МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

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

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Щепілової Карини Юріївни
(прізвище, ім'я, по батькові здобувача)

на тему **Розробка програмного забезпечення купівлі-продажу картин**

(повна назва теми)

за матеріалами праць провідних спеціалістів з розробки ПЗ та проектування БД

(повна назва бази дослідження)

науковий керівник **д.т.н., професор** _____ **Зеленський О. С.**
(наук. ступінь, вчене звання) *(підпис)* *(прізвище, ініціали)*

Робота допущена до захисту в ЕК
Протокол засідання кафедри
від 11 червня 2025 № 12
Завідувач кафедри _____
(підпис)
д.т.н., професор Зеленський О.С.
Наук. ступень, вчене звання *Ініціали, прізвище*

Кривий Ріг – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

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

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ

1. Тема роботи Розробка програмного забезпечення купівлі-продажу картин

Керівник роботи д.т.н., професор Зеленський О. С.
затверджені наказом закладу вищої освіти від «04» квітня 2025 р. № 222-ст

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

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

Розділ 1.
ПОСТАНОВКА ЗАДАЧІ
Розділ 2.
РОЗРОБКА АЛГОРИТМУ РОЗВ’ЯЗАННЯ ЗАДАЧІ
Розділ 3.
ОРГАНІЗАЦІЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ
Розділ 4.
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЗАДАЧІ
Об’єкт дослідження:
продаж та купівля картин
Предмет дослідження:
алгоритм продажу та купівлі картин
Мета кваліфікаційної роботи:
створити сайт за допомогою якого можна здійснювати купівлю та продаж картин

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

КАЛЕНДАРНИЙ ПЛАН

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

Завдання підготував науковий керівник _____ Зеленський О. С.

(підпис) (прізвище та ініціали)

Завдання одержала _____ Щепілова К. Ю.

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

на бакалаврську кваліфікаційну роботу

«Розробка програмного забезпечення купівлі-продажу картин»

Щепілової Карини Юріївни

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

Основним завданням дипломної роботи було створення, сайту, через який можна здійснювати купівлю, продаж та аукціони картин.

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

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

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

В роботі запропоновано новий підхід до автоматизації процесів роботи продажу картин, завдяки яким для роботи сервісу залучено мінімальну кількість працівників.

Ключові слова: інтернет-магазин, продаж картин, аукціон, Node JS, HTML, CSS, JavaScript.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД (база даних)	Впорядкований набір логічно взаємопов’язаних даних, що використовуються спільно та призначені для задоволення інформаційних потреб користувачів.
СУБД	Система управління базами даних.
ПЗ	Програмне забезпечення.
Node JS	Кросплатформенне середовище виконання JavaScript на сервері, яке використовує движок V8

ЗМІСТ

ВСТУП 8
РОЗДІЛ 1 ПОСТАНОВКА ЗАДАЧІ 10
1.1. Характеристика задачі 10
1.2. Вхідна інформація 12
1.3. Вихідна інформація 15
1.4. Вибір та обгрунтація технологій розробки 17
Висновки до розділу 1 19
РОЗДІЛ 2 РОЗРОБКА АЛГОРИТМУ РОЗВ'ЯЗАННЯ ЗАДАЧІ 21
2.1. Розробка алгоритму розв'язання задачі клієнтської частини 21
2.2. Розробка алгоритму розв'язання задачі серверної частини 23
Висновки до розділу 2 26
РОЗДІЛ 3 ОРГАНІЗАЦІЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ 27
3.1. Загальна характеристика інформаційного забезпечення 27
3.2. Структура баз даних та інформаційних масивів 29
Висновки до розділу 3 36
РОЗДІЛ 4 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЗАДАЧІ 38
4.1. Головна сторінка та структура навігації 38
4.2. Каталог картин та взаємодія з товарами 39
4.3. Аутентифікація та профіль користувача 41
4.4. Аукціони та покупка 45
4.5. Розробка 3-D галереї 49
Висновки до розділу 4 50
ВИСНОВКИ 52
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 54
ДОДАТКИ 55

ВСТУП

У сучасних умовах цифровізації суспільства розвиток електронної комерції має місце серед пріоритетних напрямів технологічного прогресу. Формування нових моделей бізнесу, що ґрунтуються на використанні Інтернету як основного середовища для здійснення торговельних операцій, призвело до значного поширення електронних платформ для продажу товарів і послуг. Однією з перспективних ніш є ринок мистецьких творів, де купівля, продаж та аукціон картин набувають нової форми завдяки використанню веб-ресурсів. Традиційні механізми обігу творів мистецтва поступово трансформуються під впливом цифрових технологій, що сприяє розробці сучасних веб ресурсів для забезпечення зручності, доступності та безпеки відповідних операцій.

Актуальність теми дослідження полягає в необхідності інтеграції художнього ринку у глобальний цифровий простір, де конкуренція між платформами стимулює пошук інноваційних підходів до організації онлайн-торгівлі мистецькими об'єктами. Розробка сучасного сайту для продажу, купівлі та аукціону картин дозволяє розширити аудиторію потенційних покупців і продавців художників, оптимізувати процеси взаємодії між ними, забезпечити високий рівень автоматизації та мінімізувати організаційні витрати. Водночас розвиток таких платформ сприяє підтримці митців, популяризації мистецтва та створенню нових можливостей для культурного обміну.

Існуючі рішення на ринку часто мають обмежений функціонал, недостатньо адаптовані до потреб користувачів або не відповідають сучасним вимогам безпеки даних та прозорості угод. Тому дослідження питань розробки веб-сайту, який поєднує можливості продажу, купівлі та проведення аукціонів картин, є своєчасним і має практичне значення як для розширення інструментів електронної комерції, так і для розвитку сфери мистецтва у цифровому середовищі.

Метою дослідження є розробка моделі веб-сайту для продажу, купівлі та аукціону картин, що забезпечує високу якість обслуговування користувачів,

надійність та ефективність процесів взаємодії між усіма сторонами торговельних операцій.

Для досягнення поставленої мети визначено такі завдання дослідження:

- провести аналіз існуючих рішень на ринку електронної комерції у сфері продажу мистецьких творів, виявити їх сильні та слабкі сторони.
- розробити вимоги до функціональності, безпеки та зручності використання сайту.
- побудувати архітектурну модель веб-ресурсу.
- реалізувати основні функціональні компоненти: систему реєстрації користувачів, каталог картин, механізми аукціону та продажу.
- розробити інтерфейс користувача відповідно до принципів UX/UI дизайну для забезпечення зручної взаємодії з системою.

Об’єктом дослідження є процес електронної торгівлі картин.

Предметом дослідження виступає сукупність технічних, організаційних та програмних рішень, що забезпечують створення веб-ресурсу для продажу, купівлі та аукціону картин.

Створення якісного веб-сайту для продажу, купівлі та аукціонів картин передбачає не лише вирішення технічних завдань, а й врахування особливостей художнього ринку, специфіки цільової аудиторії. Розробка повинна бути спрямована на формування довіри користувачів до платформ.

Результати роботи можуть бути використані для подальшого удосконалення інформаційних систем електронної комерції, вивчення тенденцій розвитку онлайн-ринку мистецьких творів та розробки рекомендацій щодо впровадження цифрових інновацій у галузі культури і мистецтва.

РОЗДІЛ 1
ПОСТАНОВКА ЗАДАЧІ

1.1. Характеристика задачі

Розробка веб-сайту для продажу, купівлі та аукціону картин передбачає вирішення комплексу взаємопов’язаних завдань, спрямованих на створення функціонального, безпечного та зручного у використанні електронного ресурсу. У межах проекту необхідно забезпечити організацію ефективної взаємодії між користувачами різних ролей: продавцями, покупцями та адміністраторами системи, із урахуванням специфіки обігу мистецьких об'єктів.

Сайт повинен підтримувати базові операції з реєстрації та аутентифікації користувачів, розміщення картин на продаж, участі в аукціонах, здійснення прямих покупок та комунікації між учасниками.

Необхідно забезпечити стабільність роботи платформи, запобігати втраті даних та збоїв під час проведення аукціону. Захист персональних даних, автентифікація користувачів, шифрування платіжної інформації та захист від несанкціонованого доступу є обов'язковими умовами функціонування сайту.

Інтерфейс користувача повинен бути не перевантаженим, забезпечувати швидкий доступ до основних функцій платформи, мінімізувати кількість дій для виконання операцій покупки та аукціону.

Архітектура системи має передбачати можливість розширення функціональності та збільшення навантаження зі зростанням кількості користувачів.

Для реалізації поставлених завдань обрано такі інструментальні засоби: JavaScript, Node.js та MongoDB.

JavaScript виступає основною мовою програмування для розробки клієнтської частини сайту. Завдяки широким можливостям цієї мови забезпечується динамічне оновлення контенту без перезавантаження сторінок, створення інтерактивних елементів інтерфейсу, реалізація перевірки введених

даних на стороні клієнта та забезпечення комфортної взаємодії користувача із системою.

Node.js обрано для розробки серверної частини сайту. Цей інструмент забезпечує високу продуктивність та здатність обробляти велику кількість одночасних запитів, що потребує комерційна система, де важливі швидкість обробки операцій купівлі, продажу та аукціонних ставок.

MongoDB використовується як основна система управління базами даних. Вибір даної NoSQL бази даних обумовлений її можливістю зберігання великих обсягів неструктурованих даних, високою продуктивністю при роботі з великими наборами інформації. Структура документів у форматі Binary JSON ідеально підходить для зберігання інформації про користувачів, картини, аукціонні лоти та історії угод.

Особлива увага в межах розробки приділяється забезпеченню прозорості процесу проведення аукціонів. Для цього потрібно реалізувати алгоритми ставок, які уникатимуть нечесні практики з боку учасників і гарантують достовірність результатів.

З огляду на зазначені вимоги, задачі розробки веб-сайту можна конкретизувати наступним чином:

- Проектування та створення клієнтської частини із забезпеченням динамічної взаємодії з сервером.
- Розробка серверної частини із підтримкою REST API для обробки запитів клієнтів.
- Побудова моделі бази даних забезпечення ефективного зберігання і пошуку інформації про товари мистецтва, користувачів і транзакції.
- Розробка механізму реєстрації, аутентифікації та авторизації користувачів з різними ролями (продавець, покупець).
- Реалізація функціоналу аукціонних торгів з можливістю встановлення початкової ціни, часу завершення аукціону та автоматичного визначення переможця.

1.2. Вхідна інформація

Функціонування веб-сайту для продажу, купівлі та аукціону картин базується на обробці різних типів вхідної інформації, що надходить від користувачів та продавців. Основними категоріями вхідної інформації є дані про користувачів, інформація про картини, дані аукціонних торгів, повідомлення чату та адміністративна інформація.

Інформація про користувачів містить персональні дані, які збираються під час реєстрації та використання сайту. До них належать ім'я користувача, логін, електронна пошта, пароль у зашифрованому вигляді та роль у системі. У профілі можуть також зберігатися додаткові відомості, зокрема зображення профілю. Вказані дані об'єднані у відповідній колекції бази даних «user», яка зберігає дані, які користувачі вказують у формах реєстрації або формах оновлення профілю (Рис. 1.1.).


```
_id: ObjectId('67efc93af26f48d48206b926')
firstName : "avtor"
lastName : "avtor"
nickname : "avtor"
email : "avtor@g.g"
userType : "artist"
password : "$2b$10$54rA/CtWNBZuOkKOKSfAi.GDfySf1gpI5VAGynPFJwGtawEseznMW"
createdAt : 2025-04-04T11:57:46.583+00:00
updatedAt : 2025-04-04T11:57:46.583+00:00
__v : 0
```


Рис. 1.1. Вхідні дані колекції «user»

Інформація про картини включає текстові та графічні дані, які надають продавці при створенні оголошення. Кожна картина супроводжується назвою, зазначенням логіну автора, коротким описом, категорією, типом продажу (аукціон чи звичайний продаж), ціною та відповідним зображенням. Відомості про статус картин (активна, продана, знята з продажу) зберігаються у відповідному полі документа бази даних. Вхідні дані містяться у колекції «products», де знаходяться всі перечислені дані (Рис. 1.2.).

Рис. 1.2. Вхідні дані колекції «products»

Аукціонні дані формуються у процесі торгів. Для кожної картини, що бере участь у торгах, зберігаються час початку та завершення аукціону, історія ставок, включаючи розміри ставок та ідентифікатори користувачів, що їх зробили, а також час коли було зроблено ставку. У структурі бази даних кожна ставка зберігається у колекції «bids», де ставка пов'язується з конкретним твором мистецтва та користувачем, що зробив ставку через унікальні ідентифікатори (Рис. 1.3.).

Рис. 1.3. Вхідні дані колекції «bids»

Обмін повідомленнями між користувачами здійснюється через вбудовану систему чату. Кожне повідомлення містить текст, часову мітку надсилання, ідентифікатори відправника та отримувача. Повідомлення зберігаються у відповідній колекції бази даних «messages» (Рис. 1.4.).

Рис. 1.4. Вхідні дані колекції «messages»

Нижче представлено табл. 1.1, у якій можемо побачити основні задачі програмного забезпечення, які отримують вхідну інформацію, з відповідним призначенням та періодичністю виконання:

Таблиця 1.1

Задачі програмного забезпечення

Найменування задачі	Призначення задачі	Періодичність виконання
Реєстрація користувача	Створення нового облікового запису	Під час реєстрації
Додавання картини	Розміщення твору мистецтва у каталозі або на аукціоні	За потреби користувача
Проведення аукціону	Організація торгів картинами	У встановлений період
Обмін повідомленнями	Комунікація між продавцем та покупцем	У режимі реального часу
Збереження історії ставок	Фіксація процесу участі у торгах	Під час проведення аукціону
Оновлення профілю користувача	Зміна особистих даних або інформації профілю	За ініціативи користувача
Відображення статусу картин	Інформування користувачів про наявність або продаж картин	Автоматично

Передбачено, що вхідна інформація зберігатиметься у базі даних. На основі аналізу вхідної інформації буде побудована структура майбутньої БД.

1.3. Вихідна інформація

Вихідна інформація є результатом обробки вхідних даних і забезпечує функціональність веб-сайту для продажу, купівлі та аукціону картин. Вона формується як у відповідь на запити користувачів, так і в результаті внутрішніх процесів системи. Вихідна інформація визначає ефективність взаємодії між користувачами та платформою, дозволяючи здійснювати навігацію, перегляд картин, участь в аукціонах і комунікацію між продавцями та покупцями.

Основні категорії вихідної інформації включають відображення профілів користувачів, каталогу картин, детальну інформацію про окремі товари, аукціонні дані в реальному часі та повідомлення чату.

При завантаженні головної сторінки користувач отримує сформований перелік доступних картин 10 художників з кращим рейтингом із назвою ціною та типом продажу. На сторінці окремої картини система генерує детальний опис із зазначенням характеристик товару та умов продажу або аукціону. На сторінці активного аукціону інформація оновлюється у реальному часі для відображення нових ставок та залишку часу до завершення торгів.

У процесі використання чату генеруються повідомлення, які відображаються обом сторонам розмови. Повідомлення передаються в режимі реального часу, забезпечуючи швидку комунікацію між продавцем і потенційним покупцем.

Нижче наведено табл. 1.2, у якій можемо побачити перелік і опис основних вихідних повідомлень платформи.

Вихідна інформація є інтегрованою частиною взаємодії користувача із системою та значною мірою визначає зручність і якість користування веб-сайтом.

