

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій**

ПОГОДЖЕНО
Декан факультету

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри комп'ютерних
наук

_____ **Ігор БОЛБОТ**

_____ **Белла ГОЛУБ**

« ____ » _____ 2026 р.

« ____ » _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Веб-орієнтована інформаційна система продажу моделей для 3D-друку»

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Гарант освітньої програми
Д.є.н., доцент

_____ **Руденський Р.А.**

**Керівник бакалаврської
кваліфікаційної роботи**
к.ф.-м.н., доцент

_____ **Криворучко Яніна**

Виконала

_____ 

Страшнюк Анастасія

Київ-2026

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій**

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«__» _____ 2026 р.

**ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ**

Страшнюк Анастасії Олегівні

(прізвище, ім'я, по батькові)

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Тема бакалаврської кваліфікаційної роботи

«Веб-орієнтована інформаційна система продажу моделей для 3D-друку»

затверджена наказом від «06» __ квітня ____ 2026 р. № _972_ «С»

Термін подання завершеної роботи на кафедру «22» __ травня ____ 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи:

архітектура клієнт-сервер, REST API, засоби розробки: PHP 8.2, Symfony 7, PostgreSQL 16, React 19, TypeScript, Stripe API, JWT-автентифікація, вимоги до функціональності платформи купівлі та продажу 3D-моделей.

Перелік питань, що підлягають дослідженню:

аналіз існуючих платформ торгівлі 3D-контентом, проектування реляційної бази даних, розробка серверної частини на основі REST API, розробка клієнтського односторінкового застосунку, інтеграція платіжного шлюзу, тестування програмного забезпечення.

Перелік графічних документів (за необхідності).

Дата видачі завдання «__» _____ 2026 р.

**Керівник бакалаврської
кваліфікаційної роботи**

к.ф.-м.н., доцент

Завдання взяв до виконання

(підпис)

Криворучко Яніна

(підпис)

Страшнюк Анастасія

ЗМІСТ

ВСТУП.....	5
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	7
1.1. Постановка задачі.....	7
1.2. Аналіз предметної області.....	8
1.3. Огляд існуючих рішень та інформаційних джерел.....	9
1.4. Моделювання предметної області.....	11
1.5. Формування функціональних та нефункціональних вимог.....	13
2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ.....	16
2.1. Опис логічної структури бази даних.....	16
2.2. Побудова ER-діаграми інформаційної бази.....	19
2.3. Вибір системи управління базою даних.....	21
2.4. Створення фізичної структури бази даних.....	22
2.5. Розробка SQL-запитів, тригерів та процедур.....	29
2.6. Організація доступу та захисту даних.....	32
3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ВЕБ-ПЛАТФОРМИ.....	34
3.1.Організаційна структура програмного забезпечення.....	34
3.2.Вибір технологій та інструментів розробки	38
3.3.Розробка серверної частини системи.....	40
3.4.Розробка клієнтської частини та користувацького інтерфейсу системи.....	42
3.5.Реалізація модулів системи.....	45

3.6. Алгоритмізація та програмування програмних модулів.....	48
4 ВПРОВАДЖЕННЯ СИСТЕМИ.....	55
4.1. Тестування системи.....	55
4.2. Вимоги до апаратного та програмного забезпечення.....	59
4.3. Розгортання системи та склад інсталяційного пакету.....	60
4.4. Перспективи розвитку системи.....	61
ВИСНОВКИ.....	63
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	65
ДОДАТОК А.....	68
ДОДАТОК Б.....	76
ДОДАТОК В.....	87
ДОДАТОК Г.....	96

ВСТУП

Цифровий контент стає повноцінним товаром: з розвитком ігрової індустрії, застосунків доповненої реальності та адитивного виробництва попит на готові тривимірні активи помітно зріс. Водночас інструменти для їх продажу залишаються або надто складними для незалежних авторів, або орієнтованими виключно на великі студії. Це робить розробку доступної спеціалізованої платформи актуальним завданням.

Тривимірна графіка давно вийшла за межі кіноіндустрії та великих ігрових студій. Сьогодні 3D-моделі використовують архітектори для візуалізації проєктів, інженери — для прототипування деталей, розробники ігор та застосунків доповненої реальності — для наповнення сцен. Відповідно зріс і запит на готові активи: не кожен фахівець має час і ресурси створювати модель з нуля, коли потрібний об'єкт вже існує і може бути куплений за розумну ціну.

Проте ринок таких активів досі влаштований незручно. Платформи на зразок TurboSquid чи CGTrader існують роками, але мають суттєві обмеження: складне ціноутворення, непрозорі умови ліцензування, відсутність попереднього перегляду моделі безпосередньо у браузері. Для незалежного автора, що хоче просто продати власну роботу, процес реєстрації та виведення коштів нерідко виявляється складнішим, ніж сама розробка моделі. Це і стало відправною точкою для даної роботи.

Метою роботи є розробка веб-платформи для купівлі та продажу 3D-моделей, що забезпечує зручний інтерфейс, захищену автентифікацію, онлайн-оплату та інструменти адміністрування.

Для досягнення мети вирішувалися такі завдання: дослідити предметну область і наявні аналоги; визначити вимоги до системи; спроектувати структуру бази даних; розробити серверну та клієнтську частини платформи; реалізувати основні функціональні модулі; провести тестування й підготувати систему до розгортання.

Об'єктом дослідження є процес розробки веб-платформи для торгівлі цифровим тривимірним контентом. Предметом — методи проєктування і реалізації такої платформи з використанням сучасних інструментів веб-розробки. У процесі роботи застосовувалися методи об'єктно-орієнтованого проєктування, концептуального та логічного моделювання даних, проєктування програмних інтерфейсів, а також модульного тестування програмних компонентів.

Розроблена система має практичну цінність: результатом є повноцінний програмний продукт, що забезпечує повний цикл електронної комерції — від розміщення товару продавцем до його придбання покупцем. Платформа може використовуватися незалежними авторами тривимірного контенту для монетизації власних робіт.

Робота складається з чотирьох розділів. Перший присвячено аналізу предметної області, огляду існуючих рішень та формуванню вимог до системи. У другому розглядається інформаційне забезпечення — логічна структура бази даних, ER-діаграма, вибір СУБД та організація захисту даних. Третій розділ охоплює розробку програмного забезпечення: архітектуру системи, вибір технологій та реалізацію модулів. Четвертий містить результати тестування, вимоги до середовища, опис розгортання та перспективи подальшого розвитку платформи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка задачі

Зростання попиту на готові тривимірні активи формує потребу у спеціалізованому онлайн-майданчику, де автори можуть розміщувати власні роботи, а покупці — знаходити і придбавати потрібні моделі. Існуючі рішення або надто складні для незалежного автора, або не забезпечують повного циклу торгівлі — від завантаження файлу до отримання оплати. Завдання даної роботи полягає у розробці веб-платформи, що усуває ці прогалини.

Система має обслуговувати три категорії користувачів із різними потребами. Покупець повинен мати змогу переглядати каталог моделей, фільтрувати їх за категоріями та ціною, ознайомлюватися з деталями перед покупкою і здійснювати оплату онлайн. Продавець — завантажувати моделі, описувати їх, встановлювати ціну та відстежувати продажі. Адміністратор — контролює якість контенту, модерує розміщені матеріали та керує обліковими записами користувачів.

З технічного боку платформа повинна забезпечувати надійну автентифікацію, розмежування прав доступу залежно від ролі користувача, безпечну обробку платежів та коректну роботу з файлами великого розміру. Інтерфейс має бути зрозумілим без додаткового навчання — як для технічно підготовленого спеціаліста, так і для звичайного користувача.

Таким чином задача формулюється наступним чином: спроектувати і реалізувати веб-орієнтовану платформу, яка поєднує функції каталогу цифрового контенту, особистого кабінету продавця і покупця, системи онлайн-оплати та адміністративної панелі управління — і при цьому є достатньо простою у використанні, щоб не потребувати технічної підтримки для базових операцій.

За характером використання інформації система належить до класу інформаційно-довідкових систем із функціями електронної комерції — вона

зберігає, обробляє і надає інформацію про цифрові товари та забезпечує операції їх купівлі-продажу. За масштабом це багатокористувацька веб-система, розрахована на необмежену кількість одночасних користувачів. За ступенем автоматизації — автоматизована система, що виконує операції обробки замовлень, перерахунку рейтингів і оновлення службових полів без участі людини. За підтримуваними технологіями комунікації система використовує клієнт-серверну архітектуру з REST API як інтерфейсом взаємодії між компонентами.

1.2 Аналіз предметної області

Маркетплейс 3D-моделей — це різновид платформи для торгівлі цифровим контентом. Товаром тут виступає не фізичний предмет, а файл разом із правом на його використання. Здавалося б, нічого особливого — але саме ця деталь визначає майже всі ключові рішення при проєктуванні такої системи.

Ринок тривимірного контенту неоднорідний. Архітектори замовляють або купують готові меблі, рослини, автомобілі для візуалізації інтер'єрів. Розробники ігор потребують персонажів, зброї, оточення — причому у форматах, сумісних із конкретними рушіями. Інженери завантажують технічні деталі для перевірки ще до виробництва фізичного прототипу. Для кожної з цих категорій «потрібна модель» означає щось своє — і за форматом, і за рівнем деталізації, і за умовами використання.

Останній пункт особливо важливий. Цифровий файл, на відміну від фізичного товару, можна скопіювати без обмежень — тому угода між продавцем і покупцем будується не навколо передачі об'єкта, а навколо ліцензії. Ліцензія визначає межі: чи можна використати модель у комерційному проєкті, чи дозволена модифікація, чи є обмеження на кількість кінцевих продуктів. Без чіткої системи ліцензування платформа перетворюється на звичайне файлоховище, а не повноцінний інструмент для бізнесу.

Платформа в цій схемі відіграє роль посередника — і не лише між грошима й файлом. Вона одночасно є вітриною для продавця, пошуковим інструментом для покупця та фільтром якості для обох. Саме тому в системі з'являється третя роль — адміністратор, який перевіряє завантажений контент до того, як він стане доступним у каталозі.

Окремої уваги заслуговує специфіка самих файлів. 3D-моделі бувають різних форматів: одні призначені для рендерингу, інші — для ігрових рушіїв або 3D-друку. Вага таких файлів нерідко вимірюється сотнями мегабайт, що виходить за межі звичного для веб-застосунків. До того ж покупець не може оцінити модель «наживо» до покупки — а значить, попередній перегляд безпосередньо у браузері стає не зручною опцією, а необхідністю.

Отже, предметна область охоплює одразу кілька пластів: електронну комерцію, управління цифровими правами та роботу з великими мультимедійними файлами. Кожен із них впливає на те, як саме будується система — і ці особливості послідовно враховані в наступних розділах.

1.3 Огляд існуючих рішень та інформаційних джерел

Перед проєктуванням власної платформи було проаналізовано найбільш поширені існуючі рішення у сфері торгівлі 3D-контентом. Огляд охоплює три платформи, що займають провідні позиції на ринку: TurboSquid, CGTrader та Sketchfab.

TurboSquid — одна з найстаріших платформ у цій ніші, заснована у 2000 році. Має каталог, що налічує понад мільйон моделей, і орієнтована переважно на великі студії та корпоративних замовників. Платформа запровадила власний стандарт якості StemCell, якому мають відповідати файли для отримання вищого рейтингу. Для нового продавця комісія платформи може сягати 60% від вартості продажу, що робить монетизацію не вигідною на початковому етапі. Інтерфейс

перевантажений і орієнтований на досвідченого користувача — простого завантаження моделі недостатньо, потрібно пройти кілька рівнів верифікації.

CGTrader — платформа литовського походження, заснована у 2011 році. Порівняно з TurboSquid пропонує нижчу комісію і має вбудований переглядач моделей. Додатково функціонує як фриланс-майданчик, де замовники можуть шукати авторів під конкретні проекти. Втім, інтерфейс залишається перевантаженим, а процес перевірки нових завантажень займає значний час. Платформа не має чіткого розмежування між безкоштовним і платним контентом, що ускладнює навігацію.

Sketchfab — платформа, що спеціалізується насамперед на демонстрації тривимірного контенту у браузері. Має один із найкращих WebGL-переглядачів серед усіх існуючих рішень. Проте торгівля є лише додатковою функцією, а не основним призначенням сервісу: інструменти продавця розвинені слабо, аналітика продажів мінімальна. У 2023 році Sketchfab було інтегровано до платформи Fab.com від Epic Games, що переорієнтувало його ще більше на ігрову індустрію.

Для зручного порівняння основні характеристики наведено у таблиці 1.1.

Таблиця 1.1

Порівняльний аналіз існуючих платформ

Критерій	TurboSquid	CGTrader	Sketchfab
Перегляд у браузері	Обмежений	Є	Відмінний
Комісія	До 60%	30–35%	30%
Простота для новачка	Низька	Середня	Середня
Адмін-модерація	Є	Є	Обмежена
Фільтрація каталогу	Розвинена	Розвинена	Базова
Орієнтація	Студії	Змішана	Демонстрація

Таблиця 1.2

Переваги та недоліки існуючих рішень

Платформа	Переваги	Недоліки
TurboSquid	Великий каталог; розвинена система верифікації якості	Висока комісія до 60%; складний процес onboarding; орієнтація на корпоративних клієнтів
CGTrader	Нижча комісія; вбудований переглядач; фриланс-функціонал	Перевантажений інтерфейс; повільна модерація; нечітке розмежування контенту
Sketchfab	Найкращий 3D- переглядач; велика спільнота	Торгівля є другорядною функцією; слабкі інструменти продавця; переорієнтація на ігрову індустрію

За результатами аналізу можна помітити, що жодна з розглянутих платформ не поєднує водночас зручний інтерфейс для незалежного автора, прозору систему ліцензування, якісний перегляд моделі до покупки та просту процедуру реєстрації й виведення коштів. Саме ці прогалини враховані при формуванні вимог до розроблюваної системи, розглянутих у підрозділі 1.5.

1.4 Моделювання предметної області

Для наочного відображення взаємодії між учасниками системи та її функціями використано діаграму варіантів використання (Use Case diagram) у нотації UML. Цей інструмент дозволяє абстрагуватися від технічної реалізації і зосередитися на тому, хто і що робить у системі — що особливо корисно на етапі аналізу, до початку проєктування бази даних і коду.

У системі виділено чотири актори. Гість — незареєстрований відвідувач, що має обмежений доступ до платформи. Покупець — авторизований користувач, що придбаває моделі. Продавець — користувач із розширеними

правами, що розміщує власний контент. Адміністратор — представник платформи з повним доступом до керування системою.

Гість може переглядати каталог, шукати моделі за назвою або категорією, відкривати сторінку окремої моделі та публічний профіль продавця. Для будь-якої дії, що передбачає взаємодію з акаунтом — покупки, завантаження, відгуку — він має спочатку зареєструватися або увійти.

Покупець має доступ до всього, що доступне гостю, і додатково може додавати моделі до кошика, оформлювати замовлення й оплачувати його онлайн, переглядати історію своїх покупок, завантажувати придбані файли, залишати коментарі (comment) на будь які 3д моделі або ж відгуки (review) на ті, які він уже купив. Управління власним профілем — зміна імені, аватара, пароля — також належить до його можливостей.

Продавець поєднує в собі права покупця з додатковими функціями: завантаження нової моделі з описом і зображеннями, редагування або видалення власних робіт, перегляд переліку своїх завантажень і статистики продажів. Варто зазначити, що завантажена модель хоча й з'являється у каталозі одразу — але додатково проходить перевірку адміністратором і в разі порушення правил платформи може бути видалена. Продавець отримує в такому випадку повідомлення з поясненням причини видалення та інструкцією щодо подальших дій.

Адміністратор працює переважно в окремій панелі керування. Його основні дії — перегляд моделей, що очікують на перевірку, підтвердження або відхилення кожної з них, управління обліковими записами користувачів і зміна їхніх ролей, блокування відгуків та коментарів, що порушують правила платформи, опрацювання запитів на виплату коштів продавцям, а також верифікація особистих даних продавця (KYC).

Діаграма варіантів використання наведена на рисунку 1.1.



Рис. 1.1 Діаграма варіантів використання

Побудована модель охоплює основні сценарії роботи платформи і слугує основою для формування функціональних вимог, розглянутих у підрозділі 1.5.

1.5 Формування функціональних та нефункціональних вимог

Вимоги до системи визначено за результатами аналізу предметної області, порівняння існуючих платформ і побудованої діаграми варіантів використання. За характером вони поділяються на дві групи — ті, що описують конкретні функції системи, і ті, що задають загальні якісні критерії її роботи.

Функціональні вимоги. Платформа обслуговує чотири категорії учасників, тому вимоги до функціоналу зручно розглядати окремо для кожної з них.

Відвідувач без облікового запису може вільно переглядати каталог, шукати моделі за назвою, звужувати результати за категорією чи ціною, відкривати сторінку окремої моделі й переглядати профіль її автора. Щоб придбати модель, залишити коментар або завантажити файл — потрібна реєстрація.

Зареєстрований покупець проходить повний шлях угоди: обирає модель, додає до кошика, оплачує онлайн і отримує файл для завантаження. Записи про всі придбання зберігаються в особистому кабінеті. Коментар до моделі(comment) може залишити кожний зареєстрований користувач, але відгук на модель(review) можна залишити лише після її покупки. Профіль — ім'я, фото, пароль — користувач редагує самостійно.

Продавець має власний набір інструментів: завантаження моделі з описом, зображеннями і ціною; редагування і видалення вже розміщених робіт; перегляд власного каталогу і даних про продажі. Він також може заповнити особистий профіль та подати запит на виведення зароблених коштів.

Адміністратор керує якістю контенту через окрему панель: переглядає моделі що очікують перевірки, підтверджує або відхиляє кожну з них, блокує відгуки та коментарі що порушують правила. Паралельно має доступ до списку користувачів і може змінювати їхні ролі та блокувати акаунти, опрацьовувати запити на виплату коштів і верифікувати KYC-профілі продавців, а також обробляти звернення у підтримку.

Нефункціональні вимоги. Безпека реалізована на кількох рівнях. Паролі не зберігаються у відкритому вигляді. Звернення до закритих ресурсів перевіряються на наявність дійсного токена. Платіжні дані через платформу не проходять — їх обробляє зовнішній сервіс.

Час відгуку системи має бути прийнятним для повсякденного використання — сторінки каталогу і картки моделі відкриваються без помітного очікування.

Помилки — невдала оплата, невідтримуваний формат файлу, відсутній ресурс — система обробляє коректно і повідомляє про них користувачу зрозумілим чином.

Інтерфейс не вимагає інструкцій: новий користувач має орієнтуватися на платформі без додаткового навчання. Встановлення будь-якого програмного забезпечення не потрібне — достатньо браузера.

Архітектура системи розрахована на розширення: нові модулі можна додавати не зачіпаючи вже реалізований функціонал.

Визначені вимоги стали орієнтиром при проєктуванні бази даних і розробці програмного забезпечення, що розглядаються в наступних розділах.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

2.1 Опис логічної структури бази даних

Логічна структура бази даних визначає які сутності існують у системі, які атрибути вони мають і як пов'язані між собою. На цьому етапі абстрагуються від конкретної СУБД і типів даних — головне зафіксувати змістовну модель предметної області.

У системі виділено дванадцять основних сутностей.

Користувач є центральною сутністю платформи. Він має унікальний ідентифікатор, електронну адресу, ім'я користувача, захешований пароль, роль у системі та дату реєстрації. Роль визначає рівень доступу і може набувати одного з трьох значень: покупець, продавець або адміністратор. Додатково зберігається ознака блокування облікового запису та причина блокування, що використовуються адміністратором для обмеження доступу до платформи.

Модель — основний товар платформи. Сутність зберігає назву, опис, ціну, дату завантаження, файл де зберігається модель, фото, що продавець завантажує задля можливості попереднього перегляду перед покупкою, інші унікальні дані щодо розміру, призначення та геометрії моделі та статус її верифікації. Статус верифікації відображає поточний стан моделі в процесі модерації і може бути «очікує», «підтверджено» або «відхилено». Кожна модель належить одному продавцю і може мати одну категорію.

Категорія слугує для класифікації моделей у каталозі. Вона має лише назву — проста довідникова сутність без складних зв'язків.

Ліцензія визначає умови використання придбаної моделі. Зберігає назву, опис умов та ознаку безкоштовності. Одна ліцензія може застосовуватися до багатьох моделей.

Замовлення фіксує факт покупки. Воно пов'язане з конкретним покупцем і має статус та дату створення. Одне замовлення може містити кілька позицій.

Позиція замовлення пов'язує замовлення з конкретною моделлю і фіксує ціну на момент покупки — це важливо, оскільки ціна моделі може змінитися пізніше.

Кошик і позиція кошика забезпечують тимчасове зберігання обраних моделей до оформлення замовлення. Кошик прив'язаний до користувача, а кожна позиція містить посилання на конкретну модель.

Відгук дозволяє верифікованому покупцю оцінити модель після придбання. Зберігає числову оцінку від одного до п'яти, текст та дату публікації. Право залишити відгук перевіряється на рівні сервісу — система звіряє факт покупки у таблиці позицій замовлення перед збереженням запису. Адміністратор може заблокувати відгук із зазначенням причини, для чого зберігається відповідна ознака та пояснення.

