МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Відокремлений структурний підрозділ
«Тернопільський фаховий коледж
Тернопільського національного технічного університету імені Івана Пулюя»

Відділення телекомунікацій та електронних систем

Циклова комісія комп'ютерних наук

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

фахового молодшого бакалавра

на тему: Розробка вебплатформи для дистрибуції та продажу цифрових активів «AssetMaster»

Виконав: студент IV курсу, групи КН-421
спеціальності: 122 Комп'ютерні науки
Дмитро КІЧУЛА

Керівник Руслан СЛОБОДЯН

Рецензент _____
(ім'я та прізвище)

м. Тернопіль – 2026

ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ТЕРНОПІЛЬСЬКИЙ ФАХОВИЙ КОЛЕДЖ
ТЕРНОПІЛЬСЬКОГО НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ
ІМЕНІ ІВАНА ПУЛЮЯ»

Відділення телекомунікацій та електронних систем
Циклова комісія комп'ютерних наук
Освітньо-професійний ступінь «фаховий молодший бакалавр»
Спеціальність 122 Комп'ютерні науки
Галузь знань 12 Інформаційні технології

ЗАТВЕРДЖУЮ
Голова циклової комісії
комп'ютерних наук
_____ Галина МАРЦІЯШ
« 02 » березня 2026 року

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

Кічулі Дмитру Івановичу
(прізвище, ім'я, по батькові)

1. Тема роботи: Розробка вебплатформи для дистрибуції та продажу цифрових активів «AssetMaster»,

керівник роботи Слободян Руслан Олесійович,
(прізвище, ім'я, по батькові)
затверджені наказом вищого навчального закладу № 4/9-132 від 27.02.2026 р.
2. Строк подання студентом роботи: 19.06.2026 р.
3. Вихідні дані до роботи: технічне завдання на розробку програмного забезпечення, мови програмування: Java, TypeScript; фреймворки: Spring Boot, Hibernate, Material UI; бібліотеки: React, TanStack Query, стандарти ДСТУ ISO/IEC/IEEE 29148:2025, ДСТУ ISO/IEC/IEEE 29119-1:2017, ДСТУ 3008:2015, ДСТУ 8302:2015.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):

1 Загальний розділ

1.1 Аналітичний огляд існуючих рішень

1.2 Технічне завдання

1.2.1 Найменування та область застосування

1.2.2 Призначення розробки

1.2.3 Вимоги до функціоналу вебплатформи

1.2.4 Вимоги до програмної документації

1.2.5 Техніко-економічні показники

1.2.6 Стадії та етапи розробки

1.2.7 Порядок тестування та прийому

2 Розробка технічного та робочого проєкту

2.1 Розробка структури вебплатформи

2.2 Створення та верстка сторінок вебплатформи

2.3 Розробка структури бази даних вебплатформи

2.4 Програмування вебплатформи

2.4.1 Написання клієнтської частини

2.4.2 Написання серверної частини

2.5 Тестування вебплатформи

3 Спеціальний розділ

3.1 Інструкція з розгортання вебплатформи

3.2 Інструкція з наповнення вебплатформи

3.3 Інструкція з популяризації та підтримки вебплатформи

4 Економічний розділ

4.1 Визначення стадій технологічного процесу та загальної тривалості проведення розробки вебплатформи

4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи

4.3 Розрахунок витрат на електроенергію

4.4 Розрахунок суми амортизаційних відрахувань

4.5 Обчислення накладних витрат

4.6 Складання кошторису витрат та визначення собівартості вебплатформи

4.7 Розрахунок ціни вебплатформи

4.8 Визначення економічної ефективності і терміну окупності капітальних вкладень

5 Охорона праці, техніка безпеки та екологічні вимоги

5.1 Професійні захворювання та отруєння

5.2 Шум. Основні джерела шкідливого шуму, кваліфікація шумів

6 Висновки

Додаткові вказівки: виконання кваліфікаційної роботи із розробкою програмного продукту – вебплатформи «AssetMaster».

5. Перелік графічного матеріалу:

1. Схема структурна клієнтської частини
2. UML-діаграма варіантів використання
3. ER-діаграма бази даних
4. Таблиця техніко-економічних показників

6. Консультанти розділів роботи

Розділ	Ім'я, прізвище та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний розділ	Любов КАЛУШКА		
Охорона праці, техніка безпеки та екологічні вимоги	Генадій ГОРЯЧЕК		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Отримання і аналіз технічного завдання	20.03.2026	
2	Збір і узагальнення інформації	01.05.2026	
3	Написання першого розділу	15.05.2026	
4	Розробка технічного та робочого проєкту	29.05.2026	
5	Написання спеціального розділу	05.06.2026	
6	Розрахунок економічної частини	08.06.2026	
7	Написання розділу охорони праці	09.06.2026	
8	Виконання графічної частини	10.06.2026	
9	Оформлення пояснювальної записки	11.06.2026	
10	Погодження нормоконтролю	12.06.2026	
11	Попередній захист кваліфікаційної роботи	.06.2026	
12	Захист кваліфікаційної роботи	.06.2026	

7. Дата видачі завдання: 06 березня 2026 р.

Студент _____ Дмитро КІЧУЛА
(підпис)

Керівник роботи _____ Руслан СЛОБОДЯН
(підпис)

ЗМІСТ

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи на тему «Розробка вебплатформи для дистрибуції та продажу цифрових активів AssetMaster» містить 81 сторінку, 16 рисунків, 20 таблиць, 6 додатків, 17 джерел.

У роботі розроблено вебплатформу типу маркетплейс, що забезпечує двосторонню взаємодію між авторами цифрових активів та їх покупцями. Платформа реалізує повний цикл дистрибуції: від завантаження та категоризації активів авторами до пошуку, перегляду, придбання та завантаження файлів покупцями. Для розробки серверної частини застосовано мову програмування Java 21 з фреймворком Spring Boot та СУБД PostgreSQL; клієнтську частину реалізовано засобами TypeScript, бібліотеки React, компонентної бібліотеки Material UI та бібліотеки управління серверним станом React Query.

Пояснювальна записка складається з 5 розділів.

У загальній частині описуються аналітичний огляд існуючих рішень та аналіз технічного завдання програмного продукту.

У другому розділі представлено розробку технічного та робочого проєкту інформаційної системи, зокрема розробку структури системи, створення та верстку її сторінок, розробку структури бази даних, програмування клієнтської та серверної частин, а також тестування розробленого програмного продукту.

В спеціальній частині описано процес розміщення інформаційної системи в Інтернеті, інструкцію з обслуговування та наповнення системи, а також інструкцію з популяризації та підтримки вебзастосунку «Montreux Regent Hotel».

Розрахунок вартості розробки та економічної ефективності приведено в економічній частині в четвертому розділі. Основні питання охорони праці та техніки безпеки розглянуто в п’ятому розділі.

До складу кваліфікаційної роботи входить графічна частина, яка складається з схеми структурної клієнтської частини інформаційної системи, UML-діаграми варіантів використання, ER-діаграми бази даних та таблиці техніко-економічних показників, що виконані на окремих аркушах формату А1.

ABSTRACT

The explanatory note to the qualification thesis titled "Development of the AssetMaster Web Platform for Distribution and Sale of Digital Assets" consists of 81 pages, 16 figures, 20 tables, 6 appendices, and 17 sources.

The thesis delivers the development of a marketplace-type web platform that ensures two-way interaction between digital asset creators and buyers. The platform implements a complete distribution lifecycle, spanning from asset uploading and categorization by creators to searching, viewing, purchasing, and downloading files by buyers. The backend of the system is engineered using the Java 21 programming language along with the Spring Boot framework and the PostgreSQL DBMS. The client-side is implemented via TypeScript, the React library, the Material UI component library, and the React Query server-state management library.

The explanatory note comprises 5 chapters.

The general section covers the analytical review of existing solutions and the requirements analysis for the software product.

The second chapter presents the engineering of the technical and detailed design of the information system, specifically encompassing system architecture design, page creation and layout prototyping, database schema design, client- and server-side programming, as well as testing of the developed software product.

The specialized section details the deployment process of the information system on the Internet, the system maintenance and content management manual, alongside the guidelines for promotion and support of the "Montreux Regent Hotel" web application.

The calculation of development costs and economic efficiency is detailed in the economic section within the fourth chapter. Core occupational health and safety issues are examined in the fifth chapter.

The qualification thesis includes a graphical component consisting of a client-side structural architecture diagram of the information system, a UML use case diagram, a database ER diagram, and a table of technical and economic indicators, executed on separate A1 sheets.

ВСТУП

Цифрова економіка набуває дедалі більшого значення в сучасному технологічному середовищі. Особливу роль у ній відіграє ринок цифрових активів — графічних матеріалів, шаблонів інтерфейсів, тривимірних моделей, шрифтів, аудіозаписів та ігрових ресурсів. За даними аналітичних компаній, обсяг глобального ринку цифрових творчих активів щорічно зростає на 15–20 %, що зумовлено стрімким розвитком веброзробки, геймдизайну, відеовиробництва та суміжних галузей.

Попри наявність кількох великих міжнародних платформ (Creative Market, Envato Elements, Gumroad), цей ринок залишається фрагментованим: більшість існуючих рішень орієнтовані на англомовну аудиторію, пропонують обмежені можливості для авторів-початківців або стягують завищені комісійні. Водночас українські та загалом східноєвропейські розробники та дизайнери потребують зручної та доступної платформи для монетизації власної творчої праці.

Мета кваліфікаційної роботи полягає у проєктуванні та розробці повнофункціональної вебплатформи AssetMaster, яка забезпечує двосторонній ринок цифрових активів із розширеними можливостями для авторів і зручним досвідом придбання для покупців.

Для досягнення поставленої мети необхідно вирішити такі завдання: виконати аналітичний огляд існуючих рішень і виявити їх переваги та недоліки; сформулювати технічне завдання на розробку платформи; спроєктувати структуру вебплатформи та схему бази даних; реалізувати серверну частину засобами Java 21 і Spring Boot з використанням PostgreSQL; реалізувати клієнтську частину засобами TypeScript, React, Material UI та React Query; розробити адміністративну панель для управління контентом і користувачами; протестувати розроблену систему та задокументувати результати.

Об'єктом розробки є вебплатформа для дистрибуції та продажу цифрових активів. Предметом розробки є архітектура, функціональний електронний склад і програмна реалізація двостороннього маркетплейсу.

1 ЗАГАЛЬНИЙ РОЗДІЛ

1.1 Аналітичний огляд існуючих рішень

Ринок платформ для розповсюдження цифрових активів представлений кількома категоріями рішень: спеціалізованими маркетплейсами для творчих матеріалів, стоковими сервісами та універсальними платформами для цифрових товарів. Нижче розглянуто три найбільш репрезентативні рішення, які є конкурентами розроблюваної платформи AssetMaster.

1.1.1 Creative Market

Creative Market [1] — одна з найбільших платформ для продажу шрифтів, шаблонів, графіки, ілюстрацій та UI-компонентів. Платформа була заснована у 2012 році та станом на 2024 рік налічує понад 10 мільйонів користувачів і більше 6 мільйонів цифрових продуктів. Модель монетизації передбачає стягнення комісії у розмірі 40 % з кожного продажу для нових авторів, що знижується до 30 % за умови досягнення певного обсягу продажів.

Серед переваг платформи слід виділити потужний пошуковий механізм із підтримкою фільтрації за категоріями, тегами, ціновим діапазоном і типом ліцензії, а також розвинену систему авторських профілів із відображенням портфоліо та рейтингів. Інтерфейс платформи характеризується зрозумілою навігацією та якісною передпереглядовою функціональністю для більшості типів активів.

Проте Creative Market має й суттєві недоліки. По-перше, висока комісія для нових авторів стримує вихід на платформу малодосвідчених творців. По-друге, платформа орієнтована виключно на англомовний ринок, що обмежує доступність для авторів з інших регіонів. По-третє, відсутня безкоштовна пробна ліцензія для тестування активів перед покупкою.

1.1.2 Envato Elements

Envato Elements [2] — платформа австралійської компанії Envato, що функціонує за моделлю підписки. За фіксовану щомісячну або щорічну плату користувачі отримують необмежений доступ до понад 16 мільйонів активів, включно з шаблонами WordPress, відеоматеріалами, музикою, шрифтами та графічними елементами. Для авторів передбачено роялті-систему, що базується на частці завантажень їх активів від загального обсягу завантажень на платформі.

Головною перевагою Envato Elements є модель «все включено», яка робить платформу надзвичайно привабливою для агентств і фрілансерів із великим обсягом проєктів. Технічна інфраструктура платформи відзначається високою надійністю та швидкістю завантаження файлів, а контент проходить строгу модерацію, що гарантує якість активів.

Водночас модель підписки є суттєвим бар'єром для покупців з обмеженим бюджетом або разовими потребами. Для авторів непрозорий механізм нарахування роялті, який залежить від загальної кількості завантажень на платформі, не гарантує стабільного доходу. Крім того, платформа не підтримує одноразових покупок окремих активів.

1.1.3 Gumroad

Gumroad [3] — американська платформа, орієнтована на незалежних творців і забезпечує продаж цифрових та фізичних товарів безпосередньо від автора до покупця. На відміну від Creative Market та Envato Elements, Gumroad надає максимальну гнучкість: автор самостійно встановлює ціну, налаштовує ліцензійні умови та спосіб доставки файлів. Комісія платформи становить 10 % від суми продажу.

Ключовою перевагою Gumroad є простота запуску продажів — автор може розпочати торгівлю протягом кількох хвилин без проходження тривалої модерації. Платформа підтримує гнучку цінову модель, зокрема можливість

встановлення мінімальної ціни з правом покупця доплатити більше (pay-what-you-want). Також реалізовано механізм підписки та передпродажу.

Серед недоліків Gumroad виділяється відсутність вбудованого маркетплейсу з органічним трафіком — автор самостійно відповідає за залучення аудиторії. Пошукові та рекомендаційні механізми платформи значно поступаються конкурентам. Інтерфейс сторінки продавця є шаблонним і обмеженим у кастомізації.

Узагальнений порівняльний аналіз розглянутих платформ представлено у таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз платформ цифрових активів

Критерій	Creative Market	Envato Elements	Gumroad
Модель монетизації	Разова покупка	Підписка	Разова покупка
Комісія платформи	30–40 %	Роялті від завантажень	10 %
Органічний трафік	Так	Так	Обмежено
Підтримка авторів-початківців	Обмежена	Так	Так
Мови інтерфейсу	Англійська	Англійська	Англійська

Проведений аналіз засвідчує, що існуючі платформи, попри значні переваги, мають низку системних обмежень: орієнтацію на англомовний ринок, несприятливі умови для авторів-початківців. Розробка платформи AssetMaster покликана усунути ці недоліки завдяки нижчій комісії для нових авторів, підтримці української мови, прозорій аналітиці продажів та гнучкій ліцензійній системі.

1.2 Технічне завдання

1.2.1 Найменування та область застосування

Повне найменування розробки: вебплатформа для дистрибуції та продажу цифрових активів «AssetMaster».

Платформа AssetMaster відноситься до класу двосторонніх електронних маркетплейсів і призначена для використання в галузі цифрової дистрибуції творчих матеріалів. Область застосування платформи охоплює ринок цифрових активів для веброзробки, геймдизайну, графічного дизайну та мультимедійного виробництва. Типовими об'єктами, у яких передбачається використання платформи, є дизайн-агентства, фрілансери, студії розробки ігор, незалежні творці контенту, а також будь-які юридичні або фізичні особи, що створюють або споживають цифрові творчі матеріали.

1.2.2 Призначення розробки

Експлуатаційне призначення платформи AssetMaster полягає у забезпеченні зручного та безпечного середовища для купівлі-продажу цифрових активів в режимі реального часу через мережу Інтернет. Платформа орієнтована на дві основні категорії користувачів: авторів (продавців), які розміщують власні цифрові матеріали для продажу, та покупців, які здійснюють пошук і придбання необхідних ресурсів.

Функціональне призначення платформи реалізується через такі засоби: модуль каталогу активів із розширеними можливостями пошуку та фільтрації; модуль управління обліковими записами з підтримкою ролей (покупець, автор, адміністратор); модуль завантаження та керування активами для авторів; модуль обробки замовлень та платежів; модуль аналітики продажів для авторів і адміністраторів; адміністративна панель для модерації контенту та управління користувачами.

1.2.3 Вимоги до функціоналу вебплатформи

1.2.3.1 Вимоги до функціональних характеристик

Платформа повинна забезпечувати реєстрацію та автентифікацію користувачів із підтримкою двох базових ролей — покупця та автора — а також адміністративної ролі. Авторизація здійснюватиметься за допомогою електронної пошти та пароля з підтримкою JSON Web Token (JWT) для підтримання сесії. Передбачається можливість відновлення паролю через електронну пошту.

Модуль каталогу активів повинен підтримувати перегляд усіх опублікованих активів у вигляді сітки карток із відображенням мініатюри, назви, автора, категорії та ціни. Пошук здійснюється за назвою, тегами та описом; фільтрація — за категорією, ціновим діапазоном, типом ліцензії та датою додавання. Сторінка окремого активу повинна відображати розширений попередній перегляд (до 5 зображень або аудіо/відео-прев'ю), технічні характеристики файлу (формат, розмір, дозвіл), умови ліцензування, профіль автора та відгуки покупців.

Для авторів платформа повинна надавати форму завантаження нового активу, яка включає поля для введення назви, опису, категорії, тегів, ціни, типу ліцензії та завантаження файлів (основного файлу активу і файлів попереднього перегляду). Автор повинен мати можливість редагувати та видаляти власні активи, а також переглядати статистику переглядів і продажів у розрізі окремих активів і загального портфоліо.

Модуль оформлення замовлень повинен реалізовувати кошик для накопичення активів перед оплатою, сторінку підсумку замовлення та інтеграцію з платіжним шлюзом. Після успішної оплати покупець отримує доступ до завантаження придбаних файлів із особистого кабінету. Система повинна зберігати повну історію замовлень.

Адміністративна панель повинна надавати можливість перегляду та модерації активів перед їх публікацією, управління обліковими записами користувачів (блокування, зміна ролі), перегляду фінансової звітності та

управління категоріями і тегами.

1.2.3.2 Вимоги до надійності

Платформа повинна забезпечувати валідацію всіх вхідних даних як на стороні клієнта (React-форми з бібліотекою валідації), так і на стороні сервера (Spring Validation). У разі некоректних вхідних даних система повинна повертати зрозумілі повідомлення про помилки без розкриття внутрішніх деталей реалізації. Для захисту від несанкціонованого доступу передбачається перевірка JWT-токена на кожному захищеному ендпоінті, а також розмежування прав доступу за ролями на рівні Spring Security.

З метою забезпечення цілісності даних усі операції, що змінюють стан бази даних (оформлення замовлення, зарахування коштів автору), повинні виконуватись у транзакціях PostgreSQL. Система повинна коректно обробляти виняткові ситуації (відсутність запису, недостатньо коштів на рахунку, недоступність платіжного шлюзу) та надавати відповідні відповіді клієнту з HTTP-кодами статусу згідно з REST-конвенцією.

1.2.3.3 Умови експлуатації

Платформа розрахована на одночасну роботу до 500 активних користувачів. Серверна частина розгортатиметься на хмарній інфраструктурі з підтримкою контейнеризації (Docker). Клієнтська частина повинна функціонувати у сучасних браузерах: Google Chrome версії 110+, Mozilla Firefox версії 110+, Microsoft Edge версії 110+, Safari версії 16+.

Операційна система сервера — Linux (Ubuntu 22.04 LTS або аналогічна). Мінімальні вимоги до серверних ресурсів: 2 ядра CPU, 4 ГБ RAM, 50 ГБ дискового простору для файлів активів.

1.2.3.4 Часові характеристики

Час відповіді серверного API на запити пошуку та фільтрації каталогу не повинен перевищувати 500 мс за умови нормального навантаження (до 100

одночасних запитів).

Час завантаження головної сторінки у браузері (показник Largest Contentful Paint) не повинен перевищувати 2,5 секунди на з'єднанні зі швидкістю 10 Мбіт/с.

Процес оформлення та підтвердження замовлення після завершення платежу не повинен тривати більше 5 секунд.

1.2.4 Вимоги до програмної документації

Склад програмних документів визначається відповідно до вимог кваліфікаційної роботи та включає: пояснювальну записку з повним описом архітектурних рішень, структури бази даних, алгоритмів ключових функцій і результатів тестування; інструкцію з розгортання платформи (deployment guide); інструкцію з адміністрування та наповнення сайту.

Вихідний код платформи повинен відповідати таким вимогам до документування: кожен публічний метод Java-класів сервісного рівня повинен мати Javadoc-коментар із зазначенням призначення, параметрів і значення, що повертається; React-компоненти повинні мати JSDoc-коментар із описом пропсів; усі SQL-міграції повинні мати коментар із датою та коротким описом змін. Мова коментарів — англійська.

Назви змінних, функцій і класів повинні відповідати конвенціям іменування відповідної мови програмування (camelCase для Java і TypeScript, snake_case для SQL).

1.2.5 Техніко-економічні показники

Трудові ресурси, виділені на розробку платформи, становлять 200 людино-

Загальний обсяг вихідного коду проєкту оцінюється у 15 000–20 000 рядків без урахування конфігураційних файлів і автоматично згенерованого коду.

Обсяг тестових сценаріїв — не менше 30 одиниць для серверної частини (unit та integration тести).

1.2.6 Стадії та етапи розробки

Розробка платформи AssetMaster здійснюється у чотири послідовні стадії.

На першій стадії — аналізу та проєктування — виконуються: аналіз існуючих рішень і формулювання вимог, розробка архітектури системи, проєктування схеми бази даних та розробка wireframe-макетів інтерфейсу.

На другій стадії — розробки серверної частини — реалізуються: налаштування Spring Boot проєкту та структури пакетів, реалізація схеми бази даних за допомогою Flyway-міграцій, розробка JPA-сутностей і репозиторіїв, реалізація сервісного шару і REST API, налаштування Spring Security та JWT-автентифікації.

На третій стадії — розробки клієнтської частини — реалізуються: налаштування React-проєкту з TypeScript і Material UI, розробка компонентів навігації, каталогу, сторінок продукту та авторизації, реалізація особистих кабінетів покупця та автора, розробка адміністративної панелі, інтеграція з серверним API через React Query.

На четвертій стадії — тестування та розгортання — виконуються: написання unit і integration тестів для серверної частини, функціональне тестування клієнтської частини, виправлення виявлених дефектів, підготовка Docker-конфігурації та розгортання на хостингу.

1.2.7 Порядок тестування та прийому

Тестування платформи AssetMaster повинне виконуватись у два етапи. На першому етапі (компонентне та інтеграційне тестування) перевіряється коректність роботи окремих серверних модулів і їх взаємодія. Використовуються фреймворки JUnit 5 та Mockito для unit-тестів і TestContainers для інтеграційних тестів із реальною базою даних PostgreSQL у Docker-контейнері. Мінімальне покриття коду тестами сервісного шару — 80 %.