Таблиця 1.2

Перелік і опис вихідних повідомлень

Назва вихідного повідомлення	Призначення задачі	Форма представлення	Термін і частота надходження
Список доступних картин	Інформування користувачів про наявні об'єкти	Список карток картин на сторінці	Під час кожного запиту користувача
Детальна інформація про картину	Надання повної інформації про обраний твір мистецтва	Окрема сторінка опису	Після вибору користувачем картини
Поточний стан аукціону	Відображення актуальних ставок та часу	Динамічне оновлення на сторінці	У режимі реального часу
Повідомлення чату	Забезпечення обміну інформацією між користувачами	Вікно чату	У режимі реального часу
Сповіщення про завершення аукціону	Інформування про результати аукціону	Спливаюче повідомлення або лист	Після завершення аукціону
Статус повідомлення у чаті	Відображення статусу (доставлено, прочитано)	Позначки біля повідомлення	У режимі реального часу
Повідомлення модератора	Інформування продавця про рішення щодо модерації	Повідомлення в акаунті користувача	Після перевірки контенту

Для забезпечення актуальності інформації використовуються технології асинхронного оновлення даних, що реалізовано за допомогою засобів JavaScript та Node.js та бібліотеки socket.io. Структура збереження вихідної інформації у MongoDB дозволяє оперативно формувати відповіді на запити користувачів, що

забезпечує високу продуктивність сайту при одночасному обслуговуванні великої кількості активних сесій.

1.4. Вибір та обгрунтація технологій розробки

У процесі розробки веб ресурсу для купівлі, продажу та проведення аукціонів картин було проведено процес вибору інструментів реалізації. Технологічний стек визначає архітектуру системи, швидкодію, безпеку, простоту підтримки та забезпечує зручність розширення у майбутньому. Тому вибір конкретних рішень відбувався не на інтуїтивному рівні, а в результаті аналізу характеристик альтернатив, а також порівняння з уже існуючими аналогами у сфері електронної комерції та онлайн-аукціонів.

Розробка динамічних веб ресурсів традиційно передбачає використання клієнтської та серверної частин, кожна з яких може бути реалізована з використанням різних технологічних стеків. Серед популярних фреймворків і мов для бекенду можна виокремити:

- PHP + MySQL — класичний стек для розробки сайтів, що використовується у багатьох системах управління контентом, таких як WordPress, OpenCart або Joomla.
- Python + Django / Flask + PostgreSQL — поєднання технологій для створення високорівневих веб-додатків з хорошими аналітичними можливостями.
- Node.js + MongoDB — сучасний асинхронний стек, що працює на єдиній мові (JavaScript) для клієнта і сервера, з особливою ефективністю при роботі з подієвими потоками і великим навантаженням.
- Ruby on Rails — ще одне фреймворк-орієнтоване рішення з розвиненим синтаксисом, популярне у стартап-середовищі.

Для клієнтської частини стандартом де-факто залишаються зв’язка HTML/CSS/JavaScript або фреймворки типу React, Vue.js, Angular, які працюють поверх JS.

У сфері вебаукціонів доцільно орієнтуватися на високонавантажені й динамічні моделі, де основну роль відіграє обмін даними в режимі реального часу. Такі вимоги потребують подієво-орієнтованої архітектури серверної частини, а також гнучкого документоорієнтованого підходу до збереження даних, що обґрунтовує вибір Node.js та MongoDB як основи платформи.

Node.js — це середовище виконання JavaScript, яке базується на рушії V8 від Google [13]. Його головною перевагою є неблокуюча, подієво-орієнтована модель введення-виведення. Це особливо актуально у випадках, коли система обробляє численні паралельні запити, що має місце в чатах, аукціонах і фільтрації товарів.

Node.js дозволяє створити легкий і продуктивний сервер, який обробляє події без витрат на блокування потоків. Це дає змогу досягти високої швидкодії навіть за значного навантаження. Крім того, використання однієї мови (JavaScript) для клієнта та сервера дуже спрощує підтримку і покращує інтеграцію логіки між двома частинами програми.

MongoDB — документоорієнтована система керування базами даних, яка зберігає інформацію у вигляді JSON-подібних документів. Її структура природно адаптована під гнучкі моделі даних, властиві вебплатформам із динамічно змінним вмістом. Зокрема, у розроблюваній системі товари, користувачі, аукціони та повідомлення мають неоднорідні структури, які в реляційних базах довелося б фрагментувати або зв’язувати через складні JOIN-запити.

MongoDB забезпечує високу продуктивність для операцій читання/запису та простий API. Вона також добре інтегрується з Node.js через офіційні драйвери та бібліотеки, щоб швидко і без надмірної конфігурації реалізовувати CRUD-операції.

HTML і CSS залишаються основою для створення структури та стилізації вебінтерфейсу. HTML дозволяє організувати логічну структуру сторінок: блоки товарів, аукціонів, форми авторизації тощо. CSS забезпечує адаптивність, привабливий дизайн та підтримку інтерактивних ефектів. У реалізації інтерфейсу також застосовано Flexbox та Grid-моделі для організації макетів,

медіа-запити для адаптації до екранів різних розмірів і кастомні змінні для збереження стилістичної цілісності.

JavaScript, у свою чергу, забезпечує інтерактивність сторінок: роботу фільтрів, пагінацію, асинхронне завантаження товарів, керування модальними вікнами, інтеграцію з WebSocket та REST API. Його використання на клієнті є органічним продовженням серверної логіки на Node.js, що зменшує часові витрати на узгодження даних між частинами та підключенння.

На основі порівняльного аналізу сучасних технологічних стеків, для реалізації платформи було обрано зв’язку Node.js + MongoDB як основу серверної частини та HTML/CSS/JavaScript — для клієнтської. Ці інструменти забезпечують оптимальне поєднання продуктивності, гнучкості, легкості масштабування та швидкості розробки.

Обраний стек відповідає вимогам проекту: обробка великої кількості подій у режимі реального часу, гнучкість структури даних, підтримка асинхронної взаємодії між користувачами, а також можливість легкого розширення функціоналу в майбутньому — зокрема інтеграції платіжних систем, аналітики або 3D-модулів. Технологічна однорідність дозволяє мінімізувати накладні витрати та оптимізувати процес розробки як для індивідуальних, так і командних форматів роботи.

Висновки до розділу 1

У межах даного розділу проведено комплексний аналіз задачі розробки веб-сайту для продажу, купівлі та аукціону картин. Визначено основні функціональні вимоги до програмного забезпечення, що забезпечують виконання поставленої мети та завдань дослідження. Охарактеризовано природу задачі, її структуру, особливості обробки даних та взаємодії між учасниками системи. Розглянуто склад вхідної інформації, яка включає персональні дані користувачів, відомості про картини, дані щодо ставок на аукціоні й повідомлення чату.

РОЗДІЛ 2
РОЗРОБКА АЛГОРИТМУ РОЗВ’ЯЗАННЯ ЗАДАЧІ

2.1. Розробка алгоритму розв’язання задачі клієнтської частини

Клієнтська частина веб-сайту для продажу, купівлі та аукціону картин розробляється із використанням мови програмування JavaScript у поєднанні з технологіями HTML і CSS. Робота клієнта забезпечується за допомогою браузерного середовища, що обробляє запити користувачів, здійснює візуалізацію даних та забезпечує інтерактивну взаємодію з серверною частиною через API-запити.

Механізми асинхронної взаємодії відбуваються за допомогою fetch API. Завдяки цьому користувачі отримують оперативний доступ до даних без необхідності перезавантаження сторінки. Використання JavaScript дозволяє реалізувати динамічну генерацію елементів інтерфейсу, оновлення контенту у реальному часі та забезпечення зворотного зв’язку у разі виконання дій користувачем.

Клієнтська частина також інтегрується з WebSocket-з’єднанням для обміну повідомленнями у чаті в реальному часі. WebSocket — протокол зв'язку поверх TCP-з'єднання, призначений обмінюватись повідомленнями між браузером і веб-сервером, використовуючи постійне з'єднання [1]. Це дає змогу миттєво передавати текстові повідомлення між учасниками без затримок, що є необхідною умовою для комунікації на платформі.

Алгоритм роботи клієнтської частини базується на обробці подій, ініційованих користувачем, та взаємодії із сервером. На початковому етапі користувач завантажує головну сторінку, яка містить список декількох картин. При виборі окремого об’єкта система надсилає запит до сервера для отримання детальної інформації про картину або поточний статус аукціону.

У процесі участі в аукціоні користувач має можливість надсилати ставки, які миттєво обробляються серверною частиною та призводять до оновлення

інформації на клієнтській стороні. Під час роботи з чатом передача повідомлень відбувається через встановлене WebSocket-з'єднання, що дозволяє підтримувати актуальний стан діалогу без необхідності ручного оновлення інтерфейсу.

У разі змін статусу картин або завершення аукціонів клієнтська частина отримує відповідні сповіщення або дані від сервера та оновлює інтерфейс відповідно до нових умов.

Розбір клієнтської частини сайту можна уявити як послідовність основних етапів, які відповідають за взаємодію користувача з інтерфейсом та обмін даними із сервером.

Загальний алгоритм роботи клієнтської частини можна представити у вигляді послідовних етапів:

- Ініціалізація та завантаження початкових даних.
- Обробка подій користувача.
- Надсилання запитів до серверної частини або передача повідомлень через WebSocket.
- Отримання та обробка відповіді від сервера.
- Оновлення елементів інтерфейсу відповідно до отриманої інформації.

На рис. 2.1. можемо побачити блок-схему алгоритму клієнтської частини сайту.

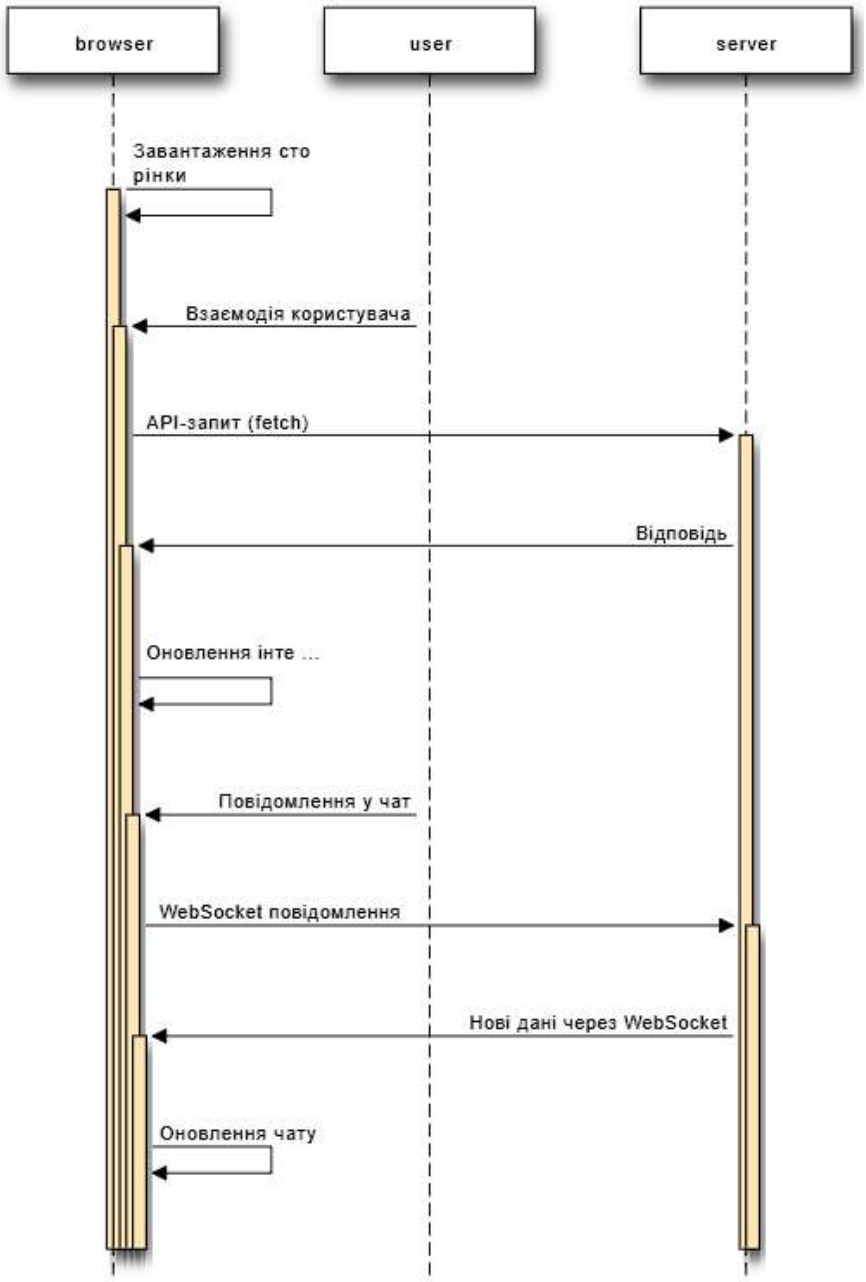

Рис. 2.1. Послідовність роботи клієнтської частини сайту

Ця діаграма ілюструє етапи взаємодії клієнта із сервером і відображає послідовність дій від завантаження даних до оновлення інтерфейсу.

2.2. Розробка алгоритму розв’язання задачі серверної частини

Серверна частина веб-сайту для продажу, купівлі і аукціону картин виконує роль посередника між клієнтськими запитами та базою даних. Її

функціонування забезпечено за допомогою середовища Node.js, яке дає змогу обробляти паралельні запити без блокування виконання процесів завдяки моделі введення-виведення.

На початковому етапі роботи сервер приймає запити користувачів через протокол HTTP або WebSocket-підключення. Далі здійснюється попередня перевірка отриманих даних: автентифікація користувача, перевірка відповідності даних очікуваному формату, виявлення потенційних загроз безпеці.

Після валідації запит обробляється відповідно до заданої логіки. Для отримання інформації про картини, користувацькі профілі або аукціонні сесії сервер звертається до бази даних MongoDB. Ця база даних обрана завдяки своїй документно-орієнтованій природі, яка дозволяє зберігати різнорідні за структурою об’єкти. Робота з MongoDB забезпечується через спеціалізовані драйвери Node.js, що мінімізує затримки у взаємодії з базою.

Особливістю архітектури є необхідність підтримки постійних з'єднань через WebSocket для чату та аукціонних процесів. У разі зміни даних сервер ініціює негайне розповсюдження оновлень усім підключеним клієнтам. Це забезпечує відображення актуального стану без затримки і виключає необхідність частого опитування сервера з боку клієнта.

Таким чином, алгоритм роботи серверної частини можна поділити на основні етапи:

- Прийом та попередня обробка запиту.
- Валідація автентичності та коректності даних.
- Виконання операцій із базою даних: зчитування, запис, оновлення.
- Формування відповіді або ініціювання подій через WebSocket.
- Логування операцій для моніторингу та подальшого аналізу.

На рис. 2.2. можемо побачити блок-схему алгоритму серверної частини сайту.

