МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії
програмного забезпечення

_____ Євген ДАВИДЕНКО

«___» _____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

Вебзастосунок продажу картин

Спеціальність 121 Інженерія програмного забезпечення
Освітня програма «Інженерія програмного забезпечення»

Здобувач _____ **Владислав Ігнатченко**

«__» _____ 2026 р.

Керівник роботи
д-рка техн. наук,
професорка _____ **Альона ШВЕД**

«__» _____ 2026 р.

Миколаїв – 2026

Завдання на виконання кваліфікаційної роботи

Чорноморський національний університет імені Петра Могили

Факультет	Комп’ютерних наук
Кафедра	Інженерії програмного забезпечення
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступінь	Бакалавр
Спеціальність	121 Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«___» _____ 2026 р.

ЗАВДАННЯ

на кваліфікаційну бакалаврську роботу здобувача

Владислав Ігнатченко

1. Тема кваліфікаційної роботи «Вебзастосунок продажу картин» затверджена наказом ректора ЧНУ ім. Петра Могили № 349 від «26» грудня 2025 р.
2. Строк представлення кваліфікаційної роботи «___» _____ 2026 р.
3. Очікуваним результатом є створення вебзастосунку для забезпечення багатокористувацького доступу до перегляду та продажу товарів зі сфери мистецтва.
4. Перелік питань, що підлягають розробці:

– провести аналіз предметної області та існуючих аналогів вебзастосунків продажу картин;

– проаналізувати програмні та інструментальні засоби для розробки інтернет-магазину;

– розробити логіку бази даних;

– розробити логіку вебзастосунку;

– виконати програмну реалізацію та тестування інтернет-магазину.

5. Перелік графічних матеріалів:

– презентація.

6. Консультанти:

Консультант	**Кафедра (організація)**	**Частина роботи**

Дата видачі завдання «03» лютого 2026 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: Вебзастосунок продажу картин

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на створення вебзастосунку продажу картин	26.12.2025	29.12.2025	Виконано
2.	Перегляд літератури за обраною темою	05.01.2026	16.01.2026	Виконано
3.	Складання календарного плану КБР	19.01.2026	21.01.2026	Виконано
4.	Аналіз подібних вебзастосунків	22.01.2026	30.01.2026	Виконано
5.	Проєктування схеми бази даних та вибір стеку технологій	02.02.2026	18.02.2026	Виконано
6.	Моделювання та конструювання вебзастосунку, розробка бази даних	19.02.2026	23.03.2026	Виконано
7.	Кодування функціоналу, перевірка роботи вебзастосунку, тестування безпеки	24.03.2026	17.04.2026	Виконано
8.	Отримання відгуку керівника КБР	20.04.2026	23.04.2026	Виконано
9.	Оформлення КБР та презентації	24.04.2026	15.05.2026	Виконано
10.	Попередній захист	27.05.2026	27.05.2026	Виконано
11.	Завершення оформлення КБР та презентації	01.06.2026	12.06.2026	Виконано
12.	Рецензування	08.06.2026	12.06.2026	Виконано
13.	Захист кваліфікаційної роботи	23.06.2026	23.06.2026	Виконано

Здобувач _____ **Владислав ІГНАТЧЕНКО**

«__» _____ 2026 р.

Керівник роботи

д-рка техн. наук,

професорка _____ **Альона ШВЕД**

«__» _____ 2026 р.

АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи

Вебзастосунок продажу картин

Здобувач 409 гр.: Ігнатченко Владислав

Керівник: д-рка техн. наук, професорка Швед Альона

Актуальність теми даної роботи обумовлена стрімкою цифровізацією культурної сфери та трансформацією арт-ринку під впливом сучасних вебтехнологій. У часи переходу бізнес-процесів у віртуальний простір, онлайн-галереї стають ключовим інструментом, що забезпечує доступність мистецтва для широкої аудиторії. Це дозволяє поціновувачам економити час на пошук творів, а митцям – ефективно презентувати та реалізовувати свої роботи без географічних обмежень.