На другому етапі (функціональне тестування) перевіряється виконання всіх

функціональних вимог, описаних у підрозділі 1.2.3. [4]

Тестування здійснюється вручну за підготовленими тест-кейсами. Контрольний приклад включає сценарії реєстрації та авторизації користувача, завантаження активу автором, придбання активу покупцем, адміністративного управління активами та модерації. Платформа вважається прийнятою за умови успішного проходження всіх функціональних тест-кейсів і відсутності критичних дефектів (за класифікацією Blocker та Critical).

2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ

2.1 Розробка структури вебплатформи

Проєктування структури вебплатформи AssetMaster є першим і визначальним етапом розробки, результати якого зумовлюють архітектурні рішення всіх наступних підрозділів. Вебплатформа AssetMaster реалізує двосторонню маркетплейс-модель, що обслуговує три ролі користувачів: покупця (ROLE_USER), автора (ROLE_AUTHOR) та адміністратора (ROLE_ADMIN). Визначення функціонального складу системи здійснювалось методом аналізу варіантів використання (Use Case Analysis) відповідно до виявлених ролей і сценаріїв взаємодії користувачів із платформою.

За результатами аналізу побудовано UML-діаграму варіантів використання, що відображає 39 варіантів використання, згрупованих у чотири функціональні зони: публічну зону (доступну без авторизації), зону покупця, зону автора та адміністративну зону. Між варіантами використання визначено зв'язки типу «include» (обов'язкові залежності, наприклад перегляд каталогу обов'язково включає фільтрацію) та «extend» (необов'язкові розширення, наприклад вхід може розширюватись перевіркою TOTP-коду). Діаграму варіантів використання наведено у графічній частині 2026.КВР.122.421.10.00.00 ДВ.

Гість може переглядати каталог із фільтрацією за категорією та ціновим діапазоном, виконувати повнотекстовий пошук, переглядати сторінки активів і блогу, реєструватись і входити до системи з підтримкою двофакторної автентифікації. Покупець, окрім функцій гостя, управляє кошиком і вішлістом, оформлює замовлення через платіжний шлюз, завантажує придбані файли через presigned MinIO URL із TTL 15 хвилин, залишає відгуки та управляє налаштуваннями безпеки. Автор додатково завантажує нові активи, управляє портфоліо, переглядає аналітику продажів. Адміністратор здійснює модерацію активів, управляє обліковими записами, категоріями, блогом та фінансовими виплатами.

На основі виявлених функціональних вимог розроблено схему структурну вебплатформи, що відображає ієрархічну організацію сторінок і навігаційні зв'язки між ними.

Схема структурна охоплює чотири рівні навігаційного простору. Перший рівень — публічна зона — включає 16 маршрутів, доступних без авторизації: головну сторінку, каталог із підтримкою категоріальної фільтрації, сторінку окремого активу, пошук, блог та п'ять статичних сторінок (Про нас, Ліцензії, FAQ, Контакти). Блок авторизації (реєстрація, вхід із підтримкою TOTP, відновлення пароля, підтвердження email) виокремлено в окрему підзону без загального макету. Схему структурну наведено у графічній частині 2026.КВР.122.421.10.00.00 СС.

Другий і третій рівні утворюють особистий кабінет, що розгалужується залежно від ролі. Покупець має доступ до шести маршрутів: покупки, список бажань, оформлення та підтвердження оплати, профіль, сповіщення, платежі та безпека. Автор отримує чотири додаткових маршрути: перелік власних активів, форма завантаження нового активу, редагування активу та аналітика продажів. Четвертий рівень — адміністративна панель (ROLE_ADMIN) — містить сім маршрутів: дашборд із аналітикою продаж, управління користувачами, фінансовий модуль, аналітика платформи, управління категоріями та блогом. Системні сторінки (404, 500, технічне обслуговування) утворюють окрему зону поза основною навігацією.

2.2 Створення та верстка сторінок вебплатформи

Верстка сторінок платформи AssetMaster виконана засобами TypeScript 6.0.2 і React 19.2.7 з використанням компонентного підходу. Кожна сторінка є деревом React-компонентів, де кожен компонент несе відповідальність за відображення і поведінку конкретного фрагмента інтерфейсу. Стилізація реалізована через систему sx-prop бібліотеки MUI v9.1.0 та кастомну тему, визначену у src/app/theme.ts. Збірка проекту здійснюється через Vite 7.3.5.

Розглянемо базові компоненти вебплатформи, створені в ході розробки програмного забезпечення.

Компонент Navbar реалізований як widget і є постійно присутнім на всіх публічних сторінках та сторінках особистого кабінету. В основі верстки — MUI AppBar і Toolbar із кастомними стилями. Для реалізації ефекту «прилипання» navbar до верху сторінки під час прокручування застосовано CSS-властивість position: sticky із z-index рівня 1100. Ефект glassmorphism досягається за допомогою властивостей backdrop-filter: blur та напівпрозорого фону. Компонент містить логотип платформи, основне меню навігації, компонент SearchBar, іконку кошика з badge лічильника елементів та аватар авторизованого користувача або кнопки входу і реєстрації для гостей. На мобільному breakpoint (< 768 пікселів) горизонтальне меню трансформується у бічний Drawer, що відкривається іконкою «бургер». Компонент SearchBar реалізує автодоповнення з затримкою запиту 300 мс через хук useDebounce та відображає dropdown із результатами пошуку, згрупованими за типом: активи, автори та категорії .

Сторінка каталогу CatalogPage складається з трьох зон: бічної панелі фільтрів, рядка сортування та основної сітки карток. Бічна панель містить акордеонні секції MUI Accordion для фільтрації за категорією, ціновим діапазоном (MUI Slider), типом ліцензії та рейтингом. Стан фільтрів синхронізується з URL-параметрами через хук useSearchParams, що забезпечує можливість збереження та передачі посилань на відфільтровані результати. Компонент AssetCard відображає мініатюру активу у фіксованому співвідношенні сторін 16:10 (реалізовано через CSS aspect-ratio), назву, ім'я автора з аватаром, badge категорії та ціну. Hover-стан картки реалізує накладення оверлею (rgba(0,0,0,0.4)) із кнопкою «Переглянути» та іконкою вішліста; усі переходи виконуються з тривалістю 200 мс та функцією ease-in-out . При завантаженні відображається MUI Skeleton-компонент. Компонент AssetGrid реалізує нескінченне прокручування через Intersection Observer API та хук useInfiniteQuery бібліотеки TanStack Query v5: sentinel-елемент наприкінці списку карток спостерігається хуком useIntersectionObserver; при входженні у видиму область

автоматично викликається fetchNextPage. Адаптивна сітка карток реалізована через MUI Grid2 із параметром responsive columns: 4 стовпці на desktop, 3 на tablet і 2 на mobile .

Сторінка AssetPage організована у дві колонки. Ліва колонка (60 % ширини) містить компонент ImageSlider із до п'яти зображень попереднього перегляду зі стрілками і dot-індикатором; для активів типу 3D-model замість слайдера відображається інтерактивний WebGL-перегляд засобами бібліотеки Three.js v0.184.0. Нижче розміщено MUI Tabs із трьома вкладками: опис активу (підтримує Markdown-форматування), технічні характеристики (формат файлу, розмір, кількість компонентів) та відгуки покупців. Права колонка (40 %) реалізована як sticky-блок придбання: при прокручуванні опису вона залишається у видимій зоні екрана. Sticky-блок містить назву активу, ім'я автора з посиланням на профіль, ціну, LicenseToggle (MUI ToggleButtonGroup для перемикання між стандартною та комерційною ліцензіями), кнопки «Додати до кошика» і «Купити зараз», а також trust bar із рейтингом, кількістю продажів і badge верифікованого автора.

Сторінки авторизації реалізовані через React Hook Form 7.78.0 із Zod-схемами валідації 4.4.3. Цей підхід дозволяє декларативно описати правила валідації окремо від компонента форми та отримати типобезпечну валідацію завдяки автоматичній генерації TypeScript-типів зі схеми Zod. Кожне поле відображає повідомлення про помилку безпосередньо під елементом введення у момент події onBlur або після першої невдалої спроби відправки форми. Форма входу реалізує двокроковий процес: на першому кроці вводяться email та пароль; якщо сервер повертає ознаку requiresTwoFactor, форма переходить до другого кроку — введення шестизначного TOTP-коду. QR-код для налаштування TOTP генерується на сторінці /dashboard/security компонентом бібліотеки qrcode.react v4.2.0 на основі URI, отриманого від ендпоінту GET /api/v1/auth/2fa/setup.

Особистий кабінет організований як окрема маршрутна зона зі спільним компонентом DashboardLayout, що містить sticky-сайдбар шириною 240 пікселів із рольовою навігацією та badge лічильника непрочитаних сповіщень. На

мобільному breakpoint сайдбар переходить у тимчасовий MUI Drawer. Активний пункт меню підсвічується акцентним кольором теми. Сторінка «Мої покупки» відображає таблицю MUI DataGrid із колонками: мініатюра, назва, дата покупки, тип ліцензії та кнопка завантаження. Завантаження файлу ініціюється GET-запитом до захищеного API-ендпоінту, що повертає presigned MinIO URL з TTL 15 хвилин. Сторінка аналітики автора відображає числові KPI-картки та динамічний AreaChart бібліотеки Recharts 3.8.1 із фільтрацією за діапазоном дат через MUI DateRangePicker.

Сторінка AssetUploadPage реалізує покроковий майстер із чотирма кроками на основі компонента MUI Stepper. Крок 1 «Основна інформація» містить поля назви, опису (Markdown), категорії та тегів. Крок 2 «Файли» реалізує завантаження основного файлу та до п'яти зображень попереднього перегляду. Крок 3 «Ліцензія та ціна» дозволяє обрати цінову модель. Крок 4 «Перегляд» підсумовує введені дані перед відправкою на модерацію. Стан усіх чотирьох кроків зберігається у єдиному об'єкті React Hook Form. Завантаження файлів виконується у два етапи: клієнт отримує від StorageService presigned PUT URL до MinIO, після чого виконує HTTP PUT безпосередньо до сховища, не навантажуючи API-сервер.

Адміністративна панель організована у сім розділів у рамках DashboardLayout із рольовою перевіркою ProtectedRoute requiredRole='ROLE_ADMIN'. AdminDashboardPage відображає чотири KPI-картки (кількість користувачів, активів, замовлень, загальний дохід). ModerationPage містить картки активів зі статусом PENDING та кнопками схвалення і відхилення з обов'язковим коментарем. AdminFinancePage організована у вкладки MUI Tabs: фінансовий огляд і виплати авторів через Stripe Connect. AdminAnalyticsPage відображає dual-axis chart із двома осями Y для порівняння трафіку та доходу. CategoriesPage та AdminBlogPage реалізують повний CRUD відповідних сутностей через модальні вікна MUI Dialog.

2.3 Розробка структури бази даних вебплатформи

База даних вебплатформи AssetMaster реалізована засобами СУБД PostgreSQL 15. Проєктування схеми виконано відповідно до принципів третьої нормальної форми (3NF): усунено надлишкове дублювання даних, кожна таблиця описує одну сутність предметної області, а зв'язки між сутностями реалізовані через зовнішні ключі.

Загальна структура бази даних складається з тринадцяти таблиць, що утворюють три функціональні групи: управління користувачами (users), каталог активів (categories, assets, reviews), комерційна обробка (orders, order_items, wishlist_items) та розширена функціональність (notifications, payouts, blog_posts, password_reset_tokens, email_verification_tokens). Зв'язки між усіма таблицями відображено на ER-діаграмі, що представлена в графічній частині 2026.КВР.122.421.10.00.00 БД.

2.3.1 Таблиця users

Таблиця users є центральною сутністю схеми і зберігає облікові записи всіх зареєстрованих користувачів незалежно від їхньої ролі. Первинний ключ — автоінкрементне ціле число типу BIGSERIAL. Роль зберігається як VARCHAR(50) зі значеннями 'ROLE_USER', 'ROLE_AUTHOR' та 'ROLE_ADMIN'.

Поле password_hash містить хеш пароля, обчислений алгоритмом BCrypt. Поле avatar_url зберігає ключ файлу у сховищі MinIO — безпосереднє посилання не зберігається, а генерується динамічно при кожному запиті. Поля totp_secret і totp_enabled забезпечують двофакторну автентифікацію, реалізовану власним алгоритмом HMAC-SHA1. Поля stripe_account_id і stripe_onboarding_complete забезпечують інтеграцію Stripe Connect для авторів. Унікальне обмеження накладено на поле email.

Структуру таблиці наведено у таблиці 2.1.

Таблиця 2.1 – Структура таблиці users

Поле	Тип	Обмеження	За замовч.	Опис
id	BIGSERIAL	PK	—	Унікальний ідентифікатор
email	VARCHAR(255)	NOT NULL, UNIQUE	—	Email-адреса (логін)
password_hash	VARCHAR(255)	NOT NULL	—	Хеш пароля (BCrypt)
role	VARCHAR(50)	NOT NULL	ROLE_USER	Роль користувача
display_name	VARCHAR(255)	—	NULL	Публічне ім'я користувача
avatar_url	VARCHAR(2048)	—	NULL	Ключ файлу аватара у MinIO
bio	TEXT	—	NULL	Біографія (для авторів)
is_verified	BOOLEAN	NOT NULL	false	Верифікований автор
email_verified	BOOLEAN	NOT NULL	false	Email підтверджено
totp_secret	VARCHAR(64)	—	NULL	Секрет TOTP (HMAC-SHA1)
totp_enabled	BOOLEAN	NOT NULL	false	Двофакторна автентифікація активна

2.3.2 Таблиця categories

Таблиця categories зберігає ієрархію категорій активів. Її структуру наведено у таблиці 2.2. Таблиця реалізує дворівневу ієрархію через самопосилання: поле parent_id посилається на первинний ключ тієї ж таблиці, що дозволяє визначати підкатегорії. Якщо parent_id має значення NULL, категорія є кореневою. Глибина ієрархії обмежена двома рівнями на рівні застосунку. Поле slug містить URL-безпечний ідентифікатор у форматі kebab-case та використовується для формування SEO-дружніх маршрутів виду /catalog/:categorySlug. Унікальні обмеження накладено на поля name і slug.

Таблиця 2.2 – Структура таблиці categories

Поле	Тип	Обмеження	За замовч.	Опис
id	BIGSERIAL	PK	—	Унікальний ідентифікатор
name	VARCHAR(100)	NOT NULL, UNIQUE	—	Назва категорії
slug	VARCHAR(100)	NOT NULL, UNIQUE	—	URL-slug (kebab-case)
parent_id	BIGINT	FK → categories(id)	NULL	Батьківська категорія (self-ref)
icon_url	VARCHAR(2048)	—	NULL	URL іконки категорії

2.3.3 Таблиця assets

Таблиця assets є центральною таблицею каталогу і зберігає всі метадані цифрових активів. Поле status визначає стан активу в циклі модерації та може

приймати значення DRAFT (чернетка), PENDING (очікує перевірки), PUBLISHED (опубліковано) та REJECTED (відхилено). Поле rejection_reason дозволяє адміністратору вказати причину відхилення, яка надсилається автору в email-сповіщенні. Ключовим архітектурним рішенням є зберігання тегів безпосередньо у полі tags типу TEXT[] — масив PostgreSQL — замість окремої таблиці зв'язків. Це усуває необхідність JOIN-операції при запитах каталогу та спрощує повнотекстовий пошук. Поле preview_urls типу JSONB зберігає впорядкований масив ключів файлів попереднього перегляду у сховищі MinIO. Поле file_key містить ключ основного файлу активу; безпосереднє посилання ніколи не зберігається у базі даних — воно генерується динамічно сервером при кожному запиті на завантаження як presigned URL із TTL 15 хвилин. Структуру таблиці assets наведено у таблиці 2.3

Таблиця 2.3 – Структура таблиці assets

Поле	Тип	Обмеження	За замовч.	Опис
1	2	3	4	5
id	BIGSERIAL	PK	—	Унікальний ідентифікатор
author_id	BIGINT	NOT NULL, FK → users	—	Автор активу
category_id	BIGINT	FK → categories	NULL	Категорія активу
title	VARCHAR(500)	NOT NULL	—	Назва активу
description	TEXT	—	NULL	Опис активу
price	NUMERIC(10,2)	NOT NULL	—	Ціна у USD
license_type	VARCHAR(50)	NOT NULL	STANDARD	STANDARD / COMMERCIAL

Продовження таблиці 2.3

1	2	3	4	5
status	VARCHAR(50)	NOT NULL	DRAFT	DRAFT / PENDING / PUBLISHED / REJECTED
file_key	VARCHAR(2048)	—	NULL	Ключ основного файлу у MinIO
preview_urls	JSONB	NOT NULL	'[]'	Масив ключів файлів прев'ю (JSON)
tags	TEXT[]	NOT NULL	'{}'	Масив тегів
downloads_count	INTEGER	NOT NULL	0	Лічильник завантажень
views_count	INTEGER	NOT NULL	0	Лічильник переглядів
rejection_reason	TEXT	—	NULL	Причина відхилення при модерації
created_at	TIMESTAMPTZ	NOT NULL	NOW()	Дата публікації

2.3.4 Таблиці orders та order_items

Таблиці orders та order_items разом реалізують модель замовлень платформи. Їх структуру наведено у таблиці 2.4. Таблиця orders фіксує кожне замовлення: поле status відображає його стан зі значеннями PENDING (сформовано, оплата не підтверджена), PAID (оплачено) або REFUNDED (повернення коштів). Таблиця order_items деталізує склад замовлення на рівні

окремих активів. Поле price_at_purchase фіксує ціну активу на момент купівлі незалежно від подальших змін ціни автором — це гарантує коректну фінансову звітність. Поле license_type вказує тип придбаної ліцензії. Зовнішній ключ asset_id має обмеження ON DELETE RESTRICT, що не дозволяє видалити актив, якщо він входить до складу хоча б одного замовлення.

Таблиця 2.4 – Структура таблиці orders та order_items

Поле	Тип	Обмеження	Опис
1	2	3	4
orders			
id	BIGSERIAL	PK	Унікальний ідентифікатор замовлення
buyer_id	BIGINT	NOT NULL, FK → users CASCADE	Покупець
total_amount	NUMERIC(10,2)	NOT NULL	Загальна сума замовлення (USD)
status	VARCHAR(20)	NOT NULL / PENDING	PENDING / PAID / REFUNDED
created_at	TIMESTAMPTZ	NOT NULL / NOW()	Дата створення замовлення
order_items			
id	BIGSERIAL	PK	Унікальний ідентифікатор рядка
order_id	BIGINT	NOT NULL, FK → orders CASCADE	Замовлення

Продовження таблиці 2.4

1	2	3	4
asset_id	BIGINT	NOT NULL, FK → assets RESTRICT	Придбаний актив
price_at_purchase	NUMERIC(10,2)	NOT NULL	Ціна активу на момент покупки
license_type	VARCHAR(20)	NOT NULL	STANDARD / COMMERCIAL

2.3.5 Таблиця wishlist_items

Таблиця wishlist_items реалізує список бажань користувача. Її структуру наведено у таблиці 2.5. Обидва зовнішні ключі визначені з ON DELETE CASCADE: видалення користувача або активу автоматично очищає відповідні записи вішліста. Складений унікальний індекс UNIQUE(user_id, asset_id) на рівні бази даних унеможливлює повторне додавання одного активу до списку бажань одним користувачем.

Таблиця 2.5 – Структура таблиці wishlist_items

Поле	Тип	Обмеження	Опис
id	BIGSERIAL	PK	Унікальний ідентифікатор запису
user_id	BIGINT	NOT NULL, FK → users CASCADE	Власник списку бажань
asset_id	BIGINT	NOT NULL, FK → assets	Актив у списку
created_at	TIMESTAMPTZ	NOT NULL, DEFAULT NOW()	Дата додавання

2.3.6 Таблиця reviews

Таблиця reviews зберігає відгуки покупців на придбані активи. Її структуру наведено у таблиці 2.6. Унікальне обмеження UNIQUE(asset_id, author_id) гарантує, що один користувач може залишити не більше одного відгуку на один актив. Поле rating обмежено CHECK-constraint значеннями від 1 до 5. Середній рейтинг активу не зберігається у таблиці assets — він обчислюється агрегатним запитом SQL через метод getStats сервісу ReviewService, що гарантує актуальність значення.

Таблиця 2.6 – Структура таблиці reviews

Поле	Тип	Обмеження	За замовч.	Опис
id	BIGSERIAL	PK	—	Унікальний ідентифікатор відгуку
asset_id	BIGINT	NOT NULL, FK → assets CASCADE	—	Актив, якому написано відгук
author_id	BIGINT	NOT NULL, FK → users CASCADE	—	Автор відгуку
rating	INTEGER	NOT NULL, CHECK (1..5)	—	Оцінка від 1 до 5
comment	TEXT	—	NULL	Текст коментаря
created_at	TIMESTAMPTZ	NOT NULL	NOW()	Дата публікації відгуку

2.3.7 Таблиця notifications

Таблиця notifications зберігає системні сповіщення користувачів. Поле type класифікує сповіщення (наприклад, ASSET_APPROVED, ASSET_REJECTED, PAYOUT_PROCESSED). Поле link містить URL для переходу за сповіщенням.

Поле is_read відстежує статус прочитання; для прискорення запиту лічильника непрочитаних сповіщень визначено частковий індекс idx_notifications_user_unread (WHERE is_read = FALSE), що мінімізує розмір індексу. Структуру таблиці notifications наведено у таблиці 2.7

Таблиця 2.7 – Структура таблиці notifications

Поле	Тип	Обмеження	За замовч.	Опис
id	BIGSERIAL	PK	—	Унікальний ідентифікатор
user_id	BIGINT	NOT NULL, FK → users CASCADE	—	Отримувач сповіщення
type	VARCHAR(50)	NOT NULL	—	Тип сповіщення
title	VARCHAR(255)	NOT NULL	—	Заголовок сповіщення
body	TEXT	—	NULL	Текст сповіщення
link	VARCHAR(500)	—	NULL	URL для переходу
is_read	BOOLEAN	NOT NULL	false	Статус прочитання
created_at	TIMESTAMP	NOT NULL	NOW()	Дата створення

2.3.8 Таблиця payouts

Таблиця payouts фіксує виплати авторам через Stripe Connect. Її структуру наведено у таблиці 2.8. Поле status є власним enum-типом payout_status зі значеннями PENDING, PROCESSING, PAID та FAILED, що відображають повний цикл виплати.

Поля period_start та period_end визначають розрахунковий період, за який нараховується виплата. Поле stripe_transfer_id зберігає ідентифікатор переказу в системі Stripe і дозволяє відстежити транзакцію та здійснити повернення у разі необхідності.