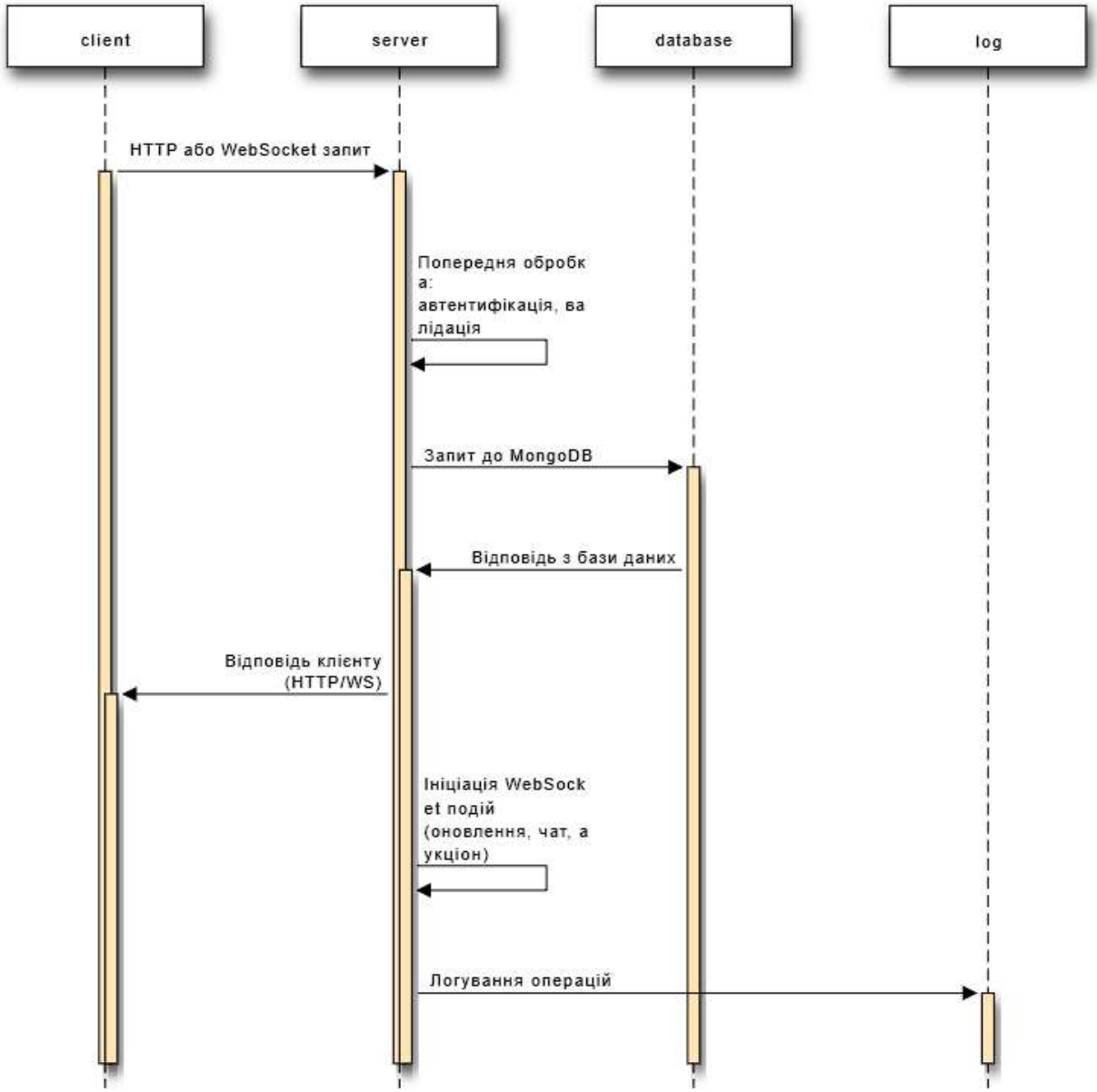

Рис. 2.2. Послідовність дій серверної частини

Сервер приймає запити від клієнта через протоколи HTTP або WebSocket, після чого виконує попередню обробку, що включає автентифікацію та валідацію даних. В разі потреби сервер звертається до бази даних MongoDB для зчитування або оновлення інформації. Отримані результати надсилаються назад клієнту у вигляді відповіді. Сервер ініціює події через WebSocket для підтримки оновлення інформації в реальному часі, для оновлень чату та аукціонів.

Висновки до розділу 2

У межах другого розділу було здійснено розробку алгоритмів роботи клієнтської та серверної частин веб-сайту для продажу, купівлі та аукціону картин. Проаналізовано технологічні основи реалізації системи, розглянуто використання JavaScript у клієнтській частині для забезпечення інтерактивної взаємодії та Node.js із MongoDB на серверній стороні для обробки запитів і зберігання даних.

Робота клієнтської частини базується на обробці подій користувача, асинхронній взаємодії із сервером та підтримці оновлення даних чату і аукціонів в реальному часі. Важливим аспектом є мінімізація часу відповіді та постійне оновлення інтерфейсу без перезавантаження сторінки.

Серверна частина реалізує обробку запитів, валідацію даних, взаємодію з базою даних і підтримку двостороннього з’єднання через WebSocket. Алгоритм роботи сервера забезпечує узгодженість стану системи, актуальність інформації для користувачів та можливість адаптування функціональності відповідно до потреб проекту.

Проведений аналіз створює концептуальну основу для переходу до наступного етапу роботи — безпосередньої розробки структурних елементів сайту, розробки бази даних та проектування користувацького інтерфейсу.

РОЗДІЛ 3
ОРГАНІЗАЦІЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Загальна характеристика інформаційного забезпечення

Інформаційне забезпечення сайту для продажу, купівлі та аукціону картин включає сукупність інформаційних ресурсів, способів організації даних, процедур обробки та актуалізації інформації. Якість і повнота інформаційного забезпечення безпосередньо впливають на функціональні можливості сайту, рівень зручності користування, швидкість виконання операцій та достовірність оброблюваної інформації.

Функціонування веб-ресурсу неможливе без визначення структури та змісту даних, що циркулюють у системі. Структура інформаційних потоків будується на взаємодії між користувачами, базою даних і сервером. Дані в системі поділяються на основні категорії: інформація про користувачів, інформація про картини, дані про аукціони та історію повідомлень чату.

Для забезпечення ефективного доступу до даних і їх обробки обрано використання бази даних MongoDB. Вона надає можливість гнучко організувати дані у вигляді документів JSON-формату, що особливо зручно для швидкого розширення або модифікації структури даних у разі зміни проекту. Відмова від жорстких реляційних схем дозволяє зберігати різнорідну інформацію без істотних втрат продуктивності системи.

Одним із принципів проектування бази даних стала мінімізація дублювання даних. Кожна колекція містить атрибути, які відповідають конкретному об’єкту. Для покращення цілісності даних використовується унікальна ідентифікація записів за допомогою автоматично згенерованих ідентифікаторів.

Користувачі взаємодіють із системою через введення, зміну і перегляд інформації. Це пояснює необхідність наявності механізмів валідації введених

даних на стороні клієнта і сервера, що забезпечує достовірність бази даних і захищає її від потенційних помилок або шкідливих втручань.

Структура інформаційного забезпечення можемо побачити у табл. 3.1 характеристик основних інформаційних об'єктів.

Таблиця 3.1

Опис таблиць бази даних

Назва таблиці	Ідентифікатор	Опис
Users	_id	Дані про користувачів сайту: • Ім'я користувача; • Електронна пошта; • Пароль (захешований); • Дата реєстрації; • Дата редагування акаунту • Роль у системі (продавець/покупець).
Paintings	_id	Інформація про картини: • Назва; • Автор; • Опис; • Вартість; • Тип продажу (аукціон/фіксований продаж); • Зображення. • Категорія
Bids	_id	Дані про ставку: • Ідентифікатор картини; • Поточна ціна; • Дата призначення ставки; • Ідентифікатор користувача, який зробив ставку.
Messages	_id	Повідомлення користувачів у чаті: • Відправник; • Отримувач; • Час відправлення; • Текст повідомлення.

Продовження таблиці 3.1

Chats	_id	• Масив об’єктів користувачів, учасників чату • Масив повідомлень • Дата створення • Дата оновлення

Інформація про картини включає як базові характеристики (назва, опис, ціна), так і додаткові атрибути (статус продажу, інформація про аукціон). Інформація про аукціони містить дані про поточні ставки, учасників та часові межі проведення торгів. Це дозволяє системі динамічно оновлювати статуси картин і забезпечувати користувачів актуальними відомостями без необхідності ручного оновлення сторінки. Повідомлення в чаті між користувачами зберігаються у вигляді окремих документів, для того щоб зберегти історію переписки, організовувати відновлення сеансів спілкування та будувати системи сповіщень.

3.2. Структура баз даних та інформаційних масивів

Розробка бази даних для інформаційної системи продажу, купівлі та аукціону картин передбачає побудову логічно завершеної моделі з урахуванням структуризації об'єктів, які відображають реальні процеси взаємодії користувачів у системі. Для реалізації зберігання інформації обрано документно орієнтовану систему управління даними MongoDB, що забезпечує гнучкість у роботі з різнотипними наборами даних та високу продуктивність при обробці запитів.

Кожна основна сутність предметної області представлена окремою колекцією документів. В межах системи визначено п’ять основних колекцій:

- Users — зберігання даних користувачів;
- Paintings — опис картин, що доступні для купівлі або участі в аукціонах;

- Auctions — фіксація параметрів проведення аукціонів;
- Messages — зберігання окремих текстових повідомлень та відправника;
- Chats — забезпечення можливості текстової комунікації між користувачами.

Структура документів у кожній колекції була побудована відповідно до вимог до мінімізації надлишковості даних і забезпечення необхідного рівня цілісності.

Кожне поле у документі має визначені характеристики, такі як тип даних, довжина, обов'язковість заповнення, наявність індексації та первинний ключ. Докладні характеристики полів наведено у відповідних таблицях до кожної колекції.

Колекція Users (табл. 3.2) є центральною у системі автентифікації та управління правами доступу. Вона містить унікальні облікові записи користувачів, що включають особисті дані, електронну адресу, зашифрований пароль та роль (покупець або продавець).

Таблиця 3.2

Колекція Users

Найменування	Поле	Тип поля	Довжина	Кількість знаків після десяткової точки	Первинний ключ	Умова на значення	Обов'язкове поле	Індексне поле
Номер	_id	ObjectId	—	—	+	Унікальність	+	+
Псевдонім	nickname	String	50	—	—	Не менше 3 символів	+	+

Продовження таблиці 3.2

Ім’я	firstname	String	50	—	—	Не менше 1 символу	+	+
Прізвище	lastname	String	50	—	—	Не менше 1 символу	+	+
Пошта	email	String	100	—	—	Формат електронної пошти	+	+
Пароль	password	String	256	—	—	Мінімальна довжина 8 символів	+	—
Дата створення	createtAt	Date	—	—	—	Дата у форматі ISO	+	—
Дата оновлення	updateAt	Date	—	—	—	Дата у форматі ISO	—	—
Тип користувача	userType	String	20	—	—	Значення: "buyer", "seller"	+	—

Колекція Prosucts (табл. 3.3) містить інформацію про твори мистецтва, представлені на платформі. Вона забезпечує зберігання даних про назву картини, автора, опис, ціну, статус та посилання на зображення.

Таблиця 3.3

Колекція Products

Найменування	Поле	Тип поля	Довжина	Кількість знаків після десяткової точки	Первинний ключ	Умовна значення	Обов'язкове поле	Індексне поле
Номер	_id	ObjectId	—	—	+	Унікальність	+	+
Назва	name	String	100	—	—	Не менше 1 символу	+	+
Автор	author	Object	50	—	—	Може бути порожнім для анонімів	—	+
Опис	description	String	500	—	—	Будь-який текст	—	—
Ціна	price	Decimal	10	2	—	Додатне значення	+	+
Статус	status	String	20	—	—	Значення: "In Stock", "Out of stock"	+	—
Посилання на фото	photo	String	200	—	—	Посилання на зображення	+	—

Продовження таблиці 3.3

Категорія	category	String	50	—	—	Не менше 1 символу	+	—
Тип продажу	SaleType	String	50	—	—	Значення: "fixed", "auction"	+	—

Колекція Bids (табл. 3.4) зберігає дані про ставку з аукціону для картин.

Таблиця 3.4

Колекція Bids

Найменування	Поле	Тип поля	Довжина	Кількість знаків після десяткової точки	Первинний ключ	Умова на значення	Обов'язкове поле	Індексне поле
Номер	_id	ObjectId	—	—	+	Унікальність	+	+
Картина	productId	ObjectId	—	—	—	Посилання на картину	+	+
Ставка	amount	Decimal	10	2	—	Більше ніж стартова ціна + крок ставки	+	—
Час	time	Date	—	—	—	Пізніше за start_date	+	—
Автор ставки	user	ObjectId	—	—	—	Ідентифікатор	—	—

						користу вача		

Кожен запис містить ідентифікатор відповідної картини, розмір ставки, дату призначення та інформацію про користувача, який зробив ставку.

Колекція Messages (табл. 3.5) виконує функцію зберігання особистих повідомлень між користувачами.

Таблиця 3.5

Колекція Messages

Найменування	Поле	Тип поля	Довжина	Кількість знаків після десяткової точки	Первинний ключ	Умова на значення	Обов'язкове поле	Індексне поле
Номер	_id	ObjectId	—	—	+	Унікальність	+	+
Відправник	sendeId	ObjectId	—	—	—	Ідентифікатор користувача	+	+
Отримувач	receivId	ObjectId	—	—	—	Ідентифікатор користувача	+	+
Повідомлення	message	String	1000	—	—	Текст повідомлення	+	—
Створено	createdAt	Date	—	—	—	Дата і час надсилання	+	—
Оновлено	updatedAt	Date	—	—	—	Дата і час оновлення	+	—

Вона дозволяє реалізувати внутрішню комунікацію без потреби у зовнішніх чат-платформах.

Колекція Chats (табл. 3.6) зберігає відомості про чат між 2 користувачами сайту. Вона включає масив усіх повідомлень між користувачами колекції Messages і масив самих користувачів-учасників чату.

Таблиця 3.6

Колекція Chats

Найменування	Поле	Тип поля	Довжина	Кількість знаків після десяткової точки	Первинний ключ	Умова на значення	Обов'язкове поле	Індексне поле
Номер	_id	ObjectId	—	—	+	Унікальність	+	+
Учасники	participants	Array	—	—	—	2 об’єкта користувачів	+	+
Отримувач	messages	Array	—	—	—	—	—	+
Створено	createdAt	Date	—	—	—	Дата і час надсилання	+	—
Оновлено	updatedAt	Date	—	—	—	Дата і час оновлення	+	—

Хоча MongoDB не є класичною реляційною базою даних, логічні зв'язки між об'єктами все ж моделюються через зберігання ідентифікаторів суміжних документів.

Встановлено такі ключові логічні зв'язки:

- Між Users та Products: кожна картина належить певному користувачу (продавцю). В полі документу картини зберігається ідентифікатор користувача, який її додав.
- Між Users та Messages: кожне повідомлення містить ідентифікатор відправника та одержувача, що забезпечує можливість відстеження комунікації між користувачами.
- Між Users та Chats: кожен чат містить відправника та одержувача, які являються користувачами сайту, що забезпечує можливість відстеження комунікації між користувачами.
- Між Messages та Chats: кожен чат складається з масиву відправлених користувачами повідомлень. Колекція чату зберігає масив ідентифікаторів повідомлень.
- Між Products та Bids: аукціон обов'язково прив'язаний до конкретної картини. В полі документу аукціону зберігається ідентифікатор картини, що бере участь у торгах.
- Між Users та Bids: в процесі проведення аукціону зберігається ідентифікатор користувача, який зробив ставку.

Висновки до розділу 3

У даному розділі було розроблено структуру бази даних та здійснено детальний опис інформаційних масивів, що забезпечують функціонування сайту для купівлі, продажу та участі в аукціонах картин. Застосування документоорієнтованої моделі MongoDB пояснено потребою у гнучкому зберіганні неоднорідних даних та швидкій обробці великої кількості запитів.

В результаті аналізу предметної області визначено основні об’єкти інформаційної системи: користувачі, картини, аукціони та повідомлення. Для кожної з цих сутностей сформовано окремі колекції з відповідною структурою полів, що враховує типи даних, обмеження цілісності, обов'язковість заповнення та індексацію для оптимізації пошуку.

Розглянуто основні логічні зв'язки між об'єктами бази даних, які реалізуються за допомогою зберігання у документах ідентифікаторів суміжних сутностей. Така організація структури забезпечує збереження слабкої зв'язності між колекціями, що є характерним для NoSQL-підходу та сприяє підвищенню адаптивності системи.