Коментар є окремою від відгуку сутністю і призначений для вільного обговорення під сторінкою моделі. На відміну від відгуку, коментар не містить числової оцінки і доступний будь-якому зареєстрованому користувачу — незалежно від факту покупки. Сутність зберігає текст, дату публікації, посилання на автора і модель, а також ознаку блокування з причиною — для адміністративної модерації.

Профіль продавця зберігає персональні дані для ідентифікації та здійснення виплат: ім'я, прізвище, дата народження, адреса. Пов'язаний із користувачем у відношенні «один до одного». Статус верифікації (KYC) може бути «очікує», «верифіковано» або «відхилено» і опрацьовується адміністратором.

Виплата фіксує запит продавця на виведення зароблених коштів. Зберігає суму, спосіб виплати (банківський переказ або PayPal), статус обробки («очікує», «підтверджено», «виплачено», «відхилено»), а також нотатки продавця й адміністратора.

Тікет підтримки дозволяє користувачу звернутися до адміністрації. Зберігає тему, текст звернення, тип («загальна підтримка», «скарга на модель», «скарга на коментар»), статус («відкрито», «в обробці», «вирішено») та посилання на пов'язаний об'єкт.

Зв'язки між сутностями відображають бізнес-логіку платформи. Продавець може мати багато моделей, але кожна модель має одного продавця. Замовлення належить одному покупцю і містить одну або більше позицій. Відгук пов'язує користувача з моделлю і може бути залишений лише після підтвердженої покупки. Коментар пов'язує користувача з моделлю без додаткових обмежень. Профіль продавця пов'язаний з користувачем відношенням «один до одного». Виплата належить одному продавцю. Тікет підтримки пов'язаний із користувачем і опціонально — з моделлю або коментарем. Ці правила реалізовані на рівні бази даних і детально розглянуті у підрозділі 2.4.

Нормалізація бази даних. Спроектowana база даних відповідає вимогам третьої нормальної форми (3НФ). Перша нормальна форма дотримана — кожен атрибут містить атомарне значення і повторюваних груп немає. Друга нормальна форма виконана — усі неключові атрибути повністю залежать від первинного ключа: наприклад, ціна моделі зберігається у таблиці `models`, а не дублюється у кожній позиції замовлення — у `order_items` зберігається лише зафіксована ціна на момент покупки. Третя нормальна форма забезпечена — транзитивних залежностей між неключовими атрибутами немає: назва категорії і умови ліцензії винесені у окремі довідникові таблиці і не зберігаються безпосередньо у таблиці моделей.

Побудована логічна модель охоплює всі сценарії використання системи, визначені у розділі 1, і слугує основою для побудови ER-діаграми та фізичної реалізації бази даних.

Повний перелік атрибутів кожної таблиці, їхні типи даних та обмеження цілісності відображені на ER-діаграмі у підрозділі 2.2.

2.2 Побудова ER-діаграми інформаційної бази

ER-діаграма є графічним відображенням логічної структури бази даних — вона показує сутності системи, їхні атрибути та зв'язки між ними. Побудована діаграма охоплює всі таблиці що беруть участь у роботі платформи.

Центральною сутністю є `models` — саме навколо неї будується більшість зв'язків. Модель належить одному користувачу з таблиці `users`, має одну категорію з таблиці `categories` та одну ліцензію з таблиці `licenses`. До моделі прив'язані зображення попереднього перегляду з таблиці `model_preview_images` та файли з таблиці `model_files` — обидві таблиці знаходяться у відношенні «один до багатьох» з моделлю, оскільки одна модель може мати кілька зображень і кілька файлів різних форматів.

Таблиця `users` є другою за кількістю зв'язків. Користувач може мати багато моделей, багато відгуків, багато коментарів, багато позицій кошика та багато замовлень. Крім основних даних облікового запису, таблиця зберігає інформацію для інтеграції з платіжною системою, ознаку верифікації електронної адреси та поля блокування.

Процес покупки відображають три пов'язані таблиці. `orders` фіксує факт замовлення і належить одному користувачу. `order_items` містить конкретні позиції — кожна прив'язана до одного замовлення і однієї моделі, а ціна фіксується на момент покупки. `payments` зберігає інформацію про платіж і пов'язана з конкретним замовленням.

Таблиця `cart_items` реалізує кошик — тимчасове сховище обраних моделей до оформлення замовлення. Кожна позиція кошика прив'язана до користувача і моделі.

Таблиця `reviews` пов'язує користувача з моделлю через відгук із числовою оцінкою. Право залишити відгук перевіряється на сервісному рівні — система підтверджує наявність відповідного запису у `order_items` перед збереженням.

Схема бази даних не накладає унікального обмеження на пару «користувач — модель», тому кількість відгуків одного покупця на одну модель не обмежена.

Таблиця `comments` реалізує вільне обговорення під сторінкою моделі — незалежно від факту покупки. Кожен запис пов'язаний із користувачем і моделлю та містить текст, дату і поля адміністративного блокування. На відміну від `reviews`, таблиця `comments` не зберігає числової оцінки і не впливає на розрахунок середнього рейтингу моделі.

Таблиця `seller_profiles` зберігає верифікаційні та адресні дані продавця і пов'язана з `users` відношенням «один до одного». Таблиця `payouts` фіксує запити на виведення коштів і належить одному користувачу. Таблиця `support_tickets` реалізує систему звернень до адміністрації — кожен тикет пов'язаний із користувачем і опціонально містить посилання на модель або коментар.

ER-діаграма наведена на рисунку 2.1. На ній показано всі таблиці з повним складом атрибутів; у тексті детальніше описані лише ті з них, що мають нетривіальну семантику або виконують ключову роль у бізнес-логіці.

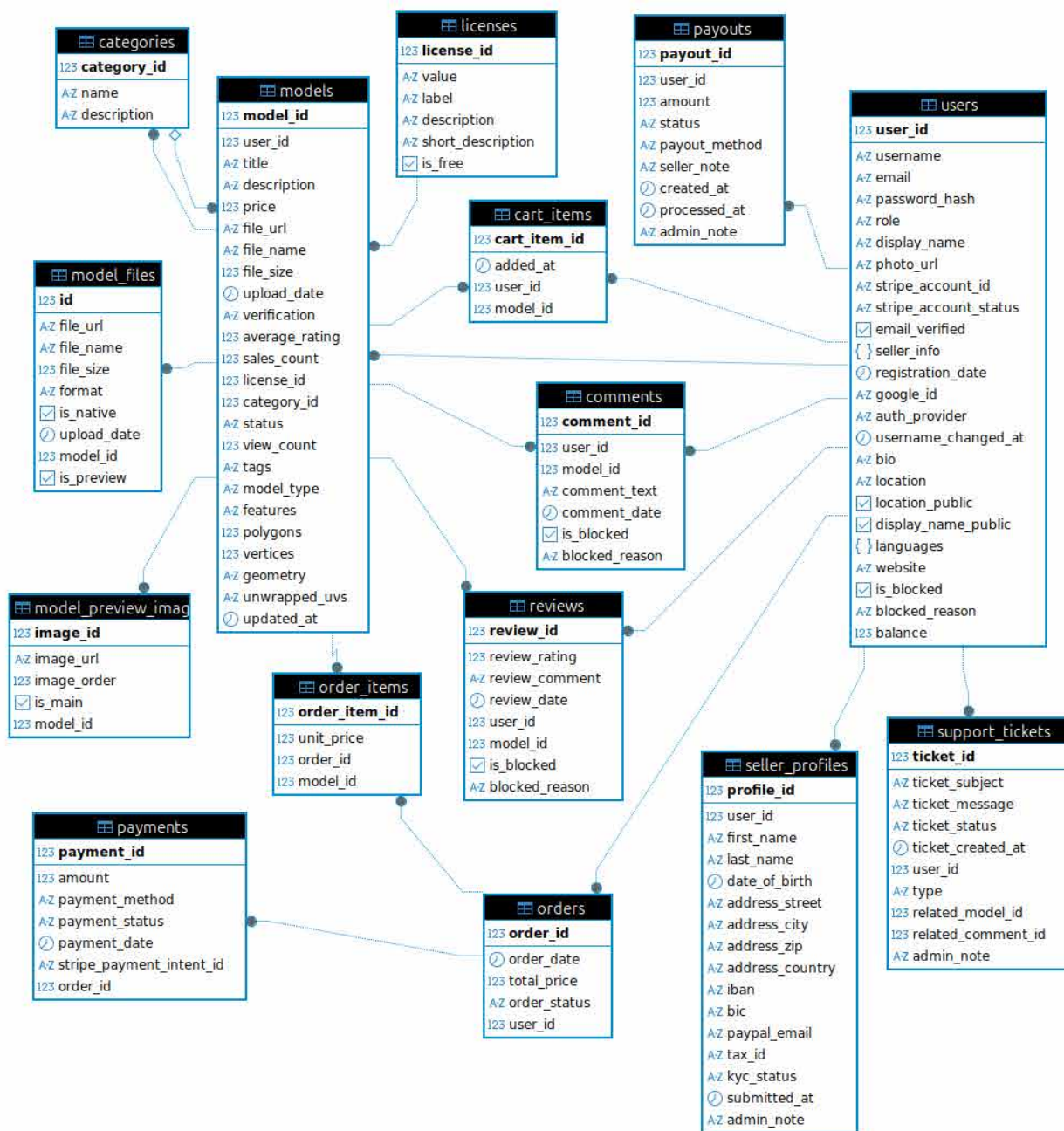


Рис. 2.1 ER-діаграма бази даних

2.3 Вибір системи управління базою даних

Вибір СУБД визначався структурою даних платформи та вимогами до надійності й масштабованості. Система охоплює кілька взаємопов'язаних предметних областей — користувачів, каталог моделей, замовлення та платежі — що передбачає велику кількість зв'язків між таблицями, транзакційну обробку та можливе зростання навантаження з часом.

Для обґрунтованого вибору було проведено порівняльний аналіз чотирьох СУБД: PostgreSQL, MySQL/MariaDB, SQLite та Microsoft SQL Server. Результати наведено у таблиці 2.1.

Таблиця 2.1

Порівняльний аналіз СУБД

Критерій	PostgreSQL	MySQL/MariaDB	SQLite	MS SQL Server
Тип	Об'єктно-реляційна	Реляційна	Вбудована	Реляційна
Ліцензія	Open Source	Open Source	Безкоштовна	Частково безкоштовна
Підтримка транзакцій	Повна (ACID)	Повна (InnoDB)	Обмежена	Повна
Робота з багатьма користувачами	Відмінна	Добра	Погана	Відмінна
Складні запити	Дуже сильна	Середня	Обмежена	Дуже сильна
Тригери і функції	Розвинені	Обмежені	Відсутні	Розвинені
Масштабованість	Висока	Середня	Відсутня	Дуже висока
Підходить для веб-систем	Так	Так	Ні	Так

SQLite відразу відпадає як варіант для багатокористувацької системи — вона не розрахована на паралельні запити і не підтримує повноцінних транзакцій. Microsoft SQL Server є потужним рішенням, але поширюється за комерційною ліцензією і орієнтований переважно на корпоративне середовище. MySQL є прийнятним варіантом для простих схем, проте має обмеженішу підтримку тригерів і збережених процедур та є лише частково безкоштовною.

За результатами аналізу для проєкту обрано PostgreSQL. Ця СУБД є об'єктно-реляційною системою з відкритим вихідним кодом, що забезпечує повну підтримку транзакцій, розвинену систему тригерів і збережених функцій, а також надійну роботу в багатокористувацькому середовищі. Саме тригери відіграють важливу роль у даній системі — вони використовуються для автоматичного оновлення службових полів і перерахунку середнього рейтингу моделей. Додатковим аргументом стала повноцінна підтримка PostgreSQL у Doctrine ORM, що використовується на серверній стороні платформи.

2.4 Створення фізичної структури бази даних

Фізична структура бази даних визначає конкретні таблиці, типи полів, обмеження та індекси — тобто те, як логічна модель реалізована безпосередньо у PostgreSQL. Всі таблиці створено та оновлюються через міграції Doctrine, що забезпечує версійність схеми і можливість відтворення бази даних на будь-якому середовищі.

Таблиця users зберігає облікові записи всіх користувачів системи. Поле role обмежене трьома значеннями: buyer, seller, admin. Електронна адреса унікальна в межах системи. Поля is_blocked і blocked_reason використовуються адміністратором для обмеження доступу. Для інтеграції з платіжною системою зберігається stripe_account_id та stripe_account_status. Склад полів наведено у таблиці 2.2.

Таблиця 2.2

Сутність users

Поле	Тип	Опис
user_id	SERIAL PRIMARY KEY	Унікальний ідентифікатор
username	VARCHAR(100)	Ім'я користувача
email	VARCHAR(255) UNIQUE	Електронна адреса

password_hash	VARCHAR(255)	Захешований пароль
role	VARCHAR(20)	Роль: buyer / seller / admin
registration_date	TIMESTAMP	Дата реєстрації
email_verified	BOOLEAN	Підтвердження email
is_blocked	BOOLEAN	Ознака блокування
blocked_reason	TEXT	Причина блокування
stripe_account_id	VARCHAR	Ідентифікатор Stripe

Таблиця models є центральною таблицею системи. Поле verification відображає статус модерації і може мати значення pending, approved або rejected. Поле average_rating оновлюється автоматично тригером при зміні відгуків. Детальніше наведено у таблиці 2.3.

Таблиця 2.3

Сутність models

Поле	Тип	Опис
model_id	SERIAL PRIMARY KEY	Унікальний ідентифікатор
user_id	INTEGER FK	Продавець
category_id	INTEGER FK	Категорія
license_id	INTEGER FK	Ліцензія
title	VARCHAR(255)	Назва моделі
description	TEXT	Опис
price	NUMERIC(10,2)	Ціна
verification	VARCHAR(20)	Статус модерації
average_rating	NUMERIC(3,2)	Середній рейтинг
sales_count	INTEGER	Кількість продажів
view_count	INTEGER	Кількість переглядів
upload_date	TIMESTAMP	Дата завантаження

updated_at	TIMESTAMP	Дата оновлення
status	VARCHAR(20)	Статус публікації: published / draft

Сутність orders, що наведено у таблиці 2.4, фіксує замовлення користувача. Статус замовлення відстежує його поточний стан — від створення до завершення.

Таблиця 2.4

Сутність orders

Поле	Тип	Опис
order_id	SERIAL PRIMARY KEY	Унікальний ідентифікатор
user_id	INTEGER FK	Покупець
order_date	TIMESTAMP	Дата створення
total_price	NUMERIC(10,2)	Загальна сума
order_status	VARCHAR(20)	Статус замовлення

Таблиця order_items містить позиції замовлення. Ціна фіксується на момент покупки і не залежить від подальших змін ціни моделі. Поля наведено нижче у таблиці 2.5.

Таблиця 2.5

Сутність order_items

Поле	Тип	Опис
order_item_id	SERIAL PRIMARY KEY	Унікальний ідентифікатор
order_id	INTEGER FK	Замовлення
model_id	INTEGER FK	Модель
unit_price	NUMERIC(10,2)	Ціна на момент покупки

Таблиця reviews зберігає відгуки верифікованих покупців. Право залишити відгук перевіряється на сервісному рівні — наявність відповідного запису у order_items. Адміністратор може заблокувати відгук із зазначенням причини. Склад полів наведено у таблиці 2.6

Таблиця 2.6

Сутність reviews

Поле	Тип	Опис
review_id	SERIAL PRIMARY KEY	Унікальний ідентифікатор
user_id	INTEGER FK	Автор відгуку
model_id	INTEGER FK	Модель
review_rating	INTEGER	Оцінка (1–5)
review_comment	TEXT	Текст відгуку
review_date	TIMESTAMP	Дата публікації
is_blocked	BOOLEAN	Ознака блокування
blocked_reason	TEXT	Причина блокування

Сутність comments, наведена у таблиці 2.7, реалізує відкрите обговорення під сторінкою моделі. На відміну від reviews, не містить числової оцінки і не вимагає підтвердженої покупки — доступна будь-якому зареєстрованому користувачу. Адміністратор може заблокувати коментар із зазначенням причини.

Таблиця 2.7

Сутність comments

Поле	Тип	Опис
comment_id	SERIAL PRIMARY KEY	Унікальний ідентифікатор
user_id	INTEGER FK	Автор коментаря

model_id	INTEGER FK	Модель
comment_text	TEXT	Текст коментаря
comment_date	TIMESTAMP	Дата публікації
is_blocked	BOOLEAN	Ознака блокування
blocked_reason	TEXT	Причина блокування

Таблиця cart_items зберігає поточний вміст кошика користувача. При оформленні замовлення позиції кошика переносяться до order_items і видаляються з кошика. Склад полів наведено нижче у таблиці 2.8.

Таблиця 2.8

Сутність cart_items

Поле	Тип	Опис
cart_item_id	SERIAL PRIMARY KEY	Унікальний ідентифікатор
user_id	INTEGER FK	Користувач
model_id	INTEGER FK	Модель
added_at	TIMESTAMP	Дата додавання

Таблиця seller_profiles зберігає персональні дані продавця, необхідні для KYC-верифікації та здійснення виплат. Пов'язана з users відношенням «один до одного» через унікальний зовнішній ключ. Деталі наведено у таблиці 2.9.

Таблиця 2.9

Сутність seller_profiles

Поле	Тип	Опис
profile_id	SERIAL PRIMARY KEY	Унікальний ідентифікатор
user_id	INTEGER FK UNIQUE	Продавець
first_name	VARCHAR(100)	Ім'я

last_name	VARCHAR(100)	Прізвище
date_of_birth	DATE	Дата народження
address_street	TEXT	Вулиця та будинок
address_city	VARCHAR(100)	Місто
address_zip	VARCHAR(20)	Поштовий індекс
address_country	VARCHAR(100)	Країна
kyc_status	VARCHAR(20)	Статус верифікації: pending / verified / rejected
admin_note	TEXT	Нотатка адміністратора
iban	VARCHAR(50)	IBAN рахунку NULL
bic	VARCHAR(20)	BIC/SWIFT код NULL
paypal_email	VARCHAR(255)	Email PayPal NULL
tax_id	VARCHAR(100)	Ідентифікаційний номер платника NULL

Сутність payouts, наведена у таблиці 2.10, фіксує запити продавців на виведення зароблених коштів. Статус відображає поточний стан обробки запиту.

Таблиця 2.10

Сутність payouts

Поле	Тип	Опис
payout_id	SERIAL PRIMARY KEY	Унікальний ідентифікатор
user_id	INTEGER FK	Продавець
amount	NUMERIC(12,2)	Сума виплати
status	VARCHAR(20)	Статус: pending / approved / paid / rejected
payout_method	VARCHAR(20)	Спосіб: bank / paypal
seller_note	TEXT	Коментар продавця

admin_note	TEXT	Нотатка адміністратора
created_at	TIMESTAMP	Дата запиту
processed_at	TIMESTAMP	Дата обробки

Таблиця support_tickets реалізує систему звернень до адміністрації. Поле type визначає характер звернення, а related_model_id і related_comment_id — необов'язкові посилання на об'єкт скарги. Склад полів наведено у таблиці 2.11.

Таблиця 2.11

Сутність support_tickets

Поле	Тип	Опис
ticket_id	SERIAL PRIMARY KEY	Унікальний ідентифікатор
user_id	INTEGER FK	Автор звернення
type	VARCHAR(30)	Тип: support / report_model / report_comment
ticket_subject	VARCHAR(255)	Тема
ticket_message	TEXT	Текст звернення
ticket_status	VARCHAR(20)	Статус: open / in_progress / resolved
related_model_id	INTEGER FK (nullable)	Пов'язана модель
related_comment_id	INTEGER FK (nullable)	Пов'язаний коментар
admin_note	TEXT	Відповідь адміністратора
created_at	TIMESTAMP	Дата звернення

Допоміжні таблиці categories і licenses є довідниковими — вони містять лише ідентифікатор і назву та слугують для класифікації моделей. Таблиці model_files і model_preview_images зберігають посилання на файли і зображення моделей відповідно, кожне з яких прив'язане до конкретної моделі. Таблиця

payments пов'язана з конкретним замовленням і зберігає технічну інформацію про платіжну транзакцію: ідентифікатор Stripe, суму та статус оплати.

Усі зовнішні ключі реалізовані з обмеженням цілісності — видалення пов'язаних записів контролюється на рівні бази даних. Це унеможливорює появу «осиротілих» записів при видаленні моделі або користувача.

2.5 Розробка SQL-запитів, тригерів та процедур

Окрім базової структури таблиць, у системі реалізовано додаткову логіку на рівні бази даних — тригери, що автоматично підтримують актуальність службових полів без участі серверного коду.

Тригери. Перший тригер - оновлення дати зміни моделі - відстежує будь-яке оновлення запису в таблиці models і автоматично записує поточний час у поле updated_at. Це усуває необхідність явно передавати цю дату з серверного коду при кожній операції редагування.