Таблиця 2.8 – Структура таблиці payouts

Поле	Тип	Обмеження	За замовч.	Опис
1	2	3	4	5
id	BIGSERIAL	PK	—	Унікальний ідентифікатор виплати
author_id	BIGINT	NOT NULL, FK → users CASCADE	—	Отримувач виплати
amount	NUMERIC(12,2)	NOT NULL	—	Сума виплати у USD
status	payout_status	NOT NULL	PENDING	PENDING / PROCESSING / PAID / FAILED
period_start	DATE	NOT NULL	—	Початок розрахункового періоду

Продовження таблиці 2.8

1	2	3	4	5
period_end	DATE	NOT NULL	—	Кінець розрахункового періоду
processed_at	TIMESTAMP	—	NULL	Дата фактичної виплати
notes	TEXT	—	NULL	Коментар адміністратора
stripe_transfer_id	VARCHAR(100)	—	NULL	ID переказу Stripe
created_at	TIMESTAMP	NOT NULL	NOW()	Дата створення запису

2.3.9 Таблиця blog_posts

Таблиця blog_posts зберігає статті вбудованого блогу платформи. Її структуру наведено у таблиці 2.9. Поле slug є унікальним URL-ідентифікатором статті та використовується у маршруті /blog/:slug для SEO-дружньої навігації. Поле published є булевим прапорцем публікації: статті з published = false є чернетками і не відображаються у публічному блозі. Поле author_id має обмеження ON DELETE SET NULL, тому видалення облікового запису автора не призводить до видалення його статей.

Таблиця 2.9 – Структура таблиці blog_posts

Поле	Тип	Обмеження	За замовч.	Опис
1	2	3	4	5
id	BIGSERIAL	PK	—	Унікальний ідентифікатор
slug	VARCHAR(200)	NOT NULL, UNIQUE	—	URL-slug статті

Продовження таблиці 2.9

1	2	3	4	5
title	VARCHAR(500)	NOT NULL	—	Заголовок статті
tag	VARCHAR(100)	—	NULL	Тег статті
excerpt	TEXT	—	NULL	Короткий анонс
content	TEXT	NOT NULL	—	Повний текст статті
published	BOOLEAN	NOT NULL	false	Опублікована / Чернетка
read_time	VARCHAR(50)	—	NULL	Приблизний час читання
author_id	BIGINT	FK → users SET NULL	NULL	Автор статті
created_at	TIMESTAMP	NOT NULL	NOW()	Дата створення
updated_at	TIMESTAMP	NOT NULL	NOW()	Дата останнього оновлення

2.3.10 Таблиці password_reset_tokens та email_verification_tokens

Таблиці password_reset_tokens та email_verification_tokens забезпечують безпечні flow відновлення пароля та підтвердження адреси електронної пошти. Їх структуру наведено у таблиці 2.10. Обидві таблиці мають схожу будову: поле token містить унікальний криптографічно стійкий рядок довжиною 64 символи, що надсилається користувачеві в email-листі. Поле expires_at визначає термін дії токена. Для таблиці password_reset_tokens додатково передбачено поле used (булевий прапорець), що запобігає повторному використанню токена після першого успішного скидання пароля. Обидва зовнішні ключі визначені з ON DELETE CASCADE.

Таблиця 2.10 – Структура таблиць password_reset_tokens та email_verification_tokens

Поле	Тип	Обмеження	Опис
Поля таблиці password_reset_tokens			
id	BIGSERIAL	PK	Унікальний ідентифікатор
user_id	BIGINT	NOT NULL, FK → users CASCADE	Власник токена
token	VARCHAR(64)	NOT NULL, UNIQUE	Токен відновлення пароля
expires_at	TIMESTAMPTZ	NOT NULL	Термін дії токена
used	BOOLEAN	NOT NULL, DEFAULT false	Токен вже використано
created_at	TIMESTAMPTZ	NOT NULL, DEFAULT NOW()	Дата створення
Поля таблиці email_verification_tokens			
id	BIGSERIAL	PK	Унікальний ідентифікатор
user_id	BIGINT	NOT NULL, FK → users CASCADE	Власник токена
token	VARCHAR(64)	NOT NULL, UNIQUE	Токен підтвердження email
expires_at	TIMESTAMPTZ	NOT NULL	Термін дії токена
created_at	TIMESTAMPTZ	NOT NULL, DEFAULT NOW()	Дата створення

2.4 Програмування вебплатформи

2.4.1 Написання клієнтської частини

Клієнтська частина являє собою SPA, ініціалізований через Vite 7.3.5 із шаблоном react-ts. Кореневий компонент App.tsx поєднує три провайдери: QueryClientProvider (TanStack Query), MUI ThemeProvider та AuthContext — власний React Context, що зберігає стан автентифікації та надає методи login, logout, refreshToken.

API-клієнт у файлі src/shared/api/client.ts створює єдиний екземпляр Axios із базовою URL зі змінної середовища. Перехоплювач запитів додає заголовок Authorization: Bearer <token> із JWT, що зберігається у AuthContext [13]. Перехоплювач відповідей обробляє статус 401: виконує запит до /api/v1/auth/refresh, і у разі успіху повторює оригінальний запит із новим токеном; при невдачі — очищає стан авторизації та перенаправляє на /auth/login. Повний текст програмного коду наведено у Додатку А.

Маршрутизація та захист маршрутів

Маршрутизація у src/app/router.tsx реалізована через createBrowserRouter із React Router v7.17.0. Компонент ProtectedRoute (src/shared/components/) перевіряє наявність авторизованого користувача в AuthContext; при необхідності — перевіряє requiredRole. Неавторизований запит перенаправляє на /auth/login, запит із недостатньою роллю — на /. Усі компоненти сторінок завантажуються через lazy() + Suspense. Повний текст програмного коду наведено у Додатку Б.

Управління серверним станом

Запити до API виконуються виключно через хуки TanStack Query v5. Для кожного ресурсу у відповідних директоріях entities/ та features/ визначено хуки з Query Key Factories. Нескінченне прокручування каталогу реалізовано через useInfiniteQuery у компоненті AssetGrid: Intersection Observer спостерігає за sentinel-елементом і при його появі у видимій зоні викликає fetchNextPage. Вішліст реалізує optimistic update: при додаванні або видаленні активу стан кешу

оновлюється негайно, а у разі помилки сервера відкочується назад. Текст програмного коду файлу AssetGrid.tsx наведено у Додатку В.

Двофакторна автентифікація (TOTP)

Сторінка входу /auth/login реалізує двокроковий процес. На першому кроці перевіряються email та пароль; якщо сервер повертає поле requiresTwoFactor: true, форма переходить до другого кроку — введення шестизначного TOTP-коду. QR-код для налаштування TOTP генерується на сторінці /dashboard/security через компонент бібліотеки qrcode.react v4.2.0 на основі URI, отриманого від ендпоінту GET /api/v1/auth/2fa/setup. Повний текст програмного коду файлу «LoginPage.tsx» наведено у Додатку Г

2.4.2 Написання серверної частини

Серверна частина реалізована як Spring Boot 3.3.5 застосунок із 11 REST-контролерами, 15 сервісними класами та 12 JPA-сутностями [3, 4]. Збірка виконується Maven. Клас ApiApplication.java анотований @SpringBootApplication та @EnableAsync — остання анотація дозволяє асинхронне виконання методів сервісів (зокрема відправку email через EmailService).

Налаштування безпеки (SecurityConfig)

Клас SecurityConfig у пакеті config налаштовує Spring Security у stateless-режимі. Анотація @EnableMethodSecurity дозволяє використання @PreAuthorize на рівні методів сервісів для тонкого контролю доступу. CSRF-захист вимкнено (stateless API не потребує). CORS налаштовано для origins із application.yml (localhost:5173, localhost:5174) із дозволеними методами GET, POST, PUT, PATCH, DELETE, OPTIONS та credentials: true для cookie-based refreshToken.

Правила авторизації запитів визначено у ланцюжку authorizeHttpRequests: GET /api/v1/auth/me, PATCH /api/v1/auth/me та POST /api/v1/auth/me/avatar вимагають автентифікації; усі /api/v1/auth/** дозволені публічно (окрім перелічених вище); GET /api/v1/assets/**, /api/v1/categories/**, /api/v1/blog/** дозволені публічно; POST /api/v1/stripe/webhooks дозволено публічно

(верифікація через підпис Stripe); Swagger UI та /actuator/health дозволені публічно; усі інші запити вимагають автентифікації. Кастомний JwtAuthFilter (OncePerRequestFilter) витягує Bearer-токен, парсить і валідує його через JwtService (JJWT 0.12.6) та встановлює SecurityContext.

REST-контролери та сервісний шар

Платформа реалізує 11 REST-контролерів. AuthController (/api/v1/auth) охоплює 15 ендпоінтів, включно з повним циклом верифікації email, відновлення пароля та управління TOTP (setup, enable, disable, verify). AssetController (/api/v1/assets) надає ендпоінти каталогу, trending, new, search та mine. AdminController (/api/v1/admin) є найбільшим — 20+ ендпоінтів для управління активами, користувачами, фінансами, категоріями та блогом. StripeController (/api/v1/stripe) забезпечує Stripe Connect onboarding для авторів та обробку webhook-сповіщень. Перелік розроблених REST-контролерів та їх маршрути наведено у таблиці 2.11.

Таблиця 2.11 – .Перелік REST-контролерів та їх маршрути

Клас	@RequestMapping	Ключові ендпоінти
1	2	3
AuthController	/api/v1/auth	register, login, refresh, logout, me, 2fa/*, verify-email, forgot/reset-password
AssetController	/api/v1/assets	GET /, /trending, /new, /search, /{id}, /mine; POST, PUT, DELETE /{id}
CategoryController	/api/v1/categories	GET /
OrderController	/api/v1/orders	POST /, GET /, GET /{id}, GET /{orderId}/download/{assetId}
WishlistController	/api/v1/wishlist	GET /, POST /{assetId}, DELETE /{assetId}, GET /check/{assetId}

Продовження таблиці 2.11

1	2	3
ReviewController	/api/v1/assets/{asset Id}/reviews	GET /stats, GET /, POST /
NotificationControlle r	/api/v1/notifications	GET /, GET /unread-count, PATCH /{id}/read, PATCH /read-all
DashboardController	/api/v1/dashboard	GET /analytics, GET /payouts
BlogPostController	/api/v1/blog	GET /, GET /{slug}
StripeController	/api/v1/stripe	POST /connect/onboard, GET /connect/status, POST /webhooks
AdminController	/api/v1/admin	stats, assets/pending, approve/reject, users, finance/payouts/transfer, analytics, categories, blog

Також вебплатформа містить 15 сервісних класів. AuthService реалізує повний цикл автентифікації з 13 методами, включно з власною реалізацією TOTP через TotpService (HMAC-SHA1 без зовнішніх бібліотек) Код класу AuthService наведено у додатку Е.

StorageService інкапсулює роботу з MinIO 8.5.11 і надає методи uploadFile, uploadAvatarFile та generatePresignedUrl. StripeService забезпечує Stripe Connect onboarding для авторів (createExpressAccount, createOnboardingLink) та виконання виплат (createTransfer, constructWebhookEvent) через stripe-java SDK 25.3.0. EmailService відправляє транзакційні листи через SMTP Gmail (spring-boot-starter-mail).

Серверна частина використовує Spring Boot 3.3.5 як батьківський POM. Ключові залежності: spring-boot-starter-web (embedded Tomcat), spring-boot-starter-data-jpa (Hibernate ORM), spring-boot-starter-security, spring-boot-starter-validation

(Jakarta Bean Validation), spring-boot-starter-mail (SMTP), spring-boot-starter-actuator (/actuator/health).

2.5 Тестування вебплатформи

Тестування вебплатформи AssetMaster здійснювалось методом мануального функціонального тестування основних сторінок і сценаріїв використання. Мета тестування — перевірка відповідності реалізованого функціоналу вимогам, визначеним у підрозділі 1.2.3, та виявлення дефектів інтерфейсу й логіки роботи перед розгортанням платформи.

Тестування охоплює вісім функціональних блоків, що відповідають основним зонам вебплатформи: публічна зона (головна сторінка, каталог, сторінка продукту), блок авторизації (реєстрація, вхід, двофакторна автентифікація), особистий кабінет покупця (кошик, оформлення замовлення, завантаження файлу), кабінет автора (завантаження активу, аналітика), адміністративна панель (модерація, управління користувачами, фінанси). Для кожного блоку визначено тест-кейси з кроками відтворення, очікуваним та фактичним результатами. Скріншоти основних сторінок і станів інтерфейсу наведені у тексті поряд із відповідними тест-кейсами.

2.5.1 Тестування публічної зони

Публічна зона охоплює сторінки, доступні без авторизації: головну сторінку, каталог активів та сторінку окремого активу. Тестування цього блоку спрямоване на перевірку коректності відображення контенту, роботи фільтрів, пошуку та навігації.

Головна сторінка. При відкритті головної сторінки у браузері відображаються всі заплановані секції: Hero-блок із заголовком і кнопками заклику до дії, смуга категоріальних фільтрів, секції «У тренді цього тижня» та «Нові надходження» із сітками карток активів, блок топ-авторів, CTA-банер для

продавців і відгуки покупців. Зовнішній вигляд головної сторінки наведено на рисунку 2.1.

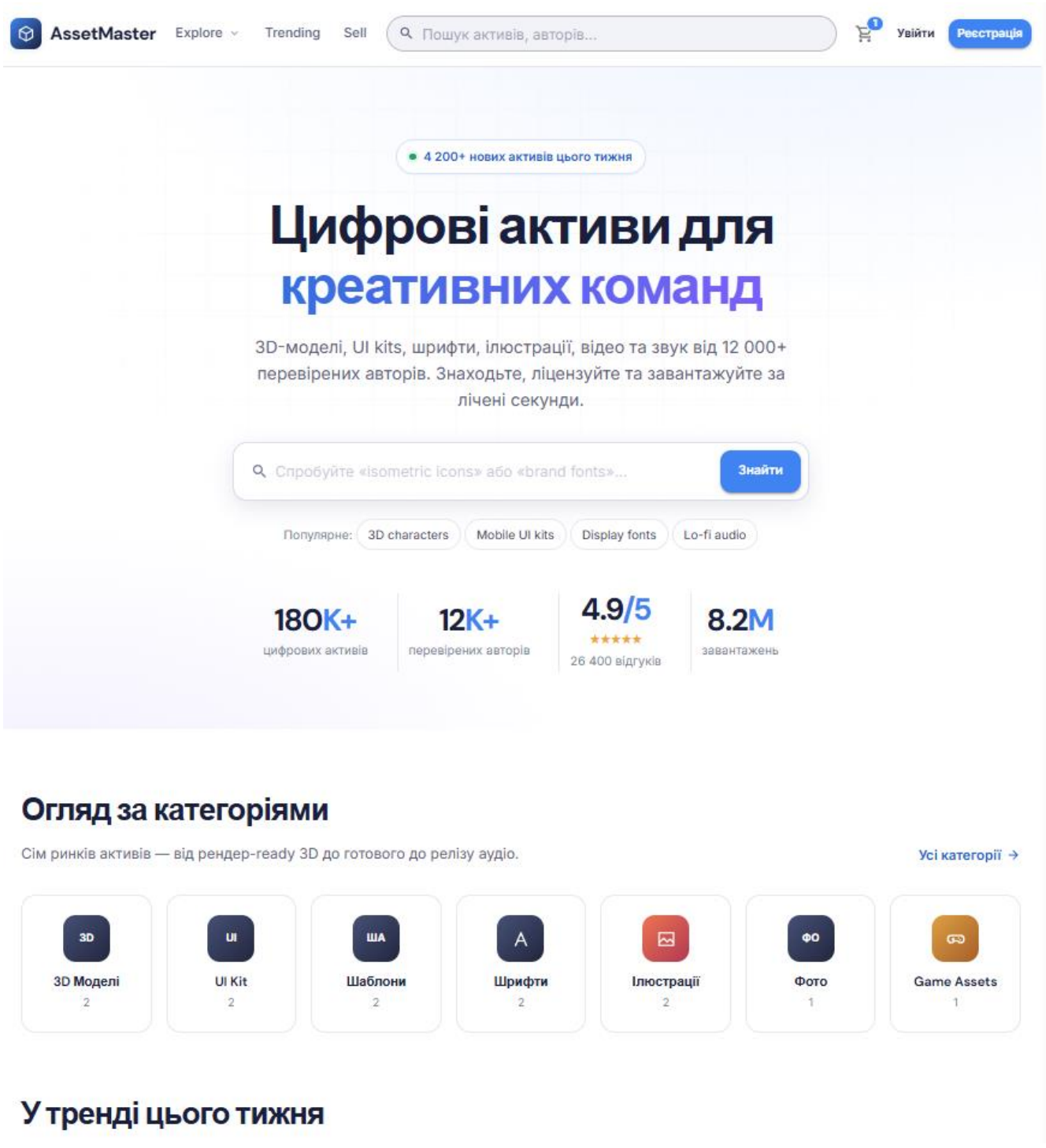


Рисунок 2.1 — Головна сторінка платформи AssetMaster

Каталог активів (/catalog). Перевірено роботу фільтрів бічної панелі: при виборі категорії, діапазону цін або типу ліцензії сітка карток оновлюється без

перезавантаження сторінки, а параметри фільтрів відображаються в URL для можливості збереження та передачі посилання. Перевірено нескінченне прокручування: при досягненні нижньої частини списку автоматично завантажується наступна порція активів. Зовнішній вигляд сторінки каталогу з активними фільтрами наведено на рисунку 2.2.

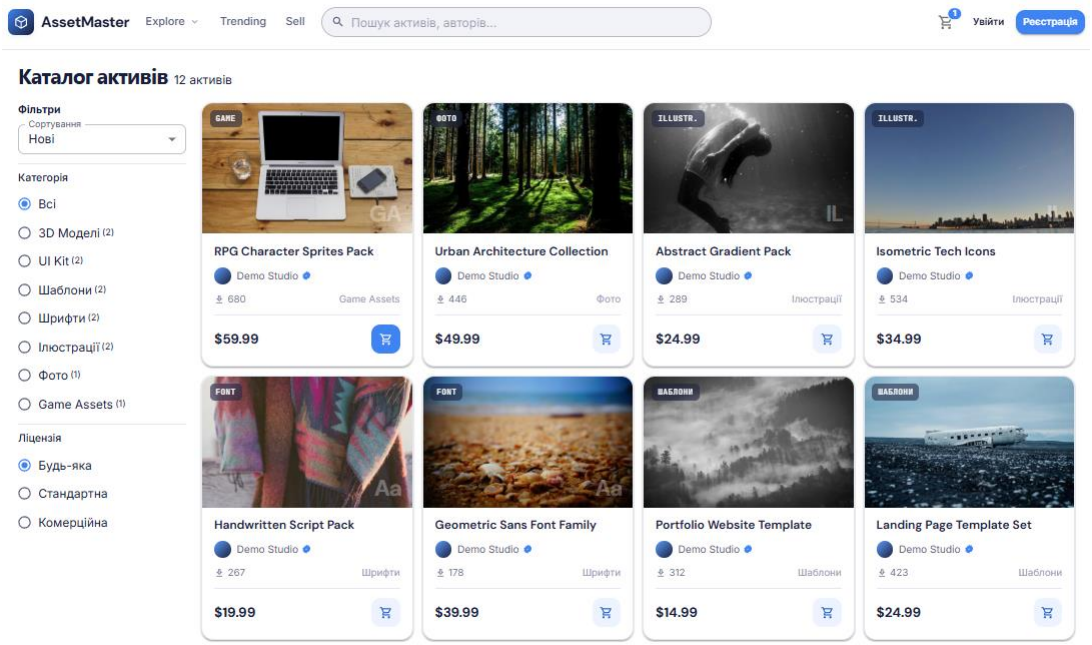


Рисунок 2.2 — Сторінка каталогу активів

Глобальний пошук. При введенні запиту в поле пошуку Navbar після затримки 300 мс відображається dropdown із автодоповненням, згрупованим за типом: активи, автори, категорії. При переході на сторінку /search результати коректно фільтруються відповідно до запиту завдяки повнотекстовому пошуку PostgreSQL. Роботу пошукового dropdown наведено на рисунку 2.3.

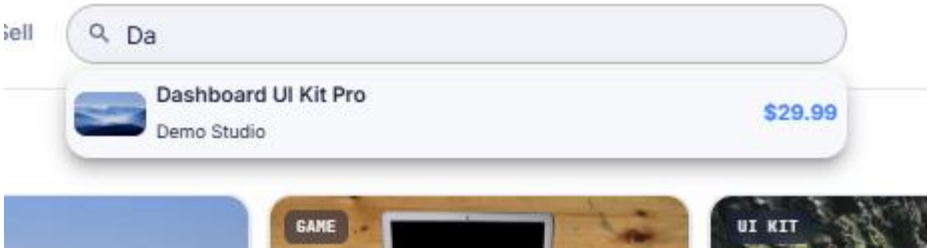


Рисунок 2.3 — Автодоповнення у компоненті SearchBar

Сторінка продукту (/assets/:id). Перевірено коректність відображення слайдера прев'ю, sticky-блоку придбання у правій колонці, вкладок із описом та характеристиками, а також секції відгуків. Перевірено переключення між стандартною та комерційною ліцензіями через LicenseToggle: ціна оновлюється відповідно до обраного типу. Зовнішній вигляд сторінки продукту наведено на рисунку 2.4.

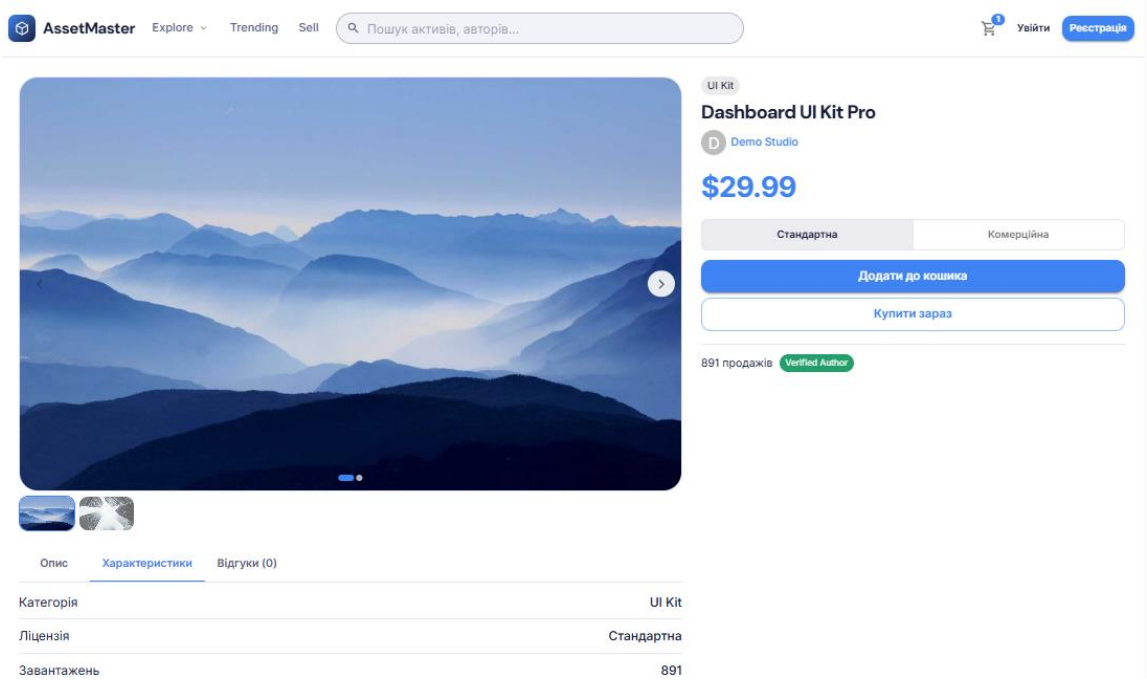


Рисунок 2.4 — Сторінка окремого активу

2.5.2 Тестування блоку авторизації

Блок авторизації охоплює сторінки реєстрації, входу з підтримкою TOTP та відновлення пароля. Тестування спрямоване на перевірку валідації форм, коректності повідомлень про помилки та двокрокового процесу входу.