Створена структура бази даних дозволяє підтримувати основні функціональні вимоги розроблюваного вебресурсу, реєстрацію та автентифікацію користувачів, відображення товарів, організацію аукціонних торгів та забезпечення внутрішньої комунікації між учасниками.

Структуризація інформаційних масивів є логічно завершеною, обґрунтованою з погляду потреб системи і придатною для подальшої інтеграції у процес розробки клієнтської та серверної частин програмного забезпечення.

РОЗДІЛ 4
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЗАДАЧІ

4.1. Головна сторінка та структура навігації

Після переходу на вебресурс користувач потрапляє на головну сторінку (Рис. 4.1.), яка виконує ознайомчу та орієнтувальну функцію [Додаток А]. У верхній частині інтерфейсу розміщено привітальний блок, що містить стислий опис призначення платформи — забезпечення зручного простору для представлення, перегляду, купівлі та продажу картин, а також участі в аукціонах. Цей блок також містить кнопку для переходу до більш детальної інформації про функціональні можливості сайту.

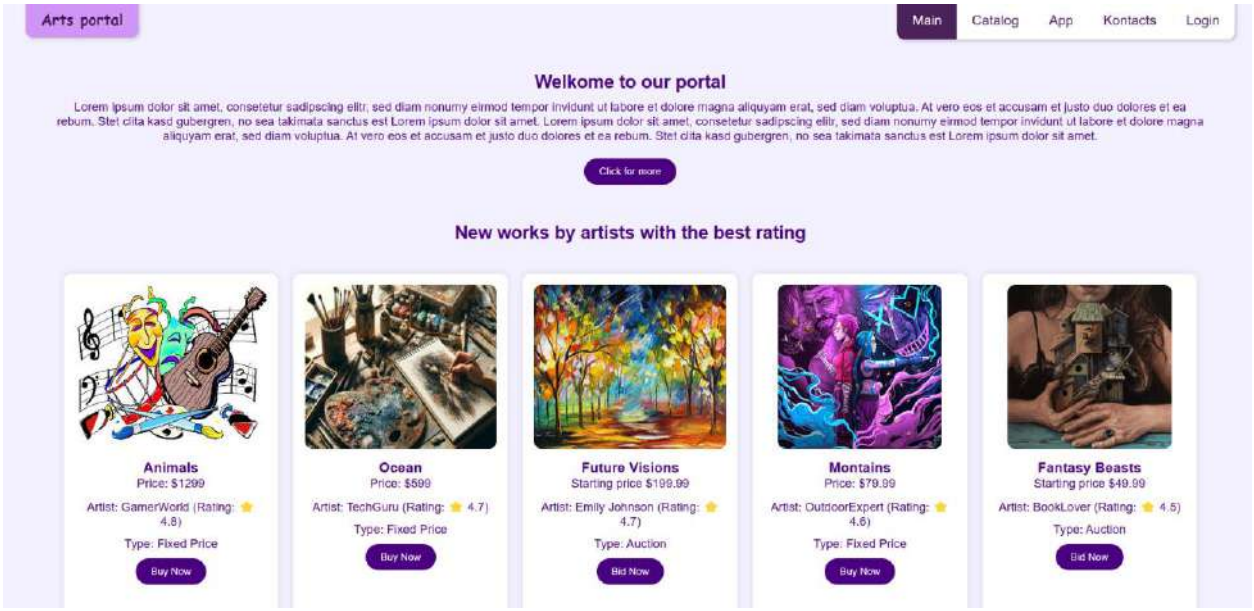

Рис. 4.1. Головна сторінка сайту

Нижче по структурі сторінки знаходиться інтерактивний компонент, що відображає список нових товарів. Система формує добірку автоматично на основі дати публікації та рейтингу художників: у цьому блоці виводяться нові роботи десяти авторів, які мають найвищий середній показник відгуків [Додаток Б]. Таким чином, головна сторінка виконує роль динамічного вхідного інтерфейсу, де користувач одразу бачить оновлення контенту та отримує перше

уявлення про якість робіт. В той же час, художники одразу можуть побачити спосіб просування своїх робіт через головну сторінку, методом отримання гарних оцінок та відгуків.

Навігаційна система побудована у вигляді горизонтального меню, що розміщене в шапці сайту (Рис. 4.2.). Вона залишається доступною незалежно від прокрутки, забезпечуючи постійний доступ до основних функціональних розділів. При авторизованому доступі останній пункт автоматично змінюється на ім’я користувача з можливістю переходу до особистого кабінету. При наведені на пункт з’являється випадаюче меню, через яке користувач може вийти із системи (Рис. 4.3.).

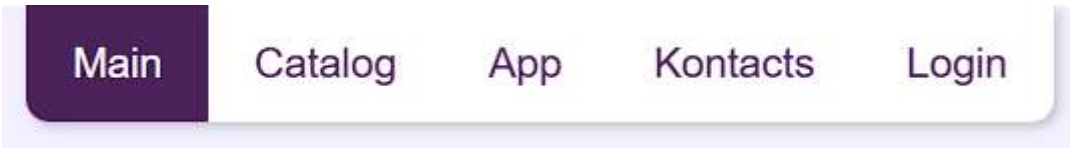

Рис. 4.2. Меню сайту

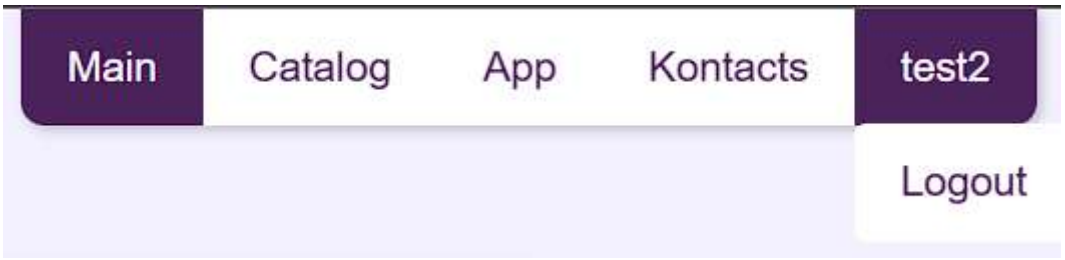

Рис. 4.3. Меню авторизованого користувача

До складу меню входять наступні пункти: «Головна», «Каталог», «Додаток», «Контакти» та «Логін / Профіль».

4.2. Каталог картин та взаємодія з товарами

Інтерфейс каталогу складається зі списку карток товарів, кожна з яких містить зображення картини, назву, ім’я автора, короткий опис, вартість, рейтинг, тип аукціону і кнопку «Купити» або «До аукціону», в залежності від типу продажу. Розміщення елементів реалізовано у вигляді сітки з підтримкою

адаптивності для коректного відображення на мобільних пристроях. На Рис. 4.4. можемо побачити загальний вигляд сторінки каталогу.

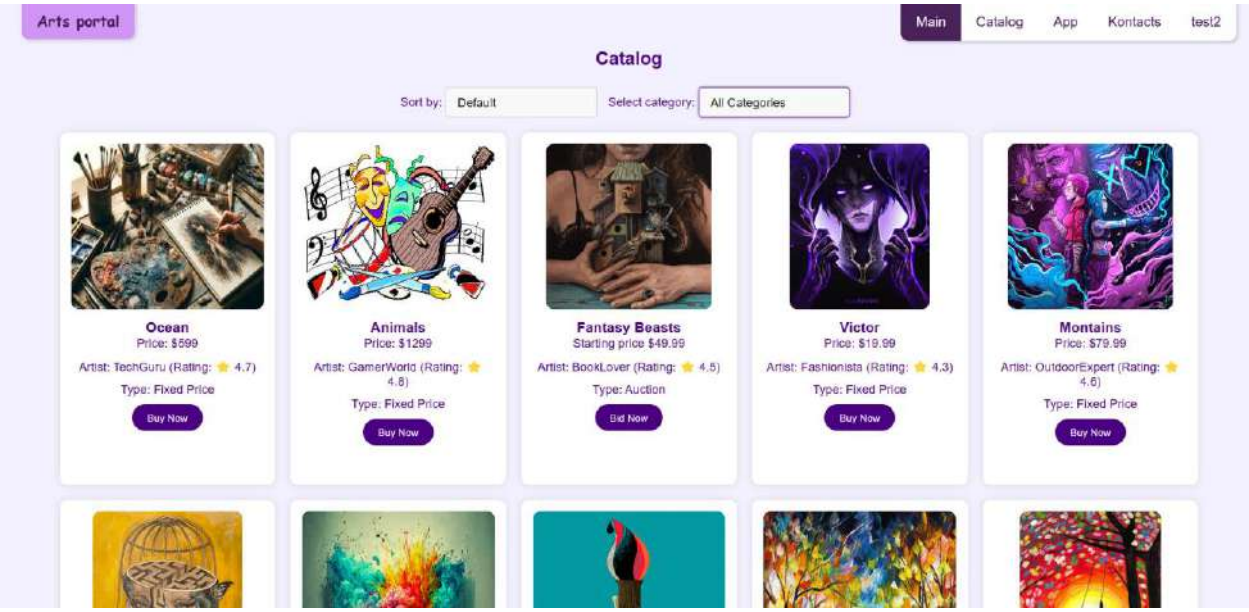

Рис. 4.4. Каталог картин

Для полегшення навігації серед великої кількості картин реалізовано механізм фільтрації. Користувач має можливість обрати критерії за категоріями, ціною та рейтингом. Застосування фільтрів не потребує повного перезавантаження сторінки — результати оновлюються динамічно, завдяки використанню асинхронних запитів до серверної частини.

Окремою функціональною складовою є система пагінації, яка забезпечує поступове завантаження контенту. Це дозволяє уникнути перевантаження на клієнтську частину та покращує продуктивність при роботі з великими обсягами даних.

При натисканні на будь-яку картку користувач перенаправляється на сторінку конкретного товару (Рис. 4.5.). У межах цієї сторінки надається розширена інформація: повне зображення картини у високій роздільній здатності, ціна, ім’я та рейтинг автора, а також відгуки про автора. Якщо товар бере участь в аукціоні, користувач може перейти до відповідної сторінки зі ставками. Якщо товар виставлено просто на продаж, користувач

перенаправляється до чату з автором. Ці дії реалізуються за допомогою відповідних кнопок «Купити» або «До аукціону».

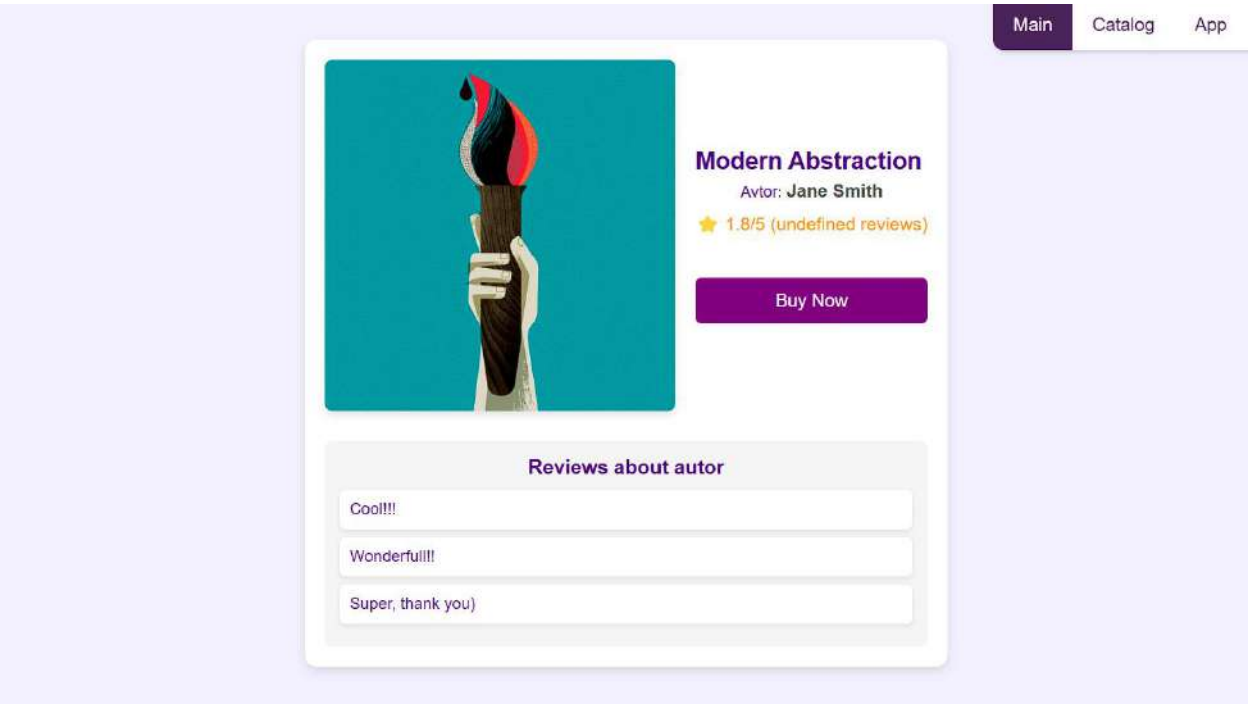

Рис. 4.5. Сторінка товару

Сайт надає можливість скачати інтерактивний десктоп додаток 3д галереї. Це зовнішній модуль, що дозволяє здійснити віртуальну прогулянку галереєю з роботами топ-художників. Сторінка має коротке відео-представлення додатку, для кращого розуміння користувачів, та посилання на безпосередньо завантаження програми.

4.3. Аутентифікація та профіль користувача

Функціонал аутентифікації на вебресурсі реалізовано з метою персоналізації взаємодії користувачів із системою, розмежування прав доступу та забезпечення можливості здійснення дій, пов’язаних з публікацією, редагуванням або придбанням картин.

Інтерфейс сторінки реєстрації (Рис. 4.6.) передбачає введення базового набору інформації: ім’я, прізвище, нікнейм, електронна пошта, пароль, підтвердження пароля, тип користувача (покупець чи продавець). Передбачено

як клієнтську перевірку даних, таких як довжина пароля, відповідність полів (Рис. 4.7.), так і серверну валідацію, таку як перевірка на унікальність електронної адреси (Рис. 4.8.).