Тригер спрацьовує до збереження змін (BEFORE UPDATE), тому нове значення updated_at записується в рамках тієї ж транзакції що і саме оновлення. Лістинг тригера наведено на рисунку 2.2.

```

> CREATE OR REPLACE FUNCTION fn_set_model_updated_at()
  RETURNS TRIGGER AS $$
  BEGIN
    NEW.updated_at = NOW();
    RETURN NEW;
  END;
  $$ LANGUAGE plpgsql;

> CREATE TRIGGER trg_model_updated_at
  BEFORE UPDATE ON models
  FOR EACH ROW
  EXECUTE FUNCTION fn_set_model_updated_at();

```

Рис. 2.2 Тригер автоматичного оновлення поля updated_at

Другий тригер - перерахунок середнього рейтингу - підтримує актуальне значення поля average_rating у таблиці models. Він спрацьовує після будь-якої зміни у таблиці reviews — додавання, оновлення або видалення відгуку — і

одразу перераховує середнє значення оцінок для відповідної моделі. Лістинг наведено на рисунку 2.3.

```

CREATE OR REPLACE FUNCTION fn_recalc_model_avg_rating()
RETURNS TRIGGER AS $$
DECLARE
    target_model_id INTEGER;
BEGIN
    target_model_id := COALESCE(NEW.model_id, OLD.model_id);
    UPDATE models
    SET average_rating = (
        SELECT COALESCE(AVG(review_rating), 0)
        FROM reviews
        WHERE model_id = target_model_id
    )
    WHERE model_id = target_model_id;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_review_rating
AFTER INSERT OR UPDATE OR DELETE ON reviews
FOR EACH ROW
EXECUTE FUNCTION fn_recalc_model_avg_rating();

```

Рис. 2.3 Тригер перерахунку середнього рейтингу моделі

Використання COALESCE гарантує що при відсутності відгуків поле отримає значення 0, а не NULL. Змінна target_model_id визначається через COALESCE(NEW.model_id, OLD.model_id) — це необхідно для коректної обробки операції видалення, коли NEW є порожнім.

Типові SQL-запити. Окрім тригерів, у системі активно використовуються параметризовані запити для вибірки даних. Зокрема, фільтрація каталогу моделей, лістинг якої наведено на рисунку 2.4, підтримує одночасний пошук за назвою, категорією та діапазоном цін:

```

SELECT m.*, u.username AS seller_name
FROM models m
JOIN users u ON m.user_id = u.user_id
WHERE m.verification != 'rejected'
    AND m.status = 'published'
    AND (m.title ILIKE :search OR :search IS NULL)
    AND (m.category_id = :category OR :category IS NULL)
    AND (m.price BETWEEN :min_price AND :max_price)
ORDER BY m.upload_date DESC;

```

Рис. 2.4 Запит фільтрації каталогу моделей

Умова `verification != 'rejected'` дозволяє відображати моделі зі статусом `pending` — покупці можуть бачити модель поки вона проходить перевірку адміністратором. Відхилені моделі приховані повністю.

Запит для отримання історії покупок користувача об'єднує кілька таблиць і повертає повну інформацію про придбані моделі. Лістинг наведено на рисунку 2.5.

```
SELECT m.title, m.file_url, oi.unit_price, o.order_date
FROM order_items oi
JOIN orders o ON oi.order_id = o.order_id
JOIN models m ON oi.model_id = m.model_id
WHERE o.user_id = :user_id
      AND o.order_status = 'completed'
ORDER BY o.order_date DESC;
```

Рис. 2.5 Запит отримання історії покупок користувача

Перед збереженням відгуку система перевіряє, чи справді користувач придбав модель. Для цього виконується окремий запит до таблиці `order_items`, наведений на рисунку 2.6.

```
SELECT COUNT(*) > 0 AS has_purchased
FROM order_items oi
JOIN orders o ON oi.order_id = o.order_id
WHERE o.user_id = :user_id
      AND oi.model_id = :model_id
      AND o.order_status = 'completed';
```

Рис. 2.6 Запит перевірки права користувача на відгук

Якщо користувач не має жодного завершеного замовлення з цією моделлю, сервіс повертає помилку до звернення до бази даних — це унеможливорює залишення відгуку без підтвердженої покупки.

Наведені тригери і запити реалізують частину бізнес-логіки безпосередньо на рівні бази даних, що зменшує навантаження на серверний код і гарантує цілісність даних незалежно від того, через який інтерфейс вони змінюються.

2.6 Організація доступу та захисту даних

Захист даних у системі реалізований на кількох рівнях — від зберігання паролів до розмежування прав доступу між різними категоріями користувачів.

Автентифікація побудована на основі токенів. Після успішного входу користувач отримує підписаний токен, який передається з кожним наступним запитом до захищених ресурсів. Сервер перевіряє дійсність токена і на основі цього визначає особу користувача та його права. Токен має обмежений термін дії, після якого користувач має пройти автентифікацію повторно. Такий підхід не вимагає зберігання сесій на сервері і добре підходить для клієнт-серверної архітектури.

Паролі зберігаються виключно у захешованому вигляді — відкритий пароль ніде не зберігається і не передається. При вході в систему введений пароль хешується і порівнюється зі збереженим значенням. Навіть у разі витоку бази даних отримати оригінальні паролі з хешів практично неможливо.

Розмежування доступу реалізовано через рольову модель. Кожен користувач має одну з трьох ролей — покупець, продавець або адміністратор.

Роль визначає які ресурси і операції доступні конкретному користувачу. Захищені маршрути перевіряють роль перш ніж виконати запит — наприклад, завантаження моделі доступне лише продавцям, а панель модерації лише адміністраторам. Додатково адміністратор може заблокувати обліковий запис користувача, що позбавляє його доступу до системи незалежно від наявності дійсного токена, а також заблокувати окремі відгуки чи коментарі — для модерації контенту без повного обмеження доступу.

Платіжні дані не обробляються і не зберігаються на стороні платформи. Вся відповідальність за обробку карткових даних делегована зовнішньому платіжному сервісу. Платформа отримує лише підтвердження успішності або невдачі транзакції — без жодної чутливої інформації про картку чи банківський рахунок.

Міжсайтові запити контролюються через налаштування CORS — система приймає запити лише з дозволених адрес, що унеможливорює несанкціоновані звернення до API з сторонніх ресурсів.

Сукупність цих заходів забезпечує базовий рівень захисту як для користувачів платформи, так і для даних що в ній зберігаються. Конкретна реалізація кожного з механізмів детально розглядається у третьому розділі.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ВЕБ-ПЛАТФОРМИ

3.1 Організаційна структура програмного забезпечення

Система побудована за архітектурою клієнт-сервер із чітким розділенням на дві незалежні частини — серверну і клієнтську. Вони взаємодіють виключно через REST API: клієнт надсилає HTTP-запити, сервер повертає відповіді у форматі JSON. Такий підхід дозволяє розробляти і розгортати обидві частини незалежно одна від одної.

Серверна частина організована за шаровою архітектурою і містить чотири основні шари.

Шар контролерів (Controller) обробляє вхідні HTTP-запити, валідує параметри і делегує виконання логіки до відповідних сервісів. У системі реалізовано чотирнадцять контролерів: AuthController, ModelController, CategoryController, LicenseController, CartController, OrderController, PaymentController, ReviewController, CommentController, UserController, SellerProfileController, PayoutRequestController, AdminController та SupportTicketController.

Шар сервісів (Service) містить бізнес-логіку системи і не залежить від HTTP-контексту. Кожен сервіс відповідає за окрему предметну область: AuthService — автентифікація та управління обліковими записами, ModelService — робота з моделями, OrderService — створення та обробка замовлень, PaymentService — взаємодія з платіжним шлюзом, ReviewService — відгуки верифікованих покупців, CommentService — публікація та модерація коментарів під моделями, FileUploadService — завантаження файлів.

Шар сутностей (Entity) містить класи що відображають таблиці бази даних. Кожна сутність відповідає одній таблиці і описує її поля та зв'язки.

Шар репозиторіїв (Repository) забезпечує доступ до даних через Doctrine ORM — інструмент що дозволяє працювати з базою даних через об'єкти PHP замість написання SQL вручну. Кожен репозиторій відповідає за одну сутність і містить методи вибірки, збереження та видалення записів. Міграції Doctrine забезпечують версійність схеми бази даних — кожна зміна структури таблиць фіксується у окремому файлі міграції і може бути відтворена на будь-якому середовищі.

Окремо виділено модуль Security що реалізує рольову перевірку доступу через клас RoleManager. Загальну структуру пакетів серверної та клієнтської частин наведено на рисунку 3.1.

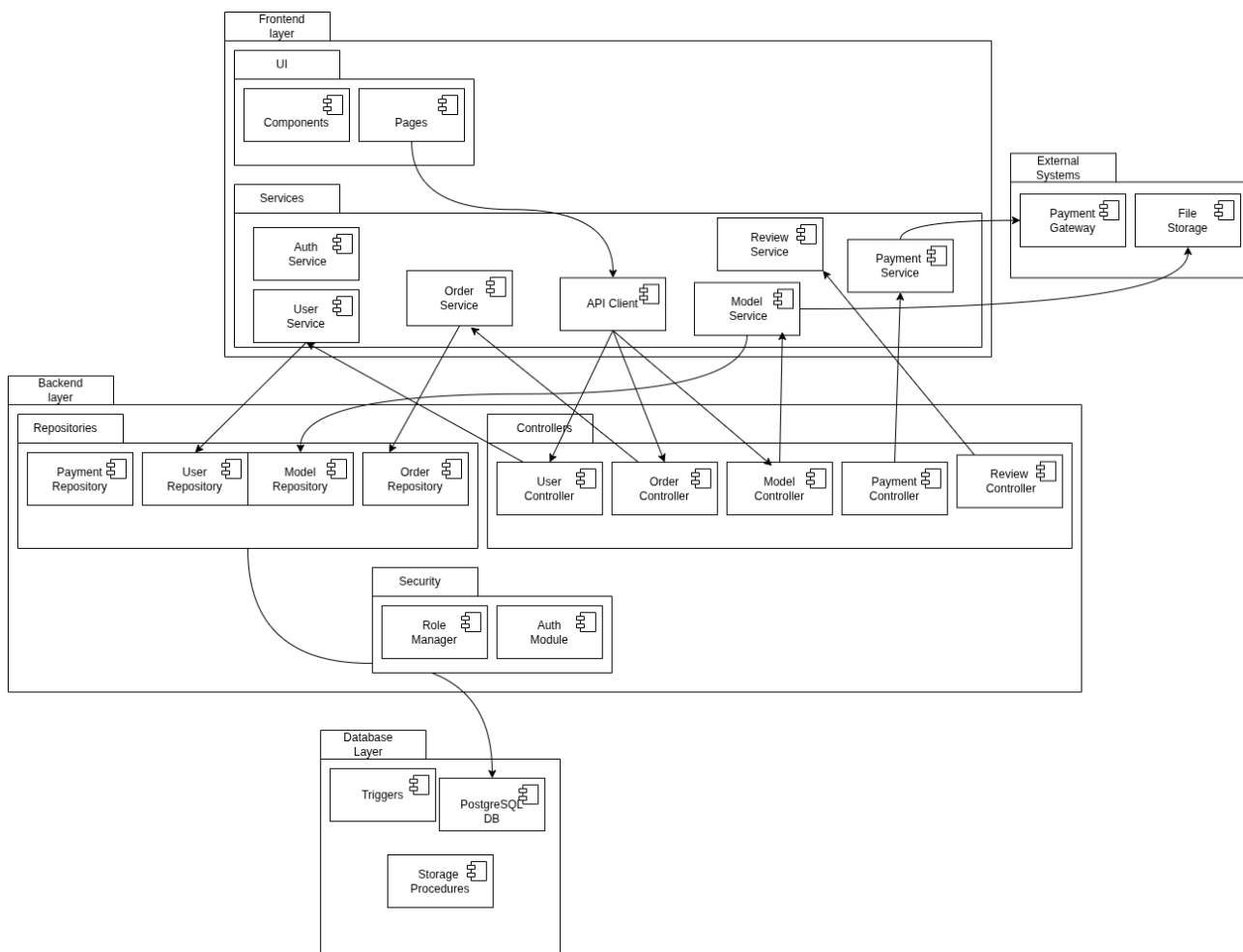


Рис. 3.1 Діаграма пакетів системи

Для наочності у таблиці 3.1 наведено перелік контролерів серверної частини та їхні зони відповідальності.

Таблиця 3.1

Контролери серверної частини

Контролер	Маршрут	Відповідальність
AuthController	/api/auth	Реєстрація, вхід, управління профілем
ModelController	/api/models	Каталог, завантаження, редагування моделей
CategoryController	/api/categories	Перелік категорій
LicenseController	/api/licenses	Перелік ліцензій
CartController	/api/cart	Управління кошиком
OrderController	/api/orders	Створення та перегляд замовлень
PaymentController	/api/payments	Обробка платежів через Stripe
ReviewController	/api/models/{id}/reviews	Відгуки верифікованих покупців
CommentController	/api/models/{id}/comments	Коментарі під моделями
UserController	/api/users	Публічні профілі користувачів
AdminController	/api/admin	Модерація контенту, управління користувачами
SupportTicketController	/api/support	Звернення до служби підтримки
PayoutRequestController	/api/seller/payouts	Запити на виплату коштів продавця
SellerProfileController	/api/seller/profile	Управління персональним профілем продавця (KYC)

Клієнтська частина організована як односторінковий застосунок і має таку структуру:

- `pages` — сторінки застосунку, кожна відповідає окремому маршруту
- `components` — перевикористовувані компоненти інтерфейсу
- `contexts` — глобальний стан застосунку (автентифікація, кошик)
- `services` — функції для звернення до API
- `hooks` — користувацькі React-хуки

Для розуміння взаємодії між шарами розглянемо типовий сценарій — покупець додає модель до кошика. Клієнт надсилає POST-запит із токеном автентифікації. `CartController` отримує запит, перевіряє наявність токена і передає ідентифікатор моделі до сервісного шару. Сервіс звертається до репозиторію щоб перевірити чи модель існує і чи вона вже не в кошику. Репозиторій виконує запит до бази даних і повертає результат. Сервіс створює новий запис у таблиці `cart_items`, репозиторій зберігає його, і контролер повертає клієнту підтвердження у форматі JSON.

Окрім внутрішніх компонентів система інтегрована з двома зовнішніми сервісами. Платіжний шлюз обробляє фінансові транзакції — серверна частина створює платіжний намір і отримує підтвердження, не торкаючись карткових даних. Файли 3D-моделей та зображення зберігаються локально на сервері у папці `public/uploads/` через сервіс `FileUploadService`.

Конфігурація серверної частини винесена у окремі файли — налаштування безпеки, маршрутизації та сервісів зберігаються у папці `config/`. Змінні середовища — ключі зовнішніх сервісів, параметри підключення до бази даних — винесені у `.env` файли і не потрапляють до репозиторію. Структура проєкту також включає папку `migrations/` з версійованими міграціями бази даних та папку `tests/` з модульними тестами.

3.2 Вибір технологій та інструментів розробки

Вибір технологій для кожної частини системи визначався кількома критеріями: зрілість інструменту, активна підтримка спільноти, сумісність між компонентами та відповідність технічним вимогам проєкту.

Серверна частина. Основою серверної частини є PHP 8.2 у поєднанні з фреймворком Symfony 7. Symfony є одним із найзріліших PHP-фреймворків з чіткою архітектурою, розвиненою системою компонентів і активною підтримкою. Він забезпечує структуру для побудови REST API, управління залежностями через сервіс-контейнер і гнучку систему конфігурації. Для роботи з базою даних використовується Doctrine ORM — стандартний інструмент в екосистемі Symfony, що реалізує патерн репозиторій і підтримує версійовані міграції схеми.

Автентифікація реалізована через LexikJWTAuthenticationBundle — бібліотеку що забезпечує генерацію і перевірку JWT-токенів у Symfony-додатках. Для автентифікації через Google-акаунт підключено офіційну бібліотеку google/apiclient, яка дозволяє перевірити автентичність токена отриманого від клієнта і отримати дані профілю користувача без прямого доступу до його пароля. Для обробки міжсайтових запитів підключено NelmioCorsBundle, що дозволяє налаштувати політику CORS між серверною і клієнтською частинами.

Інтеграція з платіжним шлюзом реалізована через офіційну бібліотеку stripe-php. Хешування паролів виконується вбудованим компонентом Symfony PasswordHasher.

Клієнтська частина. Клієнтський застосунок побудований на базі React 19 — бібліотеки для побудови користувацьких інтерфейсів із компонентним підходом. React забезпечує реактивне оновлення інтерфейсу при зміні стану і має широку екосистему готових рішень.

Як мова програмування використовується TypeScript — розширення JavaScript із статичною типізацією, що зменшує кількість помилок і спрощує підтримку коду.

Збірка і локальний сервер розробки забезпечуються інструментом Vite, що відрізняється високою швидкістю збірки порівняно з альтернативами.

Маршрутизація між сторінками реалізована через React Router v7. Стилiзація виконується за допомогою Tailwind CSS — утилітарного CSS-фреймворку що дозволяє будувати інтерфейс без написання окремих CSS-файлів. Компоненти інтерфейсу засновані на бібліотеці Radix UI, що забезпечує доступні і гнучко налаштовувані елементи.

Для відображення 3D-моделей безпосередньо у браузері використовується Three.js разом із бібліотекою @react-three/fiber — React-обгорткою над Three.js.

Інструменти розробки та тестування. Серверні тести написані з використанням PHPUnit 10 — стандартного фреймворку для модульного тестування PHP-коду. Клієнтські тести реалізовані за допомогою Vitest і @testing-library/react.

Керування залежностями серверної частини здійснюється через Composer, клієнтської — через npm.

У таблиці 3.2 наведено узагальнений перелік використаних технологій.

Таблиця 3.2

Технологічний стек системи

Категорія	Технологія	Призначення
Серверна мова	PHP 8.2	Серверна логіка
Серверний фреймворк	Symfony 7	REST API, DI-контейнер
ORM	Doctrine	Робота з базою даних
Автентифікація	LexikJWT	JWT-токени

Google OAuth	google/apiclient	Вхід через Google-акаунт
Платежі	Stripe	Обробка транзакцій
СУБД	PostgreSQL	Зберігання даних
Клієнтська бібліотека	React 19	Інтерфейс користувача
Мова	TypeScript	Типізований JavaScript
Збірник	Vite	Збірка і dev-сервер
Стилізація	Tailwind CSS	Оформлення інтерфейсу
Компоненти UI	Radix UI	Готові UI-елементи
3D-перегляд	Three.js / @react-three/fiber	Відображення моделей
Тести бекенд	PHPUnit	Модульне тестування
Тести фронтенд	Vitest	Тестування компонентів

3.3 Розробка серверної частини системи

Серверна частина реалізована як REST API на основі Symfony 7 і обробляє всі запити від клієнтського застосунку. Кожен маршрут повертає відповідь у форматі JSON, а доступ до захищених ресурсів перевіряється через JWT-токен.

Організація маршрутизації. Маршрути визначаються безпосередньо в контролерах за допомогою PHP-атрибутів. Кожен контролер має базовий префікс маршруту, а окремі методи уточнюють шлях і HTTP-метод. Лістинг наведено на рисунку 3.2.

```
#[Route('/api/auth')]
class AuthController extends AbstractController
{
    #[Route('/register', methods: ['POST'])]
    public function register(Request $request): JsonResponse { ... }

    #[Route('/login', methods: ['POST'])]
    public function login(Request $request): JsonResponse { ... }
}
```

Рис. 3.2 Визначення маршрутів через PHP-атрибути

Такий підхід дозволяє бачити маршрут і логіку обробки в одному місці без потреби у окремих файлах конфігурації.

Автентифікація та безпека. Автентифікація побудована на JWT-токенах. Після успішного входу або реєстрації сервер генерує підписаний токен і повертає його клієнту. Клієнт зберігає токен і передає його у заголовок кожного наступного запиту. Сервер перевіряє підпис токена і визначає користувача без звернення до бази даних.

Розмежування доступу реалізовано на двох рівнях. Перший — конфігурація `security.yaml`, яка визначає які маршрути є публічними, а які вимагають автентифікації або конкретної ролі. Другий — клас `RoleManager` у сервісному шарі, що надає методи перевірки ролі безпосередньо в коді. На рисунку 3.3 наведено лістинг описаного методу.

```
public function requireRole(User $user, string ...$roles): void
{
    if (!in_array($user->getRole(), $roles, true)) {
        throw new AccessDeniedHttpException('Access denied.');
```

Рис. 3.3 Метод перевірки ролі у класі `RoleManager`

Така дворівнева перевірка гарантує захист навіть якщо конфігурація маршрутів містить помилку — контролер додатково перевіряє права перед виконанням операції.