Реєстрація (/auth/register). Перевірено валідацію полів форми: при спробі відправки порожньої форми під кожним полем відображається відповідне повідомлення про помилку (React Hook Form + Zod). При введенні некоректного формату email відображається повідомлення «Некоректна адреса електронної пошти». При успішній реєстрації користувач перенаправляється на головну

сторінку, а на вказану адресу надсилається лист підтвердження. Сторінку реєстрації із помилками валідації наведено на рисунку 2.5.

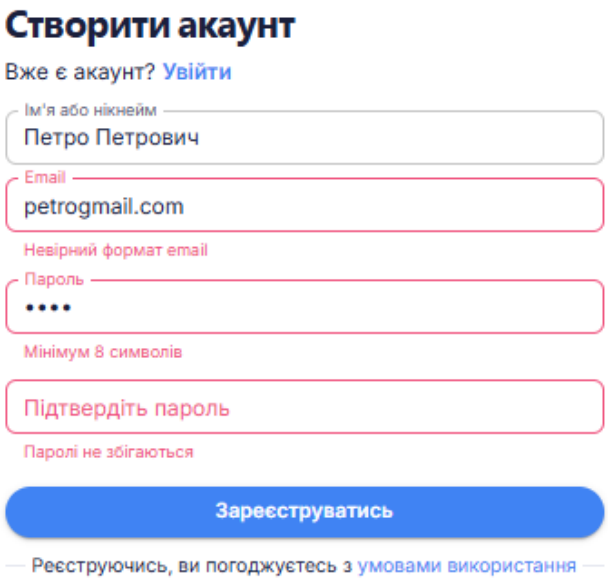


Рисунок 2.5 — Сторінка реєстрації із помилками валідації

Вхід (/auth/login). Перевірено двокроковий процес входу. На першому кроці при введенні некоректних облікових даних відображається повідомлення про помилку без розкриття деталей («Невірний email або пароль»). При активованому TOTP після успішного введення пароля форма автоматично переходить до другого кроку — введення шестизначного коду. При введенні простроченого або невірного коду відображається відповідне повідомлення. Двокроковий процес входу наведено на рисунку 2.6.

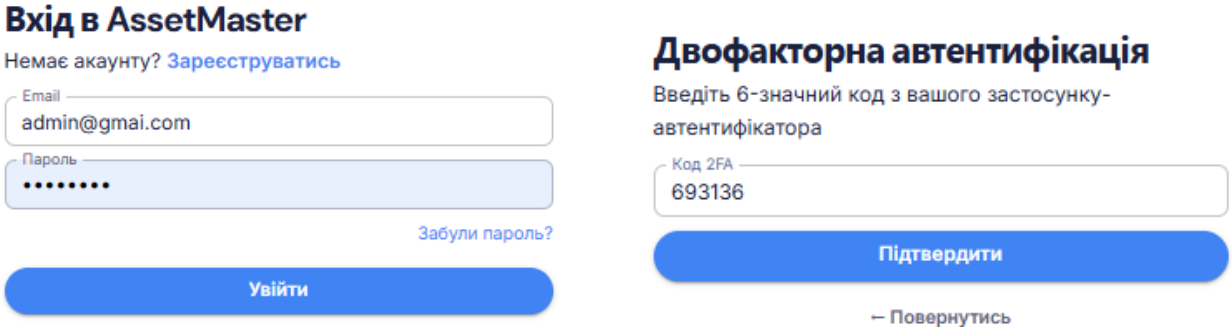


Рисунок 2.6 — Двокроковий процес входу до платформи

Відновлення пароля. Перевірено flow відновлення: при введенні зареєстрованого email система надсилає лист із посиланням на /auth/reset-password із токеном. Форма скидання пароля валідує відповідність двох введених паролів. Зовнішній вигляд форми відновлення пароля наведено на рисунку 2.7.

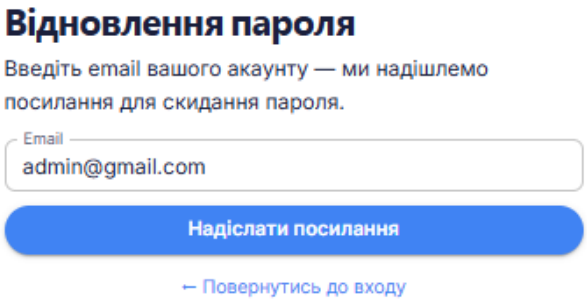


Рисунок 2.7 — Сторінка відновлення пароля

2.5.3 Тестування особистого кабінету покупця

Тестування кабінету покупця охоплює управління кошиком, процес оформлення замовлення та завантаження придбаних файлів.

Кошик (/cart). Перевірено додавання активів до кошика зі сторінки продукту та через кнопку на картці. Стан кошика зберігається у localStorage: при перезавантаженні сторінки або відкритті нової вкладки браузера активи залишаються у кошику. Кількість елементів коректно відображається у badge іконки кошика в Navbar. Зовнішній вигляд кошика наведено на рисунку 2.8.

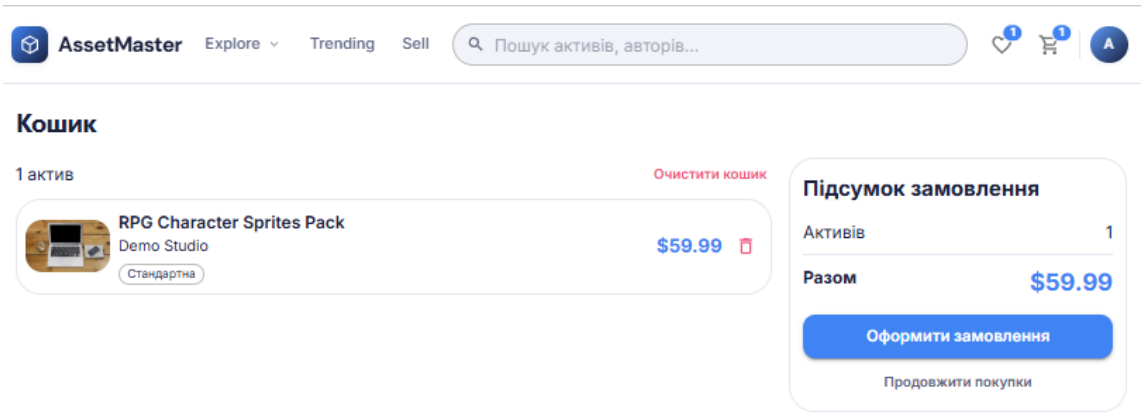


Рисунок 2.8 — Сторінка кошика

Оформлення замовлення (/checkout). Перевірено перехід до checkout із кошика, відображення підсумку замовлення та форми оплати. Після успішної оплати відбувається перенаправлення на /checkout/success, де відображається підтвердження із номером замовлення. Замовлення з'являється у розділі «Мої покупки» особистого кабінету. Сторінку підтвердження успішної оплати наведено на рисунку 2.9.

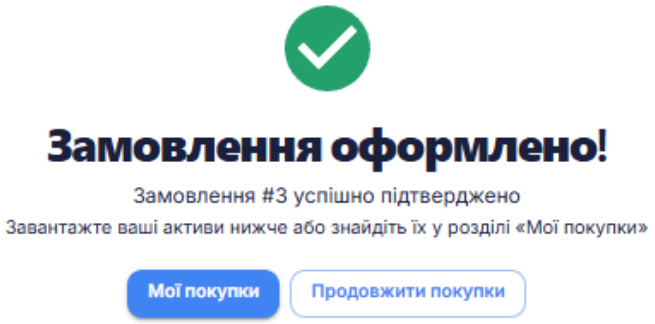


Рисунок 2.9 — Сторінка успішного оформлення замовлення

Завантаження придбаних активів (/dashboard/purchases). Перевірено відображення таблиці придбаних активів із колонками: мініатюра, назва, дата покупки, тип ліцензії та кнопка завантаження. При натисканні кнопки виконується запит до API, сервер перевіряє факт покупки і повертає presigned MinIO URL з TTL 15 хвилин; файл завантажується у браузері. Перевірено, що спроба отримати файл непридбаного активу через API повертає 403 Forbidden. Сторінку «Мої покупки» наведено на рисунку 2.10.

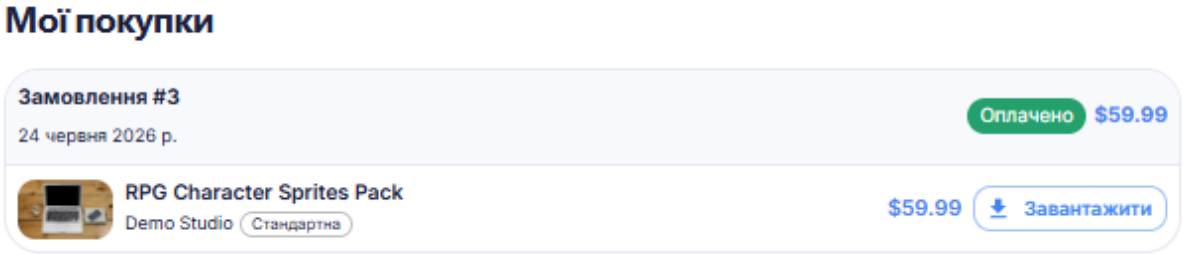


Рисунок 2.10 — Сторінка «Мої покупки» в особистому кабінеті

2.5.3 Тестування кабінету автора

Тестування кабінету автора охоплює процес завантаження нового активу, управління портфоліо та перегляд аналітики продажів.

Завантаження нового активу (/dashboard/assets/new). Перевірено чотириступеневий покроковий майстер: на кожному кроці валідуються обов'язкові поля, перехід до наступного кроку заблоковано при наявності помилок. На кроці завантаження файлів перевірено пряме завантаження основного файлу та зображень прев'ю до MinIO через presigned PUT URL. Після відправки форми актив отримує статус PENDING і з'являється у черзі модерації адміністратора. Форму завантаження активу (крок 1) наведено на рисунку 2.11.

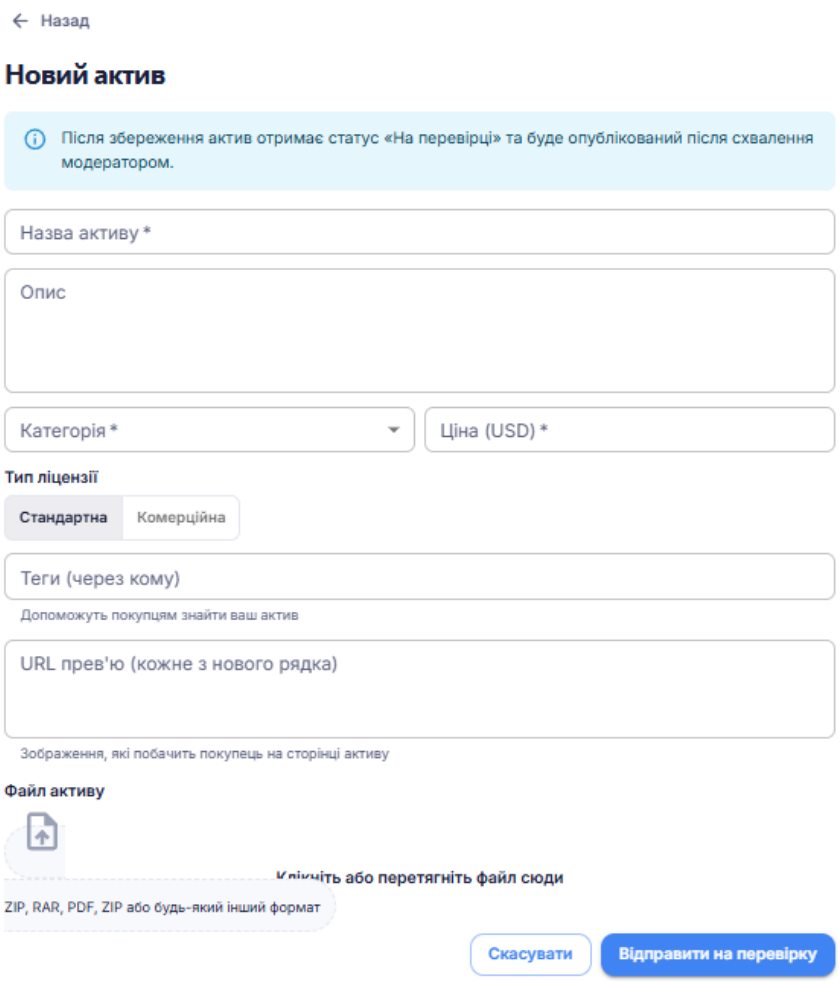


Рисунок 2.11 — Покроковий майстер завантаження нового активу (крок 1)

Портфоліо автора (/dashboard/assets). Перевірено відображення таблиці власних активів із кольоровими badge-статусів (DRAFT, PENDING, PUBLISHED, REJECTED). Для відхиленого активу відображається причина відмови від адміністратора. Перевірено редагування метаданих активу та його видалення. Зовнішній вигляд сторінки портфоліо наведено на рисунку 2.12.

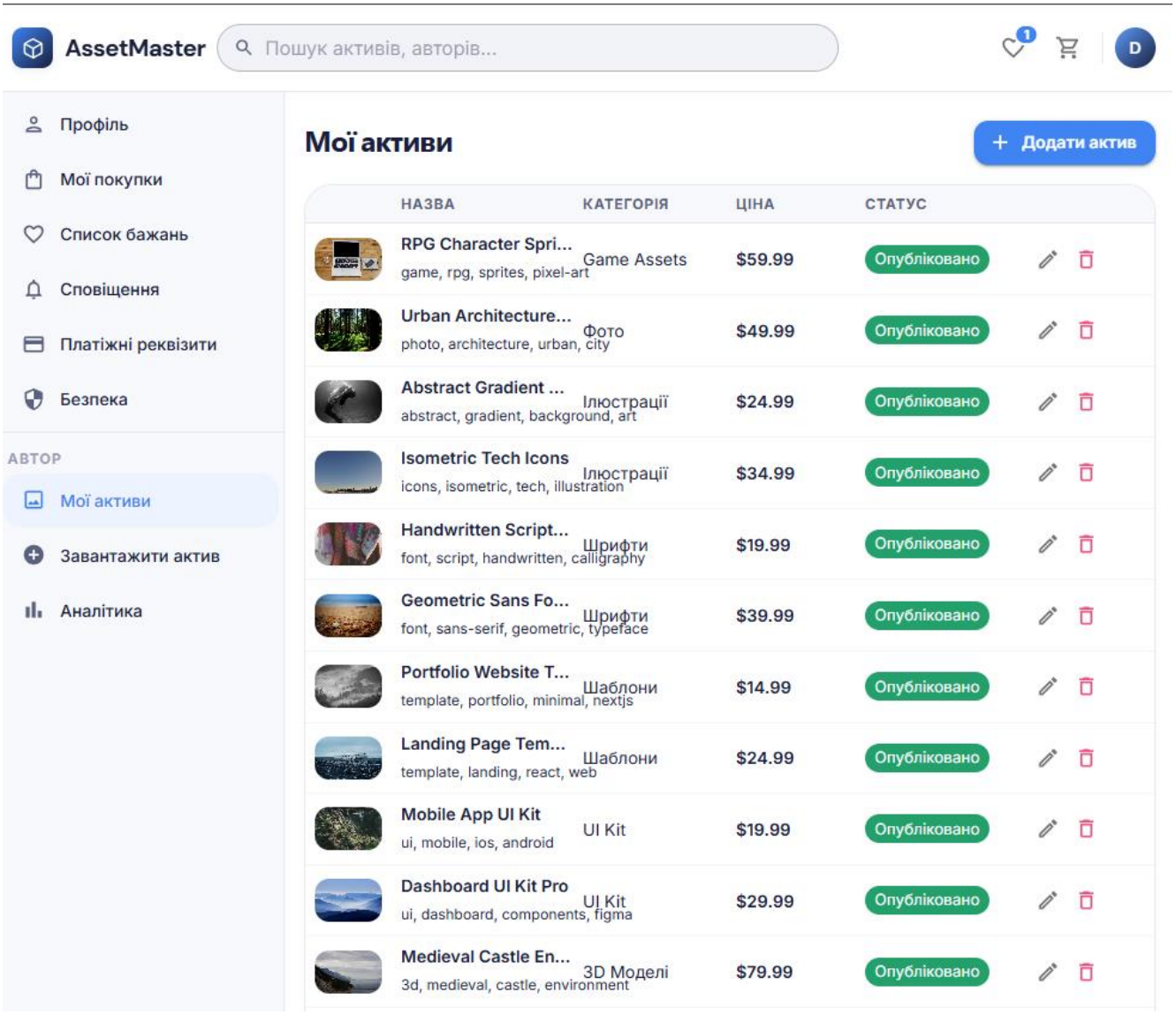


Рисунок 2.12 — Сторінка «Мої активи» в кабінеті автора

Аналітика продажів (/dashboard/analytics). Перевірено відображення KPI-карток (загальний дохід, кількість продажів, кількість переглядів) та графіка динаміки продажів (Recharts AreaChart) із фільтрацією за діапазоном дат. При

зміні діапазону дат графік та картки оновлюються коректно. Зовнішній вигляд сторінки аналітики наведено на рисунку 2.13.

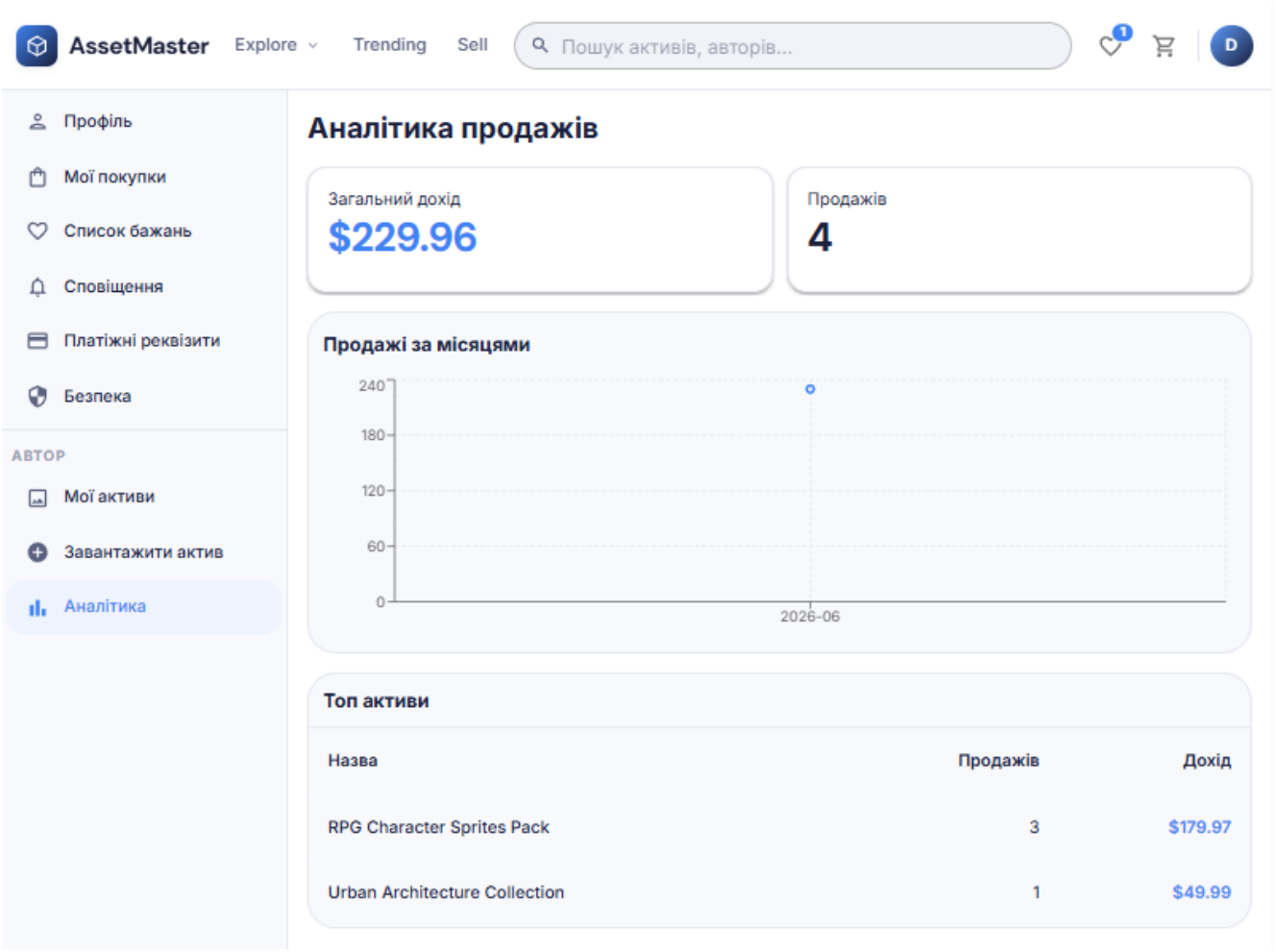


Рисунок 2.13 — Сторінка аналітики продажів в кабінеті автора

2.5.4 Тестування адміністративної панелі

Тестування адміністративної панелі охоплює модерацію активів, управління обліковими записами користувачів та фінансовий модуль.

Модерація активів (/admin/moderation). Перевірено відображення черги активів зі статусом PENDING. При натисканні «Схвалити» статус активу змінюється на PUBLISHED, актив з'являється у публічному каталозі, а автор отримує email-сповіщення. При натисканні «Відхилити» відкривається модальне вікно для введення причини відмови; після підтвердження статус змінюється на

REJECTED, автор отримує email із причиною. Перевірено, що спроба схвалити або відхилити актив зі статусом відмінним від PENDING повертає 409 Conflict. Сторінку модерації наведено на рисунку 2.14.