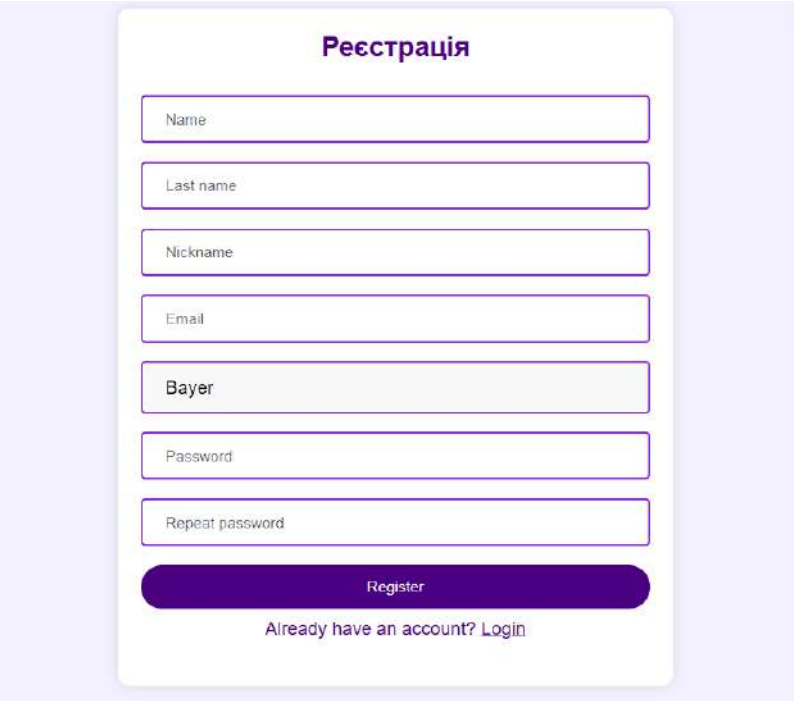

Рис. 4.6. Форма реєстрації

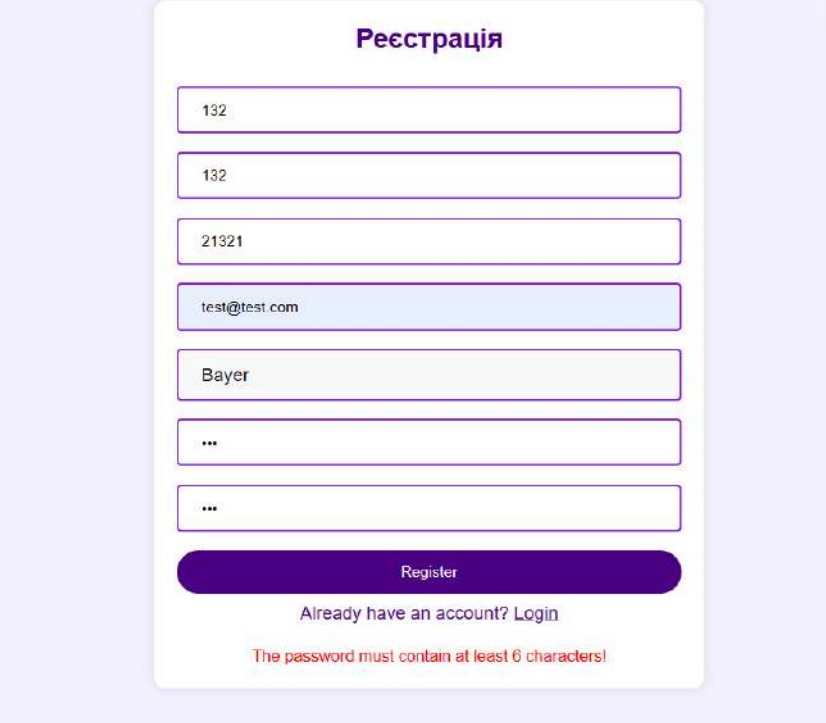

Рис. 4.7. Повідомлення про невірно введений пароль

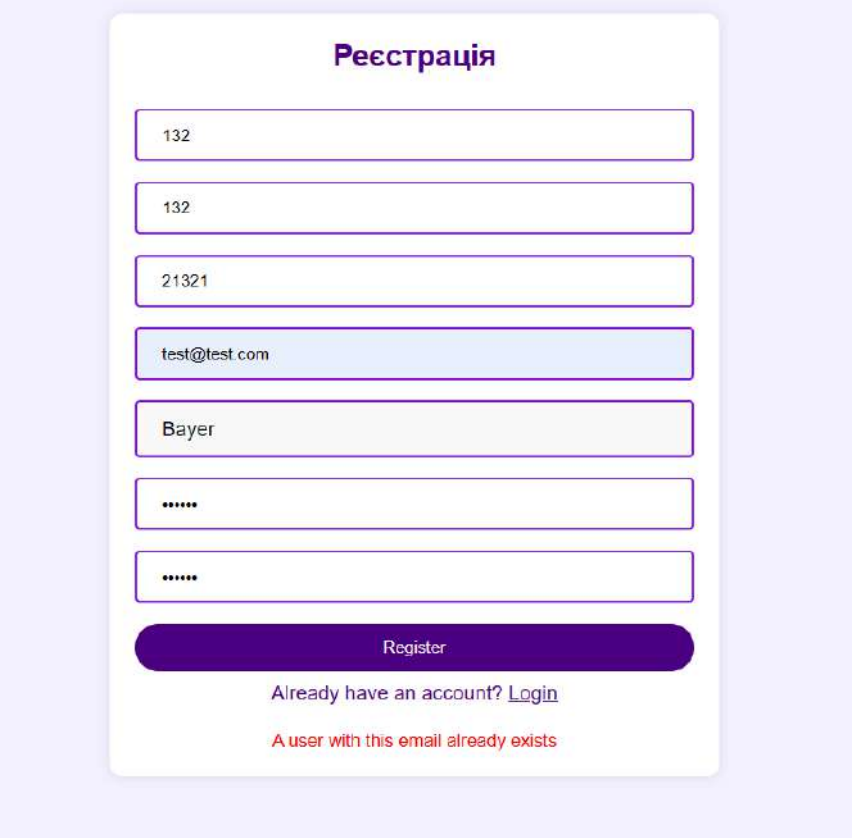

Рис. 4.8. Повідомлення про введення зареєстрованої пошти

У разі успішної реєстрації користувач перенаправляється на сторінку авторизації. Тут користувач повинен ввести пошту та пароль для успішної авторизації (Рис. 4.9.)

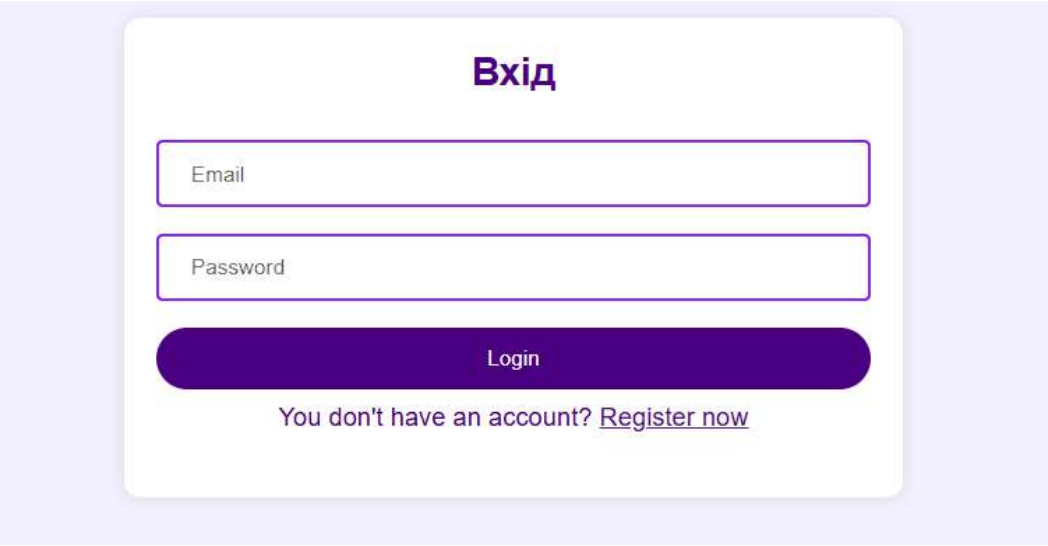

Рис. 4.9. Форма входу

Після авторизації стає доступним профіль користувача (Рис. 4.10.), який слугує персональним простором для керування активністю на платформі. У профілі відображається:

- персональна інформація (ім’я, фото, нік);
- перелік опублікованих користувачем картин;
- портфоліо;
- зараз у продажу;
- збережені чи вподобані роботи;
- відгуки.

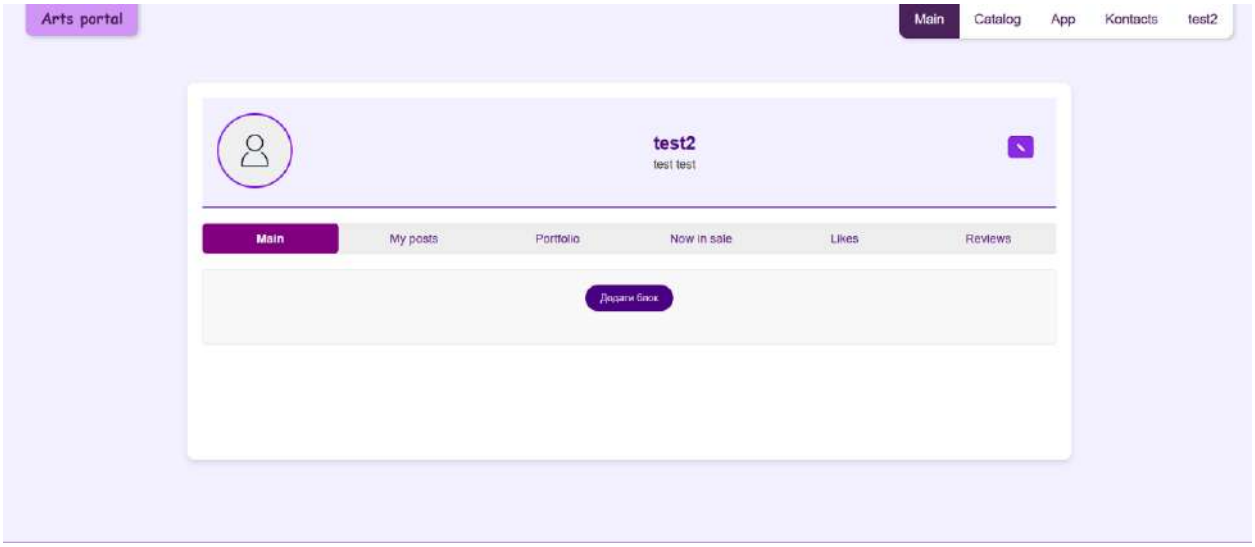

Рис. 4.10. Профіль користувача

Користувач може перейти на окрему сторінку редагування профілю (Рис. 4.11.), де дозволяється змінювати персональні дані, пароль, додавати аватар. Для оновлення даних застосовується механізм попередньої перевірки значень та підтвердження змін.

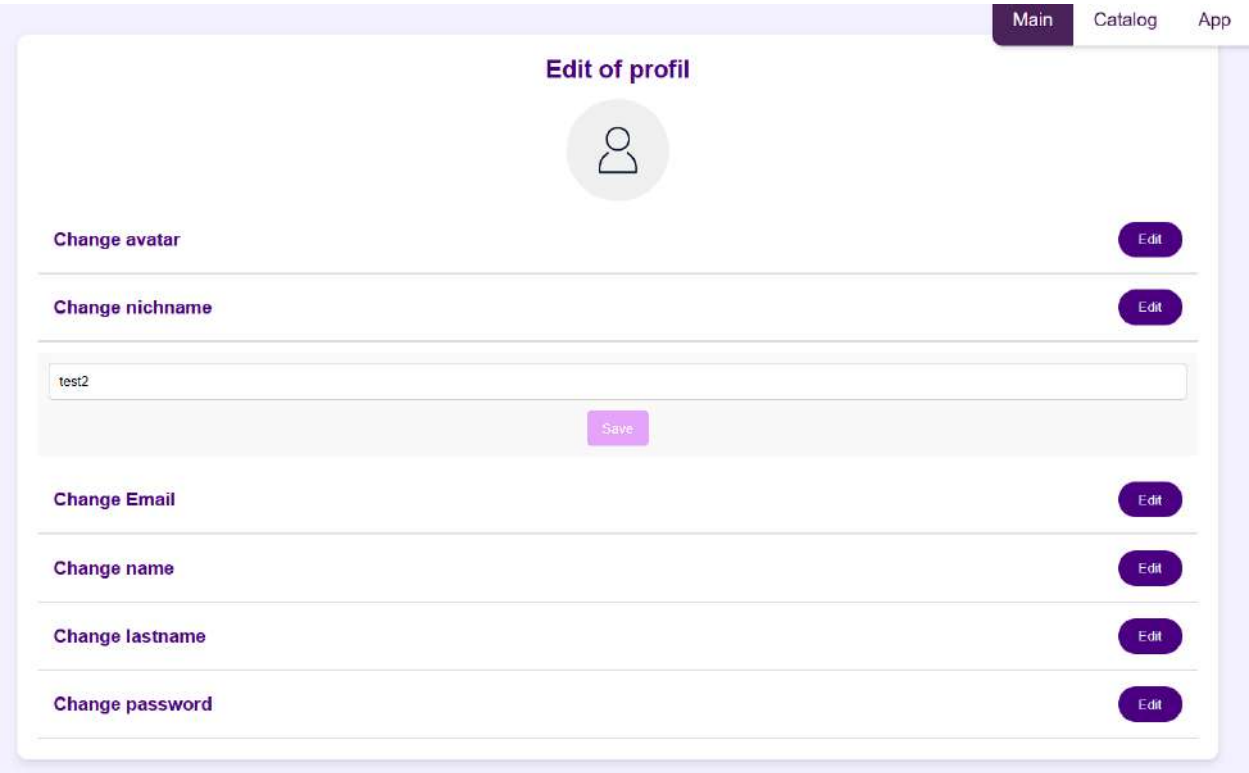

Рис. 4.11. Редагування профілю

Серверна частина аутентифікації реалізована з використанням захищених сесій. Паролі зберігаються у хешованому вигляді за допомогою бібліотеки bcrypt, що забезпечує додатковий рівень безпеки. Кожен користувач має унікальний ідентифікатор у базі даних, що використовується для зв’язування з іншими сутностями системи: роботами, повідомленнями, ставками на аукціоні.

4.4. Аукціони та покупка

Система аукціонів реалізована як інтегрована функція веб ресурсу. Завдяки ній користувачі можуть брати участь у конкурентному процесі придбання картин [Додаток В]. Вона надає інструменти для організації торгів, розміщення ставок та відстеження результатів у реальному часі. На відміну від фіксованих цін у звичайному каталозі, аукціонна модель передбачає поступове підвищення вартості товару, ініційоване діями користувачів.

Інтерфейс сторінки аукціону (Рис. 4.12.) містить зображення картини, ім’я автора, поточну найвищу ставку, таймер до завершення торгів, крок ставки а

також кнопку для того, щоб здійснити нову ставку. Для кожної ставки зберігається інформація про суму, час подачі та користувача, який її зробив. Лише зареєстровані користувачі можуть брати участь в аукціоні.

