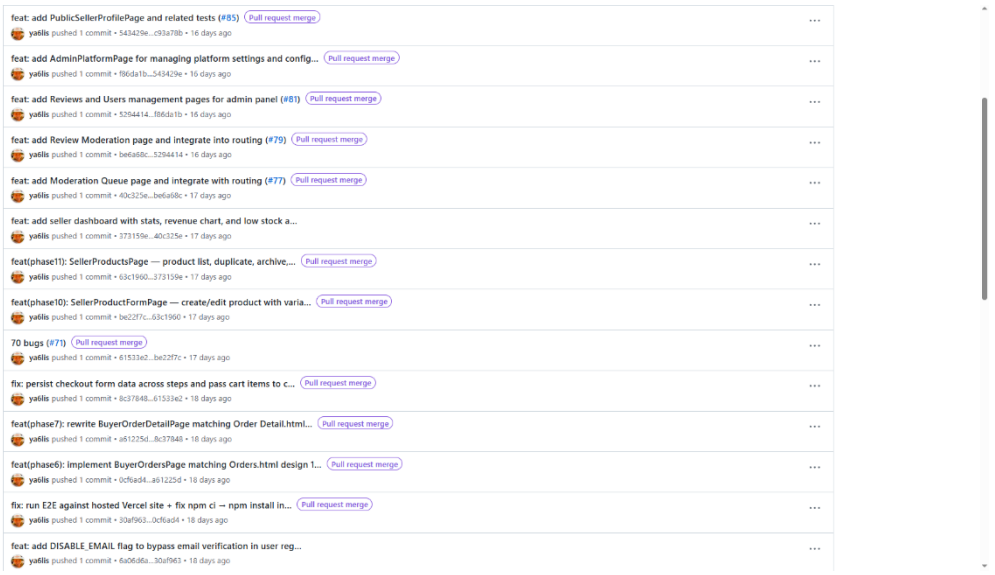



Рисунок 2.12 – Дерево комітів репозиторію Krydix (візуалізація гілок та злиттів)

## 2.10 Висновок до розділу

У другому розділі описано архітектурні рішення системи Krydix. Обрано інкрементальну модель розробки з чіткою модульною структурою та обґрунтовано вибір кожного компонента технологічного стеку. Спроєктовано схему бази даних із понад 30 моделями з підтримкою багатомовності через таблиці перекладів. GraphQL API зібраний з 20 доменних модулів із суворим розподілом відповідальностей між шарами resolver – service – repository. Реалізовано клієнтський SPA–застосунок на React 18 з MUI 6 та повною бібліотекою базових UI–компонентів. Описано організацію системи контролю версій на основі GitHub Flow з трасуванням кожної зміни через Pull Request та GitHub Issues. Ключовим архітектурним рішенням є механізм fee snapshots в OrderItem, що гарантує незмінність фінансових даних незалежно від подальших змін налаштувань платформи.

# РОЗДІЛ 3 ТЕСТУВАННЯ, РОЗГОРТАННЯ ТА ПІДТРИМКА

У цьому розділі описано стратегію тестування платформи Krydix, середовище та процедуру розгортання, а також проведено верифікацію відповідності реалізованої системи функціональним і нефункціональним вимогам, сформульованим у першому розділі.

## 3.1 Підходи до тестування системи

Якість програмного продукту неможливо гарантувати без системного підходу до тестування. У цьому підрозділі описано загальну стратегію тестування платформи Krydix, що базується на трирівневій піраміді тестування, а також конкретні інструменти та методики, застосовані для unit/integration–тестів (Jest) та E2E–тестів (Cypress), включно з підходом до мокування зовнішніх сервісів.

### 3.1.1 Загальна стратегія тестування

Тестування системи Krydix організоване за трирівневою пірамідою тестування, яка передбачає переважну більшість unit–тестів, меншу кількість integration–тестів та обмежений набір E2E–тестів для критичних сценаріїв [9]. Така пірамідальна структура забезпечує оптимальний баланс між швидкістю виконання тестів та ступенем впевненості у правильності роботи системи. Unit–тести тестують ізольовані одиниці коду без залежностей від зовнішніх систем, integration–тести – взаємодію між компонентами зі спрощеними замінниками залежностей, E2E–тести – повні користувацькі сценарії від UI до мокованого API.

Принцип «тести пишуться одразу після реалізації функціональності» застосовується протягом усього процесу розробки, що дозволяє уникати накопичення технічного боргу з тестуванням і виявляти регресії одразу після їх появи.

### 3.1.2 Unit та integration тести (Jest)

Інструментальна база unit–тестування включає Jest 29, ts–jest (TypeScript–підтримка), @testing–library/react для тестування React–компонентів та @testing–library/jest–dom для розширених matchers. Backend–тести розміщені в піддиректоріях __tests__ поряд із основним кодом: 28 тестових файлів для сервісів у backend/src/services/__tests__/, по 3 файли для репозиторіїв і валідаторів, ще 2 – для утиліт. Кожен backend–сервіс тестується з мокуванням репозиторіїв через jest.mock(), що дозволяє перевірити бізнес–логіку сервісу ізольовано від бази даних.

Frontend–тести охоплюють понад 30 тестових файлів для базових UI–компонентів у frontend/src/components/ui/__tests__/, тести для компонентів каталогу (ProductCard, CatalogFilters), для сторінок покупця, продавця та публічних сторінок, для cartStore та утиліт roleAccess, authSession, notificationUtils. Тестування React–компонентів через @testing–library/react орієнтоване на user–centric підхід: тести перевіряють поведінку, яку бачить і відчуває користувач, а не деталі реалізації. Приклад unit–тесту з мокуванням репозиторію наведено в додатку Є.

### 3.1.3 E2E–тести (Cypress)

E2E–тести організовані у e2e/cypress/e2e/. Всі 22 тести використовують fixture–based підхід: GraphQL–запити перехоплюються через cy.intercept() та повертають заздалегідь підготовлені JSON–фікстури з e2e/cypress/fixtures/. Це дозволяє тестувати UI без запуску реального backend і бази даних, забезпечуючи детерміновані та відтворювані результати.

Файл auth.cy.ts охоплює повний цикл автентифікації: реєстрацію з валідними та невалідними даними, верифікацію email з валідним та недійсним токеном, вхід, захищені маршрути та вихід. Файл catalog.cy.ts перевіряє відображення каталогу, фільтрацію за категорією та ціною, пагінацію та порожній стан. Файл cart.cy.ts охоплює додавання товарів, зміну кількості, видалення та перехід до оформлення замовлення. Файл orders.cy.ts перевіряє список та деталі замовлень покупця. Файл productReviews.cy.ts тестує написання, відображення та рейтинг відгуків.

Файли moderation.cy.ts та reviewModeration.cy.ts перевіряють відповідно модерацію товарів та відгуків. Файли sellerProducts.cy.ts і sellerProductsList.cy.ts тестують CRUD товарів продавця. Файл sellerDashboard.cy.ts перевіряє відображення аналітики. Файл importExport.cy.ts тестує завантаження xlsx–файлу, попередній перегляд рядків та підтвердження імпорту, а також експорт у форматі xlsx.

Файли adminUsers.cy.ts, adminProducts.cy.ts, adminReviews.cy.ts, adminCategories.cy.ts, adminAudit.cy.ts, adminPlatform.cy.ts охоплюють усі адміністративні функції. Файл chat.cy.ts тестує ініціацію чату та відправлення повідомлень. Файл publicSellerProfile.cy.ts перевіряє публічний профіль продавця. Файл shell.cy.ts тестує навбар, перемикач мови та навігацію між розділами.

### 3.1.4 Мокування зовнішніх сервісів

У тестах зовнішні залежності замінюються on мокованими реалізаціями. Платіжна логіка є mock–реалізацією – PaymentRecord створюється зі статусом PAID без реального виклику PSP. Доставка мокується через генерацію довільного трекінг–коду без зворотнього зв’язку з реальним перевізником. Email–відправлення мокується через jest.mock('../utils/email') у backend–тестах. Cloudinary SDK мокується при тестуванні завантаження медіа. Socket.IO у unit–тестах замінюється mock event emitter, що дозволяє перевіряти логіку сповіщень без реального WebSocket–з’єднання.