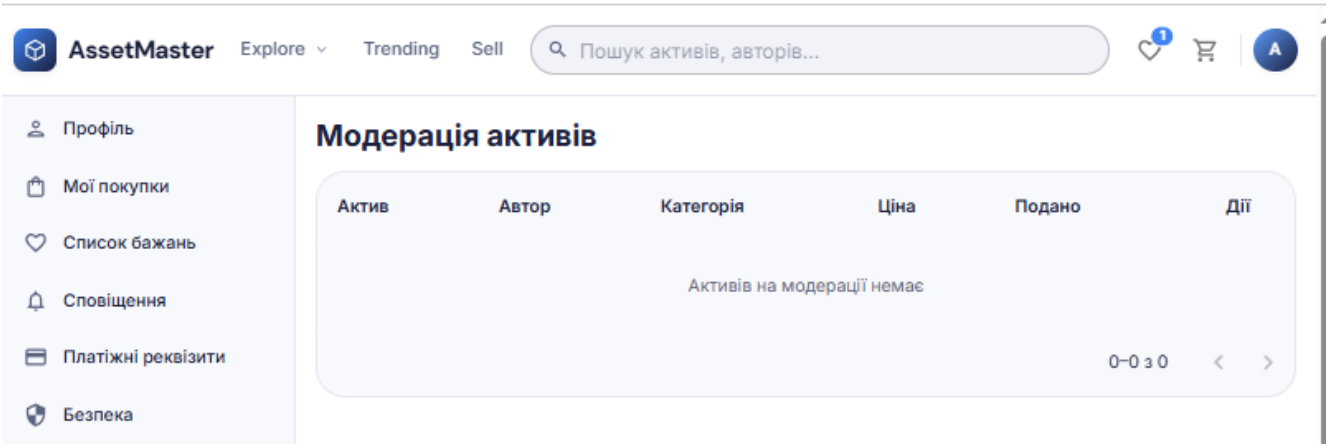


Рисунок 2.14 — Сторінка модерації активів в адміністративній панелі

Управління користувачами (/admin/users). Перевірено пошук та фільтрацію користувачів за роллю. Перевірено зміну ролі ROLE_USER → ROLE_AUTHOR: після зміни відповідні маршрути кабінету автора стають доступними для користувача. Перевірено блокування облікового запису: заблокований користувач отримує 401 при спробі входу. Зовнішній вигляд розділу управління користувачами наведено на рисунку 2.15.

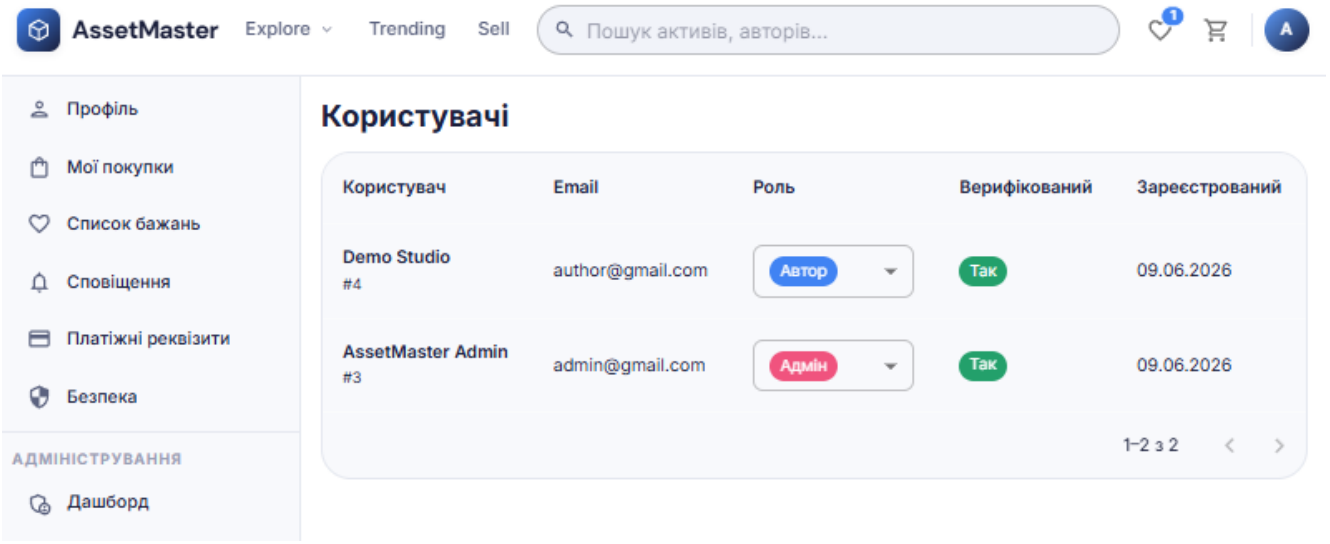


Рисунок 2.15 — Розділ управління користувачами в адміністративній панелі

Фінансовий модуль (/admin/finance). Перевірено відображення загального фінансового огляду та вкладки виплат авторам. Перевірено зміну статусу виплати та виконання переказу через Stripe Transfer. Зовнішній вигляд фінансового модуля наведено на рисунку 2.16.

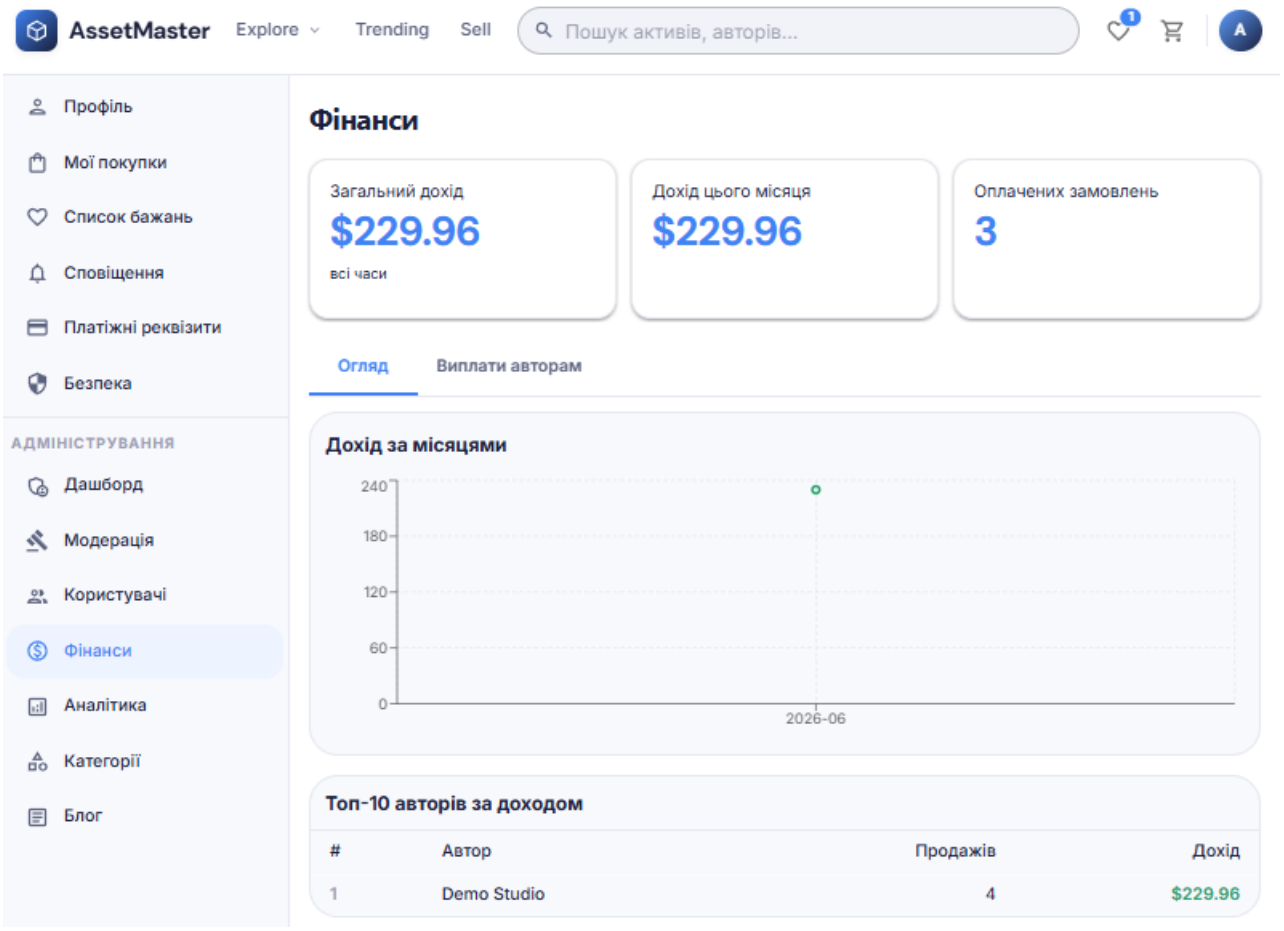


Рисунок 2.16 — Фінансовий модуль в адміністративній панелі

За результатами мануального функціонального тестування перевірено 31 тест-кейс, що охоплює всі основні зони вебплатформи: публічну (6 тест-кейсів), авторизацію (8), кабінет покупця (6), кабінет автора (5) та адміністративну панель (6). Усі тест-кейси пройдено успішно зі статусом Pass. Критичних дефектів (Blocker, Critical), що унеможливлюють використання основного функціоналу, не виявлено. Платформа AssetMaster визнана такою, що відповідає функціональним вимогам технічного завдання та готова до розгортання в експлуатаційному середовищі.

3 СПЕЦІАЛЬНИЙ РОЗДІЛ

3.1 Інструкція з розгортання вебплатформи

Розгортання вебплатформи AssetMaster передбачає послідовне налаштування трьох компонентів: серверної частини на основі Spring Boot, клієнтської частини на основі React та бази даних PostgreSQL. Усі компоненти контейнеризовані за допомогою Docker, що забезпечує відтворюваність середовища незалежно від операційної системи цільового сервера.

Для розгортання платформи необхідний сервер або хмарна інстанція з такими мінімальними характеристиками: процесор з двома і більше ядрами (архітектура x86_64), оперативна пам'ять обсягом не менше 4 ГБ, дисковий простір не менше 20 ГБ для системи та файлів застосунку і додатково не менше 50 ГБ для сховища файлів активів. Рекомендована операційна система — Ubuntu 22.04 LTS. На сервері попередньо мають бути встановлені: Docker Engine версії 24.0 або вище, Docker Compose версії 2.20 або вище, Git версії 2.34 або вище.

Для взаємодії платформи із зовнішніми сервісами необхідно мати облікові записи та отримати відповідні ключі доступу: ключі API для S3-сумісного хмарного сховища файлів (AWS S3, Cloudflare R2 або аналог), ключі платіжного шлюзу (Stripe або аналог), SMTP-дані для відправки транзакційних електронних листів.

3.1.1 Отримання вихідного коду

Перед початком розгортання необхідно отримати вихідний код платформи. Для цього на сервері виконується клонування репозиторію командою git clone, після чого відбувається перехід до кореневого каталогу проекту. Результатом є наявність двох підкаталогів — frontend та backend — та файлів конфігурації Docker у корені проекту.

3.1.2 Налаштування змінних середовища

Усі конфіденційні налаштування платформи зберігаються у файлі .env, який не включається до репозиторію з міркувань безпеки. У кореневому каталозі проекту наявний файл .env.example, що містить повний перелік необхідних змінних із коментарями щодо їх призначення. Для створення робочого файлу налаштувань необхідно скопіювати .env.example у .env та заповнити значення кожної змінної відповідно до отриманих реєстраційних даних зовнішніх сервісів.

Обов'язкові змінні середовища поділяються на такі групи. Змінні бази даних визначають ім'я, пароль та назву схеми PostgreSQL. Змінні безпеки містять секретний ключ для підпису JWT-токенів (рядок довжиною не менше 64 символів) та час дії access- і refresh-токенів. Змінні хмарного сховища включають endpoint, ідентифікатор ключа доступу, секретний ключ та назву бакета для збереження файлів активів. Змінні платіжного шлюзу містять публічний та секретний ключі API, а також URL для отримання webhook-сповіщень про статус платежів. Змінні електронної пошти визначають SMTP-хост, порт, ім'я користувача та пароль для відправника транзакційних листів.

3.1.3 Розгортання через Docker Compose

Після налаштування змінних середовища виконується збірка та запуск усіх сервісів однією командою через Docker Compose. Файл docker-compose.yml у корені проекту описує три сервіси: db (контейнер PostgreSQL), backend (Spring Boot API) та frontend (React-застосунок, що обслуговується через Nginx). Між сервісами визначені залежності: backend запускається лише після успішного проходження health check бази даних, frontend — після готовності backend.

Під час першого запуску Docker автоматично завантажує необхідні базові образи, збирає образи застосунків та ініціалізує базу даних. Flyway при старті backend автоматично застосовує всі SQL-міграції у порядку їх версіонування, створюючи повну схему бази даних. Час першого запуску залежить від швидкості

інтернет-з'єднання та потужності сервера і зазвичай становить від 3 до 10 хвилин.

Після успішного запуску всіх контейнерів сервіси доступні за такими адресами: клієнтська частина — за HTTP на порті 80 (або HTTPS на порті 443 після налаштування SSL); серверний API — на порті 8080; документація API (Swagger UI) — за адресою /swagger-ui.html відносно адреси серверного API. Перевірити стан усіх сервісів можна командою docker-compose ps, а стан здоров'я серверного API — через ендпоінт /actuator/health.

3.1.4 Налаштування SSL та доменного імені

Для забезпечення захищеного з'єднання HTTPS необхідно налаштувати SSL-сертифікат. Рекомендованим безкоштовним рішенням є Let's Encrypt у поєднанні з клієнтом Certbot. Налаштування виконується у конфігураційному файлі Nginx, що міститься у каталозі frontend/nginx. У цьому файлі визначається доменне ім'я платформи, шлях до сертифіката та ключа, а також правила проксіювання запитів до серверного API.

Після отримання сертифіката та оновлення конфігурації Nginx необхідно перезапустити контейнер frontend командою docker-compose restart frontend. Автоматичне оновлення сертифіката рекомендується налаштувати через планувальник задач cron із запуском certbot renew двічі на день.

3.1.5 Резервне копіювання даних

Регулярне резервне копіювання є обов'язковою умовою надійної експлуатації платформи. Резервні копії охоплюють два типи даних: вміст бази даних PostgreSQL та файли активів у хмарному сховищі. Для автоматичного резервного копіювання бази даних рекомендується налаштувати щоденний cron-скрипт, що виконує pg_dump усередині контейнера db та зберігає дамп у стиснутому форматі з датою у назві файлу. Зберігання резервних копій рекомендується організувати у окремому бакеті хмарного сховища з політикою

автоматичного видалення копій, старших за 30 днів.

3.2 Інструкція з наповнення вебплатформи

Наповнення платформи AssetMaster здійснюється через два незалежних інтерфейси: адміністративну панель — для управління структурою контенту та модерації, та особистий кабінет автора — для завантаження і публікації цифрових активів. Цей розділ описує обидва процеси у порядку, рекомендованому для початкового наповнення нової інсталяції платформи.

Перед завантаженням будь-яких активів необхідно створити структуру категорій, оскільки кожен актив обов'язково прив'язується до категорії. Для входу в адміністративну панель необхідно перейти за адресою /admin та автентифікуватись із використанням облікового запису, якому присвоєно роль ROLE_ADMIN. Обліковий запис адміністратора створюється вручну безпосередньо в базі даних під час першого розгортання або через SQL-міграцію початкових даних.

Після входу необхідно перейти до розділу «Категорії» та створити ієрархічну структуру категорій активів. Платформа підтримує дворівневу ієрархію: категорії верхнього рівня (наприклад, «3D-моделі», «UI Kit», «Шрифти», «Ілюстрації», «Відео», «Аудіо», «Game Assets») та підкатегорії (наприклад, для «3D-моделі»: «Архітектура», «Персонажі», «Транспорт», «Природа»). Для кожної категорії вказуються: назва, URL-slug (генерується автоматично або задається вручну), іконка з бібліотеки Tabler Icons та батьківська категорія для підкатегорій.

Після створення категорій рекомендується налаштувати перелік стандартних тегів для полегшення пошуку активів. Теги вводяться у розділі «Категорії / Теги» у вигляді рядків через кому. Вони є допоміжним засобом пошуку і доповнюють категоріальну класифікацію.

3.2.1 Реєстрація та верифікація авторів

Для завантаження активів користувач повинен мати роль ROLE_AUTHOR. Отримати цю роль можна двома способами: через форму реєстрації автора на сторінці /auth/register (де обирається тип облікового запису «Автор») або через підвищення ролі наявного покупця адміністратором у розділі «Користувачі» адмін-панелі.

В адмін-панелі у розділі «Користувачі» адміністратор може переглянути перелік усіх зареєстрованих користувачів із фільтрацією за роллю, статусом верифікації та датою реєстрації. Для зміни ролі користувача необхідно відкрити його профіль, натиснути кнопку «Змінити роль» та обрати нову роль зі списку. Зміна набуває чинності негайно, без необхідності перезапуску сервісів.

3.2.2 Завантаження активів авторами

Автор завантажує нові активи через особистий кабінет, розділ «Портфоліо» — кнопка «Додати актив». Форма завантаження складається з чотирьох кроків, реалізованих у вигляді покрокового майстра (stepper).

На першому кроці «Основна інформація» автор заповнює назву активу (до 120 символів), розгорнутий опис (до 5000 символів із підтримкою Markdown-форматування), обирає категорію зі спадного списку та вводить теги через кому (до 20 тегів). На другому кроці «Файли» завантажується основний файл активу (підтримувані формати та обмеження розміру визначаються налаштуваннями платформи) та файли попереднього перегляду (до 5 зображень у форматі JPG або PNG розміром не менше 1200×800 пікселів). Зображення попереднього перегляду є обов'язковими і безпосередньо впливають на конверсію активу.

На третьому кроці «Ліцензія та ціна» автор визначає цінову модель: безкоштовний актив, одноразова покупка зі стандартною ліцензією або подвійна ціна зі стандартною і розширеною комерційною ліцензіями. Стандартна ліцензія дозволяє використання у некомерційних та особистих проектах; розширена

комерційна — у комерційних продуктах, що перепродаються. На четвертому кроці «Перегляд» автор перевіряє всі введені дані та надсилає актив на модерацію натисканням кнопки «Надіслати на перевірку».

Після надсилання актив отримує статус PENDING і потрапляє до черги модерації адміністратора. Автор отримує email-сповіщення про зміну статусу активу: схвалення (статус PUBLISHED) або відхилення (статус REJECTED) із коментарем причини відмови.

3.2.3 Модерація активів адміністратором

Адміністратор переглядає активи, що очікують перевірки, у розділі «Модерація» адмін-панелі. Список відображає картки активів із мініатюрами, назвою, автором і датою подачі. Для перегляду деталей необхідно відкрити сторінку активу, де доступні усі завантажені матеріали, метадані та інформація про автора.

При перевірці активу адміністратор керується такими критеріями: відповідність контенту категорії та опису; якість файлів попереднього перегляду (чіткість, достатня роздільна здатність, відсутність водяних знаків сторонніх сервісів); технічна якість основного файлу; відсутність порушень авторських прав та ліцензійних обмежень; коректність цінової моделі та типу ліцензії. За результатами перевірки адміністратор натискає «Схвалити» або «Відхилити». У разі відхилення обов'язково вводиться коментар із поясненням причини, який надсилається автору в email-сповіщенні.

3.3 Інструкція з популяризації та підтримки вебплатформи

Успішний розвиток платформи AssetMaster залежить від двох взаємопов'язаних напрямків: залучення органічного трафіку через пошукову оптимізацію та системна підтримка технічної і контентної якості платформи. Цей розділ описує конкретні інструменти та практики для обох напрямків, що

доступні адміністратору та власнику платформи.

Платформа реалізує низку технічних SEO-механізмів на рівні серверного рендерингу. Кожна сторінка отримує унікальний title та meta description, що формуються динамічно на основі даних активу або категорії. Сторінка продукту включає структуровані дані у форматі JSON-LD (schema.org/Product) із зазначенням назви, опису, автора, ціни та рейтингу, що дозволяє пошуковим системам відображати розширені сніппети у результатах пошуку.

Для індексації всього контенту платформи автоматично генерується XML-sitemap, доступний за адресою /sitemap.xml. Sitemap включає URL усіх опублікованих активів, сторінок категорій і авторів із зазначенням дати останньої зміни та пріоритету індексації. Файл robots.txt, доступний за адресою /robots.txt, дозволяє індексацію публічних сторінок і забороняє індексацію адміністративної панелі та API-ендпоінтів.

Для контролю індексації та виявлення технічних SEO-проблем рекомендується зареєструвати платформу у Google Search Console та Bing Webmaster Tools. Після реєстрації необхідно завантажити sitemap.xml в обидва інструменти та налаштувати отримання сповіщень про помилки індексації. Регулярний моніторинг Search Console дозволяє відстежувати динаміку органічного трафіку, позиції ключових запитів та технічні помилки, що впливають на ранжування.

Семантичне ядро для платформи цифрових активів формується навколо трьох типів запитів: транзакційних (наприклад, «купити 3D-модель будинку», «завантажити UI Kit для Figma»), інформаційних (наприклад, «що таке комерційна ліцензія шрифту») та навігаційних (прямі запити назви платформи або імен авторів). Для покращення позицій за транзакційними запитами рекомендується наповнювати описи категорій і сторінки авторів текстовим контентом обсягом від 300 слів із природним входженням ключових фраз.

4 ЕКОНОМІЧНИЙ РОЗДІЛ

Метою економічної частини кваліфікаційної роботи є проведення економічних розрахунків, спрямованих на визначення економічної ефективності розробки вебплатформи для дистрибуції та продажу цифрових активів «AssetMaster», прийняття рішення про подальший розвиток і впровадження або ж недоцільність проведення відповідної розробки.

Об’єктом розробки є для дистрибуції та продажу цифрових активів «AssetMaster».

Розрахунок вартості розробки виконується в декілька етапів:

– описати технологічний процес розробки із зазначенням трудомісткості кожної операції;

– визначити суму витрат на оплату праці основного і допоміжного персоналу, включаючи відрахування на соціальні заходи;

– обчислити витрати на електроенергію;

– нарахувати суму амортизаційних відрахувань;

– визначити суму накладних витрат;

– скласти кошторис та визначити собівартість робіт;

– розрахувати ціну робіт;

– визначити економічну ефективність та термін окупності.

4.1 Визначення стадій технологічного процесу та загальної тривалості проведення НДР

В цьому підрозділі розглянемо основні етапи технологічного процесу для розробки вебплатформи «AssetMaster». Для визначення загальної тривалості проведення робіт доцільно дані витрат часу по окремих операціях технологічного процесу звести у таблицю 4.1.

Таблиця 4.1 - Середній час виконання робіт по обслуговуванню та стадії (операції) технологічного процесу

№ п/п	Назва операції (стадії)	Виконавець	Середній час виконання операції, год.
1	Планування та аналіз	Кер. проєкту (Pm)	8
		Інженер (l1)	8
2	Розробка технічного завдання	Кер. проєкту (Pm)	10
		Інженер (l1)	10
3	Дизайн інтерфейсу	Інженер (l1)	20
		Інженер (l2)	20
4	Розробка функціоналу	Інженер (l1)	40
5	Тестування та відладка	Тестувальник	16
6	Документування	Інженер (l1)	2
7	Розгортання та підтримка	Інженер (l2)	8
8	Управління проєктом	Кер. проєкту (Pm)	8
Разом			150

Сумарний час виконання операцій технологічного процесу становить 150 годин.

4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи

У даному підрозділі проводиться аналіз і розрахунок витрат, пов'язаних з оплатою праці та відрахуваннями на соціальні заходи, що необхідні для розробки вебплатформи «AssetMaster».