Вебзастосунки для продажу картин дозволяють не лише масштабувати мистецький бізнес, а й забезпечують інтерактивну взаємодію між автором та покупцем, пропонуючи зручні інструменти візуалізації, фільтрації та безпечних транзакцій.

Робота присвячена розробці вебзастосунку продажу картин, який надає користувачам естетичний та функціональний інструмент для перегляду, вибору та придбання творів мистецтва онлайн.

Об’єктом роботи є процес автоматизації продажу картин.

Предметом роботи є програмні засоби автоматизації продажу картин.

Метою кваліфікаційної роботи є розробка вебзастосунку для забезпечення багатокористувацького доступу до каталогу товарів та послуг з їх продажу.

Для досягнення мети необхідно виконати наступні завдання:

– проаналізувати предметну область та сучасні тенденції ринку онлайн-продажів картин;

– провести порівняльний аналіз існуючих програмних аналогів;

– обґрунтувати вибір стеку технологій для реалізації системи;

– розробити логічну та фізичну структуру реляційної бази даних;

– спроєктувати архітектуру вебзастосунку та користувацький інтерфейс;

– виконати програмну реалізацію та тестування функціональних можливостей інтернет-магазину.

Кваліфікаційна робота бакалавра включає вступ, чотири розділи, висновки та перелік джерел посилання.

У вступі розкривається актуальність теми, визначається мета, об’єкт, предмет та методи дослідження, а також надається короткий огляд поставлених задач.

У першому розділі досліджується предметна область, проводиться глибокий аналіз світових аналогів та особливостей їх роботи, на основі чого формуються вимоги до розробки.

У другому розділі проводиться аналіз інструментарію та методів моделювання об’єкту. Формується специфікація функціональних та нефункціональних вимог до ПЗ, що визначає повну логіку поведінки системи.

У третьому розділі обґрунтовується вибір стеку технологій, описується конструювання бази даних та проєктування інтерфейсу користувача. Також в розділі представлено низку UML-діаграм, що візуалізують архітектуру та процеси взаємодії в системі.

У четвертому розділі описується програмна реалізація вебзастосунку, зокрема розробка клієнтської частини, а також результати тестування продукту.

Кваліфікаційна робота викладена на 100 сторінках машинописного тексту, складається із вступу, 4 розділів, загальних висновків, переліку джерел посилання з 18 найменувань та 2 додатків. Праця містить 8 таблиць та 31 рисунків.

Ключові слова: продаж картин, онлайн-галерея, вебзастосунок, електронна комерція, користувач, Angular, SQL Server, Node.js.

ABSTRACT

of the Bachelor's Thesis

Web application for selling paintings

Student of group 409: Vladyslav Ihnatchenko

Supervisor: doctor of technical sciences, professor Alyona Shved

The relevance of the topic of this work is driven by the rapid digitalization of the cultural sphere and the transformation of the art market under the influence of modern web technologies. In times of business process transition into virtual space, online galleries become a key tool ensuring the accessibility of art to a wide audience. This allows art lovers to save time searching for works, and artists to effectively present and sell their works without geographical restrictions.

Web applications for selling paintings not only allow scaling the art business but also provide interactive interaction between the artist and the buyer, offering convenient tools for visualization, filtering, and secure transactions.

The work is dedicated to the development of a web application for selling paintings, which provides users with an aesthetic and functional tool for viewing, selecting, and purchasing works of art online.

The object of the work is the process of automating the sale of paintings.

The subject of the work is the software tools for automating the sale of paintings.

The aim of the qualification work is to develop a web application to provide multi-user access to a catalog of goods and services for their sale.

To achieve the aim, the following tasks must be accomplished:

– analyze the subject area and current trends of the online painting sales market;

– conduct a comparative analysis of existing software analogs;

– justify the choice of technology stack for system implementation;

– develop the logical and physical structure of the relational database;

– design the architecture of the web application and user interface;

– perform software implementation and testing of the functional capabilities of the online store.

The bachelor's degree thesis includes an introduction, four chapters, conclusions, and a list of references.

The introduction reveals the relevance of the topic, defines the aim, object, subject and research methods, and provides a brief overview of the tasks set.