Шаровий розподіл логіки. Контролер не містить бізнес-логіки — він лише отримує дані з запиту, передає їх до сервісу і повертає результат. Вся логіка зосереджена у сервісному шарі. Наприклад, при отриманні сторінки моделі `ModelService` перевіряє статус верифікації і чи має поточний користувач право її бачити. Лістинг котролера наведено на рисунку 3.4.

```

if ($model->getVerification() === 'rejected' || $model->getStatus() !== 'published') {
    if (!$currentUser) {
        throw new NotFoundException();
    }
    if (!$this->roleManager->isAdmin($currentUser)
        && $model->getUser()->getId() !== $currentUser->getId()) {
        throw new NotFoundException();
    }
}
}

```

Рис. 3.4 Перевірка прав доступу до моделі у ModelService

Умова перевіряє два випадки: модель відхилена адміністратором або знаходиться у статусі чернетки. Моделі зі статусом pending відображаються публічно — покупці можуть переглянути сторінку моделі поки вона очікує на верифікацію. Власник і адміністратор мають доступ до моделі за будь-якого статусу.

Робота з базою даних. Doctrine ORM забезпечує взаємодію з PostgreSQL через об'єкти PHP. Кожна таблиця бази даних відповідає класу-сутності, а репозиторій містить методи вибірки. Зміни у схемі бази даних вносяться через міграції — кожна міграція є окремим PHP-класом із методами `up()` і `down()` для застосування і відкату змін.

Обробка помилок. Symfony автоматично перетворює HTTP-виключення на відповіді з відповідним статус-кодом. Якщо сервіс кидає `NotFoundException` — клієнт отримує відповідь зі статусом 404, `AccessDeniedHttpException` — 403, `BadRequestHttpException` — 400. Це дозволяє обробляти помилки в сервісному шарі без прямого маніпулювання HTTP-відповідями.

3.4 Розробка клієнтської частини та користувацького інтерфейсу системи

Клієнтська частина реалізована як односторінковий застосунок на основі React 19. Усі переходи між сторінками відбуваються без перезавантаження

браузера — React Router перехоплює навігацію і підмінює вміст сторінки, зберігаючи стан застосунку.

Маршрутизація. Усі маршрути визначені у кореневому компоненті `App.tsx`. Кожен маршрут відповідає окремому компоненту-сторінці. Система має як публічні маршрути — каталог, сторінка моделі, профіль продавця — так і захищені, доступ до яких перевіряється на рівні компонента. Наприклад, сторінка адмін-панелі при завантаженні перевіряє роль поточного користувача і перенаправляє на головну якщо роль не відповідає.

Глобальний стан. Стан застосунку що має бути доступний у різних частинах інтерфейсу зберігається через React Context. У системі реалізовано два контексти.

`AuthContext` зберігає інформацію про поточного користувача і надає методи для входу, реєстрації, виходу та оновлення профілю. Після успішної автентифікації токен зберігається у `localStorage` і використовується при кожному наступному запиті до API. При завантаженні застосунку контекст перевіряє наявність збереженого токена і автоматично відновлює сесію.

`CartContext` відповідає за стан кошика — перелік доданих моделей, загальну суму і кількість позицій. Кошик синхронізується з сервером при кожній зміні і оновлюється автоматично при вході або виході користувача.

Компоненти інтерфейсу. Інтерфейс побудований з перевикористовуваних компонентів. Базові елементи — кнопки, діалоги, меню, поля введення — засновані на бібліотеці Radix UI і стилізовані через Tailwind CSS. Складніші компоненти реалізовані самостійно:

- `Navbar` — навігаційна панель із меню категорій, пошуком і кошиком; для адміністратора відображає посилання на панель керування;
- `ModelCard` — картка моделі в каталозі з галереєю зображень і інформацією про ціну;

- Model3DViewer — компонент перегляду 3D-моделі безпосередньо у браузері на основі Three.js;
- ModelUpload — форма завантаження моделі з підтримкою перетягування файлів;
- CheckoutModal — модальне вікно оформлення замовлення з інтеграцією платіжної форми.

Сторінка ModelDetail містить два окремі блоки взаємодії. Блок відгуків доступний лише верифікованим покупцям — система перевіряє наявність підтвердженої покупки перед відображенням форми. Блок коментарів відкритий для будь-якого авторизованого користувача і призначений для запитань та обговорення моделі без прив'язки до факту покупки.

Взаємодія з API. Усі звернення до серверного API проходять через єдину функцію apiCall у модулі services/api.js. Вона автоматично додає токен автентифікації до заголовків запиту, обробляє помилки і перенаправляє на сторінку входу при отриманні відповіді 401. Це централізує логіку роботи з API і позбавляє кожен компонент від необхідності самостійно керувати токеном.

Сторінки застосунку. Застосунок містить двадцять шість сторінок що охоплюють усі сценарії використання: Home, Marketplace, ModelDetail, Login, Signup, UploadPage, ModelEdit, UserProfile, MyModels, MyPurchases, SalesOverview, CartPage, PurchaseSuccess, EarningsPage, PublicProfile, LearnMore, SellerOnboarding, SupportTicketPage, AdminPanel та сторінки правової інформації — Impressum, Datenschutz, AGB, Lizenzen — і підтримки — HilfeCenter, Kontakt, FAQ. Кожна сторінка є окремим компонентом що самостійно завантажує необхідні дані при монтуванні.

Адмін-панель організована у п'ять вкладок: Models — перегляд і верифікація завантажених моделей, Users — управління обліковими записами і ролями, Tickets — перегляд та обробка звернень користувачів до підтримки, Payouts — опрацювання запитів на виплату коштів продавцям, KYC —

верифікація персональних даних продавців. Кожна вкладка підтримує пагінацію і редагування записів безпосередньо з панелі.

3.5 Реалізація модулів системи

Модуль автентифікації. Система підтримує два способи входу — через email і пароль та через обліковий запис Google.

При стандартній реєстрації AuthService перевіряє унікальність email і імені користувача, хешує пароль і зберігає нового користувача з роллю buyer. У відповідь повертається JWT-токен і дані користувача. При вході система знаходить користувача за email, перевіряє пароль і додатково перевіряє чи не заблокований обліковий запис — заблокований користувач отримує відповідь з кодом помилки USER_BLOCKED.

Вхід через Google реалізований через OAuth 2.0. Клієнт отримує токен доступу від Google і передає його на сервер. Сервер звертається до Google API щоб отримати дані користувача — email, ім'я, фото. Якщо користувач з таким email вже існує — він входить у систему, якщо ні — автоматично створюється новий обліковий запис.

Модуль оплати. Обробка платежів реалізована через Stripe PaymentIntent API. Процес складається з кількох кроків. Спочатку сервер створює замовлення і додає до нього обрані моделі, фіксуючи ціну кожної на момент покупки. Потім створюється PaymentIntent у Stripe із сумою у центах і метаданими замовлення. Лістинг коду показано на рисунку 3.5.

```
$intent = PaymentIntent::create([
    'amount' => $totalCents,
    'currency' => 'eur',
    'metadata' => ['order_id' => $order->getId()],
]);
```

Рис. 3.5 Створення PaymentIntent через Stripe API

Сервер повертає клієнту `clientSecret` — унікальний ключ для підтвердження платежу. Клієнт передає його до форми Stripe Elements, яка безпечно збирає карткові дані і підтверджує платіж безпосередньо з браузера без участі сервера платформи. Після успішної оплати клієнт надсилає підтвердження на сервер, який перевіряє статус `PaymentIntent` у Stripe і лише тоді переводить замовлення у статус `completed`.

Додатково реалізовано Stripe Webhook — сервер приймає і перевіряє підписані події від Stripe для надійного підтвердження платежів навіть у разі збою на стороні клієнта.

Модуль завантаження файлів. `FileUploadService` обробляє завантаження трьох типів файлів: 3D-моделей, зображень попереднього перегляду та аватарів. Для кожного типу визначено допустимі формати і максимальний розмір:

- 3D-моделі: GLB, GLTF, OBJ, FBX, STL, BLEND, DAE, 3DS та інші — до 200 МБ;
- Зображення попереднього перегляду: JPG, PNG, WEBP — до 10 МБ;
- Аватари: ті ж формати що і зображення.

Файл перевіряється за розширенням і розміром до збереження. У разі порушення вимог сервіс кидає `BadRequestHttpException` із описом помилки. Прийнятий файл отримує унікальне ім'я і зберігається у відповідній папці на сервері.

Модуль перегляду 3D-моделей. Перегляд тривимірних моделей безпосередньо у браузері реалізований через компонент `Model3DViewer` на основі `Three.js` і `@react-three/fiber`. Компонент завантажує файл моделі за URL, відображає її у WebGL-сцені з освітленням і дозволяє користувачу обертати та масштабувати модель мишею. Підтримуються формати GLB і GLTF — найпоширеніші формати для веб-відображення.

Модуль відгуків та коментарів. Система розрізняє два типи текстової взаємодії користувача з моделлю, кожен з яких обслуговується окремим сервісом.

ReviewService відповідає за відгуки з числовою оцінкою. Перед збереженням відгуку сервіс перевіряє наявність підтвердженої покупки — запис у таблиці order_items із відповідним model_id і статусом completed у пов'язаному замовленні. Якщо покупки немає — сервіс повертає помилку без звернення до бази даних. Після збереження або видалення відгуку тригер бази даних автоматично перераховує середній рейтинг моделі.

CommentService відповідає за текстові коментарі без оцінки. Коментар може залишити будь-який авторизований користувач незалежно від факту покупки. Власник коментаря може видалити його самостійно — сервіс перевіряє відповідність ідентифікатора автора перед видаленням.

Модуль модерації. Адміністративна панель надає інструменти модерації трьох типів контенту:

- Верифікація моделей: адміністратор переглядає завантажені моделі зі статусом pending і підтверджує або відхиляє кожен. Моделі зі статусом pending залишаються видимими у каталозі — повністю приховуються лише відхилені моделі зі статусом rejected. При редагуванні продавцем статус моделі автоматично скидається до pending і модель знову проходить перевірку.
- Управління користувачами: адміністратор може змінити роль користувача або заблокувати його обліковий запис із зазначенням причини. Заблокований користувач не може увійти в систему навіть з правильним паролем.
- Підтримка та скарги: адміністратор переглядає звернення користувачів, змінює статус тікетів (open → in_progress → resolved) і залишає відповідь у полі adminNote.

- **Виплати:** адміністратор переглядає запити продавців на виведення коштів, підтверджує, позначає як виплачені або відхиляє. При відхиленні сума автоматично повертається на баланс продавця.
- **KYC:** адміністратор переглядає персональні дані продавців, встановлює статус верифікації (`pending` / `verified` / `rejected`) і залишає коментар.

Модуль профілю продавця. Перед тим як подати запит на виплату, продавець заповнює персональний профіль — ім'я, прізвище, дату народження та адресу. Ці дані необхідні для KYC-верифікації і зберігаються у відношенні «один до одного» з обліковим записом користувача.

`SellerProfileController` надає маршрути для перегляду і збереження профілю (`GET` і `POST /api/seller/profile`). Адміністратор переглядає профілі всіх продавців і встановлює статус верифікації — `pending`, `verified` або `rejected` — через окремий маршрут в `AdminController`.

Модуль виплат. `PayoutRequestController` дозволяє продавцю подати запит на виведення зароблених коштів. При створенні запиту сервіс обчислює доступний баланс — різницю між загальними надходженнями від продажів і сумою вже виведених або очікуваних коштів — і повертає помилку якщо запитана сума перевищує залишок. Статус запиту проходить через стани `pending` → `approved` → `paid` або `rejected`. При відхиленні адміністратором сума автоматично повертається на баланс продавця — це відбувається безпосередньо у методі `updatePayout` `AdminController` без окремої задачі чи черги.

Модуль підтримки. `SupportTicketController` дозволяє будь-якому авторизованому користувачу створити звернення до адміністрації. Тікет має тип — загальна підтримка (`support`), скарга на модель (`report_model`) або скарга на коментар (`report_comment`) — і опціонально містить посилання на пов'язаний об'єкт. Користувач бачить лише власні тікети; адміністратор через `AdminController` отримує доступ до всіх звернень з фільтрацією за статусом.

Відповідь адміністратора зберігається у полі `adminNote`, статус тикета змінюється з `open` через `in_progress` до `resolved`.

3.6 Алгоритмізація та програмування програмних модулів

Алгоритм пошуку та фільтрації каталогу. Фільтрація моделей у каталозі реалізована через динамічно побудований запит у `ModelRepository`. Базовий запит вибирає моделі зі статусом `published` що не є відхиленими — умова `verification != 'rejected'` означає що моделі зі статусом `pending` також відображаються у каталозі і приховуються лише після явного відхилення адміністратором. До базового запиту послідовно додаються умови залежно від переданих параметрів — кожен параметр є необов'язковим і додається лише якщо він присутній у запиті. Пошук за текстом перевіряє одночасно назву, опис і теги моделі. Сортювання є динамічним — поле і напрямок передаються як параметри, але перевіряються на відповідність дозволеному переліку щоб унеможливити SQL-ін'єкцію. Результати повертаються посторінково через `limit` і `offset`.

Блок-схему алгоритму фільтрації наведено на рисунку 3.6.

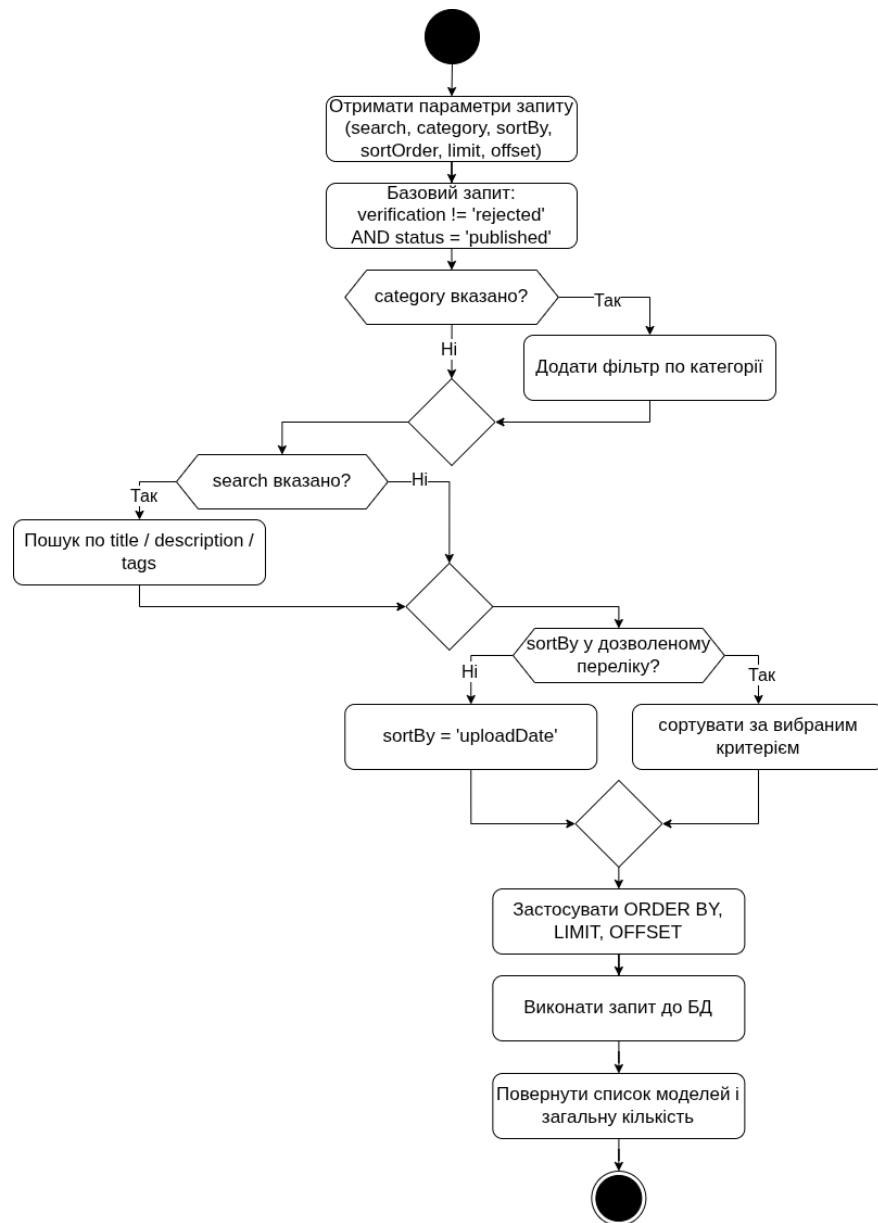


Рис. 3.6 Блок-схема алгоритму пошуку та фільтрації

Алгоритм створення замовлення та оплати. Процес покупки складається з чітко визначених кроків із перевітками на кожному етапі. При додаванні моделі до замовлення OrderService послідовно перевіряє існування замовлення і належність його поточному користувачу, статус замовлення, наявність і статус моделі, чи не намагається продавець купити власну модель, і чи модель вже не додана до замовлення. Лише після проходження всіх перевірок створюється позиція із зафіксованою ціною і загальна сума перераховується.

Після підтвердження оплати сервер перевіряє статус у Stripe і реалізує ідемпотентність — повторне підтвердження одного платежу блокується.

Блок-схему алгоритму оплати наведено на рисунку 3.7.

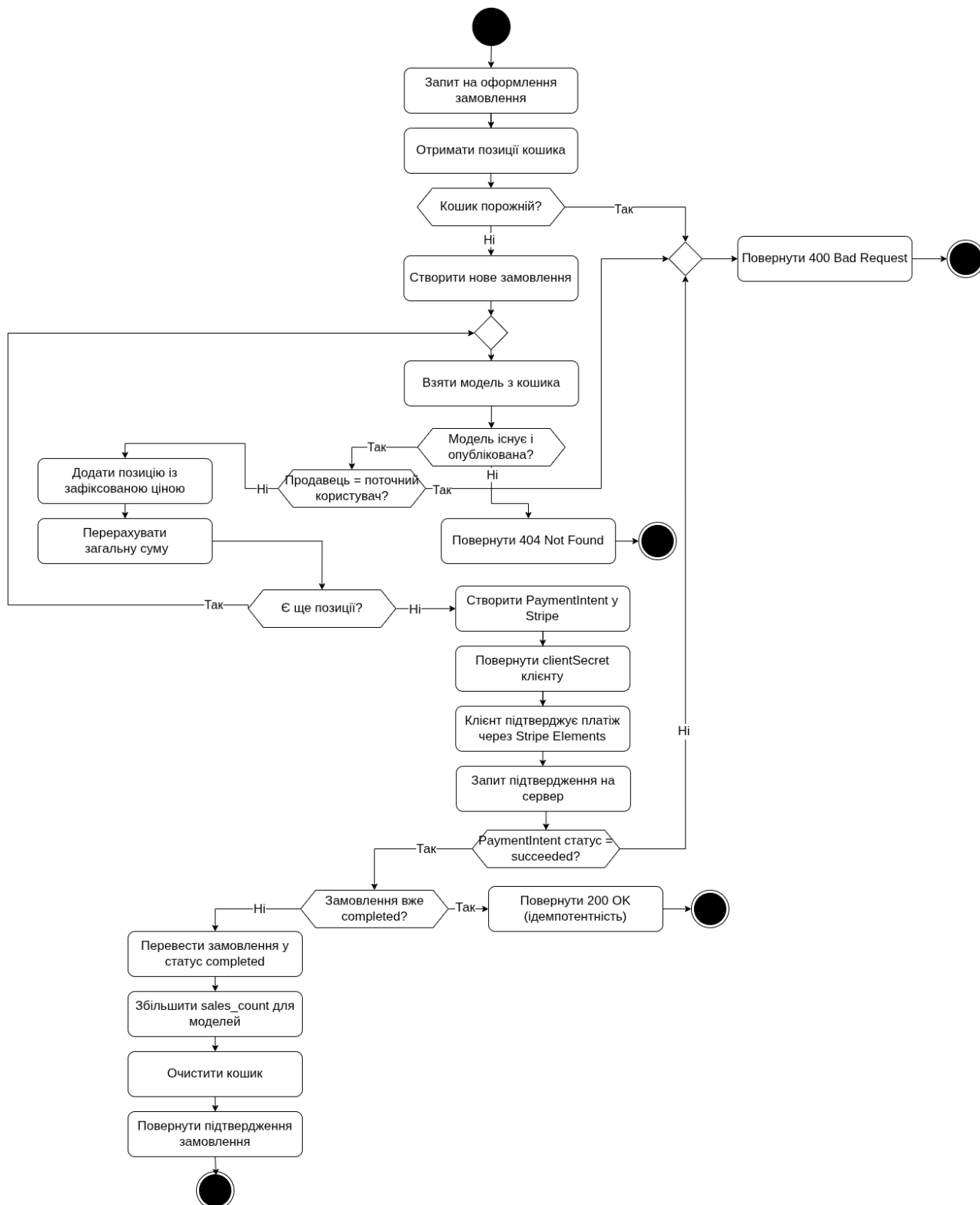


Рис. 3.7 Блок-схема алгоритму створення замовлення та оплати

Алгоритм визначення видимості моделі. При запиті сторінки моделі ModelService застосовує алгоритм визначення видимості. Якщо модель відхилена адміністратором або знаходиться у статусі чернетки — перевіряється хто робить запит: анонімний користувач отримує 404, авторизований користувач що не є власником і не є адміністратором також отримує 404. Власник і адміністратор бачать модель незалежно від статусу. Модель зі статусом pending доступна публічно — покупці можуть переглянути її сторінку поки вона очікує на верифікацію. Додатково перевіряється чи придбав користувач цю модель — результат впливає на відображення кнопки завантаження і форми відгуку.