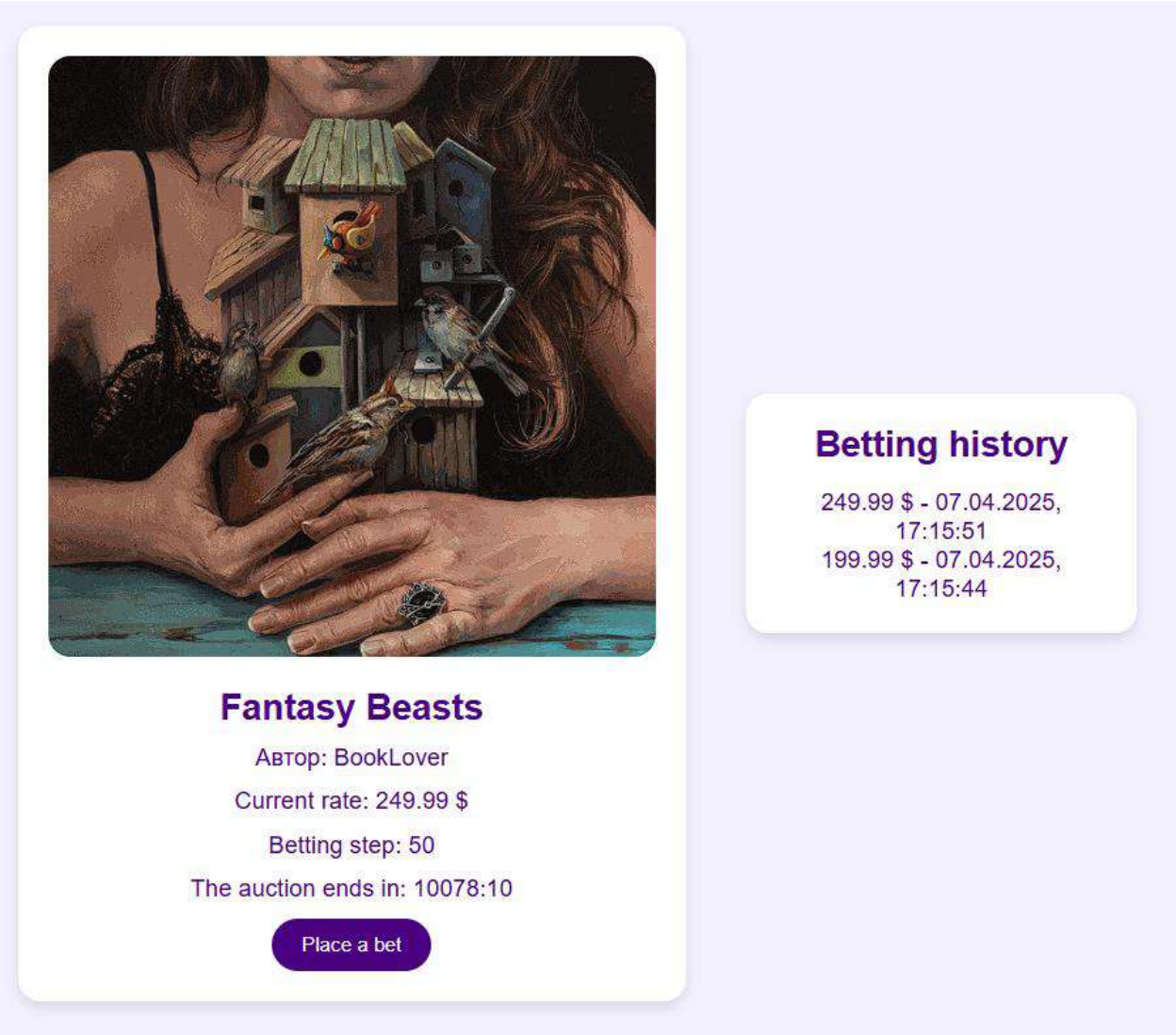

Рис. 4.12. Сторінка аукціону

Після натискання кнопки «Встановити ставку» активується модальне діалогове вікно, яке слугує інтерфейсом взаємодії з механізмом аукціонного підвищення ціни (Рис. 4.13.). У вікні відображається поточна найвища ставка, а також елемент керування у вигляді повзунка, який дозволяє користувачу вибрати нове значення ставки. Початкове положення повзунка обчислюється програмно та відповідає сумі останньої зафіксованої ставки й кроку аукціону — це забезпечує дотримання правил мінімального підвищення ціни.

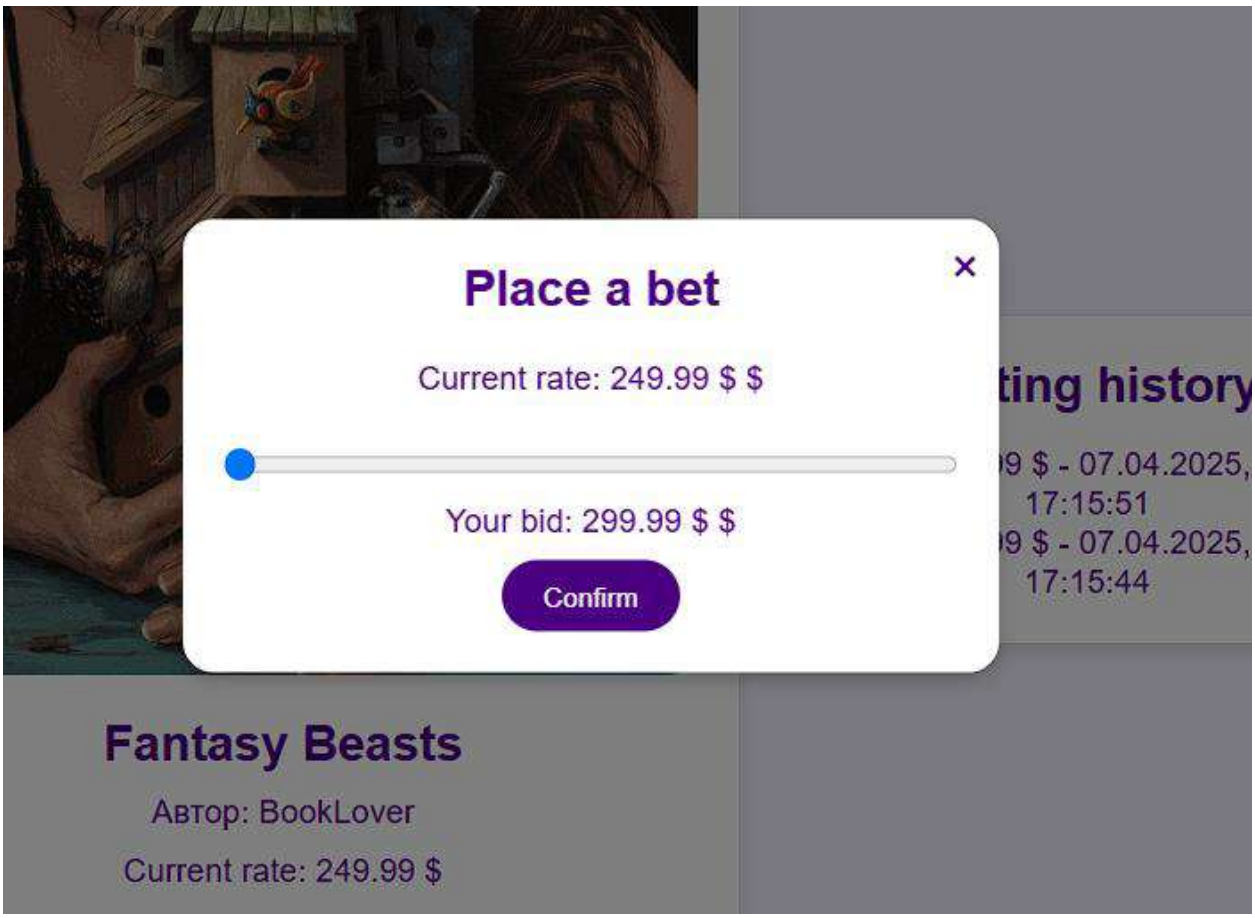

Рис. 4.13. Діалогове вікно, для додавання ставки

У реальному часі, зі зміною положення повзунка, динамічно оновлюється поле з відображенням обраної суми ставки, що дозволяє користувачу негайно бачити результат взаємодії. Передбачено також кнопку «Погодитись», натискання якої підтверджує внесення ставки. Після цього вікно автоматично закривається, а нова ставка додається до списку поточних учасників у режимі реального часу, без перезавантаження сторінки, що реалізовано через WebSocket-з’єднання (Рис. 4.14.).

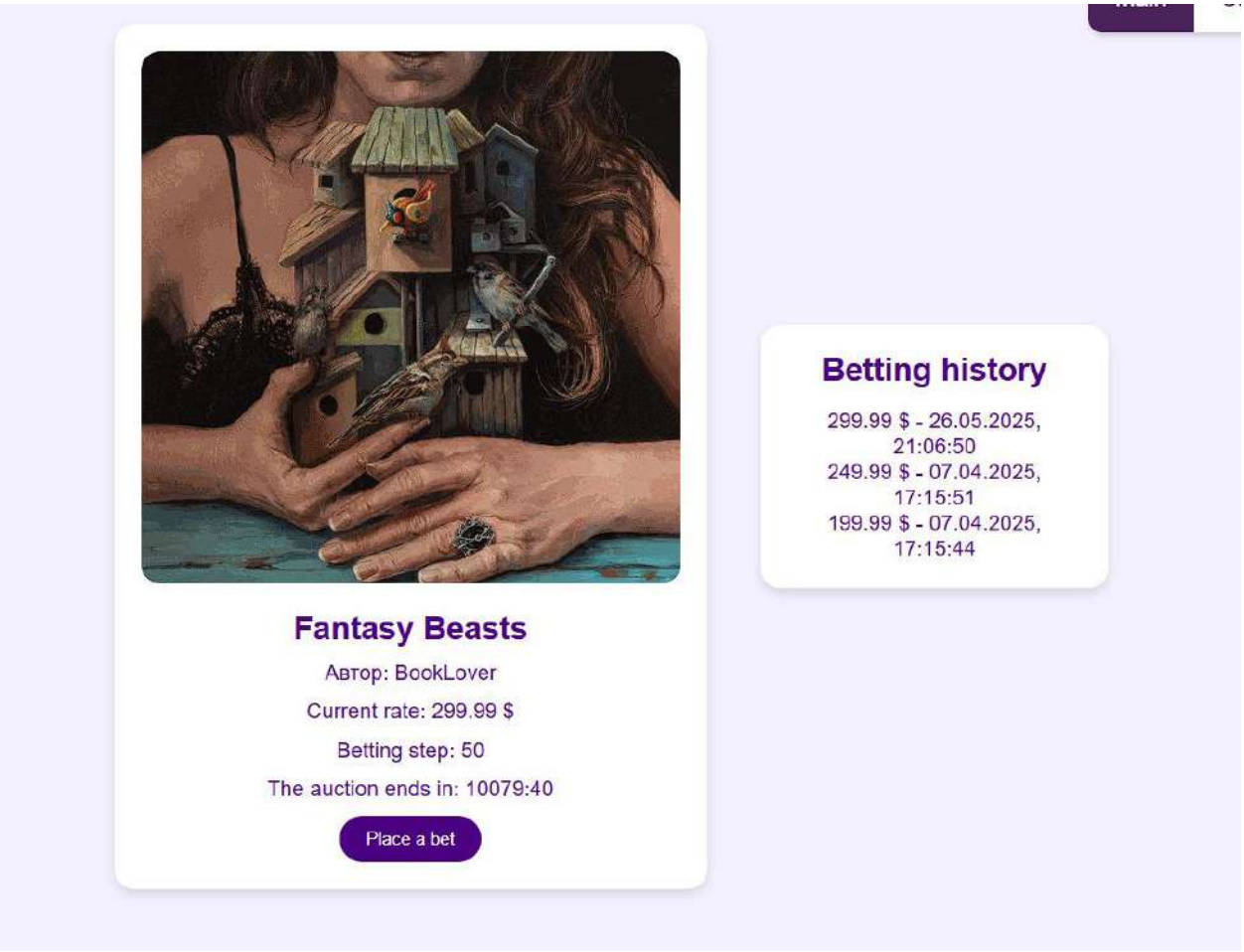

Рис. 4.14. Нова ставка відображається на сайті

Інтерфейс чату реалізований у вигляді окремої сторінки з централізованим розміщенням основного вікна діалогу. Центральну частину сторінки займає область чату. Угорі розміщено заголовок, який вказує на те, що користувач перебуває в модулі обміну повідомленнями. Основна частина вікна — це поле відображення повідомлень. Повідомлення поділяються за відправником: ті, що надіслав поточний користувач, мають вирівнювання праворуч і відрізняються стилістично, наприклад кольором фону або рамкою. Отримані повідомлення вирівнюються ліворуч (Рис. 4.15.).

Нижня частина інтерфейсу містить поле введення тексту, де користувач може набрати нове повідомлення. Праворуч від поля розміщено кнопку для надсилання повідомлення. Весь процес взаємодії з чатом реалізований таким чином, щоб забезпечити плавну та безперебійну комунікацію. Повідомлення

з’являються в реальному часі завдяки технології вебсокетів, що дозволяє миттєво відображати нові записи без потреби оновлення сторінки.

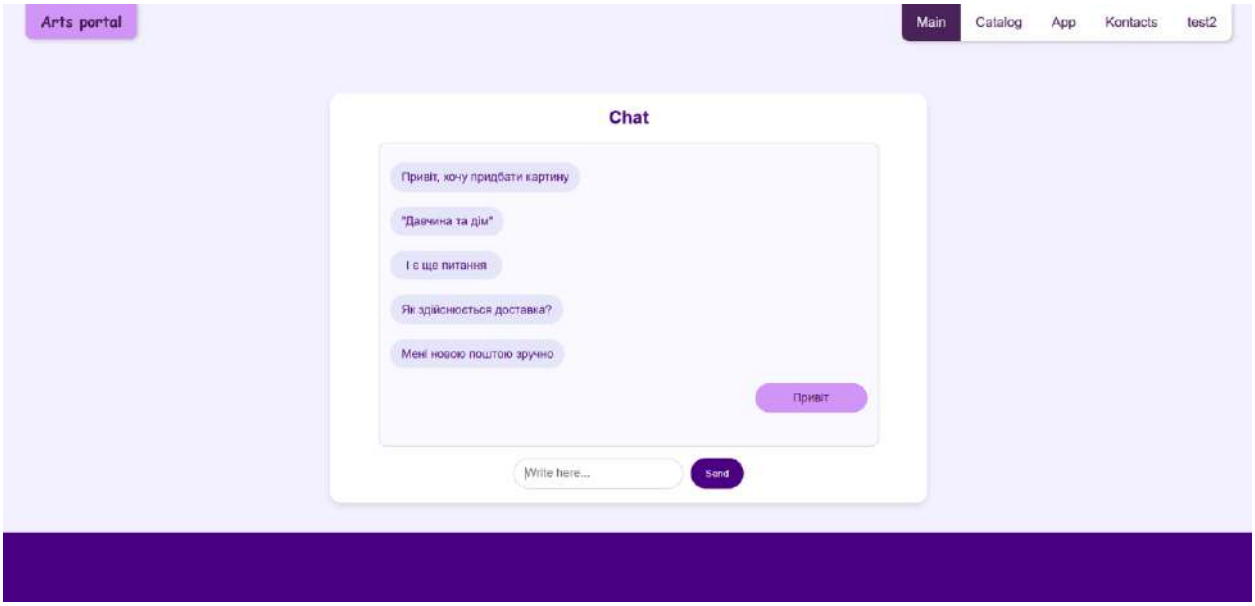

Рис. 4.15. Сторінка чату на сайті

Після завантаження сторінки програма отримує ідентифікатор поточного діалогу, який вбудовується в URL-адресу. Це дозволяє однозначно ідентифікувати бесіду. Якщо діалогова сесія вже існує, вона завантажується разом з усією історією повідомлень. Якщо чат створюється вперше, для нього ініціалізується нова структура даних, яка об'єднує двох учасників. Повідомлення передаються через сервер і миттєво розповсюджуються між клієнтами, що беруть участь у сесії. Усі повідомлення зберігаються на сервері. Таким чином завжди надається можливість переглянути минулі повідомлення про домовленості, та зафіксувати умови покупки картини.

4.5. Розробка 3-D галереї

Програма реалізована у вигляді Windows Forms-додатку з інтегрованим OpenGL, який відповідає за рендеринг тривимірної сцени (Рис. 4.16.). Центральним елементом інтерфейсу є головне вікно, в якому розміщено компонент рендеренгу. Саме через нього відбувається виведення віртуальної

галереї — кімнати, оформленої у вигляді виставкового простору з підлогою, стелею та чотирма стінами. По периметру цих стін розташовуються картини.