## 3.2 Середовище розгортання та системні вимоги

Для забезпечення стабільної роботи системи у виробничому середовищі необхідно чітко визначити технічні вимоги до серверної та клієнтської частин, а також описати процедуру розгортання. Цей підрозділ охоплює мінімальні та рекомендовані системні вимоги, конфігурацію середовища через змінні оточення, схему розгортання backend на Render та frontend на Vercel, а також організацію CI/CD через GitHub Actions.

### 3.2.1 Системні вимоги

Backend–сервер потребує Node.js 20 LTS як мінімум (22 LTS рекомендовано), не менше 512 MB RAM (1–2 GB рекомендовано), PostgreSQL 14 або вище та Linux як операційну систему. Клієнтський застосунок підтримує Chrome 100+, Firefox 100+, Safari 16+ та Edge 100+, з підтримкою ES2020 та WebSocket; дозвіл екрану від 320px (mobile) до 2560px (4K desktop). Для повного функціонування системи необхідні: managed PostgreSQL, Cloudinary (медіа), Gmail SMTP (email), а також хостинг для backend і frontend.

Таблиця 3.1 – Системні вимоги до серверного середовища

| **Параметр** | **Мінімум** | **Рекомендоване** |
|---|---|---|
| Node.js | 20 LTS | 22 LTS |
| RAM | 512 MB | 1–2 GB |
| CPU | 1 vCPU | 2 vCPU |
| PostgreSQL | 14 | 15–16 |
| ОС | Linux (Ubuntu 22.04+) | Linux (Ubuntu 24.04 LTS) |

### 3.2.2 Конфігурація середовища

Конфігурація backend зберігається у .env та зчитується через backend/src/config/env.ts. Обов’язкові змінні: DATABASE_URL та DIRECT_URL (два рядки підключення PostgreSQL – через pgBouncer для трафіку та пряме для міграцій), JWT_SECRET та JWT_REFRESH_SECRET, CLOUDINARY_CLOUD_NAME / _API_KEY / _API_SECRET, GMAIL_USER та GMAIL_PASS, FRONTEND_URL для CORS та PORT для HTTP–сервера. Frontend налаштовується через VITE_GRAPHQL_URL та VITE_SOCKET_URL.

### 3.2.3 Розгортання backend (Render)

Розгортання серверної частини виконано на платформі Render (Node.js Web Service) з конфігурацією у файлі render.yaml. При кожному push у гілку main Render автоматично виконує послідовність кроків: завантаження вихідного коду з GitHub, встановлення npm–залежностей, генерацію Prisma Client (npx prisma generate), компіляцію TypeScript (npm run build), застосування міграцій бази даних (npx

prisma migrate deploy) та запуск production–сервера через node dist/index.js. Для роботи системи в Render налаштовано набір змінних середовища (.env–конфігурація), що включає рядки підключення до PostgreSQL (DATABASE_URL і DIRECT_URL), секрети JWT (JWT_SECRET, JWT_REFRESH_SECRET), облікові дані Cloudinary (CLOUDINARY_CLOUD_NAME, CLOUDINARY_API_KEY, CLOUDINARY_API_SECRET), параметри Gmail SMTP та дозволені CORS–origins. Render забезпечує автоматичний перезапуск сервісу при аварійному завершенні та health check ендпоінт GET /health.

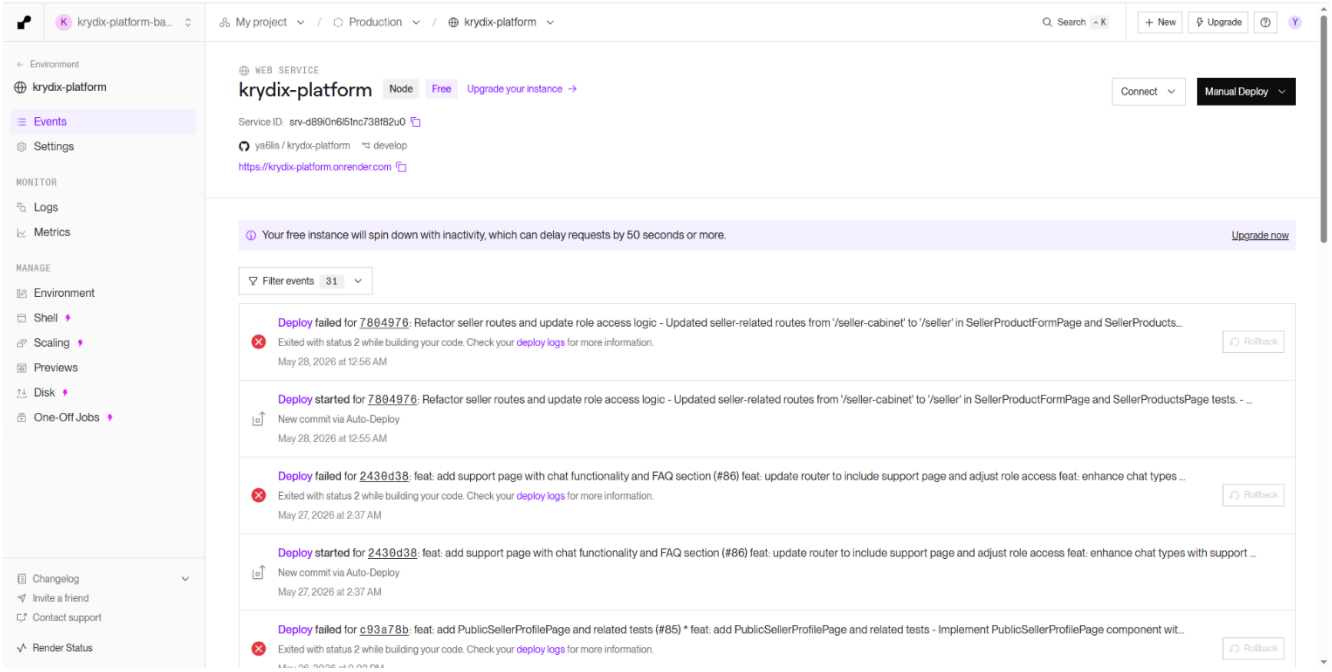



Рисунок 2.13 – Dashboard платформи Render: статус серверного сервісу Krydix

### 3.2.4 Розгортання frontend (Vercel)

Розгортання клієнтської частини виконано на платформі Vercel з конфігурацією у vercel.json. Vercel автоматично збирає Vite–проєкт (npm run build), розгортає статичні файли на власному CDN та налаштовує SPA–роутинг через redirect rule, що перенаправляє будь–який невідомий шлях на index.html. Окремою перевагою Vercel є генерація preview deployments для кожного Pull Request, що дозволяє переглянути зміни перед злиттям у основну гілку.

### 3.2.5 CI/CD через GitHub Actions

Сучасні вебзастосунки зі складною багаторівневою архітектурою – серверна частина, клієнтський застосунок, база даних і сервіси реального часу – роблять ручне тестування та розгортання неефективним і ризикованим [27]. Методологія CI/CD (Continuous Integration / Continuous Deployment) забезпечує автоматизацію процесів перевірки, складання та доставки змін до виробничого середовища, суттєво підвищуючи надійність і швидкість випуску нових версій. Типовий процес автоматизації включає статичний аналіз коду, автоматичне тестування та доставку артефактів; інструменти на кшталт GitHub Actions дозволяють описати ці етапи декларативно у вигляді YAML–конфігурацій, що запускаються автоматично при кожному коміті або pull request [27].

У системі Krydix CI/CD конфігурований через GitHub Actions. Pipeline складається з трьох послідовних кроків, що запускаються при кожному push та Pull Request: статичний аналіз коду (npm run lint), перевірка типів TypeScript (npm run typecheck) та запуск Jest unit/integration тестів (npm run test). У разі невдачі будь–якого з кроків злиття у main блокується. При успішному злитті в main спрацьовують webhook–и Render і Vercel, що ініціюють автоматичне розгортання нової версії в обох середовищах. Така схема забезпечує скорочення часу від коміту до виробничого розгортання з кількох годин ручної роботи до 5–7 хвилин автоматизованого pipeline.