Блок-схему алгоритму наведено на рисунку 3.8.

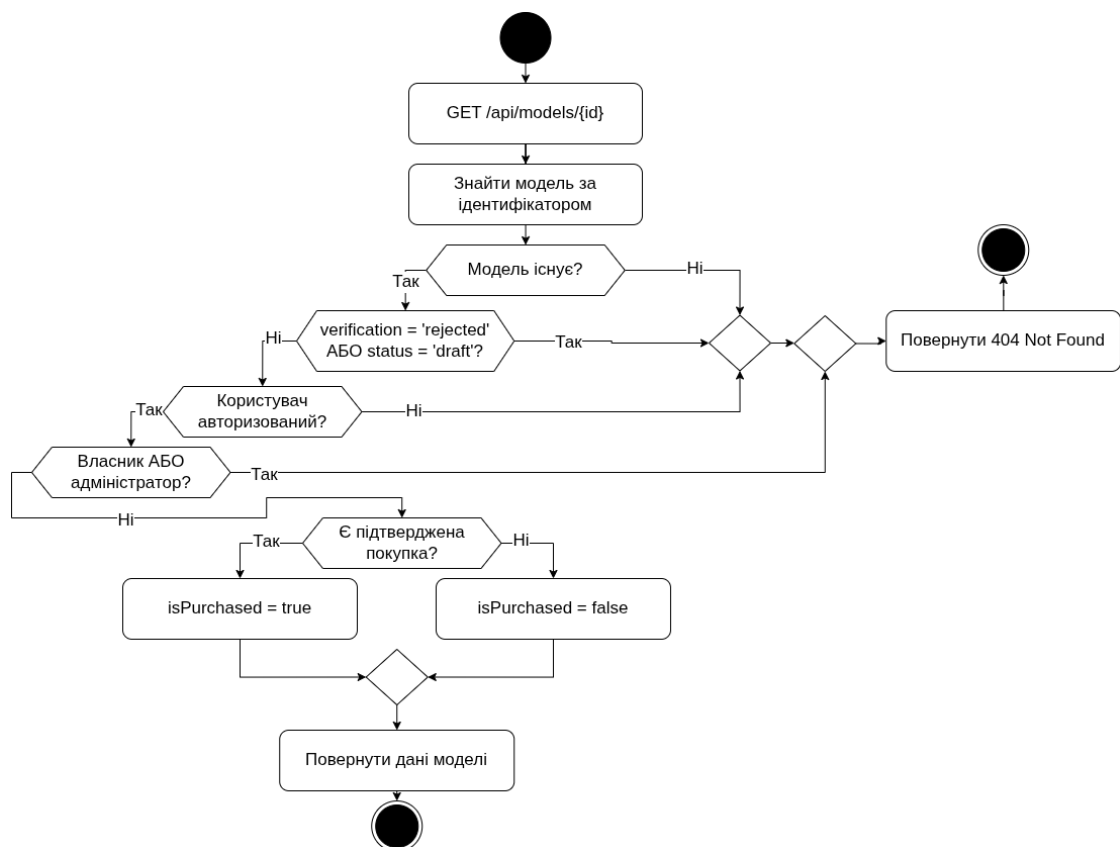


Рис. 3.8 Блок-схема алгоритму визначення видимості моделі

Алгоритм публікації відгуку. Перед збереженням відгуку ReviewService виконує послідовну перевірку умов. Спочатку перевіряється існування моделі —

якщо модель не знайдена повертається 404. Далі система перевіряє чи є у користувача підтверджена покупка цієї моделі — запит до таблиці `order_items` звіряє наявність запису із відповідним `model_id` і статусом `completed` у пов'язаному замовленні. Якщо покупки немає — повертається помилка без збереження запису. Якщо перевірка пройдена — відгук зберігається, а тригер бази даних автоматично перераховує середній рейтинг моделі. Такий підхід гарантує що оцінку може залишити лише користувач який реально придбав модель.

Блок-схему алгоритму наведено на рисунку 3.9.

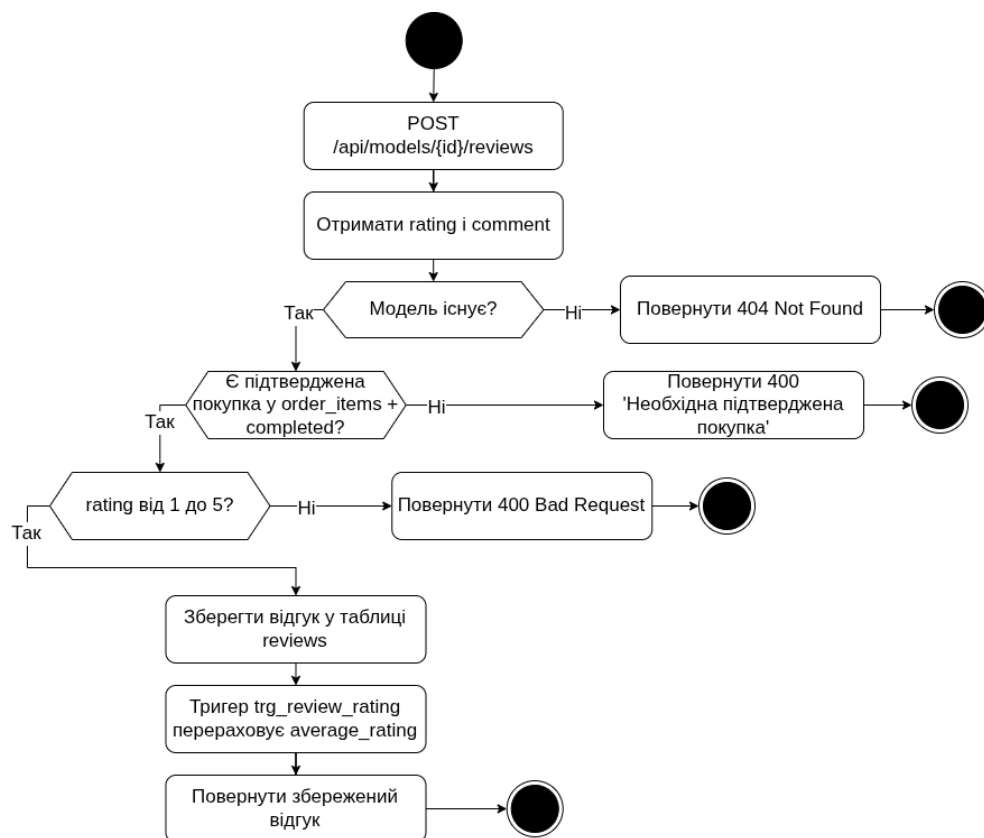


Рис. 3.9 Блок-схема алгоритму публікації відгуку

Алгоритм перерахунку рейтингу. Середній рейтинг моделі підтримується актуальним автоматично через тригер бази даних. При будь-якій зміні у таблиці `reviews` — додаванні, оновленні або видаленні відгуку — тригер `trg_review_rating` викликає функцію яка перераховує середнє значення оцінок і

одразу записує його у поле `average_rating`. Використання `COALESCE` гарантує що при відсутності відгуків поле отримає значення 0, а не `NULL`. Це означає що при читанні каталогу сервер не виконує жодних додаткових обчислень — значення вже актуальне у базі даних.

Блок-схему алгоритму наведено на рисунку 3.10.

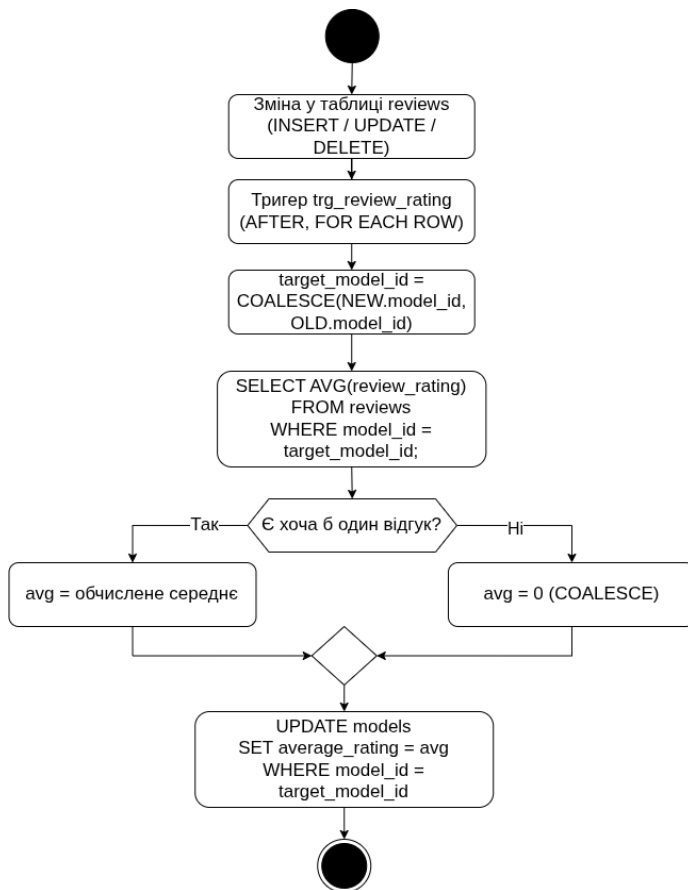


Рис. 3.10 Блок-схема алгоритму перерахунку рейтингу

4 ВПРОВАДЖЕННЯ СИСТЕМИ

4.1 Тестування системи

Тестування системи проводилось на двох рівнях — серверному та клієнтському — із використанням відповідних інструментів для кожної частини. Метою тестування була перевірка коректності бізнес-логіки, обробки граничних випадків та поведінки інтерфейсу користувача.

Тестування серверної частини. Серверні тести реалізовані з використанням PHPUnit 10 і охоплюють модульне тестування сервісів та модуля безпеки. Усього написано 76 тестів у шести тестових класах. Кожен клас відповідає окремому компоненту системи і перевіряє як коректні сценарії так і обробку помилкових ситуацій. Деталі наведено у таблиці 4.1.

Таблиця 4.1

Тестові класи серверної частини

Тестовий клас	Компонент	Кількість тестів
RoleManagerTest	Security/RoleManager	15
AuthServiceTest	Service/AuthService	13
ModelServiceTest	Service/ModelService	18
FileUploadServiceTest	Service/FileUploadService	11
ReviewServiceTest	Service/ReviewService	10
OrderServiceTest	Service/OrderService	9
Разом		76

Тести AuthService перевіряють реєстрацію з унікальністю email і імені користувача, коректність входу, зміну паролю, оновлення профілю та видалення облікового запису. Окремо перевіряється що заблокований користувач не може увійти в систему.

Тести `ModelService` охоплюють логіку видимості моделі — доступність для анонімного користувача, власника та адміністратора залежно від статусу верифікації, видалення моделі з перевіркою прав, підтвердження моделі адміністратором та підрахунок переглядів.

Тести `ReviewService` перевіряють `purchase-gate` логіку — спробу залишити відгук без підтвердженої покупки, некоректний діапазон оцінки, успішне збереження відгуку та видалення чужого відгуку.

Тести `OrderService` перевіряють весь ланцюг валідації при додаванні позиції до замовлення: замовлення не знайдено, замовлення завершено, модель не існує, продавець намагається купити власну модель.

Тести `FileUploadService` перевіряють фільтрацію за розширенням і розміром файлу для трьох типів завантажень — 3D-моделей, зображень попереднього перегляду та аватарів.

Тести `RoleManagerTest` перевіряють коректність рольової моделі — методи `isSeller`, `isBuyer`, `isAdmin`, `canUploadModel`, `canApproveModel` і `requireRole` для всіх комбінацій ролей.

У таблиці 4.2 наведено приклади ключових тестових випадків серверної частини.

Таблиця 4.2

Приклади тестових випадків серверної частини

Тестовий випадок	Вхідні дані	Очікуваний результат	Результат
Реєстрація з існуючим email	email що вже є в БД	<code>BadRequestHttpException</code>	Passed
Вхід з невірним паролем	правильний email, хибний пароль	<code>BadRequestHttpException</code>	Passed

Відгук без покупки	user_id без запису в order_items	BadRequestHttpException	Passed
Оцінка поза діапазоном	rating = 6	BadRequestHttpException	Passed
Перегляд відхиленої моделі анонімно	verification = rejected	NotFoundHttpException	Passed
Видалення чужого відгуку	userId != review.userId	BadRequestHttpException	Passed
Завантаження файлу з неприпустимим розширенням	file.exe	BadRequestHttpException	Passed
Покупець намагається купити власну модель	sellerId = buyerId	BadRequestHttpException	Passed
Перевірка ролі адміністратора	role = seller	AccessDeniedHttpException	Passed
Підтвердження моделі не-адміністратором	role = seller	AccessDeniedHttpException	Passed

Тестування клієнтської частини. Клієнтські тести реалізовані з використанням Vitest і `@testing-library/react`. Усього написано 27 тестів у чотирьох файлах що охоплюють функції роботи з API, компоненти інтерфейсу та контексти глобального стану.

Таблиця 4.3

Тестові файли клієнтської частини

Тестовий файл	Компонент	Кількість тестів
api.test.js	services/apiCall	6
ModelCard.test.jsx	components/ModelCard	9

CartContext.test.jsx	contexts/CartContext	6
AdminPanel.test.tsx	pages/AdminPanel	6
Разом		27

Тести apiCall перевіряють автоматичне додавання Bearer-токена до заголовків, обробку відповіді 401 та повернення помилки при невдалому запиті.

Тести ModelCard перевіряють коректне відображення назви, ціни, рейтингу та значків верифікації і типу моделі.

Тести CartContext перевіряють стан кошика — додавання і видалення позицій, метод isInCart, загальну суму і кількість позицій.

Тести AdminPanel перевіряють перенаправлення не-адміністратора на головну сторінку, відображення вкладок Models і Users, наявність кнопок Approve і Reject для моделей зі статусом pending та виклик відповідного API-ендпоінту при підтвердженні.

Таблиця 4.4

Приклади тестових випадків клієнтської частини

Тестовий випадок	Умова	Очікуваний результат	Результат
Запит з токеном	токен у localStorage	Bearer-токен у заголовку Authorization	Passed
Відповідь 401	сервер повертає 401	виклик logout і перенаправлення	Passed
Відображення ціни	price = 29.99	текст "€29.99" у картці	Passed
Значок безкоштовності	licenseIsFree = true	мітка "Free" у картці	Passed
Підрахунок кошика	3 позиції у кошику	itemCount = 3	Passed

Загальна сума	позиції по €10 і €20	total = 30	Passed
Перенаправлення не-адміна	role = buyer	navigate('/')	Passed
Кнопка підтвердження	модель зі статусом pending	виклик PATCH /admin/models/{id}/verify	Passed

Усі 103 тести пройшли успішно. Виявлені у процесі розробки помилки були виправлені до фіналізації відповідних модулів.

4.2 Вимоги до апаратного та програмного забезпечення

Система складається з двох компонентів — серверної частини і клієнтського застосунку — тому вимоги розглядаються окремо для кожного.

Для розгортання серверної частини необхідна наявність PHP версії 8.2 або вище, PostgreSQL версії 14 або вище та веб-сервера Nginx або Apache для проксування запитів. Збірка клієнтського застосунку виконується за допомогою Node.js версії 18 або вище. Керування залежностями серверної частини здійснюється через Composer, клієнтської — через npm.

З апаратного боку сервер повинен мати щонайменше одне процесорне ядро з тактовою частотою від 1.5 ГГц, 1 ГБ оперативної пам'яті та 5 ГБ дискового простору. Рекомендовані характеристики — 2 ядра, 2 ГБ пам'яті та 20 ГБ для зберігання файлів моделей і зображень. Для стабільної роботи необхідне мережеве з'єднання зі швидкістю від 10 Мбіт/с.

Клієнтська частина працює безпосередньо у браузері і не потребує встановлення додаткового програмного забезпечення. Підтримуються актуальні версії Chrome, Firefox, Safari та Edge. Для коректного відображення 3D-моделей браузер повинен підтримувати WebGL — ця технологія вбудована у всі сучасні браузери за замовчуванням. JavaScript має бути увімкнений.

Топологію розгортання компонентів системи наведено у підрозділі 4.3.

4.3 Розгортання системи та склад інсталяційного пакету

Схему розгортання компонентів системи наведено на рисунку 4.1.

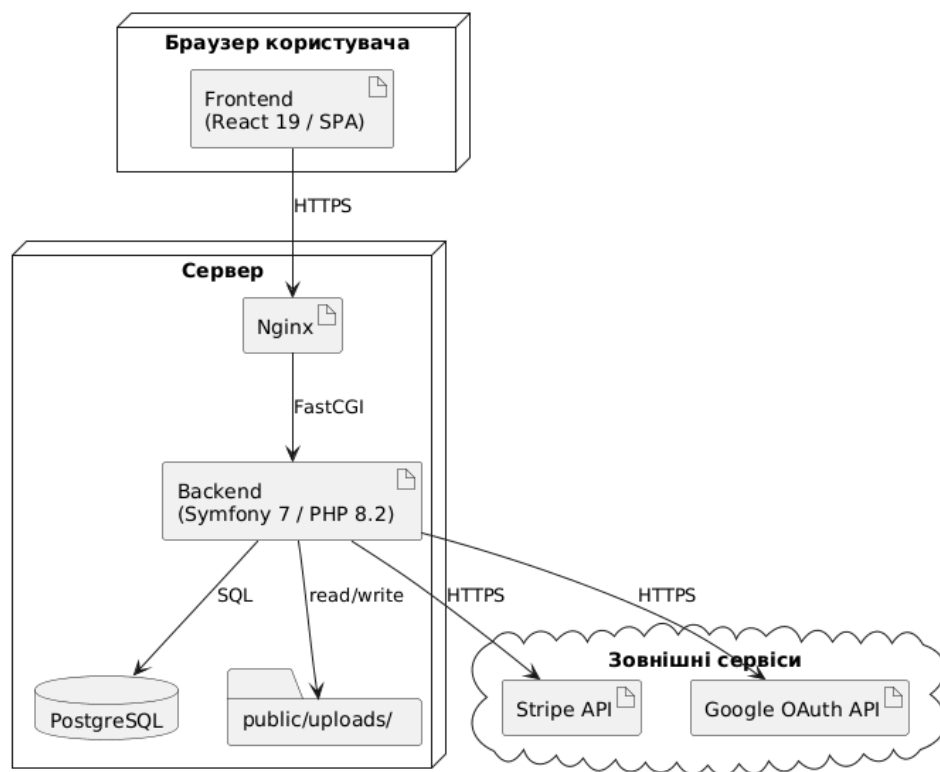


Рис. 4.1 Діаграма розгортання системи

Інсталяційний пакет системи містить такі складові. Серверна частина включає директорію `backend/` з вихідним кодом PHP-застосунку, файл `composer.json` з описом залежностей, папку `migrations/` з файлами міграцій бази даних, папку `config/` з налаштуваннями безпеки і маршрутизації та файл `.env.example` як шаблон для змінних середовища. Клієнтська частина включає директорію `frontend/` з вихідним кодом React-застосунку та файл `package.json` з переліком залежностей. До пакету не входять: файл `.env` з реальними ключами сервісів, директорія `vendor/` із встановленими PHP-бібліотеками, директорія `node_modules/` із JS-залежностями та директорія `public/uploads/` зі збереженими

файлами користувачів — усі ці компоненти формуються або заповнюються безпосередньо на цільовому сервері.

Розгортання системи виконується у кілька послідовних кроків.

Встановлюються залежності серверної і клієнтської частин командами `composer install` та `npm install`. На основі шаблону `.env.example` створюється файл `.env` із параметрами підключення до бази даних, ключами Stripe і Google OAuth та секретом для підпису JWT-токенів. Виконуються міграції командою `php bin/console doctrine:migrations:migrate` — вони створюють усі необхідні таблиці і встановлюють тригери бази даних. Клієнтський застосунок збирається командою `npm run build`, результат розміщується у папці `dist/` і обслуговується веб-сервером. Серверна частина налаштовується як окремий віртуальний хост із кореневою директорією `public/`. Перший `ome` а `Seme` а `Seb`ліковий запис адміністратора створюється вручну через призначення ролі `admin` безпосередньо у базі даних.

4.4 Перспективи розвитку системи

Реалізована система охоплює основний функціонал платформи, але залишає простір для розвитку у кількох напрямках.

Автоматизація модерації та штучний інтелект. Верифікація моделей наразі виконується адміністратором вручну. Інтеграція ШІ дозволила б автоматично перевіряти коректність файлу, відповідність зображень вмісту моделі та відсутність забороненого контенту — адміністратор виконував би роль супервізора лише у спірних випадках. Додатково ШІ міг би генерувати опис моделі на основі її характеристик і пропонувати оптимальний діапазон ціни, аналізуючи схожі моделі в каталозі.

Покращення 3D-перегляду. Поточний переглядач відображає модель у базовому WebGL-режимі. Перспективним розвитком є додавання реалістичного рендерингу — підтримка PBR-матеріалів, налаштування освітлення, тіней і

середовища відображення (HDRI). Це дозволило б покупцям оцінити вигляд моделі в реальних умовах використання до покупки.