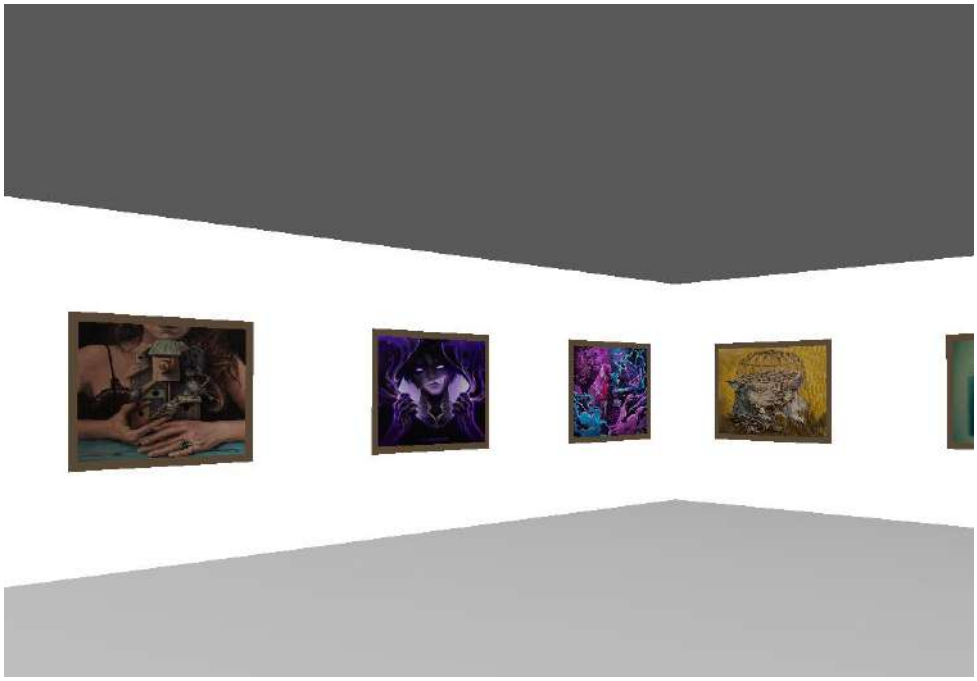

Рис. 4.16. 3-D галерея

Пересування користувача в межах сцени реалізоване у форматі камери від першої особи. Навігація здійснюється за допомогою клавіш W, A, S, D, які відповідають за рух уперед, назад та в сторони. Обертання погляду реалізовано через клік миші.

Для кожної картини створюється відповідний OpenGL-текстур. Зображення завантажуються з бази даних MongoDB. У разі її недоступності завантажується картина з демонстраційних джерел. При підключенні до MongoDB застосунок витягує до 20 картин з колекції, віддаючи пріоритет авторам з найвищим рейтингом.

Візуальне оформлення доповнене налаштуванням освітлення. Оновлення сцени відбувається регулярно завдяки таймеру, який кожні 16 мс викликає перерисовку вікна. Додатково використовується механізм обробки, який слідкує за натисканням клавіш для безперервного керування.

Висновки до розділу 4

У межах четвертого розділу було розроблено інтерфейст та функціональність веб ресурсу купівля-продажу картин. Було описано реалізоване програмне забезпечення, розробленого для забезпечення функціонування вебсистеми з купівлі, продажу та аукціонів художніх робіт. Розглянуто основні інтерфейсні компоненти та функціональні модулі, з якими взаємодіють користувачі веб ресурсу. Конкретно користувач взаємодіє з головною сторінкою, каталогом товарів, механізмами фільтрації та навігації, системою аутентифікації, профілем користувача, сторінками картин і аукціонів, а також вбудованою чат-системою для взаємодії покупців із продавцями.

Було описано логіку реалізації основних функцій клієнтської частини, зокрема механізми фільтрації та пагінації каталогу, відображення товарів і динамічних елементів, обробку подій та взаємодію з сервером. Здійснено аналіз архітектури аутентифікації, авторизації та обробки користувацьких сесій. Надано пояснення поведінки інтерфейсу під час участі користувача в аукціоні, описано принципи роботи динамічних розрахунків ставки та миттєвого оновлення результатів.

Окрему увагу приділено розробці логіки взаємодії з користувачем у межах реального часу, що реалізується без використання складних сторонніх технологій — засобами WebSocket і асинхронних оновлень, у межах обраного технологічного стеку.

ВИСНОВКИ

У межах дипломної роботи було здійснено повний цикл розробки інформаційної системи у вигляді вебплатформи для купівлі, продажу та участі в аукціонах картин. Вирішення поставленої задачі відбувалося на основі глибокого аналізу предметної області, сучасних технологічних рішень та функціональних вимог, характерних для електронної комерції у сфері мистецтва.

У теоретичній частині було обґрунтовано актуальність створення платформи, що поєднує в собі механізми продажу, демонстрації та аукціонної торгівлі. Проаналізовано особливості функціонування аукціонних систем, механіку ставок, структуру учасників процесу. У результаті було сформовано вимоги до архітектури, інтерфейсу та функціоналу системи.

На основі порівняльного аналізу сучасних технологічних рішень обрано стек розробки, що включає Node.js, MongoDB, JavaScript, HTML та CSS. Це дозволило реалізувати складний функціонал системи, асинхронної обробки подій, інтеграції чату та динамічної взаємодії з користувачем. Вибір саме цих технологій зумовлено їхньою ефективністю у побудові адаптивних, подієво-орієнтованих вебресурсів. Це було підкріплено аналітикою інструментів та дослідженням аналогів.

Під час розробки системи було реалізовано поставлені задачі та реалізовано такі частини вебресурсу:

- головна сторінка з навігацією та рекомендаційними блоками;
- каталог картин із можливістю фільтрації, сортування, пагінації;
- інтеграція з 3D-галереєю;
- реєстрація, авторизація та редагування профілю користувача;
- сторінки товарів та участь в аукціонах;
- чат для комунікації з продавцем;
- реалізація логіки ставок із динамічним оновленням у режимі реального часу.

У процесі роботи було спроектовано інформаційне забезпечення, структуру бази даних, визначено взаємозв’язки між сутностями, описано алгоритми функціонування клієнтської та серверної частини. Побудовані блок-схеми дозволяють структурувати логіку дій користувачів та взаємодії з сервером.

Результатом розробки є повнофункціональна платформа, яка відповідає сформованим на початку роботи вимогам до електронної комерції у сфері мистецтва. Її архітектура дозволяє легко адаптувати функціонал, інтегрувати додаткові модулі в майбутньому. В планах далі інтегрувати у веб сайт платіжні системи, аналітичні панелі, удосконалити системи рекомендацій та адаптувати інтерфейс до різних цільових груп користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Войтович, В. М. Основи створення веб-додатків. — Київ: Видавництво Ліра-К, 2020. — 312 с.
2. Глушаков, І. І. Проектування інформаційних систем. — Харків: ХНУРЕ, 2019. — 246 с.
3. Фленаган, Д. JavaScript: Повне керівництво. — 7-е вид. — Київ: Діалектика, 2021. — 1024 с.
4. Holmes, A. Node.js: практичний посібник. — Львів: Старий Лев, 2022. — 368 с.
5. Bank, D. MongoDB: The Definitive Guide. — 3rd ed. — O’Reilly Media, 2020. — 560 p.
6. Norman, D. The Design of Everyday Things. — MIT Press, 2013. — 368 p.
7. Козак, С. В. Технології веб-програмування. — Львів: Вид-во Львівської політехніки, 2020. — 198 с.
8. Мусієнко, І. Г. Архітектура програмного забезпечення. — Київ: НАУ, 2021. — 231 с.
9. Павленко, С. О. Методи організації баз даних. — Харків: ХНУРЕ, 2018. — 280 с.
10. Лоран, П. UX-дизайн: Розумний інтерфейс. — Київ: Наш Формат, 2020. — 240 с.
11. Krishna, V. Auction Theory. — Academic Press, 2009. — 500 p.
12. Node.js Official Documentation [Електронний ресурс]. — Режим доступу: https://nodejs.org
13. MongoDB Manual [Електронний ресурс]. — Режим доступу: https://www.mongodb.com/docs/manual/
14. React Documentation [Електронний ресурс]. — Режим доступу: https://reactjs.org
15. ISO/IEC 25010:2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE).

ДОДАТКИ

Додаток А

main.js

```
import { setupLogin } from "./js/auth/login.js";
import { setupRegister } from "./js/auth/register.js";
import { setupSession } from "./js/auth/session.js";
import { setupProfile } from "./js/components/profile.js";
import { setupEditProfile } from "./js/components/edit_profile.js";
import { setupAuction } from "./js/components/auction.js";
import { startAuctionTimer } from "./js/components/auctionTimer.js";
import {setupCatalog } from "./js/components/catalog.js";
import {setupProduct } from "./js/components/product.js";
import {setupProductsMain } from "./js/components/products_main.js";
import {setupChat } from "./js/components/chat.js";
import { toggleForms } from "./js/auth/auth.js";

document.addEventListener("DOMContentLoaded", () => {
   const page = window.location.pathname;
   setupSession();

   if (page.includes("/login")) {
      document.getElementById("toggle-login").addEventListener("click", toggleForms);
      document.getElementById("toggle-register").addEventListener("click", toggleForms);
      setupLogin();
      setupRegister();
   } else if (page.includes("/register")) {
      setupRegister();
   } else if (page.includes("/profile")) {
      setupProfile();
   } else if (page.includes("/editProfile")) {
      setupEditProfile();
   } else if (page.includes("/auction")){
      startAuctionTimer();
      setupAuction();
   } else if (page.includes("/catalog")){
      setupCatalog();
   } else if (page.includes("/product")){
      setupProduct();
   } else if (page.includes("/chat")) {
      setupChat();
   } else if (page.includes("/")){
      setupProductsMain();
   }
});
```

Додаток Б

Routers.js

```
const express = require("express");
const Bid = require("../models/Bid");

const router = express.Router();
let auctionEndTime = Date.now() + 10 * 60 * 1000;


router.get("/auction-time", (req, res) => {
   res.json({ endTime: auctionEndTime });
});


router.get("/bids/:productId", async (req, res) => {
   try {
      const { productId } = req.params;

      if (!productId) {
         return res.status(400);
      }

      const bids = await Bid.find({ productId }).sort({ time: -1 }).limit(10);
      res.json(bids);
   } catch (error) {
      console.error(error);
      res.status(500);
   }
});


router.post("/bid", async (req, res) => {
   if (!req.isAuthenticated()) {
      return res.status(401).json({ success: false, message: "You must do ligin" });
   }

   const { productId, amount } = req.body;
   const userId = req.user._id;
   const minStep = 50;

   try {
      if (!productId) {
         return res.status(400);
      }

      const lastBid = await Bid.findOne({ productId }).sort({ time: -1 });

      if (lastBid && amount < lastBid.amount + minStep) {
         return res.json({ success: false, message: "The rate is too low" });
      }

      const newBid = new Bid({ productId, user: userId, amount });
      await newBid.save();

      req.io.emit("newBid", newBid);
```

```
        res.json({ success: true, message: "The bet has been accepted", bid: newBid });
    } catch (error) {
        console.error(error);
        res.status(500);
    }
});


module.exports = router;

const express = require('express');
const passport = require('passport');
const router = express.Router();
const { isAuthenticated } = require('../config/middleware');

router.post('/login', (req, res, next) => {
    passport.authenticate('local', (err, user, info) => {
        if (err) {
            return res.status(500).json({ message: 'Server error' });
        }
        if (!user) {
            return res.status(400).json({ message: info.message });
        }
        req.logIn(user, (err) => {
            if (err) {
                return res.status(500).json({ message: 'Login error' });
            }
            res.status(200).json({
                message: 'Login successful',
                user: {
                    id: user._id,
                    email: user.email,
                },
            });
        });
    })(req, res, next);
});

router.post('/logout', (req, res) => {
    req.logout((err) => {
        if (err) {
            return res.status(500).json({ message: 'Logout failed' });
        }
        res.status(200).json({ message: 'Logged out successfully' });
    });
});

module.exports = router;

const express = require('express');
const router = express.Router();
const Goods = require('../models/Product');

router.get('/products', async (req, res) => {
    try {
        const { category, sort, page = 1, limit = 10 } = req.query;
```

```
        let query = {};
        if (category) {
            query.category = category;
        }

        let sortOptions = {};
        if (sort) {
            if (sort === 'price_asc') sortOptions.price = 1;
            if (sort === 'price_desc') sortOptions.price = -1;
        }

        const skip = (page - 1) * limit;

        const goods = await Goods.find(query)
            .sort(sortOptions)
            .skip(skip)
            .limit(Number(limit));

        const total = await Goods.countDocuments(query);

        res.json({
            products: goods,
            total,
            page: Number(page),
            pages: Math.ceil(total / limit),
        });
    } catch (err) {
        console.error(err);
        res.status(500).json({ message: 'Failed to fetch goods' });
    }
});

module.exports = router;

const express = require("express");
const router = express.Router();
const User = require("../models/User");
const bcrypt = require("bcrypt");

router.put("/edit", async (req, res) => {
    try {
        if (!req.user) {
            return res.status(401).json({ message: "User is not authorized" });
        }

        const { nickname, email, firstName, lastName, password, profilePhoto } = req.body;
        const user = await User.findById(req.user._id);

        if (!user) {
            return res.status(404).json({ message: "User didn't found" });
        }

        let updatedFields = {};

        if (nickname && nickname !== user.nickname) {
```

```
            const existingNickname = await User.findOne({ nickname });
            if (existingNickname) {
                return res.status(400).json({ message: "This nickname is already in use" });
            }
            updatedFields.nickname = nickname;
        }

        if (email && email !== user.email) {
            const existingEmail = await User.findOne({ email });
            if (existingEmail) {
                return res.status(400).json({ message: "This email is already in use" });
            }
            updatedFields.email = email;
        }

        if (firstName && firstName !== user.firstName) updatedFields.firstName = firstName;
        if (lastName && lastName !== user.lastName) updatedFields.lastName = lastName;
        if (profilePhoto && profilePhoto !== user.profilePhoto) updatedFields.profilePhoto = profilePhoto;

        if (password) {
            const salt = await bcrypt.genSalt(10);
            updatedFields.password = await bcrypt.hash(password, salt);
        }

        if (Object.keys(updatedFields).length === 0) {
            return res.json({ message: "No changes to update" });
        }

        await User.findByIdAndUpdate(req.user._id, updatedFields, { new: true });

        res.json({
            message: "Profile updated",
            user: {
                id: user._id,
                email: updatedFields.email || user.email,
                nickname: updatedFields.nickname || user.nickname,
                firstName: updatedFields.firstName || user.firstName,
                lastName: updatedFields.lastName || user.lastName,
                profilePhoto: updatedFields.profilePhoto || user.profilePhoto,
            },
        });

    } catch (error) {
        console.error("Error:", error);
        res.status(500).json({ message: "Server's error" });
    }
});

module.exports = router;

const express = require('express');
const router = express.Router();
const Product = require('../models/Product');

router.get('/:id', async (req, res) => {
    try {
```

```
    const productId = req.params.id;

    if (!productId) {
      return res.status(400).json({ message: "Error" });
    }

    const product = await Product.findById(productId);

    if (!product) {
      return res.status(404).json({ message: "Art didn't found" });
    }

    const responseData = {
      _id: product._id,
      name: product.name,
      photo: product.photo,
      price: product.price,
      saleType: product.saleType,
      category: product.category,
      author: {
        authorId: product.author.authorId,
        name: product.author.name,
        rating: product.author.rating,
      },
    };

    res.json(responseData);
  } catch (error) {
    console.error("Error:", error);
    res.status(500).json({ message: "Server's error" });
  }
});

module.exports = router;

const express = require('express');
const router = express.Router();

router.get('/profile', (req, res) => {
  if (req.isAuthenticated()) {
    res.status(200).json({
      user: {
        id: req.user._id,
        email: req.user.email,
        nickname: req.user.nickname || 'No nickname',
        firstName: req.user.firstName || 'No first name',
        lastName: req.user.lastName || 'No last name',
        avatar: req.user.avatar || 'https://cdn.pixabay.com/photo/2018/11/13/21/43/avatar-
3814049_1280.png',
      },
    });
  } else {
    res.status(401).json({ message: 'Unauthorized' });
  }
});
```

```
module.exports = router;

const express = require('express');
const router = express.Router();
const User = require('../models/User');
const bcrypt = require('bcrypt');
const { isAuthenticated } = require('../config/middleware');

router.post('/register', async (req, res) => {
   const { firstName, lastName, nickname, email, userType, password, confirmPassword } = req.body;

   if (!firstName || !lastName || !nickname || !email || !userType || !password || !confirmPassword) {
      return res.status(400).json({ message: 'All fields must be filled' });
   }

   if (password !== confirmPassword) {
      return res.status(400).json({ message: 'Passwords do not match' });
   }

   try {
      const existingEmail = await User.findOne({ email });
      if (existingEmail) {
         return res.status(400).json({ message: 'A user with this email already exists' });
      }

      const existingNickname = await User.findOne({ nickname });
      if (existingNickname) {
         return res.status(400).json({ message: 'Nickname is already in use' });
      }


      const newUser = new User({
         firstName,
         lastName,
         nickname,
         email,
         userType,
         password: password,
      });

      await newUser.save();
      res.status(201).json({ message: 'Registration is successful' });
   } catch (error) {
      console.error('Error during registration:', error);
      res.status(500).json({ message: 'Server error' });
   }
});

module.exports = router;
```