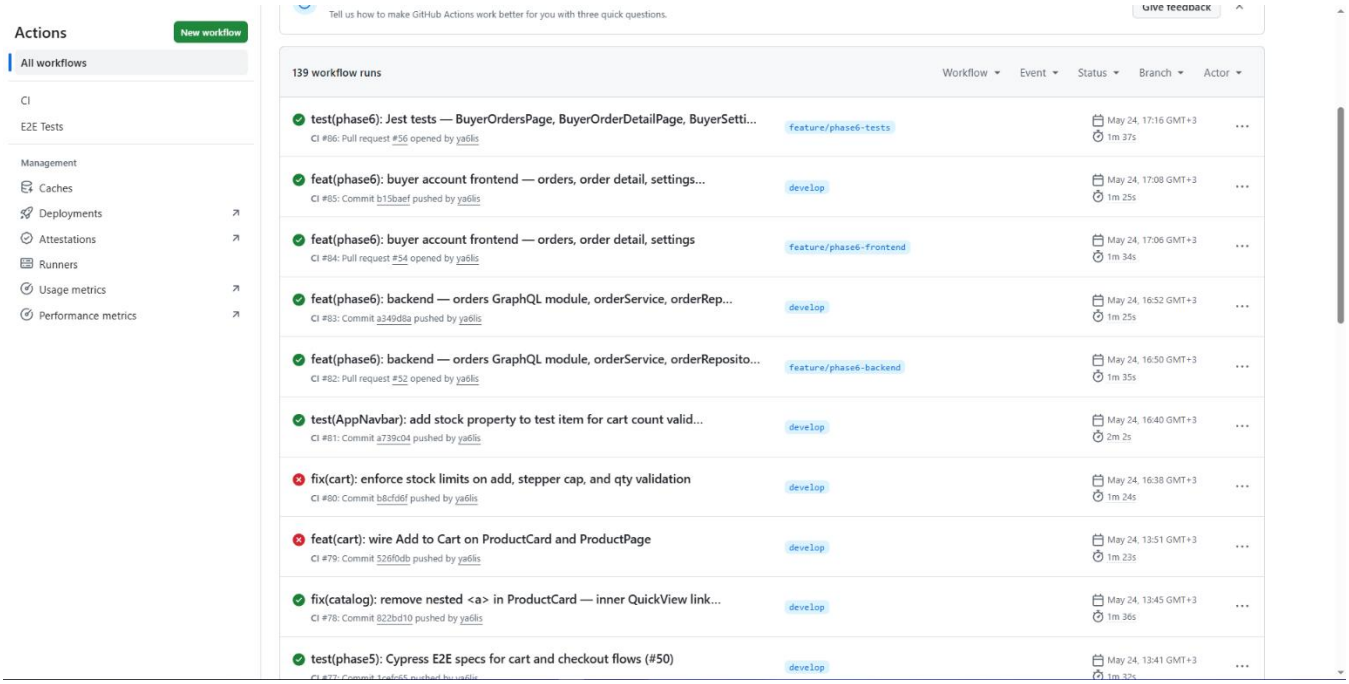



Рисунок 2.14 – Виконання CI-pipeline у GitHub Actions для Pull Request

## 3.3 Перевірка відповідності вимогам

Завершальним етапом розробки є верифікація – підтвердження того, що реалізована система дійсно відповідає вимогам, сформульованим у першому розділі. У цьому підрозділі перевіряється відповідність функціональним вимогам через аналіз тестового покриття, нефункціональним вимогам – через перевірку безпеки, надійності та локалізації, а також фіксуються відомі обмеження MVP.

### 3.3.1 Верифікація функціональних вимог

Перевірка відповідності функціональним вимогам, визначеним у розділі 1, здійснюється шляхом аналізу тестового покриття.

Вимога до каталогу і пошуку підтверджена тестами catalog.cy.ts (відображення, фільтрація, пагінація, порожній стан) та catalogService.test.ts (логіка пошуку та фільтрації на рівні сервісу). Автентифікаційні вимоги підтверджені повним E2E–тестом auth.cy.ts та unit–тестами authService.test.ts (bcrypt, JWT, ротація токенів). Вимоги до покупця верифіковані тестами cart.cy.ts, checkoutService.test.ts, orderService.test.ts та orders.cy.ts. Вимоги до продавця

підтверджені тестами sellerProducts.cy.ts, sellerProductService.test.ts, importExport.cy.ts, bulkImportService.test.ts та sellerDashboard.cy.ts. Модераційні вимоги верифіковані файлами moderation.cy.ts, reviewModeration.cy.ts та moderationService.test.ts.

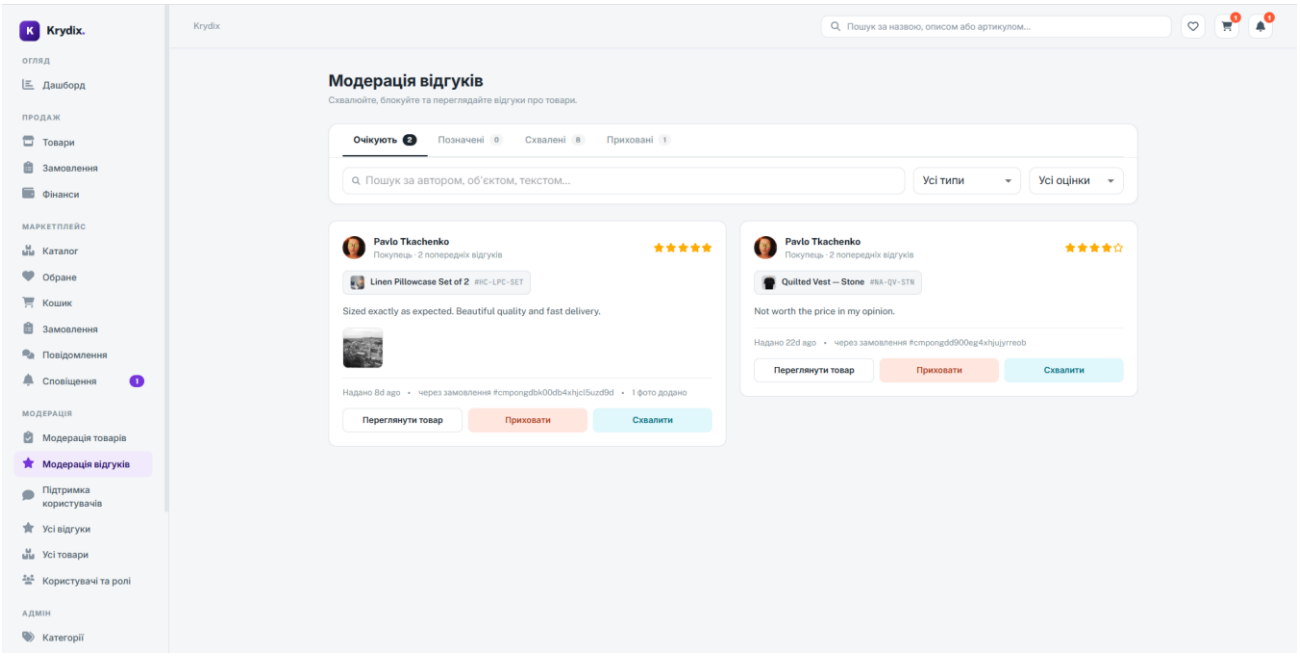



Рисунок 3.1 – Модерація відгуків: таблиця з можливістю схвалення та блокування

Адміністративні вимоги підтверджені тестами adminUsers.cy.ts, adminCategories.cy.ts, adminPlatform.cy.ts та adminAudit.cy.ts у поєднанні з відповідними unit–тестами adminService.test.ts, platformService.test.ts та adminAuditService.test.ts.