Аналітика для продавців. Наразі продавець бачить лише кількість продажів і переглядів. Розширена аналітика у вигляді інтерактивних графіків — динаміка продажів у часі, конверсія переглядів у покупки, розподіл доходу по моделях, порівняння з середніми показниками категорії — допомогла б приймати обґрунтовані рішення щодо ціноутворення і контентної стратегії.

Email-сповіщення. Інтеграція поштового сервісу (SendGrid або AWS SES) дозволила б надсилати автоматичні сповіщення: підтвердження покупки з посиланням на завантаження, зміна статусу верифікації моделі, відповідь адміністратора на тикет підтримки, сповіщення про нові роботи продавця на якого підписаний покупець.

Соціальні функції та фріланс. Вбудований чат між покупцем і продавцем, підписка на авторів і список бажань зі сповіщенням про зниження ціни — стандартні інструменти електронної комерції що підвищують повторні відвідування. На базі наявної інфраструктури можна також реалізувати модуль замовлення індивідуальних моделей з ескроу-оплатою, що перетворило б платформу на повноцінну екосистему для 3D-дизайнерів.

Технічні вдосконалення. Мобільний застосунок для iOS і Android, підтримка кількох валют і мов інтерфейсу розширили б географію платформи. Міграція файлового сховища з локального сервера на хмарне (AWS S3 або аналог) підвищила б надійність і масштабованість зберігання моделей.

ВИСНОВКИ

У бакалаврській кваліфікаційній роботі розроблено вебплатформу для купівлі та продажу тривимірних моделей, що включає повний цикл взаємодії між покупцем, продавцем і адміністратором — від реєстрації та перегляду каталогу до оплати, виведення коштів і модерації контенту.

За результатами аналізу предметної області та порівняння існуючих рішень — TurboSquid, CGTrader і Sketchfab — встановлено що жодна з розглянутих платформ не поєднує одночасно зручний інтерфейс для незалежного автора, прозору систему ліцензування, якісний перегляд моделі до покупки та просту процедуру виведення коштів. Саме ці прогалини визначили функціональні вимоги до розроблюваної системи.

Спроековано реляційну базу даних з п'ятнадцятьма таблицями у третій нормальній формі. Для підтримки актуальності службових полів реалізовано два тригери PostgreSQL: автоматичне оновлення дати зміни моделі та перерахунок середнього рейтингу після кожної зміни відгуку. Це дозволило перенести частину бізнес-логіки на рівень бази даних і гарантувати цілісність даних незалежно від джерела зміни.

Серверна частина побудована на основі Symfony 7 і реалізує чотирнадцять REST-контролерів із шаровим розподілом логіки між контролерами, сервісами і репозиторіями. Автентифікація підтримує два способи входу — через email і пароль та через обліковий запис Google — з використанням JWT-токенів. Платіжна обробка делегована сервісу Stripe через PaymentIntent API, що виключає зберігання карткових даних на стороні платформи. Клієнтська частина реалізована як односторінковий застосунок на React 19 з TypeScript і містить двадцять шість сторінок.

Новим технічним рішенням, запропонованим у роботі, є розмежування двох типів текстової взаємодії користувача з моделлю. Відгук із числовою

оцінкою доступний лише після підтвердженої покупки — система перевіряє наявність відповідного запису у таблиці позицій замовлення перед збереженням. Коментар, натомість, доступний будь-якому авторизованому користувачу без прив'язки до факту покупки. Такий підхід забезпечує достовірність рейтингу моделі і водночас не обмежує вільне обговорення. Додатково реалізовано механізм видимості моделей у каталозі: моделі зі статусом очікування залишаються публічно доступними, що скорочує час виходу нового контенту до покупців порівняно з класичною схемою де модель прихована до підтвердження.

Система включає модуль KYC-верифікації продавців і обробки запитів на виплату з автоматичним поверненням суми при відхиленні, а також модуль підтримки користувачів із системою тікетів. Це відрізняє розроблену платформу від аналогів де інструменти продавця обмежені завантаженням контенту і переглядом статистики.

Коректність реалізованої логіки підтверджена автоматизованим тестуванням: написано 76 серверних тестів з використанням PHPUnit 10 і 27 клієнтських тестів з використанням Vitest, усі 103 тести пройшли успішно. Тести охоплюють граничні випадки бізнес-логіки — спробу залишити відгук без покупки, перегляд відхиленої моделі анонімним користувачем, спробу продавця придбати власну модель та інші сценарії.

Отримані результати відповідають завданню на бакалаврську кваліфікаційну роботу в повному обсязі. Система готова до дослідної експлуатації; подальший розвиток доцільно вести у напрямку автоматизації модерації засобами штучного інтелекту, розширення аналітичних інструментів для продавців, інтеграції поштових сповіщень та покращення якості 3D-перегляду моделей з підтримкою PBR-рендерингу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sommerville I. Software Engineering. — 10th ed. — Pearson, 2016. — 816 p.
2. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : PhD dissertation. — University of California, Irvine, 2000. — 162 p.
3. Turban E., Outland J., King D. Electronic Commerce 2018: A Managerial and Social Networks Perspective. — 9th ed. — Springer, 2018. — 680 p.
4. Fortune Business Insights. 3D Digital Asset Market Size, Share & Industry Analysis. — 2024. — URL: <https://www.fortunebusinessinsights.com/3d-digital-asset-market-112221>
5. CGTrader. About CGTrader — Global 3D Model Marketplace. — URL: <https://www.cgtrader.com/about>
6. TurboSquid. Publishing & Selling 3D Models. — URL: <https://www.turbosquid.com/PublishYourStuff>
7. Sellfy Blog. Top 10 Places to Sell 3D Models in 2026. — 2026. — URL: <https://sellfy.com/blog/sell-3d-models-tips/>
8. Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide. — 2nd ed. — Addison-Wesley, 2005. — 496 p.
9. Elmasri R., Navathe S. B. Fundamentals of Database Systems. — 7th ed. — Pearson, 2016. — 1200 p.
10. PostgreSQL Global Development Group. PostgreSQL 16 Documentation. — 2024. — URL: <https://www.postgresql.org/docs/16/>

11. Obe R., Hsu L. PostgreSQL: Up and Running. — 3rd ed. — O'Reilly Media, 2017. — 338 p.
12. PostgreSQL Global Development Group. PL/pgSQL — SQL Procedural Language. — 2024. — URL: <https://www.postgresql.org/docs/current/plpgsql.html>
13. Stallings W. Cryptography and Network Security: Principles and Practice. — 8th ed. — Pearson, 2019. — 768 p.
14. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : PhD dissertation. — University of California, Irvine, 2000.
15. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). RFC 7519. — IETF, 2015. — URL: <https://datatracker.ietf.org/doc/html/rfc7519>
16. Symfony, Doctrine, PHP. Symfony Documentation. The Symfony Framework Documentation. — URL: <https://symfony.com/doc/current/>
17. OWASP Foundation. OWASP Top Ten Web Application Security Risks. — 2021. — URL: <https://owasp.org/www-project-top-ten/>
18. React Team. React Documentation. — URL: <https://react.dev/>
19. React Three Fiber. Documentation. — URL: <https://docs.pmnd.rs/react-three-fiber>
20. Stripe. Payment Intents API Reference. — URL: https://docs.stripe.com/api/payment_intents
21. PHPUnit Documentation. — URL: <https://phpunit.de/documentation.html>
22. Vitest Documentation. — URL: <https://vitest.dev/guide/>
23. Google Identity. OAuth 2.0 for Web Server Applications. — URL: <https://developers.google.com/identity/protocols/oauth2>

База даних

Сторінок 7

Київ-2026

-- Таблиці категорії та ліцензії

```
CREATE TABLE categories (
  category_id INT GENERATED BY DEFAULT AS IDENTITY NOT NULL,
  name        VARCHAR(255) NOT NULL,
  description TEXT          DEFAULT NULL,
  PRIMARY KEY (category_id)
);

CREATE TABLE licenses (
  license_id    INT GENERATED BY DEFAULT AS IDENTITY NOT NULL,
  value         VARCHAR(50) NOT NULL,
  label         VARCHAR(100) NOT NULL,
  description    TEXT          DEFAULT NULL,
  short_description VARCHAR(255) DEFAULT NULL,
  is_free       BOOLEAN        NOT NULL,
  PRIMARY KEY (license_id)
);
CREATE UNIQUE INDEX uq_licenses_value ON licenses (value);
```

-- Користувачі

```
CREATE TABLE users (
  user_id        INT GENERATED BY DEFAULT AS IDENTITY NOT NULL,
  username       VARCHAR(50) NOT NULL,
  email          VARCHAR(100) NOT NULL,
  password_hash  TEXT          DEFAULT NULL,
  google_id      VARCHAR(255) DEFAULT NULL,
  auth_provider  VARCHAR(20) NOT NULL,
  role           VARCHAR(20) NOT NULL,
  display_name   VARCHAR(100) DEFAULT NULL,
  display_name_public BOOLEAN NOT NULL,
  bio            TEXT          DEFAULT NULL,
  location       VARCHAR(100) DEFAULT NULL,
  location_public BOOLEAN NOT NULL,
  languages      JSON          DEFAULT NULL,
  website        VARCHAR(255) DEFAULT NULL,
  photo_url      TEXT          DEFAULT NULL,
  stripe_account_id VARCHAR(255) DEFAULT NULL,
  stripe_account_status VARCHAR(50) DEFAULT NULL,
  email_verified BOOLEAN NOT NULL,
  seller_info     JSON          DEFAULT NULL,
  registration_date TIMESTAMP(0) NOT NULL,
  username_changed_at TIMESTAMP(0) DEFAULT NULL,
  is_blocked      BOOLEAN NOT NULL DEFAULT FALSE,
  blocked_reason  TEXT          DEFAULT NULL,
  balance         NUMERIC(12, 2) NOT NULL DEFAULT '0.00',
  PRIMARY KEY (user_id)
);
CREATE UNIQUE INDEX uq_users_email ON users (email);
```

```
CREATE UNIQUE INDEX uq_users_google_id ON users (google_id);
```

```
-- Модели
```

```
CREATE TABLE models (
  model_id INT GENERATED BY DEFAULT AS IDENTITY NOT NULL,
  user_id INT NOT NULL,
  category_id INT DEFAULT NULL,
  license_id INT NOT NULL,
  title VARCHAR(255) NOT NULL,
  description TEXT DEFAULT NULL,
  price NUMERIC(10, 2) NOT NULL,
  file_url TEXT DEFAULT NULL,
  file_name VARCHAR(255) DEFAULT NULL,
  file_size BIGINT NOT NULL,
  upload_date TIMESTAMP(0) NOT NULL,
  updated_at TIMESTAMP(0) DEFAULT NULL,
  verification VARCHAR(50) NOT NULL,
  status VARCHAR(20) NOT NULL,
  average_rating NUMERIC(3, 2) NOT NULL,
  sales_count INT NOT NULL,
  view_count INT NOT NULL,
  tags TEXT DEFAULT NULL,
  model_type VARCHAR(20) NOT NULL,
  features TEXT DEFAULT NULL,
  polygons INT DEFAULT NULL,
  vertices INT DEFAULT NULL,
  geometry VARCHAR(100) DEFAULT NULL,
  unwrapped_uvs VARCHAR(50) DEFAULT NULL,
  PRIMARY KEY (model_id)
```

```
);
```

```
CREATE INDEX idx_models_user ON models (user_id);
```

```
CREATE INDEX idx_models_category ON models (category_id);
```

```
CREATE INDEX idx_models_license ON models (license_id);
```

```
CREATE TABLE model_preview_images (
  image_id INT GENERATED BY DEFAULT AS IDENTITY NOT NULL,
  model_id INT NOT NULL,
  image_url TEXT NOT NULL,
  image_order INT NOT NULL,
  is_main BOOLEAN NOT NULL,
  PRIMARY KEY (image_id)
```

```
);
```

```
CREATE INDEX idx_model_preview_images_model ON model_preview_images (model_id);
```

```
CREATE TABLE model_files (
  id INT GENERATED BY DEFAULT AS IDENTITY NOT NULL,
  model_id INT NOT NULL,
  file_url TEXT NOT NULL,
  file_name VARCHAR(255) NOT NULL,
```

```

file_size INT NOT NULL,
format VARCHAR(20) NOT NULL,
is_native BOOLEAN NOT NULL,
is_preview BOOLEAN NOT NULL,
upload_date TIMESTAMP(0) NOT NULL,
PRIMARY KEY (id)
);
CREATE INDEX idx_model_files_model ON model_files (model_id);

-- Замовлення та оплати

CREATE TABLE orders (
order_id INT GENERATED BY DEFAULT AS IDENTITY NOT NULL,
user_id INT NOT NULL,
order_date TIMESTAMP(0) NOT NULL,
total_price NUMERIC(10, 2) DEFAULT NULL,
order_status VARCHAR(50) NOT NULL,
PRIMARY KEY (order_id)
);
CREATE INDEX idx_orders_user ON orders (user_id);

CREATE TABLE order_items (
order_item_id INT GENERATED BY DEFAULT AS IDENTITY NOT NULL,
order_id INT NOT NULL,
model_id INT NOT NULL,
unit_price NUMERIC(10, 2) NOT NULL,
PRIMARY KEY (order_item_id)
);
CREATE INDEX idx_order_items_order ON order_items (order_id);
CREATE INDEX idx_order_items_model ON order_items (model_id);

CREATE TABLE payments (
payment_id INT GENERATED BY DEFAULT AS IDENTITY NOT NULL,
order_id INT NOT NULL,
amount NUMERIC(10, 2) NOT NULL,
payment_method VARCHAR(50) DEFAULT NULL,
payment_status VARCHAR(50) DEFAULT NULL,
payment_date TIMESTAMP(0) NOT NULL,
stripe_payment_intent_id VARCHAR(255) DEFAULT NULL,
PRIMARY KEY (payment_id)
);
CREATE UNIQUE INDEX uq_payments_order ON payments (order_id);

CREATE TABLE cart_items (
cart_item_id INT GENERATED BY DEFAULT AS IDENTITY NOT NULL,
user_id INT NOT NULL,
model_id INT NOT NULL,
added_at TIMESTAMP(0) NOT NULL,
PRIMARY KEY (cart_item_id)
);

```

```
CREATE INDEX    idx_cart_items_user    ON cart_items (user_id);
CREATE INDEX    idx_cart_items_model    ON cart_items (model_id);
CREATE UNIQUE INDEX uq_cart_items_user_model ON cart_items (user_id, model_id);
```

-- Відгуки та коментарі

```
CREATE TABLE reviews (
  review_id  INT GENERATED BY DEFAULT AS IDENTITY NOT NULL,
  user_id    INT      NOT NULL,
  model_id   INT      NOT NULL,
  review_rating INT    NOT NULL,
  review_comment TEXT  DEFAULT NULL,
  review_date TIMESTAMP(0) NOT NULL,
  is_blocked BOOLEAN  NOT NULL DEFAULT FALSE,
  blocked_reason TEXT  DEFAULT NULL,
  PRIMARY KEY (review_id)
);
CREATE INDEX idx_reviews_user ON reviews (user_id);
CREATE INDEX idx_reviews_model ON reviews (model_id);
```

```
CREATE TABLE comments (
  comment_id  INT GENERATED BY DEFAULT AS IDENTITY NOT NULL,
  user_id     INT      NOT NULL,
  model_id    INT      NOT NULL,
  comment_text TEXT    NOT NULL,
  comment_date TIMESTAMP(0) NOT NULL,
  is_blocked  BOOLEAN  NOT NULL DEFAULT FALSE,
  blocked_reason TEXT  DEFAULT NULL,
  PRIMARY KEY (comment_id)
);
CREATE INDEX idx_comments_user ON comments (user_id);
CREATE INDEX idx_comments_model ON comments (model_id);
```

-- Профіль продавця, виплати, підтримка

```
CREATE TABLE seller_profiles (
  profile_id  INT GENERATED BY DEFAULT AS IDENTITY NOT NULL,
  user_id     INT      NOT NULL,
  first_name  VARCHAR(100) DEFAULT NULL,
  last_name   VARCHAR(100) DEFAULT NULL,
  date_of_birth DATE     DEFAULT NULL,
  address_street VARCHAR(255) DEFAULT NULL,
  address_city VARCHAR(100) DEFAULT NULL,
  address_zip  VARCHAR(20)  DEFAULT NULL,
  address_country VARCHAR(100) DEFAULT NULL,
  iban        VARCHAR(50)  DEFAULT NULL,
  bic         VARCHAR(20)  DEFAULT NULL,
  paypal_email VARCHAR(255) DEFAULT NULL,
  tax_id      VARCHAR(100) DEFAULT NULL,
  kyc_status   VARCHAR(20) NOT NULL DEFAULT 'pending',
```



```

    submitted_at  TIMESTAMP(0) DEFAULT NULL,
    admin_note    TEXT          DEFAULT NULL,
    PRIMARY KEY (profile_id)
);
CREATE UNIQUE INDEX uq_seller_profiles_user ON seller_profiles (user_id);

CREATE TABLE payouts (
    payout_id  INT GENERATED BY DEFAULT AS IDENTITY NOT NULL,
    user_id    INT          NOT NULL,
    amount     NUMERIC(12, 2) NOT NULL,
    status     VARCHAR(20)  NOT NULL DEFAULT 'pending',
    payout_method VARCHAR(20) NOT NULL DEFAULT 'bank',
    seller_note TEXT          DEFAULT NULL,
    created_at  TIMESTAMP(0) NOT NULL,
    processed_at TIMESTAMP(0) DEFAULT NULL,
    admin_note  TEXT          DEFAULT NULL,
    PRIMARY KEY (payout_id)
);
CREATE INDEX idx_payouts_user  ON payouts (user_id);
CREATE INDEX idx_payouts_status ON payouts (status);

CREATE TABLE support_tickets (
    ticket_id  INT GENERATED BY DEFAULT AS IDENTITY NOT NULL,
    user_id    INT          NOT NULL,
    type       VARCHAR(30) NOT NULL DEFAULT 'support',
    ticket_subject VARCHAR(255) DEFAULT NULL,
    ticket_message TEXT          DEFAULT NULL,
    ticket_status VARCHAR(50) NOT NULL DEFAULT 'open',
    related_model_id INT          DEFAULT NULL,
    related_comment_id INT          DEFAULT NULL,
    ticket_created_at TIMESTAMP(0) NOT NULL,
    admin_note  TEXT          DEFAULT NULL,
    PRIMARY KEY (ticket_id)
);
CREATE INDEX idx_support_tickets_user ON support_tickets (user_id);

-- Зовнішні ключі

ALTER TABLE models
    ADD CONSTRAINT fk_models_user
        FOREIGN KEY (user_id) REFERENCES users (user_id) ON DELETE CASCADE,
    ADD CONSTRAINT fk_models_category
        FOREIGN KEY (category_id) REFERENCES categories (category_id) ON DELETE SET
NULL,
    ADD CONSTRAINT fk_models_license
        FOREIGN KEY (license_id) REFERENCES licenses (license_id);

ALTER TABLE model_preview_images
    ADD CONSTRAINT fk_preview_images_model
        FOREIGN KEY (model_id) REFERENCES models (model_id) ON DELETE CASCADE;

```

```

ALTER TABLE model_files
  ADD CONSTRAINT fk_model_files_model
    FOREIGN KEY (model_id) REFERENCES models (model_id) ON DELETE CASCADE;

ALTER TABLE orders
  ADD CONSTRAINT fk_orders_user
    FOREIGN KEY (user_id) REFERENCES users (user_id);

ALTER TABLE order_items
  ADD CONSTRAINT fk_order_items_order
    FOREIGN KEY (order_id) REFERENCES orders (order_id) ON DELETE CASCADE,
  ADD CONSTRAINT fk_order_items_model
    FOREIGN KEY (model_id) REFERENCES models (model_id);

ALTER TABLE payments
  ADD CONSTRAINT fk_payments_order
    FOREIGN KEY (order_id) REFERENCES orders (order_id) ON DELETE CASCADE;

ALTER TABLE cart_items
  ADD CONSTRAINT fk_cart_items_user
    FOREIGN KEY (user_id) REFERENCES users (user_id) ON DELETE CASCADE,
  ADD CONSTRAINT fk_cart_items_model
    FOREIGN KEY (model_id) REFERENCES models (model_id) ON DELETE CASCADE;

ALTER TABLE reviews
  ADD CONSTRAINT fk_reviews_user
    FOREIGN KEY (user_id) REFERENCES users (user_id),
  ADD CONSTRAINT fk_reviews_model
    FOREIGN KEY (model_id) REFERENCES models (model_id);

ALTER TABLE comments
  ADD CONSTRAINT fk_comments_user
    FOREIGN KEY (user_id) REFERENCES users (user_id),
  ADD CONSTRAINT fk_comments_model
    FOREIGN KEY (model_id) REFERENCES models (model_id);

ALTER TABLE seller_profiles
  ADD CONSTRAINT fk_seller_profiles_user
    FOREIGN KEY (user_id) REFERENCES users (user_id) ON DELETE CASCADE;

ALTER TABLE payouts
  ADD CONSTRAINT fk_payouts_user
    FOREIGN KEY (user_id) REFERENCES users (user_id) ON DELETE CASCADE;

ALTER TABLE support_tickets
  ADD CONSTRAINT fk_support_tickets_user
    FOREIGN KEY (user_id) REFERENCES users (user_id) ON DELETE CASCADE;

-- Тригери