Розмір заробітної плати залежить від складності та умов виконуваної роботи, професійно-ділових якостей працівника, результатів його праці та

діяльності підприємства.

Основна заробітна плата розраховується за формулою –

$$З_{осн.} = Т_с \cdot К_г, \quad (4.1)$$

де $Т_с$ – тарифна ставка, грн. (приймаємо для керівника проєкту (Pm) – 400 грн./год, , інженера (l2) – 320 грн./год.), інженера (l1) – 215 грн./год., тестувальник – 160 грн./год.); $К_г$ – кількість відпрацьованих годин.

Отже основна заробітна плата для:

– керівник проєкту (Pm): $З_{осн1} = 400 \cdot 26 = 10400$ грн.;

– інженера (l2): $З_{осн2} = 320 \cdot 28 = 8960$ грн.;

– інженера (l1): $З_{осн3} = 215 \cdot 80 = 17200$ грн.;

– тестувальника: $З_{осн4} = 160 \cdot 16 = 2560$ грн.

Сумарна основна заробітна плата становить –

$$З_{осн} = 10400 + 8960 + 17200 + 2560 = 39120 \text{ грн.}$$

Додаткова заробітна плата становить 10–15 % від суми основної заробітної плати.

$$З_{дод.} = З_{осн.} \cdot К_{допл.}, \quad (4.2)$$

де $К_{допл.}$ – коефіцієнт додаткових виплат працівникам (приймаємо 10%).

Отже додаткова заробітна плата по категоріях працівників становить:

– керівника проєкту $З_{дод1} = 10400 \cdot 0,1 = 1040$ грн.;

– інженера (l2): $З_{осн2} = 8960 \cdot 0,1 = 896$ грн.;

– інженера (l1): $З_{осн3} = 17200 \cdot 0,1 = 1720$ грн.;

– тестувальника $З_{дод4} = 2560 \cdot 0,1 = 256$ грн.

Загальна додаткова заробітна плата становить:

$$З_{дод} = 1040 + 896 + 1720 + 256 = 3912 \text{ грн.}$$

Звідси загальні витрати на оплату праці ($В_{о.п.}$) визначаються за формулою:

$$В_{о.п.} = З_{осн.} + З_{дод.} \quad (4.3)$$

$$В_{о.п} = 39120 + 3912 = 43032 \text{ грн.}$$

Єдиний соціальний внесок (ЄСВ – 22%) визначається за формулою:

$$В_{єсв} = Воп \cdot 0{,}22 \quad (4.4)$$

$$В_{єсв.} = 43032 \cdot 0{,}22 = 9467{,}04 \text{ грн.}$$

Проведені розрахунки витрат на оплату праці наведено у таблицю 4.2.

Таблиця 4.2 - Зведені розрахунки витрат на оплату праці

№ п/п	Категорія працівників	Основна заробітна плата, грн.			Додат-кова заробітна плата, грн.	ЄСВ, грн.	Всього витрати на оплату праці,грн.
		Тариф-на ставка, грн.	К-сть годин, год.	Фактично нарах. зарплати, грн.			
1	Кер. проєкту (Pm)	400	26	10400	1040	-	-
2	Інженера (l2)	320	28	8960	896	-	-
3	Інженера (l1)	215	80	17200	1720	-	-
4	Тестувальник	160	16	2560	256	-	-
Разом				39120	3912	9467,04	52499,04

Отже, загальні витрати на оплату праці становлять 52499,04 грн.

4.3 Розрахунок витрат на електроенергію

Розрахуємо вартість електроенергії. Затрати на електроенергію 1-ці обладнання визначаються за формулою:

$$З_в = W \cdot T \cdot S, \quad (4.5)$$

де W – необхідна потужність, кВт; Т – кількість годин роботи обладнання; S – вартість кіловат-години електроенергії (приймаємо 15,94 грн).

В нашій системі є 1 ПК. Витрати на електроенергію для цього комп'ютера обчислимо окремо, взявши за основу, що час роботи обладнання обчислюється в залежності від виконуваних робіт (згідно табл. 4.1) і споживані потужності наступні – комп'ютер – 0,8 кВт/год.

$$З_в = 0{,}8 \cdot 150 \cdot 15{,}94 = 1912{,}80 \text{ грн.}$$

Витрати на електроенергію становлять 1912,80 грн.

4.4 Розрахунок суми амортизаційних відрахувань розробки вебплатформи «AssetMaster»

Характерною особливістю застосування основних фондів у процесі виробництва є їх відновлення. Для відновлення засобів праці у натуральному виразі необхідне їх відшкодування у вартісній формі, яке здійснюється шляхом амортизації.

Амортизація – це процес перенесення вартості основних фондів на вартість новоствореної продукції з метою їх повного відновлення. Комп'ютери та оргтехніка належать до четвертої групи основних фондів.

Амортизація на них нараховується лише в випадку, якщо мінімально допустимі строки їх корисного використання 2 роки. Для визначення амортизаційних відрахувань застосовуємо формулу:

А

$$= \frac{Б_В \cdot Н_А}{100\%}$$

$\cdot T,$

де А – амортизаційні відрахування за звітний період, грн.; $Б_В$ – балансова вартість групи основних фондів на початок звітного періоду, грн.; Т - кількість годин роботи обладнання, год.; $Н_А$ – норма амортизації, 0,04%.

Оскільки для написання програми та її тестування використовується один ПК вартістю 48500 грн., тоді сума амортизаційних відрахувань становить:

$$А = \frac{48500 \cdot 0{,}04}{150} \cdot 150 = 1940 \text{ грн.}$$

4.5 Обчислення накладних витрат

Накладні витрати пов’язані з обслуговуванням виробництва, утриманням апарату управління підприємства (фірми) та створення необхідних умов праці. В залежності від організаційно-правової форми діяльності господарюючого суб’єкта, накладні витрати можуть становити 20–60 % від суми основної та додаткової заробітної плати працівників.

$$Нв = Воп. \cdot 0{,}2 \ldots 0{,}6, \quad (4.7)$$

де $Н_В$ – накладні витрати (приймемо 30%).

$$Н_В = 43032 \cdot 0{,}3 = 12909{,}60 \text{ грн.}$$

4.6 Складання кошторису витрат та визначення собівартості вебплатформи «AssetMaster»

Для складення кошторису витрат та визначення собівартості, результати проведених вище розрахунків зведемо у таблиці 4.3.

Таблиця 4.3 - Кошторис витрат розробки вебплатформи «AssetMaster»

№	Зміст витрат	Сума, грн.	В % до загальної суми
1	Витрати на оплату праці	52499,04	75,8
2	Витрати на електроенергію	1912,8	2,76
3	Амортизаційні відрахування	1940	2,8
4	Накладні витрати	12909,60	18,64
5	Собівартість	69261,44	100

Собівартість ($С_В$) НДР розрахуємо за формулою:

$$С_В = В_{О.П.} + З_М + З_е + А + Н_в \quad (4.8)$$

Отже, собівартість дорівнює:

$С_В = 52499,04 + 1912,80 + 1940 + 12909,60 = 69261,44$ грн.

4.7 Розрахунок ціни вебплатформи «AssetMaster»

Розрахунок ціни науково-дослідної роботи включає в себе урахування різноманітних факторів, таких як рівень рентабельності, собівартість та податкова ставка.

Ціну робіт можна визначити за формулою:

$$Ц = Св \cdot (1+Р_{рен}) \cdot (1+ПДВ), \quad (4.9)$$

де Св – собівартість розробки; $Р_{рен.}$ – рівень рентабельності, (30 %); ПДВ – ставка податку на додану вартість, (20 %).

$$Ц = 69261,44 \cdot \ (1 + 0,3) \ \cdot (1 + 0,2) \ = 108047,85 \quad \text{грн.}$$

4.8 Визначення економічної ефективності і терміну окупності капітальних вкладень

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу.

Для визначення ефективності продукту розраховують чисту теперішню вартість (ЧТВ) і термін окупності ($Т_{ок}$).

$$ЧТВ = -\ С_B \ + \sum_{i=1}^{t} \frac{Г_П}{(1+i)^t}, \quad (4.10)$$

де $С_В$ – собівартість розробки; $Г_п$ – грошовий потік за t – ий рік; t – відповідний рік проєкту; і - величина дисконтної ставки (10%).

$$ЧТВ = -69261,44 \ + \frac{38786,41}{(1+0,1)} + \frac{38786,41}{(1+0,1)^2} + \frac{38786,41}{(1+0,1)^3} = \ 27194,61 \ \text{грн.}$$

Якщо ЧТВ ≥ 0, то проєкт може бути рекомендований до впровадження. Термін окупності визначається за формулою:

$$Т_{ОК} \ = Т_{ПВ} \ + \frac{Н_В}{Г_{ПР}} \quad (4.11)$$

де $Т_{ПВ}$ – період до повного відшкодування витрат, років; $Н_В$ – невідшкодовані

витрати на початок року, грн.; $Г_{ПР}$ – грошовий потік на початок року, грн.

$$Т_{ок} = 2 + 1946,19 / 38786,41 = 2,1$$

Всі дані внесемо в зведену таблицю 4.4.

Таблиця 4.4 - Техніко-економічні показники вебплатформи «AssetMaster»

№ п/п	Показник	Значення
1	Собівартість, грн.	69261,44
2	Плановий прибуток, грн.	38786,41
3	Ціна, грн.	108047,85
4	Чиста теперішня вартість, грн.	27194,61
5	Термін окупності, рік	2,1

Прибутковість проєкту та термін окупності свідчать про його фінансову ефективність та здатність повернути капітальні вкладення протягом 2,1 року. Отже, на основі отриманих показників можна зробити висновок, що розробка вебплатформи «AssetMaster» є доцільною з економічної точки зору.

5 ОХОРОНА ПРАЦІ, ТЕХНІКА БЕЗПЕКИ ТА ЕКОЛОГІЧНІ ВИМОГИ

Розробка, тестування та адміністрування інформаційної системи вебплатформи «AssetMaster» здійснюються у приміщеннях з персональними комп'ютерами, серверним обладнанням та іншою оргтехнікою. Специфіка такої праці породжує характерні виробничі ризики — від психофізіологічного перевантаження до впливу шуму від вентиляційних систем та оргтехніки. Цей розділ розглядає нормативне регулювання питань професійних захворювань та шкідливого шуму на таких робочих місцях.

5.1 Професійні захворювання та отруєння

Професійне захворювання — це захворювання, що виникло внаслідок професійної діяльності та зумовлюється виключно або переважно впливом шкідливих речовин і певних видів робіт та інших факторів, пов'язаних з роботою. Відповідно до Закону України «Про охорону праці» від 14 жовтня 1992 року № 2694-XII, до виробничих шкідливих факторів, що спричиняють профзахворювання, належать фізичні, хімічні, біологічні та психофізіологічні чинники виробничого середовища. [14]

Порядок виявлення підозри на професійне захворювання, розслідування обставин та причин його виникнення, встановлення та підтвердження діагнозу регламентує постанова Кабінету Міністрів України від 17 квітня 2019 року № 337 «Про затвердження Порядку розслідування та обліку нещасних випадків, професійних захворювань та аварій». Відповідно до цього нормативного акту, роботодавець зобов'язаний подати комісії з розслідування відомості про умови праці на робочому місці, результати лабораторних досліджень, матеріали про проведення інструктажів та медичні огляди працівника. [15]

Гострим професійним захворюванням (отруєнням) вважається захворювання, що виникло після однократного впливу протягом не більш як однієї робочої зміни шкідливих факторів фізичного, біологічного та хімічного

характеру, у тому числі гострого отруєння або гострого захворювання. Хронічним є захворювання, що виникло внаслідок тривалого провадження професійної діяльності під впливом шкідливих факторів виробничого середовища та трудового процесу. [15]

Відповідно до характеру впливаючих шкідливих виробничих факторів, усі професійні захворювання поділяються на чотири групи, наведені у таблиці 5.1.

Таблиця 5.1 – Класифікація професійних захворювань за видом впливаючого чинника

Група чинників	Характер впливу	Типові захворювання
Хімічні	Вплив токсичних речовин, важких металів, розчинників, кислот, лугів	Гострі та хронічні отруєння, токсична нефропатія, гепатит, ураження нервової системи
Фізичні	Шум, вібрація, іонізуюче та неіонізуюче випромінювання, несприятливий мікроклімат, підвищений або знижений тиск	Шумова хвороба, нейросенсорна приглухуватість, вібраційна хвороба, катаракта, дерматити
Біологічні	Патогенні мікроорганізми, грибки, паразити, алергени біологічного походження	Інфекційні та паразитарні захворювання (туберкульоз, гепатит В і С), мікози, біноза
Психофізіологічні	Фізичне перевантаження, нервово-психічна напруга, одноманітність рухів, вимушена робоча поза	Захворювання опорно-рухового апарату, неврози, синдром хронічної втоми, порушення зору

Зростання рівнів професійної захворюваності в Україні пояснюється недостатньою увагою до додержання вимог чинного законодавства та

технологічної дисципліни, різким скороченням коштів на охорону праці, використанням шкідливих речовин та технологічного обладнання без належної сертифікації та гігієнічної експертизи, а також низьким рівнем виробничої та особистої гігієнічної культури. [15]

У виникненні конкретного профзахворювання вирішальну роль відіграють: доза або рівень шкідливої речовини (вібрації, шуму); тривалість дії чинника на організм; умови праці, що можуть зменшити або посилити вплив шкідливостей; наявність або відсутність засобів індивідуального захисту; індивідуальна чутливість організму людини до конкретного виробничого чинника. [15]

Розробники та оператори інформаційних систем, зокрема вебплатформи «AssetMaster», виконують роботу, що за класифікацією ДСанПіН 3.3.2.007-98 відноситься до першої категорії — творча, конструкторська, проєктна, програмістська діяльність. Попри відсутність важкої фізичної праці та впливу токсичних речовин, тривала робота за комп'ютером породжує специфічний комплекс ризиків. Характерні для цього виду праці захворювання наведено у таблиці 5.2. [16]

Таблиця 5.2 – Типові профзахворювання операторів персональних комп'ютерів

Система або орган	Захворювання	Основна причина
1	2	3
Органи зору	Комп'ютерний зоровий синдром (CVS), астенопія, міопія прогресуюча	Тривала робота з монітором, незадовільне освітлення, мерехтіння зображення
Опорно-руховий апарат	Тунельний синдром зап'ястя (CTS), тендиніт, остеохондроз шийного та поперекового відділів	Тривала одноманітна робота з клавіатурою та маніпулятором, вимушена статична поза

Продовження таблиці 5.2

1	2	3
Нервова система	Хронічна втома, неврастенія, синдром емоційного вигорання	Нервово-психічне навантаження, інтенсивна розумова праця, часті дедлайни
Серцево-судинна система	Вегетосудинна дистонія, гіпертонічна хвороба	Нервово-психічна напруга, гіподинамія, тривале перебування в закритому приміщенні
Шкіра	Алергічний дерматит, сухість шкіри	Електростатичне поле монітора, знижена вологість повітря від роботи кондиціонера

Комп'ютерний зоровий синдром (Computer Vision Syndrome, CVS) є найпоширенішим розладом здоров'я серед розробників програмного забезпечення. Він проявляється відчуттям сухості та печії в очах, нечіткістю зображення, підвищеною чутливістю до світла, головним болем. Причинами є надмірне напруження акомодаційного апарату ока, знижена частота мигання при роботі з монітором (до 5 разів на хвилину проти норми 15–20) та незадовільні параметри освітлення робочого місця. [16]

Тунельний синдром зап'ястя (Carpal Tunnel Syndrome, CTS) є хронічним профзахворюванням, що виникає внаслідок тривалої одноманітної роботи з клавіатурою та маніпулятором у незручному положенні руки. Характеризується болями та оніміням у зоні зап'ястя, кисті та пальців, ослабленням сили захоплення. При несвоєчасному лікуванні може призвести до стійкого функціонального порушення. [16]

Профілактика профзахворювань полягає в систематичному поліпшенні умов праці та ліквідації або мінімізації шкідливо діючих факторів. Для операторів і розробників системи «AssetMaster» рекомендуються такі заходи.

Організаційні заходи включають: дотримання регламентованих перерв — через кожні 2 години безперервної роботи з монітором слід робити 15-хвилинну перерву; виконання вправ для очей (зведення погляду на далекий предмет, кругові рухи очима) та вправ для кистей рук і шиї; забезпечення достатнього рівня природного освітлення та правильне розташування монітора відносно вікон. [16]

Технічні заходи охоплюють: використання моніторів з антибліковим покриттям та частотою оновлення не нижче 75 Гц; налаштування яскравості та контрасту монітора відповідно до освітленості приміщення; застосування ергономічних клавіатур та маніпуляторів для зниження навантаження на кисті; встановлення вертикальних підставок для документів поруч з монітором для зменшення рухів очей. [16]

5.2 Шум. Основні джерела шкідливого шуму, кваліфікація шумів

Шум — це будь-який небажаний звук або сукупність звуків, що заважають нормальній трудовій діяльності людини, завдають шкоди її здоров'ю або знижують якість сприйняття інформації. З фізичної точки зору шум являє собою механічні коливання пружного середовища (повітря, рідини або твердого тіла) у діапазоні частот, що сприймаються слуховим аналізатором людини — від 16 до 20 000 Гц. [17]

Звуковий тиск вимірюється у паскалях (Па), однак на практиці через великий діапазон значень використовується логарифмічна шкала — децибели (дБ). Рівень звукового тиску визначається як логарифм відношення фактичного звукового тиску до порогового значення 2×10^{-5} Па (порогу чутності людського вуха). Для санітарно-гігієнічного нормування застосовується рівень звуку в дБА — рівень, виміряний за частотною характеристикою «А» шумоміра, що відповідає суб'єктивному сприйняттю звуку людиною. [17]

Тривалий вплив шуму на організм людини спричиняє: специфічні ураження органу слуху (шумова хвороба, нейросенсорна приглухуватість); неспецифічні реакції з боку серцево-судинної системи (тахікардія, підвищення артеріального

тиску), нервової системи (підвищена втомлюваність, дратівливість, порушення сну), шлунково-кишкового тракту та ендокринних залоз. Виробнича нейросенсорна приглухуватість є одним з найпоширеніших хронічних профзахворювань у світі.

Джерела виробничого шуму поділяються на три основні групи залежно від механізму їх утворення.

Механічний шум виникає внаслідок вібрації твердих поверхонь при їх ударному, фрикційному або зубчастому взаємодії. Це найпоширеніший тип виробничого шуму: удари деталей у машинах і механізмах, тертя в підшипниках та зубчастих передачах, вібрація корпусів обладнання.

Аеродинамічний шум утворюється при русі газів або рідин: витікання стисненого повітря або пари через відкриті отвори та патрубки (шум реактивного типу), турбулентність потоків у вентиляційних каналах та повітропроводах, обтікання потоком повітря перешкод (лопаток вентилятора, крилець насоса).

Електромагнітний шум генерується електричними машинами і апаратами: трансформаторами, дроселями, електродвигунами, електромагнітами через магнітострикцію та взаємодію магнітних полів.

Для офісних приміщень, де розробляється і експлуатується вебплатформа «AssetMaster», характерними джерелами шуму є системи примусової вентиляції та кондиціонування (аеродинамічний шум від обертання крильчатки та руху повітря у каналах); вентилятори охолодження системних блоків і серверів; жорсткі диски (шум обертання та позиціонування головок); лазерні принтери та багатофункціональні пристрої (механічний шум при друку); мережеве обладнання з активним охолодженням.

Державні санітарні норми виробничого шуму, ультразвуку та інфразвуку (ДСН 3.3.6.037-99), затверджені постановою Головного державного санітарного лікаря України від 1 грудня 1999 року № 37, є основним нормативним документом, що встановлює класифікацію виробничих акустичних коливань і гігієнічні нормативи шуму на робочих місцях. Відповідно до ДСН 3.3.6.037-99, виробничі шуми класифікуються за двома ознаками, що наведені у таблиці 5.3.

Таблиця 5.3 – Класифікація виробничих шумів за ДСН 3.3.6.037-99

Ознака класифікації	Вид шуму	Характеристика
За характером спектра	Широкосмуговий	Безперервний спектр шириною більше однієї октави
	Вузькосмуговий (тональний)	В спектрі наявні виражені дискретні тони; рівень шуму в одній октавній смузі перевищує сусідні не менш ніж на 10 дБ
За часовими характеристиками	Постійний	Рівень шуму за повний робочий день змінюється не більш ніж на 5 дБА
	Непостійний	Рівень шуму за повний робочий день змінюється більш ніж на 5 дБА
Підвиди непостійного	Мінливий	Рівень безперервно змінюється у часі
	Переривчастий	Рівень шуму змінюється ступінчасто на 5 дБА і більше; тривалість інтервалів зі сталим рівнем ≥ 1 с
	Імпульсний	Один або кілька звукових сигналів тривалістю менше 1 с кожний; різниця рівнів між характеристиками «імпульс» та «повільно» ≥ 7 дБ

Відповідно до пункту 2.1.1 ДСН 3.3.6.037-99, тональний характер шуму встановлюється вимірюванням у третиноктавних смугах частот по перевищенню рівня шуму в одній смузі над сусідніми не менш ніж на 10 дБ. Для тонального та імпульсного шуму допустимі рівні знижуються на 5 дБ порівняно з нормативними значеннями таблиці. [17]

Окрім класифікації звичайного слухового шуму, ДСН 3.3.6.037-99 також

класифікує ультразвук (з частотами вище 20 000 Гц) за способом передачі на повітряний та контактний, та інфразвук (з частотами нижче 16 Гц) за часовими характеристиками на постійний та непостійний. Обидва різновиди акустичних коливань шкідливо впливають на організм людини навіть за відсутності суб'єктивного відчуття «звуку». [17]

ДСН 3.3.6.037-99 встановлює допустимі рівні звукового тиску в октавних смугах частот і допустимі еквівалентні рівні звуку на робочих місцях залежно від виду трудової діяльності. Параметрами постійного шуму, що нормуються, є рівні звукових тисків в октавних смугах з середньогеометричними частотами 31,5; 63; 125; 250; 500; 1000; 2000; 4000; 8000 Гц у децибелах. Для непостійного шуму нормується еквівалентний рівень звуку — рівень постійного шуму, дія якого відповідає дії фактичного шуму зі змінними рівнями за той самий час. [17]

Для розробників програмного забезпечення та програмістів ЕОМ — категорія 1 ДСН 3.3.6.037-99 «творча діяльність, програмування» — встановлено найбільш жорсткі нормативи допустимого шуму. Витяг з нормативної таблиці наведено у таблиці 5.4.