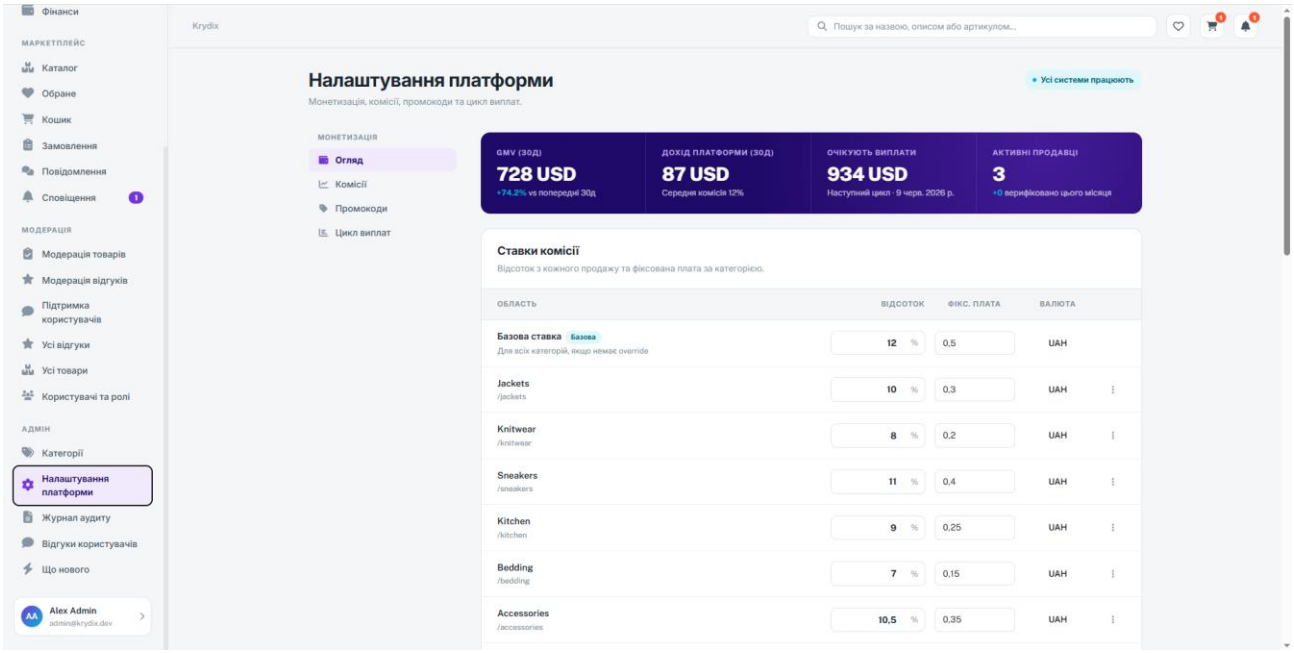



Рисунок 3.2 – Адміністративна панель: налаштування платформи (комісії, таймери виплат)

Інтерфейс налаштувань платформи надає адміністратору повний контроль над параметрами комісій та таймерів виплат без необхідності прямого доступу до бази даних, що знижує ризик некоректних змін у виробничому середовищі.

### 3.3.2 Верифікація нефункціональних вимог

Безпека системи підтверджена на кількох рівнях. authService.test.ts перевіряє, що пароль не зберігається у відкритому вигляді – bcrypt.compare() повертає true, а passwordHash не збігається з plain text. authValidators.test.ts перевіряє Zod–валідацію граничних значень (мінімальна довжина пароля, формат email). Rate limiting налаштований для /graphql–ендпоінту через express–rate–limit. CORS обмежений списком дозволених origins з FRONTEND_URL.

Надійність фінансових даних підтверджена тестом feeCalculationService.test.ts: після зміни PlatformConfig значення *Snapshot полів у вже існуючих OrderItem залишаються незмінними – перевірка реалізована через мокування репозиторію та порівняння зафіксованих значень. adminAuditService.test.ts перевіряє, що запис в AuditLog формується при кожній

модераційній та фінансовій дії з коректними полями actorId, action, targetType та targetId.

Вимога локалізації підтверджена shell.cy.ts: перемикання мови відображає правильні переклади для всіх елементів інтерфейсу, а notificationUtils.test.ts перевіряє коректність i18n–ключів у сповіщеннях.

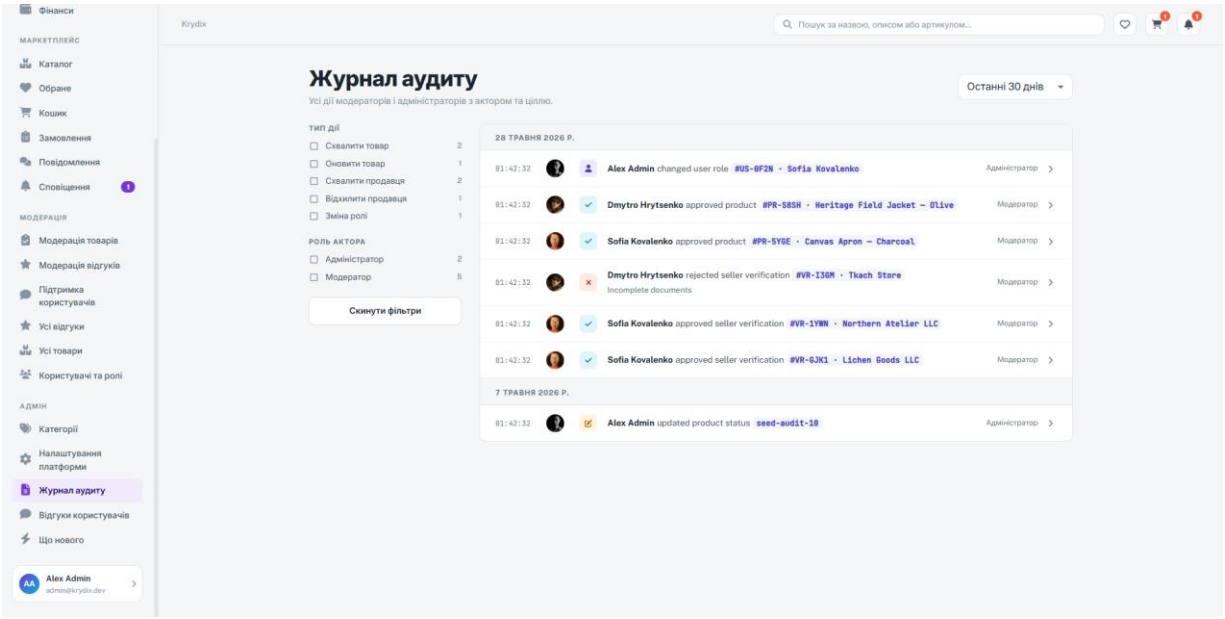



Рисунок 3.3 – Сторінка аудит журналу

Сторінка аудит журналу підтверджує реалізацію вимоги до in–app каналу комунікації для всіх восьми типів подій, визначених у NotificationEvent, з можливістю індивідуального налаштування через NotificationPreference.

### 3.3.3 Результати тестування

Нижче наведено зведену таблицю результатів тестування за основними функціональними модулями.

Таблиця 3.2 – Результати тестування за модулями

| **Модуль** | **Unit/Integration** | **E2E** | **Статус** |
|---|---|---|---|
| Автентифікація | authService.test.ts | auth.cy.ts | Пройдено |
| Каталог | catalogService.test.ts | catalog.cy.ts | Пройдено |

Продовження таблиці 3.2

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| Товар продавця | sellerProductService.test.ts | sellerProducts.cy.ts | Пройдено |
| Кошик | cartRepository.test.ts | cart.cy.ts | Пройдено |
| Checkout | checkoutService.test.ts | cart.cy.ts | Пройдено |
| Замовлення | orderService.test.ts | orders.cy.ts | Пройдено |
| Відгуки | reviewService.test.ts | productReviews.cy.ts | Пройдено |
| Модерація товарів | moderationService.test.ts | moderation.cy.ts | Пройдено |
| Модерація відгуків | reviewModerationService.test.ts | reviewModeration.cy.ts | Пройдено |
| Адмін: користувачі | adminService.test.ts | adminUsers.cy.ts | Пройдено |
| Адмін: категорії | categoryService.test.ts | adminCategories.cy.ts | Пройдено |
| Адмін: платформа | platformService.test.ts | adminPlatform.cy.ts | Пройдено |
| Аудит | adminAuditService.test.ts | adminAudit.cy.ts | Пройдено |
| Seller Dashboard | sellerDashboardService.test.ts | sellerDashboard.cy.ts | Пройдено |
| Імпорт/Експорт | bulkImportService.test.ts | importExport.cy.ts | Пройдено |
| Чат | chatService.test.ts | chat.cy.ts | Пройдено |
| Виплати (payout) | payoutService.test.ts | – | Пройдено (unit) |
| Fee calculation | feeCalculationService.test.ts | – | Пройдено (unit) |
| Сповіщення | notificationService.test.ts | – | Пройдено (unit) |

### 3.3.4 Відомі обмеження

Під час розробки MVP свідомо залишено у вигляді заглушки <Placeholder> ряд функціональних можливостей, що зафіксовано в frontend/src/router/index.tsx. Сторінка пошуку (/search), відновлення та скидання пароля (/forgot–password, /reset–password), UI–форма верифікації продавця (/seller/verification), UI для скарг покупців (/complaints) та список власних відгуків покупця (/my–reviews) відображають повідомлення «Coming soon» і заплановані до реалізації в наступній ітерації. Слід зазначити, що backend–логіка для верифікації продавців і скарг реалізована повністю – відсутнє лише клієнтське UI.