```

```

CREATE OR REPLACE FUNCTION fn_set_model_updated_at()
RETURNS TRIGGER AS $$
BEGIN
    NEW.updated_at = NOW();
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

```

```

CREATE TRIGGER trg_model_updated_at
    BEFORE UPDATE ON models
    FOR EACH ROW
    EXECUTE FUNCTION fn_set_model_updated_at();

```

```

CREATE OR REPLACE FUNCTION fn_recalc_model_avg_rating()
RETURNS TRIGGER AS $$
DECLARE
    target_model_id INTEGER;
BEGIN
    target_model_id := COALESCE(NEW.model_id, OLD.model_id);
    UPDATE models
    SET average_rating = (
        SELECT COALESCE(AVG(review_rating), 0)
        FROM reviews
        WHERE model_id = target_model_id
    )
    WHERE model_id = target_model_id;
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

```

```

CREATE TRIGGER trg_review_rating
    AFTER INSERT OR UPDATE OR DELETE ON reviews
    FOR EACH ROW
    EXECUTE FUNCTION fn_recalc_model_avg_rating();

```

Скриншоти інтерфейсу системи

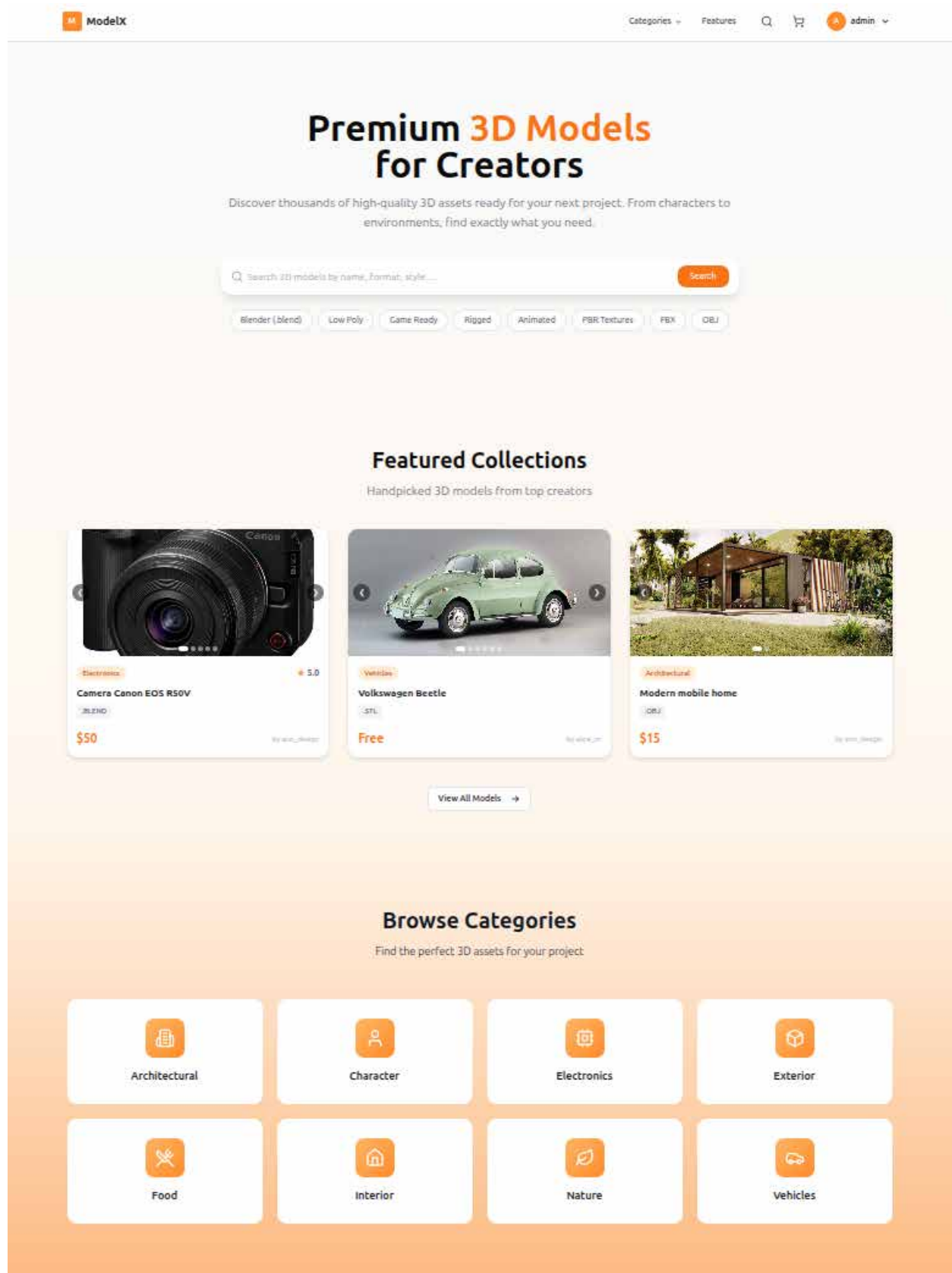


Рис. Б.1 Головна сторінка

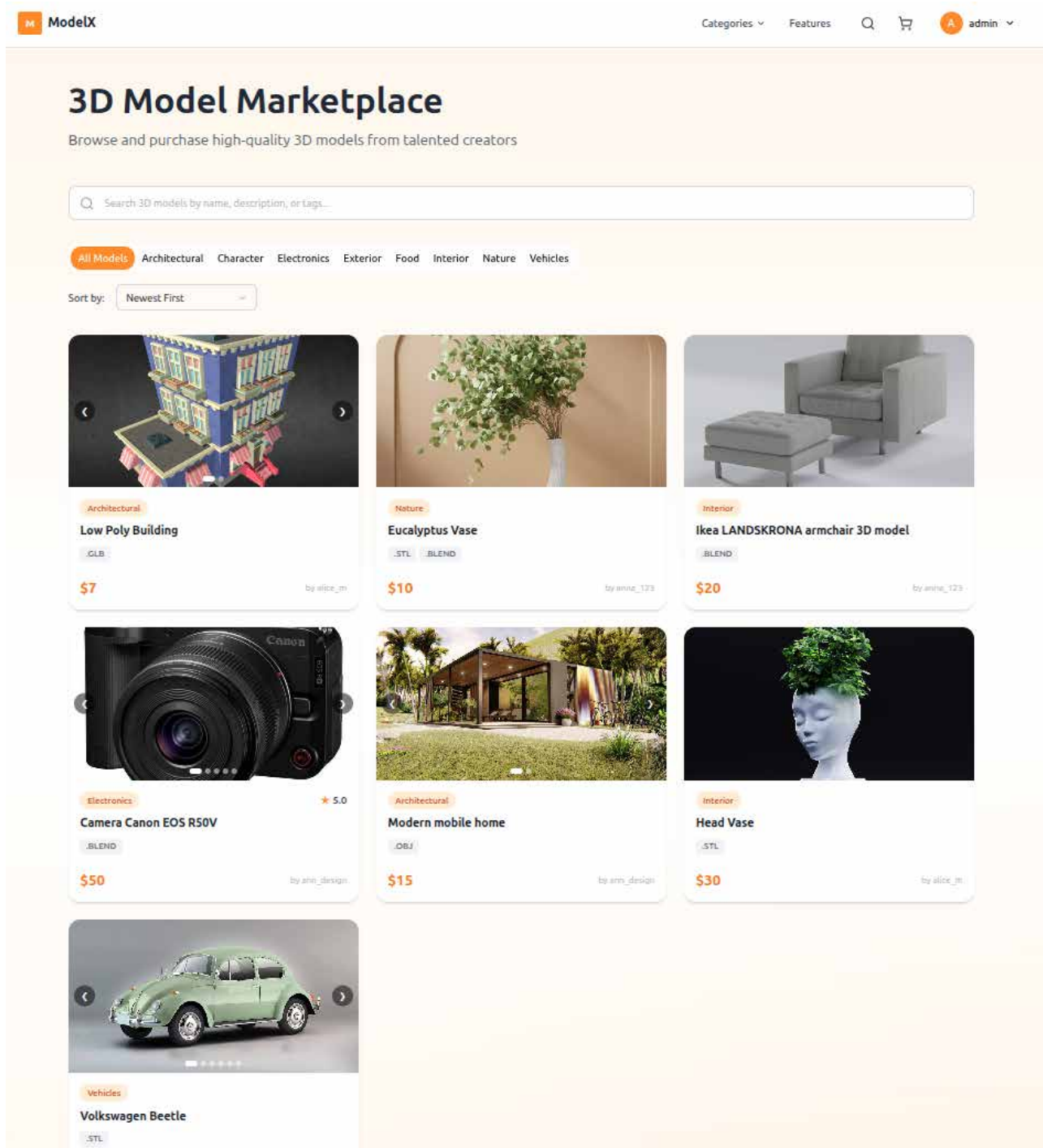


Рис. Б.2 Каталог з фільтрами

Categories ▾ Features 🔍 🛒

1 / 2

Modern mobile home

★ ★ ★ ★ ★ No ratings yet

€15.00

+ Add to cart

Technical details

CGI model

Textures Materials PBR UVW mapping Plugins used

Unwrapped UVs Geometry Mixed Polygonal quads/tris

Details

Category:	Architectural
File size:	4.77 MB
Uploaded:	2.5.2026

License

Royalty-Free

[Report this model](#)

Author

anna_design

Description

Modern designed mobile home, tiny house, vacation house on 38 m2 with terrace and fully furnished interior. Model is produced in Slovenija by my company and can be ordered and placed any where in europe. Unique designed, modern, highly detailed model with lot of useful 3D models and furniture in it. Model contains house with all the furniture in

- living room
- kitchen
- bedroom
- bathroom
- terrace, additional models

(shower, sink, faucet, toilet, bath, sofa, bed TV, outdoor lounge, pots, windows,...) included.

Show less ^

RELATED TAGS

exterior villa house

Reviews

Purchase this model to leave a review.

Leave a review

Purchase this model to leave a review.

No reviews yet.

Рис. Б.3 Сторінка моделі з 3D-переглядачем


ModelX

Categories ▾
Features
Q
Shopping cart icon
anna_123 ▾

Upload your files

Share your 3D creations with the community

Media & Files

Preview images * (1-25)



Drag & Drop images here

or

Browse

PNG, JPG, WEBP | max 10 MB each

IMAGE GUIDELINES

- First image is the cover shown in search results
- Add a wireframe or clay render
- Show the model from multiple angles
- Recommended: 1000x1000 px or higher
- PNG or JPG, max 10 MB each
- Up to 25 images total

3D model files *

3D preview supported: .GLB .GLTF .STL .OBJ Download only: .fbx .blend .c4d .dae .3ds .ma .lwo max 250 MB per file

+ Add 3D files

Texture files (optional)

Put all textures into a single .zip or .rar archive before uploading

+ Add texture archive (.zip / .rar)

Details

Title *

E.g. Modern Office Chair Low-Poly

Description

B / **I** / **S** H2 **≡** **≡**

Category Select category ▾ License * Royalty-Free Lizenz ▾

Price (EUR) *

€ 0.00

This is the price customers pay

Tags

ikea, chair, furniture ...

Press **Space** to add a tag - 0/30

Рис. Б.4 Форма завантаження моделі

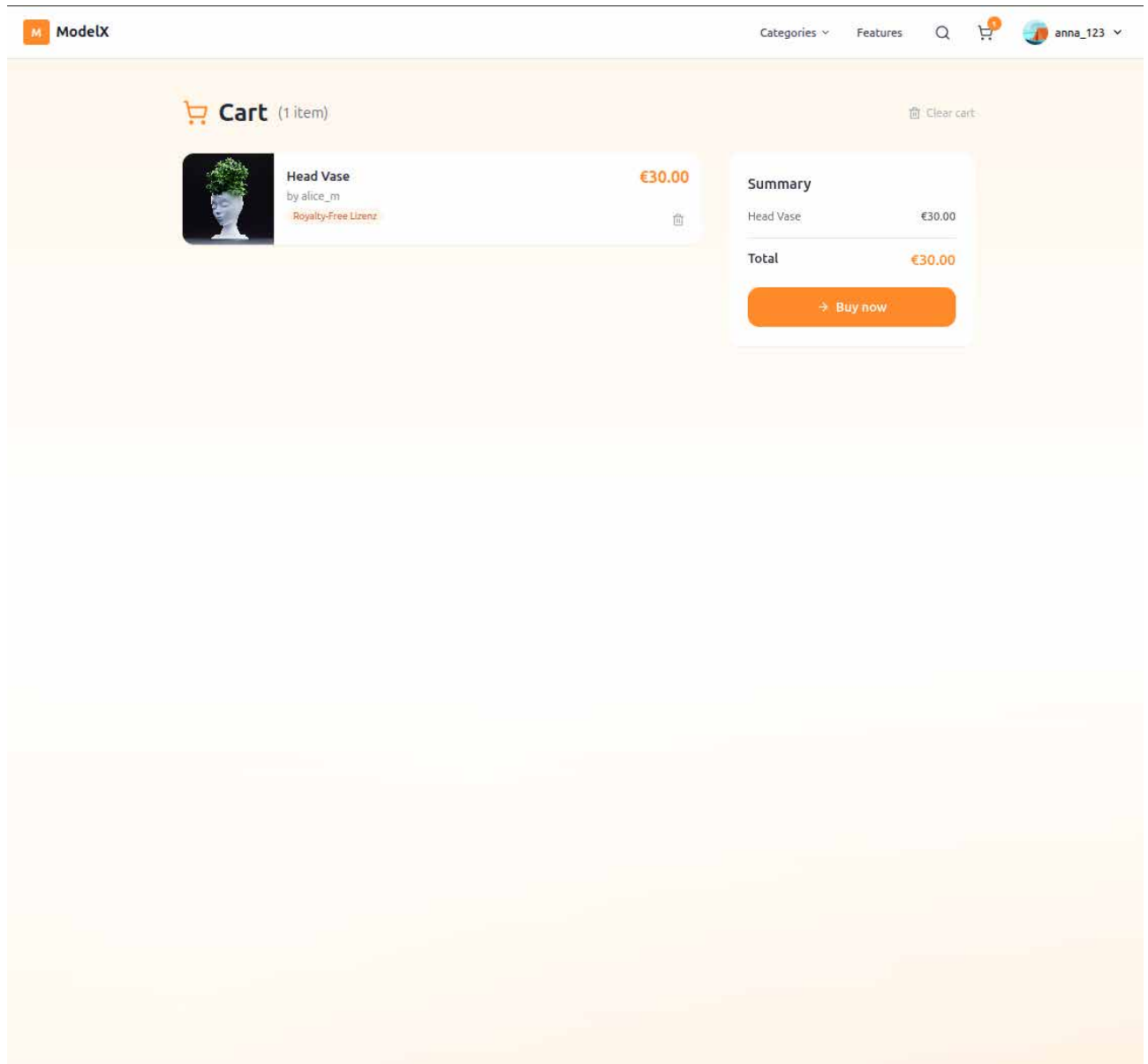


Рис. Б.5 Кошик

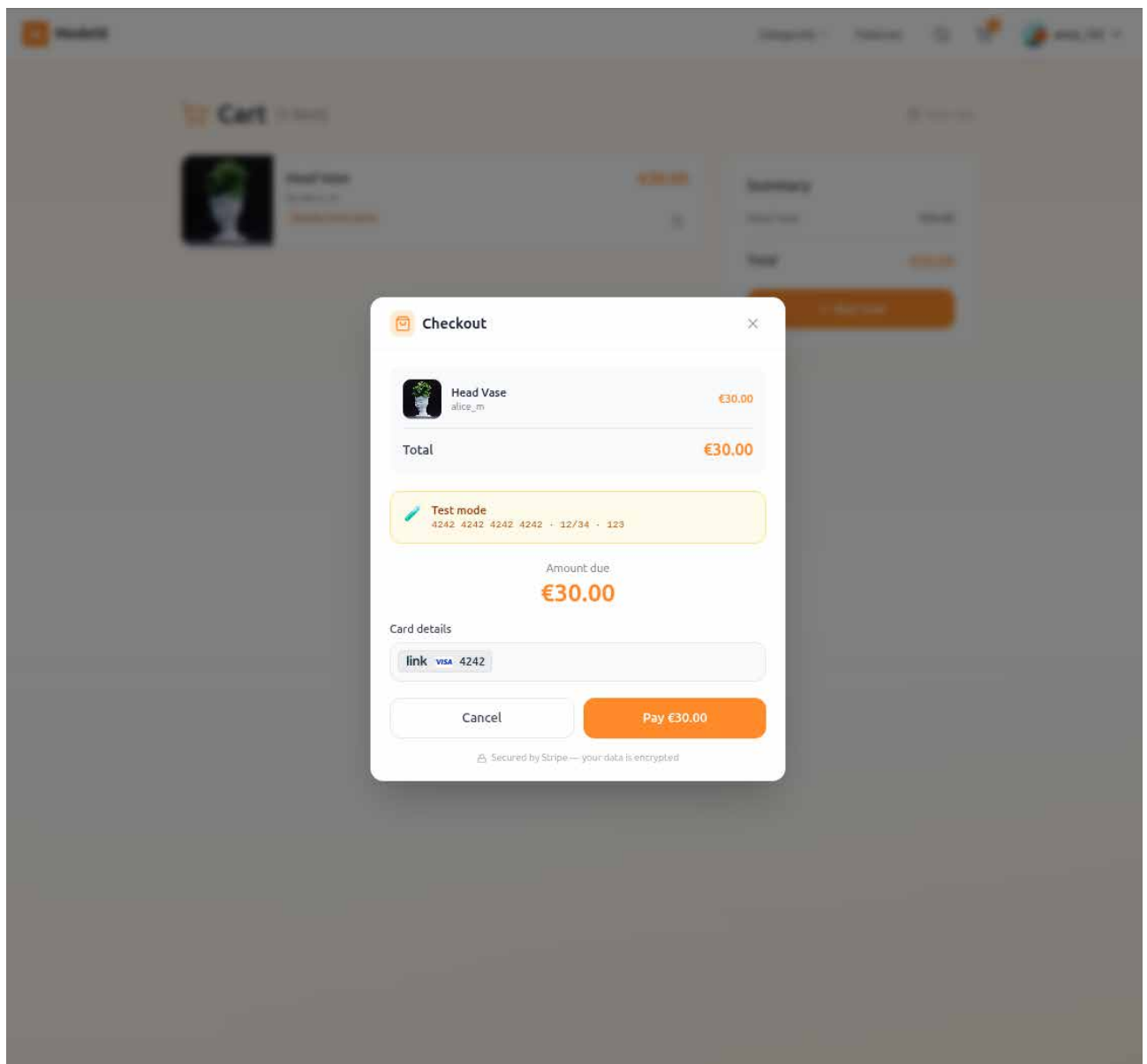


Рис. Б.6 Оплата

The image shows a web interface for user profile management. At the top, there are two tabs: "Personal Information" (selected) and "Account Information". Below the tabs, the "Profile" section contains a "Username" field with the value "ann_design" and a note: "Unique — visible to everyone. Can be changed once per month." Below this is an "About yourself" text area and a "Save" button. The "Account Information" section contains an "Email" field with "anastasiia.s333@gmail.com", a "Change" button, and a green "Email verified" status. Below the email is a "Password" field with masked characters and another "Change" button. The "Account Type" is set to "Seller". At the bottom, there is a "Log out" section with a "Logout" button and a "Delete Account" section with a warning message and a "Delete my account" button.

Personal Information Account Information

Profile

Username *

ann_design

Unique — visible to everyone. Can be changed once per month.

About yourself

Save

Account Information

Email

anastasiia.s333@gmail.com ✉ Change

✓ Email verified

Password

***** 🔒 Change

Account Type

Seller

Log out

Sign out of your account on this device 🚪 Logout

Delete Account

This action is permanent and cannot be undone. All your data, models, and purchases will be removed.