Додаток В

Frontend.js

```
export async function startAuctionTimer() {
   try {
      const response = await fetch("/api/auction-time");
      const data = await response.json();
      const auctionEndTime = new Date().getTime() + 7 * 24 * 60 * 60 * 1000;;
      const timeLeftDisplay = document.getElementById("time-left");
      const placeBidBtn = document.getElementById("place-bid-btn");

      if (!timeLeftDisplay || !placeBidBtn) return;

      function updateTimer() {
         const now = new Date().getTime();
         const timeLeft = auctionEndTime - now;

         if (timeLeft <= 0) {
            timeLeftDisplay.textContent = "Auction completed!";
            placeBidBtn.disabled = true;
            clearInterval(timerInterval);
            return;
         }

         const minutes = Math.floor(timeLeft / (1000 * 60));
         const seconds = Math.floor((timeLeft % (1000 * 60)) / 1000);
         timeLeftDisplay.textContent = `${minutes}:${seconds < 10 ? "0" : ""}${seconds}`;
      }

      updateTimer();
      const timerInterval = setInterval(updateTimer, 1000);
   } catch (error) {
      console.error(error);
   }
}

export function setupBidding(productId, initialBid, bidStep) {
   const socket = io();
   const bidModal = document.getElementById("bid-modal");
   const bidAmountDisplay = document.getElementById("bid-value");
   const bidSlider = document.getElementById("bid-slider");
   const confirmBidBtn = document.getElementById("confirm-bid");
   const bidList = document.getElementById("bids-list");
   const currentBidDisplay = document.getElementById("current-bid");
   const modalCurrentBid = document.getElementById("modal-current-bid");
   const placeBidBtn = document.getElementById("place-bid-btn");
   const closeModalBtn = document.querySelector(".close-btn");

   let minBid = parseFloat(initialBid);

   function openBidModal() {
      bidSlider.min = minBid + bidStep;
      bidSlider.step = bidStep;
      bidSlider.value = minBid + bidStep;
      bidAmountDisplay.textContent = `${bidSlider.value} $`;
      modalCurrentBid.textContent = `${minBid} $`;
```

```
    bidModal.style.display = "flex";
}

bidSlider.addEventListener("input", () => {
    bidAmountDisplay.textContent = `${bidSlider.value} $`;
});

confirmBidBtn.addEventListener("click", async () => {
    const bidAmount = parseFloat(bidSlider.value);

    try {
        const response = await fetch("/api/bid", {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({ user: "Anonim", amount: bidAmount, productId })
        });

        const data = await response.json();

        if (data.success) {
            bidModal.style.display = "none";
        } else {
            alert(data.message);
        }
    } catch (error) {
        console.error( error);
    }
});

closeModalBtn.addEventListener("click", () => {
    bidModal.style.display = "none";
});

socket.on("newBid", function (bid) {
    if (bid.productId !== productId) return;

    currentBidDisplay.textContent = bid.amount;
    minBid = bid.amount;

    const bidEntry = document.createElement("div");
    bidEntry.classList.add("bid-entry");
    bidEntry.textContent = `${bid.amount} $ - ${new Date(bid.time || Date.now()).toLocaleString()}`;

    bidList.prepend(bidEntry);
});

async function loadBidHistory() {
    try {
        const response = await fetch(`/api/bids/${productId}`);
        const bids = await response.json();

        if (!bids.length) return;

        currentBidDisplay.textContent = bids[0].amount;
        minBid = bids[0].amount;
```

```
            bidList.innerHTML = "";
            bids.forEach(bid => {
                const bidEntry = document.createElement("div");
                bidEntry.classList.add("bid-entry");
                bidEntry.textContent = `${bid.amount} $ - ${new Date(bid.time).toLocaleString()}`;
                bidList.appendChild(bidEntry);
            });
        } catch (error) {
            console.error(error);
        }
    }

    placeBidBtn.addEventListener("click", openBidModal);
    loadBidHistory();
}


import { renderProductCard } from "../utils/renderProduct.js";
import { handleBuyButtonClick } from "../utils/handleBuyButtonClick.js";

export async function setupCatalog() {
    const productContainer = document.getElementById('goods-container');
    const categorySelect = document.getElementById('category-filter');
    const sortSelect = document.getElementById('sort-select');
    const pagination = document.getElementById('pagination');

    let currentPage = 1;
    const limit = 10;

    async function loadProducts() {
        try {
            const category = categorySelect.value;
            const sort = sortSelect.value;

            const response = await fetch(`/api/products?page=${currentPage}&limit=${limit}&category=${category}&sort=${sort}`);
            const data = await response.json();

            if (data.products?.length) {
                productContainer.innerHTML = data.products
                    .map(product => renderProductCard(product))
                    .join("");
            } else {
                productContainer.innerHTML = "<p>No products available</p>";
            }

            pagination.innerHTML = Array.from({ length: data.pages }, (_, i) => {
                const page = i + 1;
                return `<button class="pagination-btn" ${page === currentPage ? "disabled" : ""}>${page}</button>`;
            }).join("");

        } catch (err) {
            console.error("Error loading products:", err);
            productContainer.innerHTML = "<p>Error loading products. Please try again later.</p>";
        }
```

```
    }

    productContainer.addEventListener("click", handleBuyButtonClick);
    categorySelect.addEventListener("change", () => { currentPage = 1; loadProducts(); });
    sortSelect.addEventListener("change", () => { currentPage = 1; loadProducts(); });
    pagination.addEventListener("click", (e) => {
        if (e.target.classList.contains("pagination-btn")) {
            currentPage = parseInt(e.target.textContent);
            loadProducts();
        }
    });

    loadProducts();
}

export function setupChat() {
    const socket = io("http://localhost:5000");
    const chatBox = document.getElementById("chat-box");
    const chatInput = document.getElementById("chat-input");
    const sendBtn = document.getElementById("send-btn");

    let userId = null;
    let chatId = null;

    const urlParams = new URLSearchParams(window.location.search);
    chatId = urlParams.get("chatId");

    if (!chatId) {
        return;
    }

    async function loadMessages() {
        try {
            const response = await fetch(`/api/chat/${chatId}`, {
                method: "GET",
                headers: { "Content-Type": "application/json" }
            });

            if (!response.ok) {
                throw new Error("Error loading chat");
            }

            const data = await response.json();
            userId = data.userId;

            chatBox.innerHTML = "";
            data.chat.messages.forEach(msg => displayMessage(msg, msg.senderId === userId));
        } catch (error) {
            console.error(error);
        }
    }

    function displayMessage(msg, isSent) {
        const msgElement = document.createElement("div");
        msgElement.classList.add("message", isSent ? "sent" : "received");
        msgElement.textContent = msg.message;
```

```
        chatBox.appendChild(msgElement);
        chatBox.scrollTop = chatBox.scrollHeight;
    }

    sendBtn.addEventListener("click", async () => {
        const message = chatInput.value.trim();
        if (!message || !userId || !chatId) return;

        try {
            const response = await fetch("/api/chatSendMessage", {
                method: "POST",
                headers: { "Content-Type": "application/json" },
                body: JSON.stringify({ chatId, senderId: userId, message })
            });

            if (!response.ok) {
                throw new Error("Failed to send message");
            }

            const data = await response.json();
            displayMessage(data, true);
            chatInput.value = "";

            socket.emit("sendMessage", { chatId, message: data });
        } catch (error) {
            console.error(error);
        }
    });

    socket.on(`message-${chatId}`, (msg) => {
        if (msg.senderId !== userId) {
            displayMessage(msg, false);
        }
    });

    loadMessages();
}

export async function setupEditProfile() {
    const editButtons = document.querySelectorAll(".edit-button");

    async function fetchUserProfile() {
        try {
            const response = await fetch("/api/profile", {
                method: "GET",
                headers: {
                    "Content-Type": "application/json",
                },
                credentials: "include",
            });

            if (!response.ok) {
                throw new Error("No user's data");
            }

            const data = await response.json();
```

```
        const user = data.user;

        if (!user) {
            console.error("No user's data");
            return;
        }

        document.getElementById("nickname")?.setAttribute("value", user.nickname || "");
        document.getElementById("email")?.setAttribute("value", user.email || "");
        document.getElementById("firstName")?.setAttribute("value", user.firstName || "");
        document.getElementById("lastName")?.setAttribute("value", user.lastName || "");

        const avatarElement = document.getElementById("current-avatar");
        if (avatarElement) {
            avatarElement.src = user.profilePhoto || "https://cdn.pixabay.com/photo/2018/11/13/21/43/avatar-
3814049_1280.png";
        }
    } catch (error) {
        console.error("Error:", error);
    }
}

await fetchUserProfile();

editButtons.forEach(button => {
    button.addEventListener("click", () => {
        const parent = button.closest(".edit-option");
        const form = parent.querySelector(".edit-form");

        if (form) {
            form.style.display = (form.style.display === "none" || form.style.display === "") ? "block" : "none";
        }
    });
});

const avatarUpload = document.getElementById("avatar-upload");
if (avatarUpload) {
    avatarUpload.addEventListener("change", async function () {
        const file = this.files[0];
        if (!file) return;

        const formData = new FormData();
        formData.append("avatar", file);

        try {
            const response = await fetch("/api/edit", {
                method: "PUT",
                credentials: "include",
                body: formData,
            });

            const result = await response.json();

            if (!response.ok) {
                throw new Error(result.message || "Error avatar updates");
```

```
            }

            document.getElementById("current-avatar").src = result.profilePhoto;
            alert("Avatas succesfull uodated");
          } catch (err) {
            console.error("Error:", err);
          }
        });
    }

    document.querySelectorAll(".save-button").forEach(button => {
      button.addEventListener("click", async function () {
        const parent = this.closest(".edit-option");
        if (!parent) return;

        const input = parent.querySelector("input");
        if (!input) return;

        const field = input.id;
        const value = input.value.trim();

        if (!value) {
          alert("The field cannot be empty");
          return;
        }

        try {
          const response = await fetch("/api/edit", {
            method: "PUT",
            headers: { "Content-Type": "application/json" },
            credentials: "include",
            body: JSON.stringify({ [field]: value }),
          });

          const result = await response.json();

          if (!response.ok) {
            throw new Error(result.message || "Error profile update");
          }

          alert(result.message);
          fetchUserProfile();
        } catch (err) {
          console.error("Error profile update", err);
        }
      });
    });
};

import { handleBuyButtonClick } from '../utils/handleBuyButtonClick.js';

export async function setupProduct() {
    const buyButton = document.getElementById("buy-button");
    const currentPage = window.location.pathname;

    if (!currentPage.includes('product.html')) return;
```

```
    const urlParams = new URLSearchParams(window.location.search);
    const productId = urlParams.get('id');

    if (!productId) {
        document.body.innerHTML = '<p>Error: not found</p>';
        return;
    }

    try {
        const response = await fetch(`/api/products/${productId}`);
        const product = await response.json();

        document.getElementById('product-image').src = product.photo;
        document.getElementById('product-title').textContent = product.name;
        document.getElementById('product-description').textContent = product.description;
        document.getElementById('artist-name').textContent = product.author.name;

        document.getElementById('artist-rating').textContent = `${product.author.rating}/5
(${product.author.reviewCount} reviews)`;
        buyButton.textContent = product.saleType === "auction" ? "Auction" : "Buy Now";

        buyButton.dataset.id = productId;
        buyButton.dataset.type = product.saleType;
        buyButton.dataset.authorId = product.author?.authorId;

        buyButton.addEventListener("click", handleBuyButtonClick);

        loadReviews(product.author._id);

    } catch (error) {
        console.error('Error:', error);
    }
}

async function loadReviews(authorId) {
    const reviewsList = document.getElementById('reviews-list');

    try {
        const response = await fetch(`/api/reviews?author=${authorId}`);
        const reviews = await response.json();

        reviewsList.innerHTML = '';

        if (reviews.length === 0) {
            reviewsList.innerHTML = '<li>No reviews</li>';
        } else {
            reviews.forEach(review => {
                const li = document.createElement('li');
                li.textContent = review.text;
                reviewsList.appendChild(li);
            });
        }
    } catch (error) {
        console.error('Error:', error);
        reviewsList.innerHTML = '<li>Error loading</li>';
```

```
    }
}


import { renderProductCard } from "../utils/renderProduct.js";
import { handleBuyButtonClick } from "../utils/handleBuyButtonClick.js";

export function setupProductsMain() {
    const topArtworksContainer = document.querySelector(".artworks");

    async function loadTopAuthorsWorks() {
        try {
            const response = await fetch("/api/products?limit=1000");
            const data = await response.json();

            if (!data.products?.length) {
                topArtworksContainer.innerHTML = "<p>No top-rated authors found.</p>";
                return;
            }

            const authorsMap = data.products.reduce((map, product) => {
                const key = product.author.name;
                if (!map[key]) {
                    map[key] = {
                        name: product.author.name,
                        rating: product.author.rating,
                        works: [],
                    };
                }
                map[key].works.push(product);
                return map;
            }, {});

            const topAuthors = Object.values(authorsMap)
                .sort((a, b) => b.rating - a.rating)
                .slice(0, 10);

            topArtworksContainer.innerHTML = topAuthors.map(author =>
                author.works.slice(0, 3).map(product => renderProductCard(product)).join("")
            ).join("");

        } catch (err) {
            console.error("Error loading top authors and works:", err);
            topArtworksContainer.innerHTML = "<p>Failed to load top-rated authors. Try again later.</p>";
        }
    }

    topArtworksContainer.addEventListener("click", handleBuyButtonClick);
    loadTopAuthorsWorks();
}

export async function setupProfile() {
    const nicknameElement = document.querySelector('.nickname');
    const fullNameElement = document.querySelector('.full-name');
    const avatarElement = document.querySelector('.profile-avatar img');
```

```
    if (!nicknameElement || !fullNameElement || !avatarElement) {
        console.warn("Profile elements not found on the page.");
        return;
    }

    try {
        const response = await fetch('/api/profile');
        if (!response.ok) throw new Error(`Failed to fetch user profile: ${response.status}`);

        const { user } = await response.json();
        nicknameElement.textContent = user.nickname || 'Unknown';
        fullNameElement.textContent = `${user.firstName} ${user.lastName}` || 'No Full Name';
        avatarElement.src = user.avatar || 'https://cdn.pixabay.com/photo/2018/11/13/21/43/avatar-
3814049_1280.png';
    } catch (err) {
        console.error('Error fetching profile data:', err);
        nicknameElement.textContent = 'Error loading profile';
        fullNameElement.textContent = 'Please try again later';
    }


    document.querySelector(".edit-profile-button").addEventListener("click", () => {
        window.location.href = "http://localhost:5000/editProfile";
    });
}
```