### 3.4 Висновок до розділу

У третьому розділі описано стратегію тестування системи Krydix за трирівневою пірамідою: понад 30 unit/integration–файлів на Jest та 22 E2E–файли на Cypress із fixture–based мокуванням GraphQL. Визначено системні вимоги до серверного та клієнтського середовища. Описано схему розгортання: backend на Render, frontend на Vercel, PostgreSQL як managed–сервіс. Проведена верифікація підтверджує, що всі критичні бізнес–потоки MVP покриті тестами та проходять успішно. Задокументовані відомі обмеження щодо нереалізованих елементів UI при повністю реалізованому backend.

# РОЗДІЛ 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

Розробка та супровід багатомодульного маркетплейс–додатку Krydix відбувається переважно за персональним комп’ютером з кольоровим монітором, з тривалим перебуванням розробника у статичному робочому положенні при високому інтелектуальному та зоровому навантаженні. Це створює сукупність ризиків, що регламентуються українськими нормативно–правовими актами та міжнародними стандартами ергономіки. Мета розділу – проаналізувати актуальність безпеки життєдіяльності у сучасних інформаційних професіях і сформулювати гігієнічні вимоги до організації та обладнання робочого місця з відеодисплейним терміналом (ВДТ).

Рисунок 4.1 – Робоче місце розробника вебзастосунку Krydix.

### 4.1 Безпека життєдіяльності

Безпека життєдіяльності (БЖД) – міждисциплінарна галузь, спрямована на збереження життя, здоров'я та працездатності людини у системі «людина – техніка – середовище» [38]. За останні десятиліття структура ризиків змінилася: на зміну переважно фізичним чинникам прийшли інформаційні, психологічні та ергономічні, особливо помітні у сфері IT. Для проєкту Krydix, команда якого працює переважно дистанційно за персональним комп'ютером, актуальними є такі групи ризиків: фізичні (тривала статична поза, зорове навантаження, неоптимальне освітлення, шум, мікрокліматичні відхилення), психосоціальні (інформаційне перевантаження, стрес від дедлайнів і релізів, синдром вигорання), інформаційні (фішинг, помилки конфігурування доступу, витік .env–файлів) та гіподинамія у разі повністю дистанційного режиму.

В Україні питання БЖД на робочому місці регулюються Законом України «Про охорону праці» [29], Кодексом законів про працю, постановами Кабінету Міністрів та галузевими нормативами МОЗ і Держпраці. Для IT–команд додаються вимоги безпечного оброблення персональних даних користувачів, верифікаційних документів продавців і банківських реквізитів – що частково перетинається з GDPR–подібними практиками.

Профілактика ризиків формується на трьох рівнях: технічному (ергономічне обладнання, монітор з низьким мерехтінням, регульоване крісло, належний мікроклімат), організаційному (чіткий графік праці і відпочинку, перерви на гімнастику, ротація задач, відсутність нічних релізів без компенсації) та особистісному (здорові звички, контроль режиму сну і фізичної активності). Для команди Krydix специфічними організаційними практиками є: автоматизація рутинних дій через CI/CD, що зменшує когнітивне навантаження і знижує ризик помилки при ручному розгортанні; журнал аудиту (AuditLog) та ескроу–логіка платежів, що дозволяють уникати поспішних рішень із критичними наслідками; окремий ChatType.SUPPORT для ізоляції інцидентів і зменшення кількості переключень між задачами.

Сучасна модель БЖД враховує надзвичайні ситуації техногенного, природного і соціально–політичного характеру. Для команди Krydix критично важливі: план реагування на виробничі інциденти, резервне копіювання репозиторію та БД у незалежних географічних регіонах (базова функція Neon), готовність до зміни локації роботи та комунікаційні протоколи на випадок відключення електропостачання чи Інтернету.

### 4.2 Гігієнічні вимоги до робочих місць з ВДТ

Робота розробника вебзастосунку Krydix належить до категорії робіт з відеодисплейними терміналами (ВДТ). Особливості охорони праці користувачів комп'ютерів – від ергономіки робочого місця до профілактики комп'ютерного зорового синдрому та синдрому зап'ястного каналу – системно викладені у фаховому підручнику [37]. Основоположним нормативним документом є ДСанПіН 3.3.2.007–98 [30], а також міжнародні стандарти серії ДСТУ ISO 9241 [36]. Правові основи закріплені у НПАОП 0.00–7.15–18 [31].

Щодо площі та розміщення робочого місця: згідно з нормативами, площа на одне робоче місце з ВДТ – не менше 6 м², об’єм приміщення – не менше 20 м³ на особу, відстань між тильними поверхнями сусідніх моніторів – не менше 2 м, бічна відстань між ними – не менше 1,2 м. Віконні прорізи розміщуються збоку від робочого місця для уникнення прямих відблисків. Робочий стіл повинен мати висоту 680–800 мм (регульована або фіксована), ширину не менше 1000 мм, матову поверхню. Ергономічне крісло з регульованою висотою сидіння 400–500 мм, нахилом спинки, поперековою підтримкою і м’якими підлокітниками є обов’язковим елементом. Монітор діагоналлю не менше 19 дюймів із частотою оновлення 60+ Гц розміщується так, щоб верхній край був на рівні очей на відстані 600–700 мм від обличчя. Клавіатура має бути відокремленою від монітора, з нахилом 5–15°.

Освітлення регулюється ДБН В.2.5–28:2018 [33]. Рівень загальної освітленості – 300–500 лк для офісних робіт інтелектуального типу; освітленість

безпосередньо на поверхні екрана – не більше 300 лк. Джерела світла розміщуються паралельно лінії зору, а не позаду чи перед екраном, із захисним кутом не менше 40° і кольоровою температурою 3500–4500 K. Мікроклімат регулюється ДСН 3.3.6.042–99 [34]: для легкої категорії робіт (Ia) допустима температура 22–25 °C залежно від сезону, відносна вологість 40–60 %, швидкість руху повітря ≤ 0,1 м/с. Додатково рекомендуються автоматичні системи вентиляції з HEPA–фільтрами та контроль $CO_2$ нижче 1000 ppm. Рівень шуму на робочих місцях із ВДТ не повинен перевищувати 50 дБА для робіт, що вимагають високої концентрації, що відповідає вимогам ДСН 3.3.6.037–99 [35].

Режим праці та відпочинку регламентовано для ВДТ–робіт інтелектуального характеру (категорія В): тривалість безперервної роботи з ВДТ – не більше 2 годин, після чого – перерва 15–20 хвилин; сумарна тривалість роботи з ВДТ – не більше 6 годин на зміну. У командах розробки добре зарекомендувала себе техніка Pomodoro (25 хв роботи + 5 хв перерви, 4 цикли + тривалий відпочинок), що узгоджується з нормативами. Під час перерв рекомендується виконувати вправу «20–20–20» для очей (кожні 20 хвилин дивитися на об’єкт на відстані 6 м протягом 20 секунд), фізичні вправи для шиї, плечей і поперекової зони, а також уникати використання мобільного телефону – він продовжує зорове навантаження, а не знижує його.

Комп’ютерний зоровий синдром (CVS) проявляється почервонінням, сухістю очей, втомою і головним болем наприкінці робочого дня. Його профілактика включає контроль яскравості та контрастності монітора, режим нічного освітлення у вечірній час, зволоження повітря, регулярне моргання і щорічний огляд офтальмолога. Синдром зап’ястного каналу (CTS) – поширене професійне захворювання ІТ–працівників – попереджується нейтральним положенням кисті під час друку, ергономічними клавіатурою і мишею, регулярними розминками зап’ясть та м’якими підлокітниками крісла.