🗑 Delete my account

Рис. Б.7 Налаштування профілю користувача

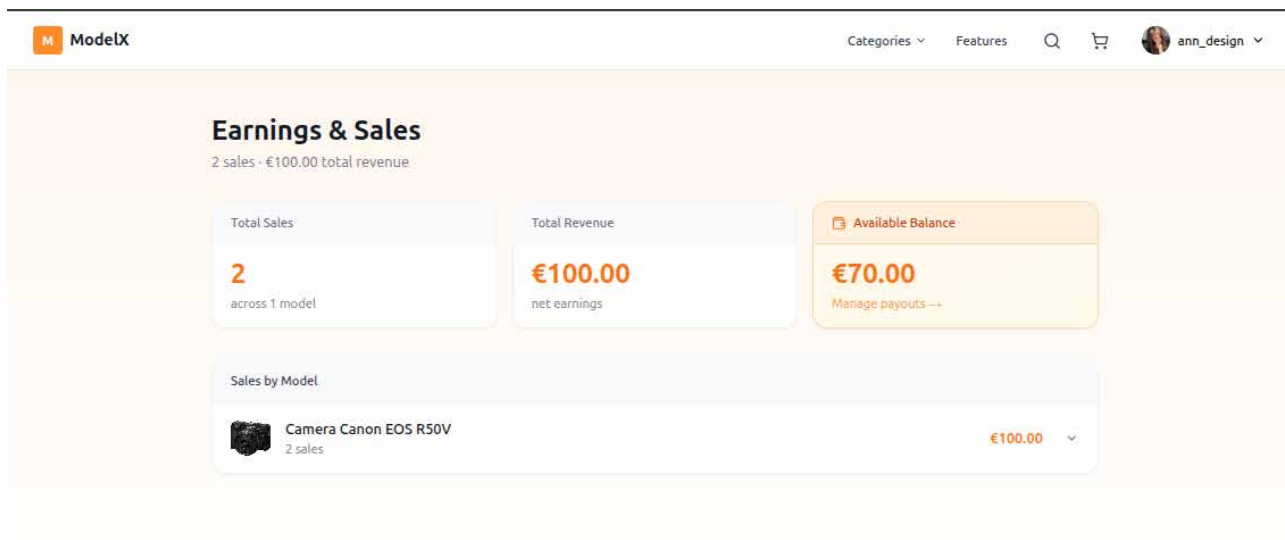


Рис. Б.8 Кабінет продавця

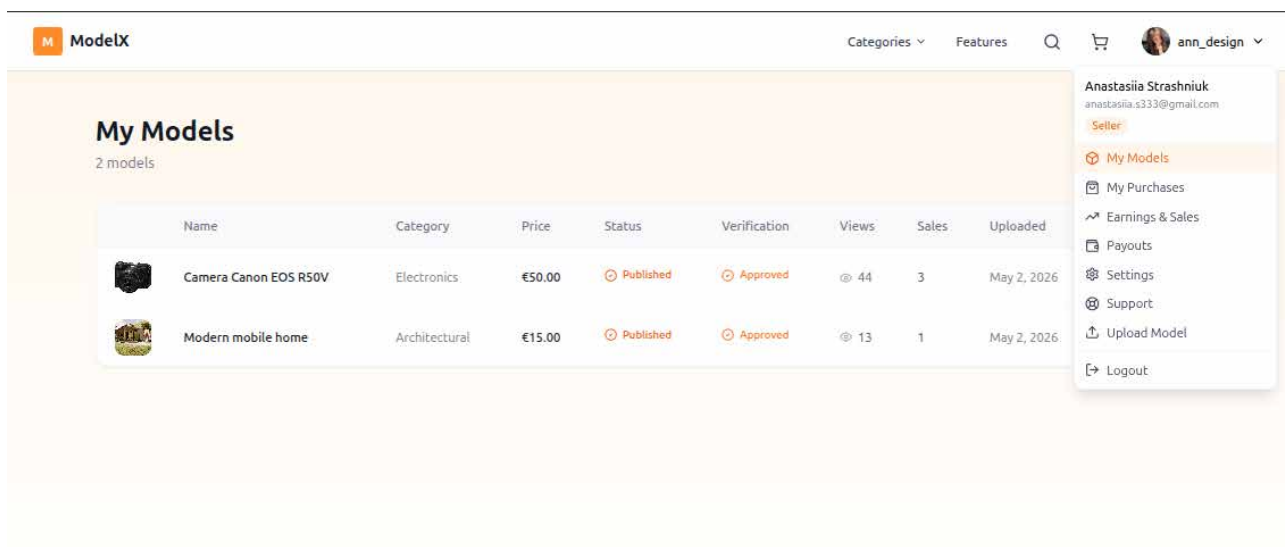


Рис. Б.9 Перегляд завантажених 3д моделей

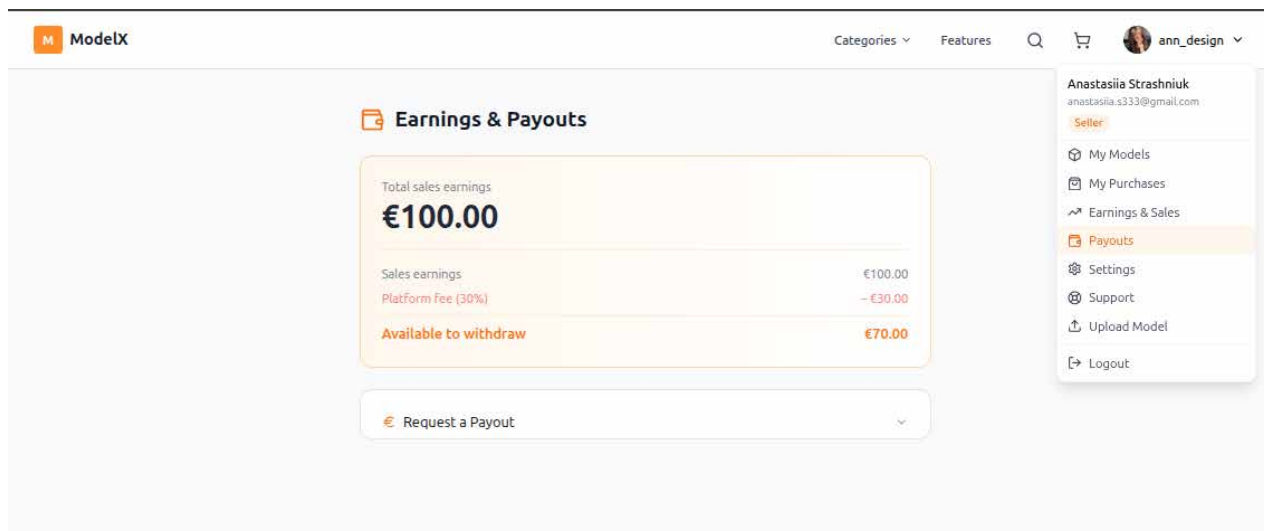


Рис. Б.10 Виведення коштів

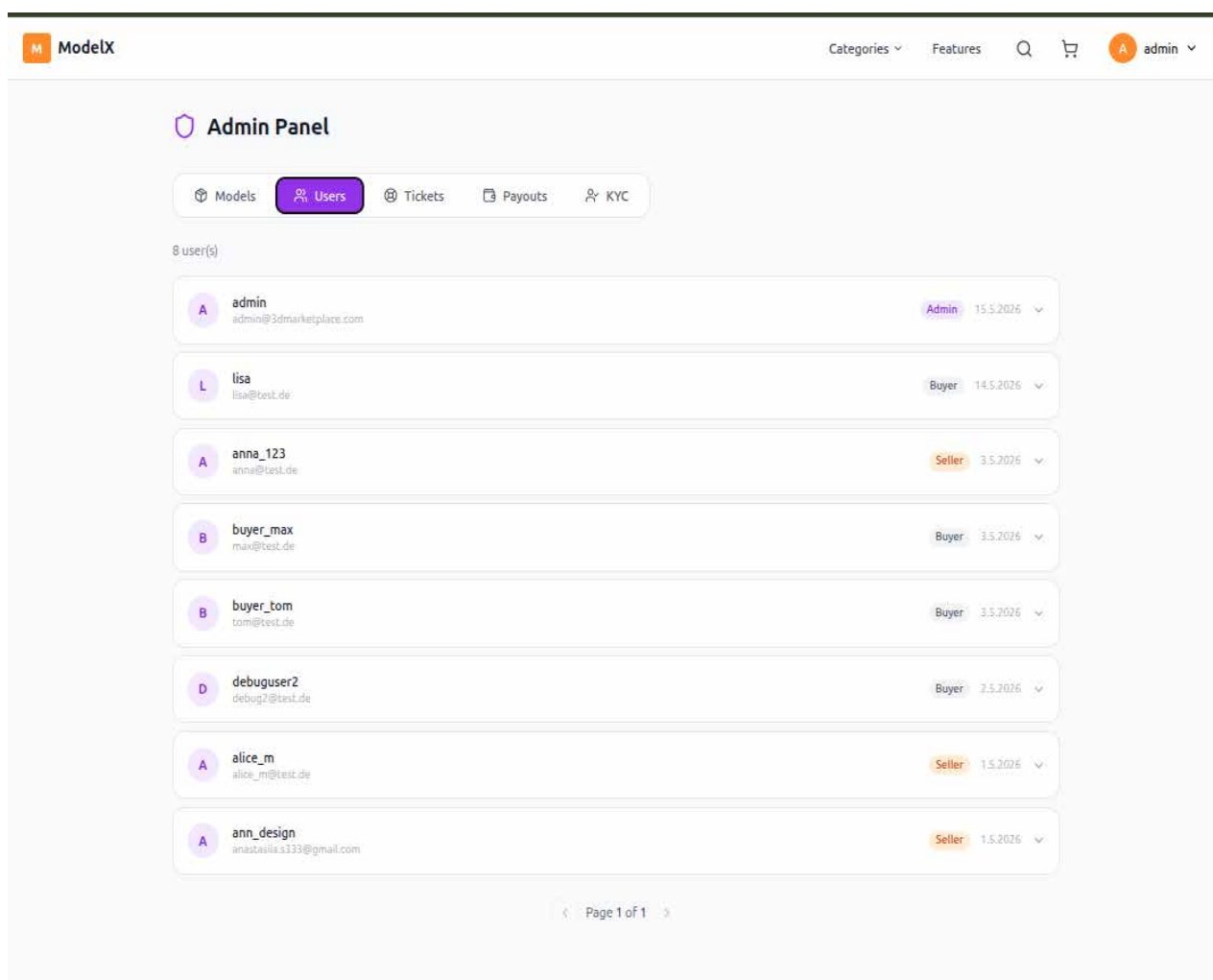


Рис. Б.11 Адмін панель

Лістинги ключового коду програми

Сторінок 9

Лістинг В.1 AuthService.php — реєстрація, вхід та автентифікація через Google

```

<?php
namespace App\Service;

use App\Entity\User;
use App\Repository\UserRepository;
use Lexik\Bundle\JWTAuthenticationBundle\Services\JWTTokenManagerInterface;
use Symfony\Component\HttpKernel\Exception\BadRequestHttpException;
use Symfony\Component\HttpKernel\Exception\ConflictHttpException;
use Symfony\Component\PasswordHasher\Hasher\UserPasswordHasherInterface;

class AuthService
{
    public function __construct(
        private readonly UserRepository $userRepository,
        private readonly UserPasswordHasherInterface $passwordHasher,
        private readonly JWTTokenManagerInterface $jwtManager,
    ) {}

    public function register(string $email, string $password, string $username): array
    {
        if ($this->userRepository->findByEmail($email)) {
            throw new ConflictHttpException('E-Mail is already registered.');
```

```

        throw new BadRequestHttpException('E-Mail or password ist false.');
```

```

    }

    if ($user->isBlocked()) {
        $reason = $user->getBlockedReason() ? ' Reason: ' . $user->getBlockedReason() : '';
        throw new BadRequestHttpException('USER_BLOCKED:' . $reason);
    }

    return [
        'token' => $this->jwtManager->create($user),
        'user' => $user->toArray(),
    ];
}

public function loginWithGoogle(string $googleToken): array
{
    $response = file_get_contents(
        'https://www.googleapis.com/oauth2/v3/userinfo',
        false,
        stream_context_create(['http' => [
            'header' => 'Authorization: Bearer ' . $googleToken,
            'ignore_errors' => true,
        ]])
    );

    if ($response === false) {
        throw new BadRequestHttpException('Google-Token couldn't be verified.');
```

```

    }

    $payload = json_decode($response, true);
    if (empty($payload['sub']) || empty($payload['email'])) {
        throw new BadRequestHttpException('Invalid Google token. Please try again.');
```

```

    }

    $googleId = $payload['sub'];
    $email = $payload['email'];
    $name = $payload['name'] ?? explode('@', $email)[0];
    $photoUrl = $payload['picture'] ?? null;

    $user = $this->userRepository->findByGoogleId($googleId)
        ?? $this->userRepository->findByEmail($email);

    if ($user === null) {
        $user = new User();
        $user->setEmail($email);
        $user->setUsername($this->generateUniqueUsername($email));
        $user->setDisplayName($name ?? null);
        $user->setRole('buyer');
        $user->setEmailVerified(true);
    }
}

```



```

    $user->setGoogleId($googleId);
    $user->setAuthProvider('google');
    if ($photoUrl) {
        $user->setPhotoUrl($photoUrl);
    }

    $this->userRepository->save($user);

    return [
        'token' => $this->jwtManager->create($user),
        'user' => $user->toArray(),
    ];
}

private function generateUniqueUsername(string $email): string
{
    $base = preg_replace('/^[^a-z0-9. _]\/', '', strtolower(explode('@', $email)[0]));
    $base = $base ?: 'user';

    if (!$this->userRepository->isUsernameTaken($base)) {
        return $base;
    }

    $i = 2;
    while ($this->userRepository->isUsernameTaken($base . $i)) {
        $i++;
    }
    return $base . $i;
}

public function changePassword(User $user, string $currentPassword, string $newPassword):
void
{
    if (!$this->passwordHasher->isPasswordValid($user, $currentPassword)) {
        throw new BadRequestHttpException('Current password is false.');
```

Лістинг В.2 ReviewService.php — purchase-gate логіка відгуків

```

<?php
namespace App\Service;

use App\Entity\Review;
use App\Entity\User;
use App\Repository\ModelRepository;
use App\Repository\OrderItemRepository;
use App\Repository\ReviewRepository;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Component\HttpKernel\Exception\BadRequestHttpException;
use Symfony\Component\HttpKernel\Exception\NotFoundHttpException;

class ReviewService
{
    public function __construct(
        private readonly ReviewRepository $reviewRepository,
        private readonly ModelRepository $modelRepository,
        private readonly OrderItemRepository $orderItemRepository,
        private readonly EntityManagerInterface $em,
    ) {}

    public function addReview(int $modelId, User $user, int $rating, ?string $comment): Review
    {
        $model = $this->modelRepository->find($modelId);
        if (!$model) {
            throw new NotFoundHttpException("Model $modelId not found.");
        }

        if (!$this->orderItemRepository->hasPurchased($user->getId(), $modelId)) {
            throw new BadRequestHttpException(
                'You must purchase this model before leaving a review.'
            );
        }

        if ($rating < 1 || $rating > 5) {
            throw new BadRequestHttpException('Rating must be between 1 and 5.');
```

```

        }

        $review = new Review();
        $review->setUser($user);
        $review->setModel($model);
        $review->setRating($rating);
        $review->setComment($comment);

        $this->reviewRepository->save($review, false);
        $this->em->flush();
    }
}
```

```

        return $review;
    }

    public function deleteReview(int $reviewId, User $user): void
    {
        $review = $this->reviewRepository->find($reviewId);
        if (!$review) {
            throw new NotFoundException("Review $reviewId not found.");
        }
        if ($review->getUser()->getId() !== $user->getId()) {
            throw new BadRequestHttpException('No permission to delete.');
```

Лістинг В.3 OrderService.php — створення замовлення та валідація позицій

```

<?php
namespace App\Service;

use App\Entity\Order;
use App\Entity\OrderItem;
use App\Entity\User;
use App\Repository\ModelRepository;
use App\Repository\OrderItemRepository;
use App\Repository\OrderRepository;
use Symfony\Component\HttpKernel\Exception\BadRequestHttpException;
use Symfony\Component\HttpKernel\Exception\NotFoundException;

class OrderService
{
    public function __construct(
        private readonly OrderRepository $orderRepository,
        private readonly OrderItemRepository $orderItemRepository,
        private readonly ModelRepository $modelRepository,
    ) {}

    public function createOrder(User $user): Order
    {
        $order = new Order();
        $order->setUser($user);
```

```

$order->setOrderStatus('created');
$order->setTotalPrice('0.00');

$this->orderRepository->save($order);
return $order;
}

public function addItem(int $orderId, int $modelId, User $user): Order
{
    $order = $this->findOrderForUser($orderId, $user);

    if ($order->getOrderStatus() !== 'created') {
        throw new BadRequestHttpException('Completed orders cannot be modified.');
```

```

public function findOrderForUser(int $orderId, User $user): Order
{
    $order = $this->orderRepository->find($orderId);
    if (!$order || $order->getUser()->getId() !== $user->getId()) {
        throw new NotFoundException("Order $orderId not found.");
    }
    return $order;
}
}

```

Лістинг В.4 PayoutRequestController.php — обчислення балансу та виплати

```

<?php
namespace App\Controller;

use App\Entity\Payout;
use App\Entity\SellerProfile;
use App\Entity\User;
use App\Repository\OrderRepository;
use App\Repository\PayoutRepository;
use App\Repository\SellerProfileRepository;
use App\Repository\UserRepository;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\JsonResponse;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpKernel\Exception\BadRequestHttpException;
use Symfony\Component\Routing\Attribute\Route;

#[Route('/api/seller/payouts')]
class PayoutRequestController extends AbstractController
{
    private const PLATFORM_FEE = 0.30;

    public function __construct(
        private readonly PayoutRepository $payoutRepository,
        private readonly OrderRepository $orderRepository,
        private readonly UserRepository $userRepository,
        private readonly SellerProfileRepository $profileRepository,
    ) {}

    private function computeBalance(User $user): array
    {
        $grossSales = $this->orderRepository->sumCompletedSalesForSeller($user->getId());
        $netEarned = round($grossSales * (1 - self::PLATFORM_FEE), 2);
        $alreadyPaid = $this->payoutRepository->sumAllocatedForUser($user->getId());
        $available = round(max(0, $netEarned - $alreadyPaid), 2);
    }
}

```

```

return [
    'grossSales' => round($grossSales, 2),
    'withdrawable' => $available,
];
}

#[Route("", methods: ['GET'])]
public function index(): JsonResponse
{
    /** @var User $user */
    $user = $this->getUser();
    $profile = $this->profileRepository->findByUser($user->getId());
    $payouts = $this->payoutRepository->findByUser($user->getId());
    $bal = $this->computeBalance($user);

    return $this->json([
        'balance' => $bal['grossSales'],
        'withdrawable' => $bal['withdrawable'],
        'platformFeePercent' => (int) (self::PLATFORM_FEE * 100),
        'savedIban' => $profile?->getIban()
            ? str_pad(substr($profile->getIban(), -4), strlen($profile->getIban()), '*',
STR_PAD_LEFT)
            : null,
        'savedPaypalEmail' => $profile?->getPaypalEmail(),
        'payouts' => array_map(fn(Payout $p) => $p->toArray(), $payouts),
    ]);
}

#[Route("", methods: ['POST'])]
public function create(Request $request): JsonResponse
{
    /** @var User $user */
    $user = $this->getUser();
    $data = json_decode($request->getContent(), true) ?? [];
    $amount = round((float) ($data['amount'] ?? 0), 2);

    if ($amount < 10.00) {
        throw new BadRequestHttpException('Minimum payout amount is €10.00.');
```

```

    }

    $bal = $this->computeBalance($user);
    if ($amount > $bal['withdrawable']) {
        throw new BadRequestHttpException(sprintf(
            'Requested amount exceeds withdrawable balance (€%.2f after %d%% platform fee).',
            $bal['withdrawable'],
            (int) (self::PLATFORM_FEE * 100)
        ));
    }
}

```

```

$method = $data['payoutMethod'] ?? 'bank';
if (!in_array($method, ['bank', 'paypal'], true)) {
    throw new BadRequestHttpException('Invalid payout method.');
```

```

}

if (!empty($data['iban']) || !empty($data['paypalEmail'])) {
    $profile = $this->profileRepository->findByUser($user->getId()) ?? new SellerProfile();
    $profile->setUser($user);
    if (!empty($data['iban'])) {
        $profile->setIban(strtoupper(preg_replace('/\s+/', '', $data['iban'])));
    }
    if (!empty($data['paypalEmail'])) {
        $profile->setPaypalEmail(trim($data['paypalEmail']));
    }
    $this->profileRepository->save($profile, false);
}

$payout = new Payout();
$payout->setUser($user);
$payout->setAmount(number_format($amount, 2, '.', ''));
$payout->setPayoutMethod($method);
$payout->setSellerNote(trim($data['sellerNote'] ?? '') ?: null);
$payout->setStatus('pending');

$this->payoutRepository->save($payout);

$newBal = $this->computeBalance($user);
return $this->json([
    'balance' => $newBal['grossSales'],
    'withdrawable' => $newBal['withdrawable'],
    'payout' => $payout->toArray(),
], 201);
}
}

```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

ДЕКЛАРАЦІЯ ЗДОБУВАЧА

Я, Страшнюк Анастасія Олегівна, здобувач(ка) НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота (проект) на тему «Веб-орієнтована інформаційна система продажу моделей для 3D-друку» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):
 - ☐ інструменти генеративного ШІ не використовувалися.
 - ☒ інструменти ШІ (вказати назву: ChatGPT, Gemini, Claude, DeepL, Chat GPT інші (вказати назву)) були використані для:
 - ☒ перекладу іншомовних джерел;
 - ☒ стилістичного редагування та перевірки граматики;
 - ☒ допомоги у написанні/відлагодженні програмного коду;
 - ☐ інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«22 » травня 2026 р.



(підпис)

Страшнюк Анастасія