Таблиця 5.4 – Допустимі рівні шуму для окремих видів праці

Вид трудової діяльності	31,5 Гц	63 Гц	125 Гц	250 Гц	500 Гц	1000 Гц	2000 Гц	Рівень, дБА
1	2	3	4	5	6	7	8	9
Творча діяльність, програмування, конструювання, проектування; робочі місця програмістів ЕОМ, лабораторії для теоретичних робіт	86	71	61	54	49	45	42	50

Продовження таблиці 5.4

1	2	3	4	5	6	7	8	9
Висококваліфікована робота, адміністративно-керівна діяльність, лабораторії	93	79	70	63	58	55	52	60
Виконання всіх видів робіт на постійних робочих місцях у виробничих приміщеннях	107	95	87	82	78	75	73	80

Таким чином, для програмістів та розробників вебплатформи «AssetMarket» допустимий еквівалентний рівень шуму на робочому місці становить 50 дБА. Для порівняння: рівень шуму тихого офісу з вентиляцією становить приблизно 45–50 дБА, шум від сучасного системного блоку — близько 30–40 дБА, а шум від промислового кондиціонера у приміщенні — 50–55 дБА, що вже може перевищувати норму. [17]

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи розроблено повнофункціональну вебплатформу для дистрибуції та продажу цифрових активів AssetMaster, що реалізує двосторонню маркетплейс-модель і забезпечує взаємодію між авторами цифрових матеріалів та їх покупцями.

За результатами виконання роботи можна зробити такі висновки.

Проведено аналітичний огляд трьох провідних платформ аналогічного призначення — Creative Market, Envato Elements і Gumroad. Виявлено їх ключові недоліки: орієнтацію виключно на англомовний ринок, несприятливі умови комісії для авторів-початківців і відсутність гнучкого поєднання разових покупок із підписковою моделлю. Визначено конкурентні переваги розробленої платформи AssetMaster: підтримка української мови, нижча початкова комісія (15–25 %), прозора аналітика продажів для авторів та двоступенева ліцензійна модель.

Розроблено технічне завдання на платформу, що включає функціональні вимоги до всіх модулів системи, вимоги до надійності, часові характеристики (LCP ≤ 2,5 с, відповідь API ≤ 500 мс при 100 одночасних запитах) та план стадій розробки. Сформульована структура вебплатформи охоплює чотири функціональні зони: публічну (16 сторінок), особистий кабінет покупця і автора, адміністративну панель та технічні системні сторінки.

Спроєктовано реляційну схему бази даних із дев'яти таблиць, нормалізованих до третьої нормальної форми. Визначено типи даних, обмеження цілісності, зовнішні ключі та індекси для прискорення запитів пошуку і фільтрації. Версіонування схеми організовано через інструмент Flyway, що гарантує відтворюваність стану бази даних у будь-якому середовищі.

Реалізовано серверну частину платформи засобами Java 21 і Spring Boot 3. Архітектура побудована за принципами шаруватої архітектури. Автентифікація реалізована на основі JWT із двоетапною моделлю токенів.

Реалізовано клієнтську частину платформи засобами TypeScript і React 18

відповідно до архітектурної методології Feature-Sliced Design. Управління серверним станом здійснюється через TanStack Query v5 з ієрархічними ключами кешування. Реалізовано нескінченне прокручування каталогу через Intersection Observer API, покрокову форму завантаження активів з прямим завантаженням файлів та адаптивний інтерфейс для трьох breakpoints.

Розроблено адміністративну панель, що забезпечує модерацію активів перед публікацією, управління обліковими записами та ролями користувачів, фінансовий моніторинг та агреговану аналітику платформи засобами нативних JPQL-запитів.

Проведено тестування платформи на трьох рівнях: модульному (JUnit 5 + Mockito), інтеграційному (TestContainers + MockMvc) та функціональному (12 ключових тест-кейсів). Усі тест-кейси пройдено успішно.

Розроблено детальні інструкції з розгортання платформи через Docker Compose, наповнення контентом через адміністративну панель та популяризації й технічної підтримки, що включають рекомендації щодо SEO-оптимізації, контент-маркетингу.

Практичним результатом роботи є готова до розгортання вебплатформа AssetMaster, що може використовуватись як самостійний комерційний продукт у сфері дистрибуції цифрових творчих активів або як основа для подальшого розвитку з розширенням функціоналу — підпискової моделі, реферальної системи та мобільного застосунку.

ПЕРЕЛІК ПОСИЛАНЬ

1) Creative Market : офіційний сайт платформи. URL: https://creativemarket.com (дата звернення: 01.03.2026).

2) Envato Elements : офіційний сайт платформи. URL: https://elements.envato.com (дата звернення: 01.03.2026).

3) Gumroad : офіційний сайт платформи. URL: https://gumroad.com (дата звернення: 02.03.2026).

4) Марціяш Г.Я., Слободян Р.О. Методичні вказівки до виконання дипломного проєкту для здобувачів фахової передвищої освіти спеціальності 122 «Комп'ютерні науки». Тернопіль : ВСП «ТФК ТНТУ ім. І. Пулюя», 2024, 48с..

5) Dorfmeister D. Practical React Query. URL: https://tkdodo.eu/blog/practical-react-query (дата звернення: 10.03.2026).

6) Material UI Documentation / MUI Team. URL: https://mui.com/material-ui/getting-started/ (дата звернення: 15.03.2026).

7) Axios HTTP Client Documentation. URL: https://axios-http.com/docs/intro (дата звернення: 22.03.2026).

8) Flyway by Redgate — Database Migrations Made Easy. URL: https://documentation.red-gate.com/flyway (дата звернення: 05.04.2026).

9) MinIO Documentation: S3-Compatible Object Storage. URL: https://min.io/docs/minio/linux/index.html (дата звернення: 08.04.2026).

10) Mockito Framework Documentation. URL: https://javadoc.io/doc/org.mockito/mockito-core/latest/org/mockito/Mockito.html (дата звернення: 14.04.2026).

11) SpringDoc OpenAPI Library Documentation. URL: https://springdoc.org (дата звернення: 16.04.2026).

12) MapStruct — Java Bean Mappings. Reference Guide. URL: https://mapstruct.org/documentation/stable/reference/html/ (дата звернення: 16.04.2026).

13) Recharts Documentation. URL: https://recharts.org/en-US/guide (дата

звернення: 18.04.2026).

14) Закон України «Про охорону праці» від 14 жовтня 1992 р. № 2694-XII. Редакція від 12.09.2026. URL: https://zakon.rada.gov.ua/go/2694-12 (дата звернення: 01.06.2026).

15) Порядок розслідування та обліку нещасних випадків, професійних захворювань та аварій: постанова Кабінету Міністрів України від 17 квітня 2019 р. № 337. URL: https://zakon.rada.gov.ua/laws/show/337-2019-п (дата звернення: 01.06.2026).

16) Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин: ДСанПіН 3.3.2.007-98, затверджено постановою Головного державного санітарного лікаря України від 10 грудня 1998 р. № 7. URL: https://zakon.rada.gov.ua/laws/show/v0007282-98 (дата звернення: 01.06.2026).

17) Санітарні норми виробничого шуму, ультразвуку та інфразвуку: ДСН 3.3.6.037-99, затверджено постановою Головного державного санітарного лікаря України від 1 грудня 1999 р. № 37. URL: https://dnaop.com/html/40957/doc-ДСН_3.3.6.037-99 (дата звернення: 01.06.2026).

ДОДАТКИ

Додаток А. Лістинг файлу «client.ts»

```
import axios from 'axios'

const apiClient = axios.create({
  baseURL: '/api/v1',
  headers: {
    'Content-Type': 'application/json',
  },
  withCredentials: true,
})

apiClient.interceptors.request.use((config) => {
  const token = localStorage.getItem('accessToken')
  if (token) {
    config.headers.Authorization = `Bearer ${token}`
  }
  return config
})

apiClient.interceptors.response.use(
  (response) => response,
  async (error) => {
    const original = error.config

    if (error.response?.status === 401 && !original._retry) {
      original._retry = true
      try {
        const { data } = await axios.post('/api/v1/auth/refresh', null, {
          withCredentials: true,
        })
        localStorage.setItem('accessToken', data.accessToken)
```

```
        original.headers.Authorization = `Bearer ${data.accessToken}`
        return apiClient(original)
      } catch {
        localStorage.removeItem('accessToken')
        window.location.href = '/auth/login'
      }
    }

    return Promise.reject(error)
  },
)

export default apiClient
```

Додаток Б. Лістинг файлу «router.tsx»

```
import { createBrowserRouter } from 'react-router-dom'
import ProtectedRoute from '../shared/components/ProtectedRoute'
import DashboardLayout from '../widgets/DashboardLayout/DashboardLayout'
import PublicLayout from '../widgets/PublicLayout/PublicLayout'

const router = createBrowserRouter([
  // ── Public (Navbar + Footer via PublicLayout) ──────────────
  {
    element: <PublicLayout />,
    children: [
      { path: '/', lazy: () => import('../pages/HomePage/HomePage').then(m
=> ({ Component: m.default })) },
      {         path:         '/catalog',         lazy:         ()         =>
import('../pages/CatalogPage/CatalogPage').then(m    =>    ({    Component:
m.default })) },
      {      path:      '/catalog/:categorySlug',      lazy:      ()      =>
import('../pages/CatalogPage/CatalogPage').then(m    =>    ({    Component:
m.default })) },
      {         path:         '/search',         lazy:         ()         =>
import('../pages/CatalogPage/CatalogPage').then(m    =>    ({    Component:
m.default })) },
      {        path:        '/assets/:id',        lazy:        ()        =>
import('../pages/AssetPage/AssetPage').then(m  =>  ({  Component:  m.default
})) },
      {         path:         '/cart',         lazy:         ()         =>
import('../pages/CartPage/CartPage').then(m => ({ Component: m.default }))
},
      { path: '/about', lazy: () => import('../pages/AboutPage').then(m =>
({ Component: m.default })) },
      { path: '/faq', lazy: () => import('../pages/FaqPage').then(m => ({
Component: m.default })) },
      {        path:        '/licenses',        lazy:        ()        =>
```

```
import('../pages/LicensesPage').then(m => ({ Component: m.default })) },
      { path: '/contact', lazy: () => import('../pages/ContactPage').then(m
=> ({ Component: m.default })) },
      { path: '/blog', lazy: () => import('../pages/BlogPage').then(m => ({
Component: m.default })) },
      {        path:        '/blog/:slug',        lazy:        ()        =>
import('../pages/BlogPostPage').then(m => ({ Component: m.default })) },
    ],
  },

  // ── Auth ──────────────────────────────────────────────────
  {
    path: '/auth/login',
    lazy: () => import('../pages/auth/LoginPage').then(m => ({ Component:
m.default })),
  },
  {
    path: '/auth/register',
    lazy:   ()   =>   import('../pages/auth/RegisterPage').then(m   =>   ({
Component: m.default })),
  },
  {
    path: '/auth/forgot-password',
    lazy:  ()  =>  import('../pages/auth/ForgotPasswordPage').then(m  =>  ({
Component: m.default })),
  },
  {
    path: '/auth/reset-password',
    lazy:  ()  =>  import('../pages/auth/ResetPasswordPage').then(m  =>  ({
Component: m.default })),
  },
  {
    path: '/auth/verify-email',
    lazy: () => import('../pages/auth/EmailVerificationPage').then(m => ({
```

```
Component: m.default })),
  },

  // ── System pages (no layout) ──────────────────────────────
  {
    path: '/500',
    lazy: () => import('../pages/ErrorPage').then(m => ({ Component:
m.default })),
  },
  {
    path: '/maintenance',
    lazy: () => import('../pages/MaintenancePage').then(m => ({ Component:
m.default })),
  },

  // ── Checkout (ProtectedRoute, own Navbar+Footer) ──────────
  {
    element: <ProtectedRoute />,
    children: [
      {
        path: '/checkout',
        lazy: () => import('../pages/CheckoutPage/CheckoutPage').then(m =>
({ Component: m.default })),
      },
      {
        path: '/checkout/success',
        lazy:                                  ()                                  =>
import('../pages/CheckoutPage/CheckoutSuccessPage').then(m => ({ Component:
m.default })),
      },
    ],
  },

  // ── Dashboard (all authenticated users) ───────────────────
```

```
  {
    element: <ProtectedRoute />,
    children: [{
      element: <DashboardLayout />,
      children: [
        { path: '/dashboard/profile', lazy: () =>
import('../pages/DashboardPage/ProfilePage').then(m => ({ Component:
m.default })) },
        { path: '/dashboard/purchases', lazy: () =>
import('../pages/DashboardPage/PurchasesPage').then(m => ({ Component:
m.default })) },
        { path: '/dashboard/wishlist', lazy: () =>
import('../pages/DashboardPage/WishlistPage').then(m => ({ Component:
m.default })) },
        { path: '/dashboard/security', lazy: () =>
import('../pages/DashboardPage/SecurityPage').then(m => ({ Component:
m.default })) },
        { path: '/dashboard/notifications', lazy: () =>
import('../pages/DashboardPage/NotificationsPage').then(m => ({ Component:
m.default })) },
        { path: '/dashboard/payments', lazy: () =>
import('../pages/DashboardPage/PaymentsPage').then(m => ({ Component:
m.default })) },
      ],
    }],
  },

  // ── Author dashboard ──────────────────────────────────────
  {
    element: <ProtectedRoute requiredRole="ROLE_AUTHOR" />,
    children: [{
      element: <DashboardLayout />,
      children: [
        { path: '/dashboard/assets', lazy: () =>
```

```
import('../pages/DashboardPage/AssetsPage').then(m => ({ Component:
m.default })) },
        { path: '/dashboard/assets/new', lazy: () =>
import('../pages/DashboardPage/AssetUploadPage').then(m => ({ Component:
m.default })) },
        { path: '/dashboard/assets/:id/edit', lazy: () =>
import('../pages/DashboardPage/AssetEditPage').then(m => ({ Component:
m.default })) },
        { path: '/dashboard/analytics', lazy: () =>
import('../pages/DashboardPage/AnalyticsPage').then(m => ({ Component:
m.default })) },
      ],
    }],
  },

  // ── Admin ──────────────────────────────────────────────
  {
    element: <ProtectedRoute requiredRole="ROLE_ADMIN" />,
    children: [{
      element: <DashboardLayout />,
      children: [
        { path: '/admin', lazy: () =>
import('../pages/AdminPage/AdminDashboardPage').then(m => ({ Component:
m.default })) },
        { path: '/admin/moderation', lazy: () =>
import('../pages/AdminPage/ModerationPage').then(m => ({ Component:
m.default })) },
        { path: '/admin/users', lazy: () =>
import('../pages/AdminPage/UsersPage').then(m => ({ Component: m.default
})) },
        { path: '/admin/finance', lazy: () =>
import('../pages/AdminPage/AdminFinancePage').then(m => ({ Component:
m.default })) },
        { path: '/admin/analytics', lazy: () =>
```

```
import('../pages/AdminPage/AdminAnalyticsPage').then(m => ({ Component:
m.default })) },
        { path: '/admin/categories', lazy: () =>
import('../pages/AdminPage/CategoriesPage').then(m => ({ Component:
m.default })) },
        { path: '/admin/blog', lazy: () =>
import('../pages/AdminPage/AdminBlogPage').then(m => ({ Component:
m.default })) },
      ],
    }],
  },

  // ── 404 ──────────────────────────────────────────────────────
  {
    path: '*',
    lazy: () => import('../pages/NotFoundPage').then(m => ({ Component:
m.default })),
  },
])

export default router
```

Додаток В. Лістинг файлу «AssetGrid.tsx»

```
import { Link } from 'react-router-dom'
import Box from '@mui/material/Box'
import Typography from '@mui/material/Typography'
import Button from '@mui/material/Button'
import AssetCard from '../../entities/asset/ui/AssetCard'
import type { AssetSummaryDto } from '../../entities/asset/types'

interface Props {
  label: string
  assets?: AssetSummaryDto[]
  isLoading?: boolean
  viewAllHref?: string
  skeletonCount?: number
}

export default function AssetGrid({ label, assets, isLoading, viewAllHref,
skeletonCount = 8 }: Props) {
  return (
    <Box>
      {label && (
        <Box sx={{ display: 'flex', alignItems: 'center', justifyContent:
'space-between', mb: 3 }}>
          <Typography variant="h2">{label}</Typography>
          {viewAllHref && (
            <Button component={Link} to={viewAllHref} size="small">
              Дивитись всі →
            </Button>
          )}
        </Box>
      )}

      <Box
```

```
        sx={{
          display: 'grid',
          gridTemplateColumns: {
            xs: 'repeat(2, 1fr)',
            sm: 'repeat(3, 1fr)',
            md: 'repeat(4, 1fr)',
          },
          gap: 3,
        }}
      >
        {isLoading
          ? Array.from({ length: skeletonCount }).map((_, i) => <AssetCard
key={i} isLoading />)
          : (assets ?? []).map((asset) => <AssetCard key={asset.id}
asset={asset} />)}
      </Box>
    </Box>
  )
}
```

Додаток Г. Лістинг файлу «LoginPage.tsx»

```
import { useState } from 'react'
import { useForm } from 'react-hook-form'
import { zodResolver } from '@hookform/resolvers/zod'
import { z } from 'zod'
import { Link, useLocation, useNavigate } from 'react-router-dom'
import Box from '@mui/material/Box'
import Button from '@mui/material/Button'
import TextField from '@mui/material/TextField'
import Typography from '@mui/material/Typography'
import Alert from '@mui/material/Alert'
import Divider from '@mui/material/Divider'
import { useAuth } from '../../features/auth/AuthContext'

const loginSchema = z.object({
  email: z.string().email('Невірний формат email'),
  password: z.string().min(8, 'Мінімум 8 символів'),
})

const totpSchema = z.object({
  code: z.string().length(6, 'Код складається з 6 цифр').regex(/^\d+$/,
'Тільки цифри'),
})

type LoginFormData = z.infer<typeof loginSchema>
type TotpFormData = z.infer<typeof totpSchema>

export default function LoginPage() {
  const { login, verify2fa } = useAuth()
  const navigate = useNavigate()
  const location = useLocation()
  const from = (location.state as { from?: Location })?.from?.pathname ??
'/'
```

```
  const [step, setStep] = useState<'credentials' | 'totp'>('credentials')
  const [partialToken, setPartialToken] = useState('')
  const [serverError, setServerError] = useState<string | null>(null)

  const loginForm = useForm<LoginFormData>({ resolver:
zodResolver(loginSchema) })
  const totpForm = useForm<TotpFormData>({ resolver:
zodResolver(totpSchema) })

  const onLoginSubmit = async (data: LoginFormData) => {
    setServerError(null)
    try {
      const result = await login(data)
      if (result.requires2fa) {
        setPartialToken(result.partialToken)
        setStep('totp')
      } else {
        navigate(from, { replace: true })
      }
    } catch {
      setServerError('Невірний email або пароль')
    }
  }

  const onTotpSubmit = async (data: TotpFormData) => {
    setServerError(null)
    try {
      await verify2fa(partialToken, data.code)
      navigate(from, { replace: true })
    } catch {
      setServerError('Невірний код 2FA. Спробуйте ще раз.')
    }
  }
```

```
  return (
    <Box sx={{ minHeight: '100vh', display: 'flex', alignItems: 'center',
justifyContent: 'center', bgcolor: 'background.default', px: 4 }}>
      <Box sx={{ width: '100%', maxWidth: 440 }}>

        {step === 'credentials' ? (
          <>
            <Typography variant="h1" sx={{ fontSize: '1.75rem', fontWeight:
700, mb: 1, letterSpacing: '-0.03em' }}>
              Вхід в AssetMaster
            </Typography>
            <Typography variant="body1" color="text.secondary" sx={{ mb: 4
}}>
              Немає акаунту?{' '}
              <Link   to="/auth/register"   style={{   color:   '#3B82F6',
fontWeight: 600, textDecoration: 'none' }}>
                Зареєструватись
              </Link>
            </Typography>

            {serverError && (
              <Alert   severity="error"   sx={{   mb:   3,   borderRadius:   2
}}>{serverError}</Alert>
            )}

            <Box                                                component="form"
onSubmit={loginForm.handleSubmit(onLoginSubmit)} noValidate sx={{ display:
'flex', flexDirection: 'column', gap: 2.5 }}>
              <TextField
                label="Email"
                type="email"
                autoComplete="email"
                autoFocus
```

```
                fullWidth
                {...loginForm.register('email')}
                error={!!loginForm.formState.errors.email}
                helperText={loginForm.formState.errors.email?.message}
              />
              <TextField
                label="Пароль"
                type="password"
                autoComplete="current-password"
                fullWidth
                {...loginForm.register('password')}
                error={!!loginForm.formState.errors.password}
                helperText={loginForm.formState.errors.password?.message}
              />

              <Box sx={{ textAlign: 'right', mt: -1 }}>
                <Link  to="/auth/forgot-password"  style={{  fontSize:  14,
color: '#3B82F6', textDecoration: 'none' }}>
                  Забули пароль?
                </Link>
              </Box>

              <Button
                type="submit"
                variant="contained"
                size="large"
                fullWidth
                disabled={loginForm.formState.isSubmitting}
                sx={{ py: 1.5, mt: 1, borderRadius: 3 }}
              >
                {loginForm.formState.isSubmitting ? 'Вхід...' : 'Увійти'}
              </Button>

              <Divider sx={{ my: 1 }}>
```

```
                <Typography                                    variant="body2"
color="text.secondary">або</Typography>
              </Divider>

              <Button  variant="outlined"  size="large"  fullWidth  disabled
sx={{ borderRadius: 3, color: 'text.secondary' }}>
                Продовжити з Google (незабаром)
              </Button>
            </Box>
          </>
        ) : (
          <>
            <Typography variant="h1" sx={{ fontSize: '1.75rem', fontWeight:
700, mb: 1, letterSpacing: '-0.03em' }}>
              Двофакторна автентифікація
            </Typography>
            <Typography variant="body1" color="text.secondary" sx={{ mb: 4
}}>
              Введіть 6-значний код з вашого застосунку-автентифікатора
            </Typography>

            {serverError && (
              <Alert   severity="error"   sx={{   mb:   3,   borderRadius:   2
}}>{serverError}</Alert>
            )}
            <Box                                          component="form"
onSubmit={totpForm.handleSubmit(onTotpSubmit)}  noValidate  sx={{  display:
'flex', flexDirection: 'column', gap: 2.5 }}>
              <TextField
                label="Код 2FA"
                type="text"
                inputMode="numeric"
                autoComplete="one-time-code"
                autoFocus
```

```
                fullWidth
                placeholder="000000"
                inputProps={{ maxLength: 6 }}
                {...totpForm.register('code')}
                error={!!totpForm.formState.errors.code}
                helperText={totpForm.formState.errors.code?.message}
              />
              <Button
                type="submit"
                variant="contained"
                size="large"
                fullWidth
                disabled={totpForm.formState.isSubmitting}
                sx={{ py: 1.5, borderRadius: 3 }}
              >
                {totpForm.formState.isSubmitting    ?    'Перевірка...'    :
'Підтвердити'}
              </Button>

              <Button
                variant="text"
                onClick={()       =>       {       setStep('credentials');
setServerError(null) }}
                sx={{ color: 'text.secondary' }}
              >
                ← Повернутись
              </Button>
            </Box>
          </>
        )}
      </Box>
    </Box>
  )
}
```