Кібербезпека є невід’ємною складовою охорони праці ІТ–фахівця. Для команди Krydix вона включає: використання менеджера паролів і двофакторної автентифікації для всіх облікових записів, розмежування прав доступу в

репозиторії і продакшені, регулярне оновлення ОС та інструментарію, безпечне зберігання .env–файлів без їх потрапляння у git–коміти, журнал аудиту критичних дій (AuditLog). Відповідно до НПАОП 0.00–4.12–05 [32], для ІТ–працівників проводяться вступний, первинний (на робочому місці), повторний (не рідше 1 разу на 6 місяців) та позаплановий інструктажі з охорони праці. Доцільно включати до програми такі теми: ергономіка робочого місця, режим праці та відпочинку, кібербезпека, поведінка при відключенні електропостачання та у надзвичайних ситуаціях.

### 4.3 Висновок до розділу

У розділі проведено аналіз актуальності безпеки життєдіяльності у сфері ІТ та сформульовано гігієнічні вимоги до робочого місця з ВДТ для команди розробки маркетплейс–додатку Krydix. Поєднання заходів технічного, організаційного та особистісного рівнів дозволяє створити безпечне і продуктивне середовище, мінімізувати ризики профзахворювань і забезпечити відповідність вимогам ДСанПіН [30], ДБН [33], ДСН [34,35], ДСТУ ISO 9241 [36] та НПАОП [31,32]. Автоматизація (CI/CD, журнал аудиту, ескалаційні канали підтримки) є не лише технічним рішенням, а й засобом зниження когнітивного навантаження і психоемоційного стресу – важливою складовою сучасної ІТ–охорони праці.

# ВИСНОВКИ

У кваліфікаційній роботі розроблено та реалізовано систему багатомовного маркетплейсу Krydix – веб–застосунок, що забезпечує структуровану та безпечну онлайн–торгівлю між покупцями та верифікованими бізнес–продавцями. Визначено п’ять ролей учасників системи з детальним описом прав і відповідальностей, сформульовано та покрито тестами вісім ключових сценаріїв використання.

Розроблено монорепозиторій із поділом на backend (Node.js + Express + Apollo GraphQL), frontend (React 18 + Vite + MUI 6) та e2e (Cypress). Спроєктовано реляційну схему бази даних PostgreSQL з понад 30 моделями; багатомовність реалізовано через окремі таблиці перекладів без зміни основних схем. GraphQL API зібрано з 20 доменних модулів за принципом суворого розподілу відповідальностей між шарами resolver – service – repository.

Ключовим архітектурним рішенням є механізм незмінних знімків комісій в OrderItem: значення комісій фіксуються на момент створення замовлення і не залежать від подальших змін налаштувань платформи, що гарантує аудитованість фінансової складової системи. Реалізовано повний цикл виплат продавцям, підсистему документарної верифікації продавців та модерацію товарів і відгуків з обов’язковим журналом аудиту.

Тестове покриття включає понад 30 Jest–файлів для модульного та інтеграційного тестування і 22 Cypress E2E–файли з фікстурним мокуванням GraphQL. Налаштовано CI/CD через GitHub Actions з автоматичним розгортанням на Render та Vercel.

Результати дослідження апробовано у формі наукової публікації; копія подана в додатку З. Серед перспективних напрямків розвитку – інтеграція з реальними платіжними провайдерами, реалізація диспутного модуля, OAuth–автентифікація, B2B–функціональність та контейнеризація через Docker Compose.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Statista. Global e–commerce revenue 2014–2027. Statista Research Department, 2024. URL: https://www.statista.com/statistics/379046/worldwide–retail–e–commerce–sales/

2. OECD. E–commerce in the time of COVID–19. OECD Policy Brief, 2020. URL: https://www.oecd.org/content/dam/oecd/en/publications/reports/2020/10/e–commerce–in–the–time–of–covid–19_bb699f3a/3a2b78e8–en.pdf

3. Cases Media. E–commerce в Україні. Київ, 2026. URL: https://cases.media/article/e–commerce–v–ukrayini–analiz–trendiv–2024–2026–na–osnovi–7–5–mln–zamovlen

4. Eisenmann T., Parker G., Van Alstyne M. Strategies for Two–Sided Markets. Harvard Business Review, 2006. Vol. 84, No. 10. P. 92–101.

5. Lee B. GraphQL: A Query Language for APIs. Facebook Open Source, 2015. URL: https://graphql.org/

6. Aggarwal S. Modern Web–Development Using ReactJS. International Journal of Recent Research Aspects, 2018. Vol. 5, No. 1. P. 133–137.

7. Node.js Foundation. Node.js Documentation. URL: https://nodejs.org/en/docs/

8. React Team. React Documentation: Concurrent Features. Meta Open Source, 2022. URL: https://react.dev/

9. Cohn M. Succeeding with Agile: Software Development Using Scrum. – Addison–Wesley, 2009. – 504 p. ISBN 978–0–321–57936–2.

10. Prisma Team. Prisma ORM Documentation. URL: https://www.prisma.io/docs/

11. Fielding R. T. Architectural Styles and the Design of Network–based Software Architectures. Doctoral dissertation, University of California, Irvine, 2000.

12. Fowler M. Patterns of Enterprise Application Architecture. – Addison–Wesley, 2002. – 533 p. ISBN 978–0–321–12752–6.

13. Hunt A., Thomas D. The Pragmatic Programmer: From Journeyman to Master. – Addison–Wesley, 1999. – 352 p. ISBN 978–0–201–61622–4.

14. Newman S. Building Microservices: Designing Fine–Grained Systems. – O'Reilly Media, 2021. – 612 p. ISBN 978–1–492–03401–4.

15. Wieruch R. The Road to React. – Robin Wieruch, 2022. – 492 p. ISBN 978–1–7165–7408–0.

16. JSON Web Token Standard. RFC 7519 / M. Jones, J. Bradley, N. Sakimura. – IETF, 2015. URL: https://datatracker.ietf.org/doc/html/rfc7519

17. OWASP Foundation. OWASP Top Ten Security Risks. – OWASP, 2021. URL: https://owasp.org/www–project–top–ten/

18. PostgreSQL Global Development Group. PostgreSQL Documentation, Version 15. URL: https://www.postgresql.org/docs/15/

19. Material UI Team. MUI Documentation v6. URL: https://mui.com/material–ui/

20. Apollo GraphQL Team. Apollo Client Documentation. URL: https://www.apollographql.com/docs/react/

21. Cypress.io. Cypress Documentation. URL: https://docs.cypress.io/

22. Jest Team. Jest Documentation. URL: https://jestjs.io/docs/

23. Cloudinary. Cloudinary Developer Documentation. URL: https://cloudinary.com/documentation

24. Socket.IO Team. Socket.IO Documentation. URL: https://socket.io/docs/v4/

25. Beck K. Test–Driven Development: By Example. – Addison–Wesley, 2002. – 240 p. ISBN 978–0–321–14653–3.

26. Vernon V. Implementing Domain–Driven Design. – Addison–Wesley, 2013. – 656 p. ISBN 978–0–321–83457–7.

27. Петришин Я. Автоматизація процесів інтеграції та розгортання вебзастосунків / Я. Петришин. – Тернопіль : ТНТУ, 2026. – УДК 004.42.

28. Martin R. C. Clean Architecture: A Craftsman’s Guide to Software Structure and Design. – Prentice Hall, 2017. – 432 p. ISBN 978–0–13–468599–1.

29. Закон України «Про охорону праці» від 14.10.1992 № 2694–XII (зі змінами). URL: https://zakon.rada.gov.ua/laws/show/2694–12

30. ДСанПіН 3.3.2.007–98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно–обчислювальних машин. – Київ : МОЗ України, 1998.

31. НПАОП 0.00–7.15–18. Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями. – Київ : Держпраці України, 2018.

32. НПАОП 0.00–4.12–05. Типове положення про порядок проведення навчання і перевірки знань з питань охорони праці. – Київ : Держпраці України, 2005.

33. ДБН В.2.5–28:2018. Природне і штучне освітлення. – Київ : Мінрегіонбуд України, 2018.

34. ДСН 3.3.6.042–99. Санітарні норми мікроклімату виробничих приміщень. – Київ : МОЗ України, 1999.

35. ДСН 3.3.6.037–99. Санітарні норми виробничого шуму, ультразвуку та інфразвуку. – Київ : МОЗ України, 1999.

36. ДСТУ ISO 9241–5:2004. Ергономічні вимоги до роботи з відеотерміналами в офісі. Частина 5. Вимоги до розміщення робочого місця та робочої пози. – Київ : Держспоживстандарт України, 2004.

37. Жидецький В.Ц. Охорона праці користувачів комп’ютерів : підручник. Львів : Афіша, 2020. 176 с.

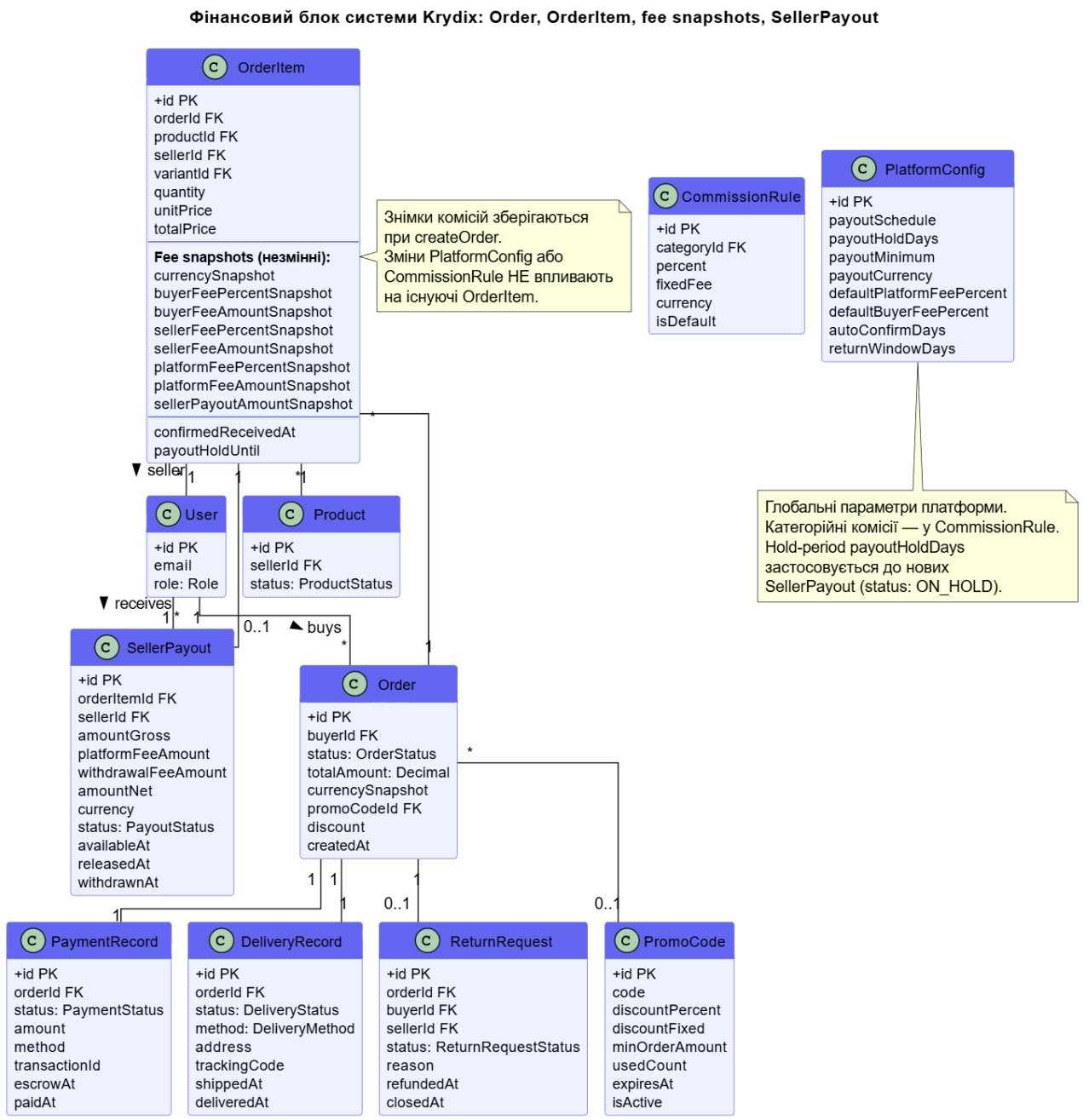
38. Желібо Є.П. Безпека життєдіяльності : підручник / В. В. Зацарний. Київ : Каравела, 2023. 344 с.

39. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для здобувачів спеціальності 121 – Інженерія програмного забезпечення, всіх форм навчання / укладачі: Михалик Д.М., Цуприк Г.Б., Бревус В.М. – Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2024. – 45 с.