Додаток Д. Лістинг файлу «JwtAuthFilter.java»

```
package com.assetmaster.api.security;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
import org.springframework.lang.NonNull;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationT
oken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.web.authentication.WebAuthenticationDetailsSou
rce;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;

@Component
@RequiredArgsConstructor
public class JwtAuthFilter extends OncePerRequestFilter {

    private final JwtService jwtService;
    private final UserDetailsServiceImpl userDetailsService;

    @Override
    protected void doFilterInternal(
            @NonNull HttpServletRequest request,
            @NonNull HttpServletResponse response,
```

```
            @NonNull FilterChain filterChain
    ) throws ServletException, IOException {
        String authHeader = request.getHeader("Authorization");
        if (authHeader == null || !authHeader.startsWith("Bearer ")) {
            filterChain.doFilter(request, response);
            return;
        }
        String token = authHeader.substring(7);
        String email;
        try {
            email = jwtService.extractUsername(token);
        } catch (Exception e) {
            filterChain.doFilter(request, response);
            return;
        }
        if (email != null &&
SecurityContextHolder.getContext().getAuthentication() == null) {
            UserDetails userDetails =
userDetailsService.loadUserByUsername(email);
            if (jwtService.isTokenValid(token, userDetails)) {
                UsernamePasswordAuthenticationToken authToken =
                        new
UsernamePasswordAuthenticationToken(userDetails, null,
userDetails.getAuthorities());
                authToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(authToken);
            }
        }
        filterChain.doFilter(request, response);
    }
}
```

Додаток Е. Лістинг файлу «AuthService.java»

```
package com.assetmaster.api.service;

import com.assetmaster.api.dto.*;
import com.assetmaster.api.entity.EmailVerificationToken;
import com.assetmaster.api.entity.PasswordResetToken;
import com.assetmaster.api.entity.User;
import com.assetmaster.api.exception.ApiException;
import com.assetmaster.api.repository.EmailVerificationTokenRepository;
import com.assetmaster.api.repository.PasswordResetTokenRepository;
import com.assetmaster.api.repository.UserRepository;
import com.assetmaster.api.security.JwtService;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseCookie;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.BadCredentialsException;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationT
oken;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.time.Duration;
import java.time.OffsetDateTime;
import java.util.UUID;

@Service
@RequiredArgsConstructor
```

```
@Transactional(readOnly = true)
public class AuthService {

    private final UserRepository                userRepository;
    private              final              PasswordResetTokenRepository
passwordResetTokenRepository;
    private            final            EmailVerificationTokenRepository
emailVerificationTokenRepository;
    private final PasswordEncoder               passwordEncoder;
    private final JwtService                    jwtService;
    private final AuthenticationManager         authenticationManager;
    private final EmailService                  emailService;
    private final TotpService                   totpService;

    @Value("${app.jwt.refresh-expiry-sec}")
    private long refreshExpirySec;

    // ── Register ──────────────────────────────────────────

    @Transactional
    public    AuthResponseDto    register(RegisterRequestDto    request,
HttpServletResponse response) {
        if (userRepository.existsByEmail(request.email())) {
            throw   new   ApiException(HttpStatus.CONFLICT,   "Email   вже
зайнятий");
        }

        User user = User.builder()
                .email(request.email())
                .passwordHash(passwordEncoder.encode(request.password()))
                .displayName(request.displayName())
                .build();
        userRepository.save(user);
```

```
        sendVerificationEmail(user);
        return issueTokens(user, response);
    }

    // ── Login ──────────────────────────────────────────────

    @Transactional
    public AuthResponseDto login(LoginRequestDto request,
HttpServletResponse response) {
        try {
            authenticationManager.authenticate(
                    new
UsernamePasswordAuthenticationToken(request.email(), request.password()));
        } catch (BadCredentialsException e) {
            throw new ApiException(HttpStatus.UNAUTHORIZED, "Невірний email
або пароль");
        }

        User user = userRepository.findByEmail(request.email())
                .orElseThrow(() -> new
ApiException(HttpStatus.UNAUTHORIZED, "Невірний email або пароль"));

        if (user.isTotpEnabled()) {
            String partialToken = jwtService.generatePartialToken(user);
            return AuthResponseDto.pending2fa(partialToken);
        }
        return issueTokens(user, response);
    }

    // ── 2FA verify ─────────────────────────────────────────

    @Transactional
    public AuthResponseDto verify2fa(VerifyTotpRequestDto request,
HttpServletResponse response) {
```

```
        String email;
        try {
            email = jwtService.extractUsername(request.partialToken());
        } catch (Exception e) {
            throw new ApiException(HttpStatus.UNAUTHORIZED, "Невалідний
токен");
        }
        if (!jwtService.isPartialToken(request.partialToken())) {
            throw new ApiException(HttpStatus.UNAUTHORIZED, "Невалідний
токен");
        }

        User user = userRepository.findByEmail(email)
                .orElseThrow(() -> new
ApiException(HttpStatus.UNAUTHORIZED, "Користувача не знайдено"));

        if (!totpService.verify(user.getTotpSecret(), request.code())) {
            throw new ApiException(HttpStatus.UNAUTHORIZED, "Невірний код
2FA");
        }
        return issueTokens(user, response);
    }

    // ── Refresh / Logout ──────────────────────────────────────

    @Transactional
    public AuthResponseDto refresh(String refreshToken, HttpServletResponse
response) {
        String email;
        try {
            email = jwtService.extractUsername(refreshToken);
        } catch (Exception e) {
            throw new ApiException(HttpStatus.UNAUTHORIZED, "Невалідний
refresh token");
```

```
        }

        User user = userRepository.findByEmail(email)
                .orElseThrow(() -> new
ApiException(HttpStatus.UNAUTHORIZED, "Користувача не знайдено"));

        if (!jwtService.isTokenValid(refreshToken, user)) {
            throw new ApiException(HttpStatus.UNAUTHORIZED, "Refresh token
прострочений або невалідний");
        }
        return issueTokens(user, response);
    }

    public void logout(HttpServletResponse response) {
        clearRefreshCookie(response);
    }

    // ── Email verification ──────────────────────────────

    @Transactional
    public void verifyEmail(String token) {
        EmailVerificationToken evt =
emailVerificationTokenRepository.findByToken(token)
                .orElseThrow(() -> new ApiException(HttpStatus.BAD_REQUEST,
"Невалідний або прострочений токен"));

        if (evt.getExpiresAt().isBefore(OffsetDateTime.now())) {
            throw new ApiException(HttpStatus.BAD_REQUEST, "Токен
прострочений");
        }
        User user = evt.getUser();
        user.setEmailVerified(true);
        userRepository.save(user);
        emailVerificationTokenRepository.deleteByUserId(user.getId());
```

```
    }

    @Transactional
    public void resendVerification(User currentUser) {

emailVerificationTokenRepository.deleteByUserId(currentUser.getId());
        sendVerificationEmail(currentUser);
    }

    // ── Password reset ──────────────────────────────────────

    @Transactional
    public void forgotPassword(String email) {
        userRepository.findByEmail(email).ifPresent(user -> {
            passwordResetTokenRepository.deleteByUserId(user.getId());
            String token = UUID.randomUUID().toString().replace("-", "");
            passwordResetTokenRepository.save(PasswordResetToken.builder()
                    .user(user)
                    .token(token)
                    .expiresAt(OffsetDateTime.now().plusHours(1))
                    .build());
            emailService.sendPasswordReset(email, token);
        });
        // Always succeed to avoid email enumeration
    }

    @Transactional
    public void resetPassword(ResetPasswordRequestDto request) {
        PasswordResetToken                          prt                          =
passwordResetTokenRepository.findByToken(request.token())
                .orElseThrow(() -> new ApiException(HttpStatus.BAD_REQUEST,
"Невалідний або прострочений токен"));

        if (prt.isUsed()) {
```

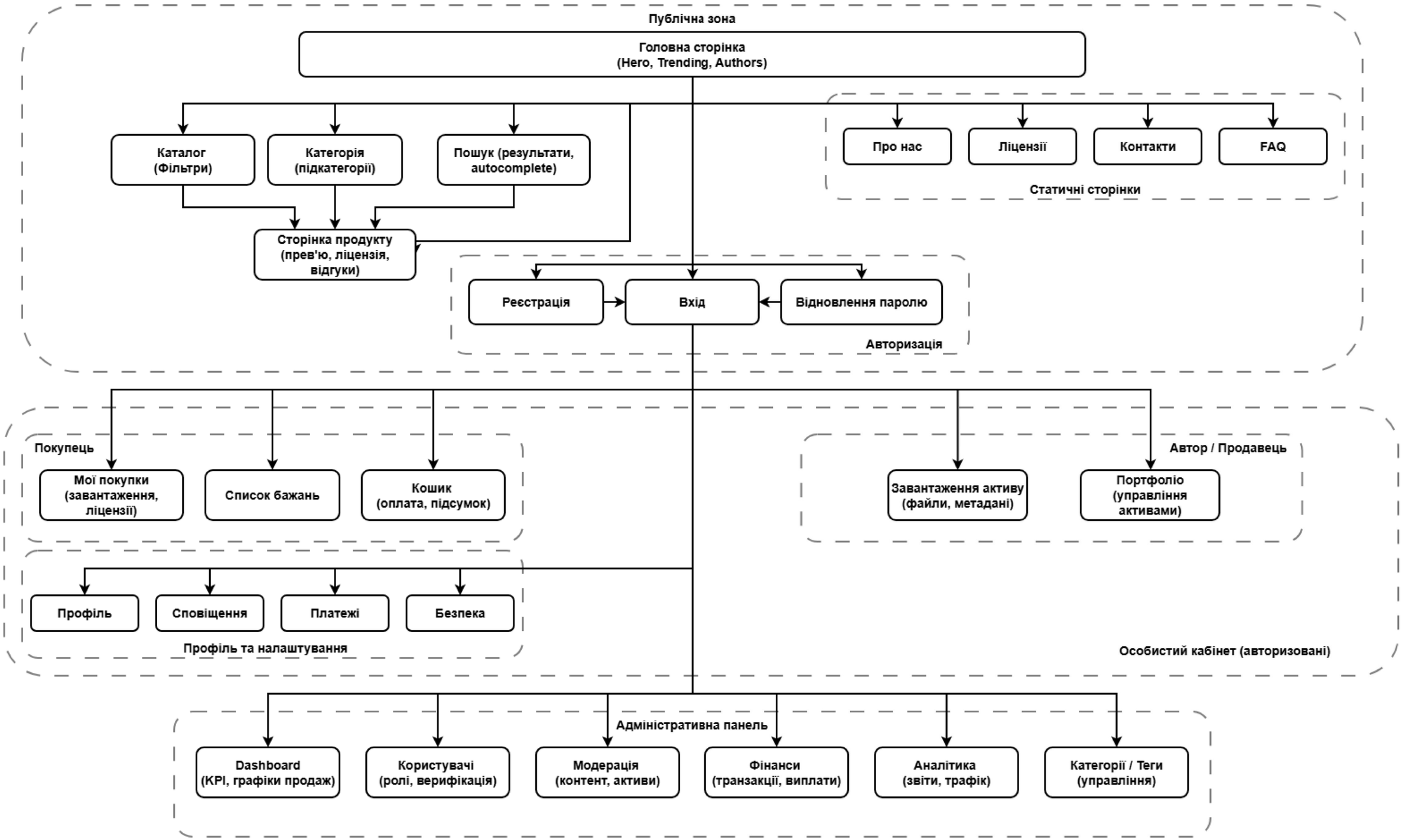
```
            throw new ApiException(HttpStatus.BAD_REQUEST, "Токен вже
використано");
        }
        if (prt.getExpiresAt().isBefore(OffsetDateTime.now())) {
            throw new ApiException(HttpStatus.BAD_REQUEST, "Токен
прострочений");
        }

        User user = prt.getUser();

user.setPasswordHash(passwordEncoder.encode(request.newPassword()));
        userRepository.save(user);
        prt.setUsed(true);
        passwordResetTokenRepository.save(prt);
    }

    // ── Profile update ──────────────────────────────────────

    @Transactional
    public UserResponseDto updateProfile(User user, UpdateProfileRequestDto
request) {
        if (request.displayName() != null &&
!request.displayName().isBlank()) {
            user.setDisplayName(request.displayName());
        }
        if (request.avatarUrl() != null) {
            user.setAvatarUrl(request.avatarUrl().isBlank() ? null :
request.avatarUrl());
        }
        if (request.bio() != null) {
            user.setBio(request.bio().isBlank() ? null : request.bio());
        }
        return UserResponseDto.fromEntity(userRepository.save(user));
    }
```

```
    // ── TOTP setup ──────────────────────────────────────────

    @Transactional
    public TotpSetupDto setupTotp(User user) {
        if (user.isTotpEnabled()) {
            throw new ApiException(HttpStatus.BAD_REQUEST, "2FA вже
увімкнено");
        }
        String secret = totpService.generateSecret();
        user.setTotpSecret(secret);
        userRepository.save(user);
        return new TotpSetupDto(secret,
totpService.generateUri(user.getEmail(), secret));
    }

    @Transactional
    public UserResponseDto enableTotp(User user, String code) {
        if (user.getTotpSecret() == null) {
            throw new ApiException(HttpStatus.BAD_REQUEST, "Спочатку
налаштуйте 2FA");
        }
        if (!totpService.verify(user.getTotpSecret(), code)) {
            throw new ApiException(HttpStatus.UNAUTHORIZED, "Невірний код
2FA");
        }
        user.setTotpEnabled(true);
        return UserResponseDto.fromEntity(userRepository.save(user));
    }

    @Transactional
    public UserResponseDto disableTotp(User user, String code) {
        if (!user.isTotpEnabled()) {
            throw new ApiException(HttpStatus.BAD_REQUEST, "2FA не
```

```
увімкнено");
        }
        if (!totpService.verify(user.getTotpSecret(), code)) {
            throw new ApiException(HttpStatus.UNAUTHORIZED, "Невірний код
2FA");
        }
        user.setTotpEnabled(false);
        user.setTotpSecret(null);
        return UserResponseDto.fromEntity(userRepository.save(user));
    }

    // ── Helpers ──────────────────────────────────────────────

    private AuthResponseDto issueTokens(User user, HttpServletResponse
response) {
        String accessToken  = jwtService.generateAccessToken(user);
        String refreshToken = jwtService.generateRefreshToken(user);
        setRefreshCookie(response, refreshToken);
        return                          AuthResponseDto.success(accessToken,
UserResponseDto.fromEntity(user));
    }

    private void sendVerificationEmail(User user) {
        String token = UUID.randomUUID().toString().replace("-", "");

emailVerificationTokenRepository.save(EmailVerificationToken.builder()
                .user(user)
                .token(token)
                .expiresAt(OffsetDateTime.now().plusHours(24))
                .build());
        emailService.sendEmailVerification(user.getEmail(), token);
    }

    private void setRefreshCookie(HttpServletResponse response, String
```

```
refreshToken) {
        ResponseCookie    cookie    =    ResponseCookie.from("refreshToken",
refreshToken)
                .httpOnly(true).secure(false).sameSite("Lax")
                .path("/api/v1/auth/refresh")
                .maxAge(Duration.ofSeconds(refreshExpirySec))
                .build();
        response.addHeader(HttpHeaders.SET_COOKIE, cookie.toString());
    }

    private void clearRefreshCookie(HttpServletResponse response) {
        ResponseCookie cookie = ResponseCookie.from("refreshToken", "")
                .httpOnly(true).secure(false).sameSite("Lax")

.path("/api/v1/auth/refresh").maxAge(Duration.ZERO).build();
        response.addHeader(HttpHeaders.SET_COOKIE, cookie.toString());
    }
}
```

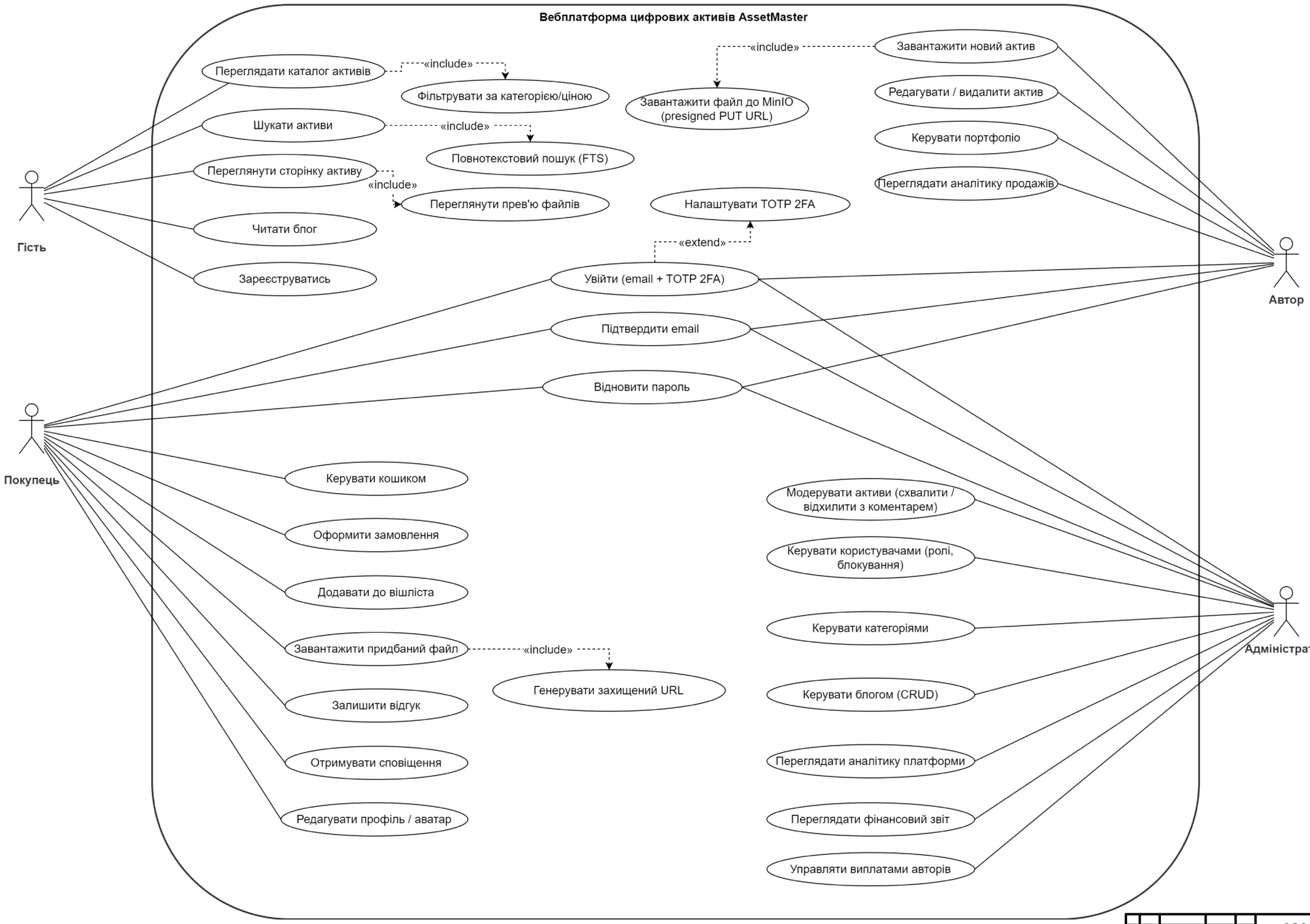
Схема структурна клієнтської частини



					2026.КВР.122.4.21.10.00.00 СС		
Зм.	Арк.	№ докум.	Підпис	Дата	Розробка вебплатформи для дистрибуції та продажу цифрових активів «AssetMaster»	Літ.	Маса	Масштаб
Розроб.		Кочула Д.І.						
Перевір.		Слободан Р.О.			Схема структурна клієнтської частини	Аркуш	Аркушів 1
Т.контр.							
Реценз.						ВСП ТФК ТНТУ КН-4.21	
Н.контр.		Приймак В.А.				м. Тернопіль	
Затв.							

Копіював Формат А1

UML-діаграма варіантів використання



					2026.КВР.122.421.10.00.00 ДВ		
Зм.	Арк.	№ докум.	Підпис	Дата	Розробка вебплатформи для дистрибуції та продажу цифрових активів «AssetMaster» UML-діаграма варіантів використання	Літ.	Маса	Масштаб
Розроб.		Кочула Д.І.						
Перевір.		Слободян Р.О.				Аркуш	Аркушів 1	
Т.контр.						ВСП ТФК ТНТУ КН-421		
Реценз.						м. Тернопіль		
Н.контр.		Приймак В.А.						
Затв.								

Копіював Формат А1

ER-діаграма бази даних

flyway_schema_history

installed_rank	integer NN
version	varchar(50)
description	varchar(200) NN
type	varchar(20) NN
script	varchar(1000) NN
checksum	integer
installed_by	varchar(100) NN
installed_on	timestamp NN
execution_time	integer NN
success	boolean NN

orders

id	**bigserial**
buyer_id	bigint NN
total_amount	numeric(10,2) NN
status	varchar(20) NN
created_at	timestamp NN

order_items

id	**bigserial**
order_id	bigint NN
asset_id	bigint NN
price_at_purchase	numeric(10,2) NN
license_type	varchar(20) NN

categories

id	**bigserial**
name	varchar(100) NN
slug	varchar(100) NN
parent_id	bigint
icon_url	varchar(2048)

assets

id	**bigserial**
author_id	bigint NN
title	varchar(500) NN
description	text
category_id	bigint
price	numeric(10,2) NN
license_type	varchar(50) NN
status	varchar(50) NN
file_key	varchar(2048)
preview_urls	jsonb NN
tags	text[] NN
downloads_count	integer NN
views_count	integer NN
created_at	timestamp NN

users

id	**bigserial**
email	varchar(255) NN
password_hash	varchar(255) NN
role	varchar(50) NN
display_name	varchar(255)
avatar_url	varchar(2048)
bio	text
is_verified	boolean NN
created_at	timestamp NN

wishlist_items

id	**bigserial**
user_id	bigint NN
asset_id	bigint NN
created_at	timestamp NN

Зм.	Арк.	№ докум.	Підпис	Дата	2026.КВР.122.421.10.00.00 БД		
Розроб.		Кичула Д.І.			Розробка вебплатформи для дистрибуції та продажу цифрових активів «AssetMaster»	Літ.	Маса	Масштаб
Перевір.		Слободян Р.О.			ER-діаграма бази даних			
Т.контр.						Аркуш	Аркушів 1	
Рецензент						ВСП ТФК ТНТУ КН-421		
Н.контр.		Прийма В.А.				м. Тернопіль		
Затв.								

Копіював Формат А1

Таблиця техніко-економічних показників

№ п/п	Показник	Одиниці вимірювання	Значення
1	Архітектура програмного забезпечення	–	Клієнт-серверна
2	Технології клієнтської частини	–	TypeScript 6.0, React 19.2, Material UI 9.1, TanStack Query 5.10
3	Технології серверної чистини	–	Java, Spring Boot 3.3.5, Maven
4	База даних	–	PostgreSQL 15
5	Зовнішні інтеграції	–	Stripe Connect, SMTP, MilO
6	Обмін даними між сервером і клієнтами	–	REST, JSON
7	Захист	–	Spring Security + JWT
8	Собівартість	грн.	69261,44
9	Плановий прибуток	грн.	38786,41
10	Ціна	грн.	108047,85
11	Чиста теперішня вартість	грн.	27194,61
12	Термін окупності	рік	2,1