# ДОДАТКИ

# ДОДАТОК А

Діаграма фінансового блоку системи

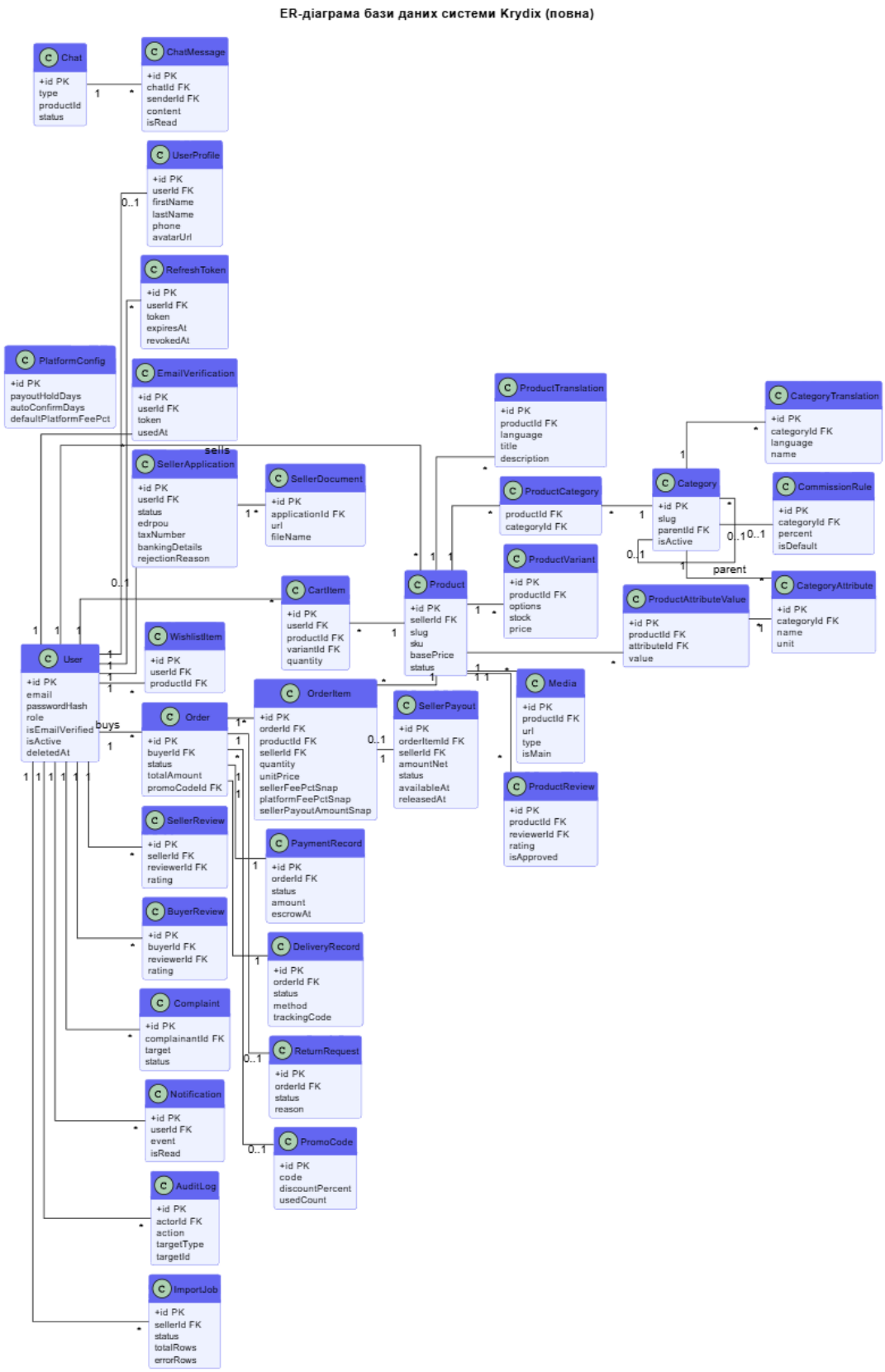


# ДОДАТОК Б

Діаграма прецедентів системи

# ДОДАТОК В

ER–діаграма бази даних



# ДОДАТОК Д

ER– Лістинг модуля контролю доступу за ролями

```
import { Role } from '@/constants/enums';

export const STAFF_ROLES: readonly Role[] = [Role.MODERATOR,
Role.ADMIN];

export enum RouteZone {
  GLOBAL = 'GLOBAL',
  AUTH = 'AUTH',
  BUYER = 'BUYER',
  BUYER_COMMERCE = 'BUYER_COMMERCE',
  SELLER = 'SELLER',
  MODERATOR = 'MODERATOR',
  ADMIN = 'ADMIN',
}

export function isStaffRole(role?: Role | string): boolean {
  return !!role && STAFF_ROLES.includes(role as Role);
}

export function isAdminRole(role?: Role | string): boolean {
  return role === Role.ADMIN;
}

export function canAccessRouteZone(
  role: Role | string | undefined,
  zone: RouteZone,
  isAuthenticated: boolean,
): boolean {
  if (zone === RouteZone.GLOBAL) return true;
  if (!isAuthenticated) return zone === RouteZone.BUYER_COMMERCE;
  if (zone === RouteZone.ADMIN) return isAdminRole(role);
  if (zone === RouteZone.MODERATOR) {
    return role === Role.MODERATOR || isAdminRole(role);
  }
  if (isStaffRole(role)) return true;

  switch (zone) {
    case RouteZone.AUTH: return true;
    case RouteZone.BUYER: return role === Role.BUYER;
    case RouteZone.BUYER_COMMERCE: return role === Role.BUYER;
    case RouteZone.SELLER: return role === Role.SELLER;
    default: return false;
  }
}
```

# ДОДАТОК Е

ER– Лістинг модуля контролю доступу за ролями

```
import * as platformRepo from
'../../repositories/platformRepository.js';
import { buildFeeSnapshots, computeLineFeeSnapshot } from
'../feeCalculationService.js';
beforeEach(() => {
  jest.clearAllMocks();
  getPlatformConfig.mockResolvedValue({
    payoutCurrency: 'UAH',
    defaultBuyerFeePercent: 0,
    defaultPlatformFeePercent: 2,
  } as never);
  findCommissionRules.mockResolvedValue([
    { id: 'default', categoryId: null, percent: 12, fixedFee: 0.5,
isDefault: true },
    { id: 'cat-rule', categoryId: 'cat-1', percent: 5, fixedFee: 1,
isDefault: false },
  ] as never);
});
describe('computeLineFeeSnapshot', () => {
  it('calculates percent + fixed fee from commission rule', () => {
    const snapshot = computeLineFeeSnapshot(
      { productId: 'p1', categoryIds: [], lineTotal: 100 },
      { currency: 'UAH', buyerFeePercent: 0, platformFeePercent: 2
},
      { percent: 12, fixedFee: 0.5 }
    );
    expect(snapshot.platformFeeAmount).toBe(12.5);
    expect(snapshot.sellerPayoutAmount).toBe(87.5);
  });
});
describe('buildFeeSnapshots', () => {
  it('uses category-specific commission rules', async () => {
    const snapshots = await buildFeeSnapshots([
      { productId: 'p1', categoryIds: ['cat-1'], lineTotal: 200 },
      { productId: 'p2', categoryIds: ['cat-2'], lineTotal: 100 },
    ]);
    expect(snapshots[0].platformFeePercent).toBe(5);
    expect(snapshots[0].sellerPayoutAmount).toBe(189);
    expect(snapshots[1].platformFeePercent).toBe(12);
    expect(snapshots[1].sellerPayoutAmount).toBe(87.5);
  });
});
```

## ДОДАТОК Ж

Лістинг GraphQL–схеми (checkout typeDefs)

```
export const checkoutTypeDefs = `#graphql
  enum DeliveryMethod { COURIER  BRANCH_PICKUP  SELF_PICKUP }
  enum OrderStatus { PENDING CONFIRMED SHIPPED DELIVERED CANCELLED
REFUNDED }   enum PaymentStatus { PENDING AUTHORIZED PAID IN_ESCROW
FAILED REFUNDED CHARGEBACK CANCELED }   enum DeliveryStatus {
PENDING PACKED SENT IN_TRANSIT DELIVERED NOT_RECEIVED RETURNED }
  type OrderItemOut {id: ID!
    productId: ID!
    variantId: ID
    quantity: Int!
    unitPrice: Float!
    totalPrice: Float!
    productTitle: String!
  }
  type PaymentOut {id: ID!
    status: PaymentStatus!
    amount: Float!
    method: PaymentMethod
  }
  type DeliveryOut {id: ID!
    status: DeliveryStatus!
    method: DeliveryMethod!
    address: String
    trackingCode: String
  }
  type OrderOut {
    id: ID!
    status: OrderStatus!
    totalAmount: Float!
    discount: Float
    items: [OrderItemOut!]!
    payment: PaymentOut
    delivery: DeliveryOut
    createdAt: String!
  }
  input CartItemInput {productId: ID!
    variantId: ID
    quantity: Int!
  }
  extend type Mutation {
    createOrder(items: [CartItemInput!]
      paymentMethod: PaymentMethod!
      deliveryMethod: DeliveryMethod!
      deliveryAddress: String
      promoCode: String
      notes: String
    ): OrderOut!
```

# ДОДАТОК З

Апробація результатів дослідження


Рисунок 3.1 – Титульна сторінка конференції

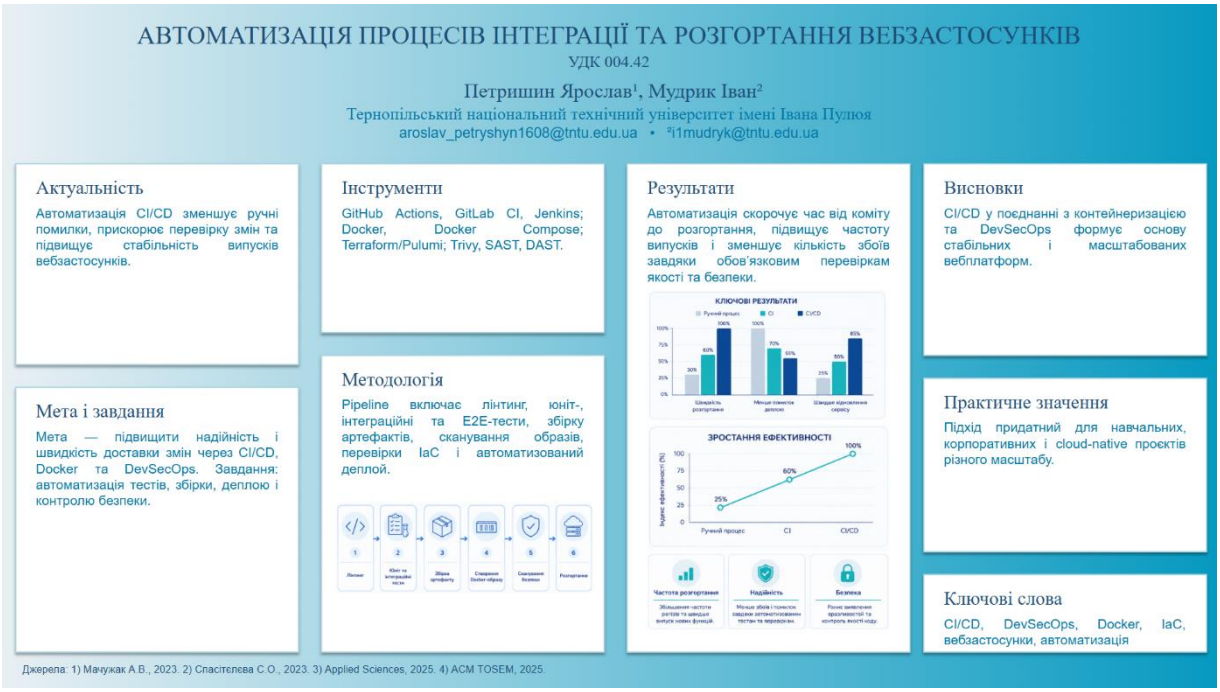



Рисунок 3.2 – Сторінка публікації