The first chapter explores the subject area, conducts an in-depth analysis of worldwide analogs and their operational features, based on which the development requirements are formed.

The second chapter analyzes the tools and methods of object modeling. A specification of functional and non-functional requirements for the software is formed, which defines the complete behavior logic of the system.

The third chapter justifies the choice of the technology stack, describes the construction of the database and the design of the user interface. The chapter also presents a number of UML diagrams visualizing the architecture and interaction processes within the system.

The fourth chapter describes the software implementation of the web application, in particular the development of the client-side part, as well as the results of product testing.

The qualification work is presented on 100 pages of typewritten text, consists of an introduction, 4 sections, general conclusions, a list of references with 18 titles and 2 appendices. The work contains 8 tables and 31 figures.

Keywords: art sales, online gallery, web application, e-commerce, user, Angular, SQL Server, Node.js.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД – база даних.

ІТ – інформаційні технології.

СУБД – система управління базами даних.

API (Application Programming Interface) – прикладний програмний інтерфейс.

CRUD (Create, Read, Update, Delete) – базові функції управління даними.

CSS (Cascading Style Sheets) – каскадні таблиці стилів.

ER (Entity-Relationship) – модель «сутність-зв'язок».

HTML (HyperText Markup Language) – мова гіпертекстової розмітки.

HTTP (HyperText Transfer Protocol) – протокол передачі гіпертексту.

HTTPS (HyperText Transfer Protocol Secure) – захищений протокол передачі гіпертексту в мережі.

JSON (JavaScript Object Notation) – текстовий формат обміну даними.

REST (Representational State Transfer) – архітектурний стиль взаємодії компонентів розподіленого застосунку.

SPA (Single Page Application) – односторінковий вебзастосунок.

SQL (Structured Query Language) – структурована мова запитів.

UI (User Interface) – інтерфейс користувача.

UML (Unified Modeling Language) – уніфікована мова моделювання.

ВСТУП

Актуальність обраної теми: У сучасному цифровому світі трансформація традиційних ринків у бік онлайн-платформ є не лише трендом, а й стратегічною необхідністю для виживання бізнесу. Ринок образотворчого мистецтва, який довгий час залишався консервативним та залежав від фізичних галерей, за останні роки зазнав кардинальних змін. Згідно зі світовими аналітичними звітами, обсяги онлайн-продажів творів мистецтва демонструють стабільну динаміку росту, щороку збільшуючи свою частку в загальному обігу. Це зумовлено зміною споживчої поведінки: сучасні колекціонери та поціновувачі мистецтва все частіше використовують мобільні та вебзастосунки для пошуку, оцінки та придбання унікальних предметів інтер’єру.

Цифровізація цієї сфери вирішує ключову проблему – географічну обмеженість. Вебзастосунок дозволяє художнику з будь-якого куточка країни вийти на глобальний ринок, а покупцю – отримати доступ до ексклюзивних робіт, не відвідуючи фізичні виставки. Крім того, розвиток технологій електронної комерції висуває нові вимоги до програмних продуктів: вони повинні бути не лише візуально привабливими, а й забезпечувати високу швидкість завантаження контенту (що критично для зображень високої якості), гарантувати безпеку персональних даних та стабільність фінансових транзакцій.

Додатковим фактором актуальності є необхідність впровадження сучасних інструментів взаємодії з користувачем, таких як адаптивні інтерфейси, системи розумної фільтрації за жанрами та техніками виконання, а також персоналізовані списки бажань. Враховуючи високу конкуренцію серед інтернет-магазинів, розробка спеціалізованого вебзастосунку, який поєднує в собі потужну серверну частину та інтуїтивно зрозумілий клієнтський інтерфейс, є вкрай актуальним завданням для підтримки та розвитку сучасної арт-індустрії.

Практичне значення: застосунок створить зручну та ефективну платформу для ознайомлення з асортиментом художніх робіт, що пропонуються авторами чи галереями. Надання детальної візуальної та описової інформації про картини,

інтеграція списків бажань та безпечних систем оплати допоможе покупцям зробити обдуманий вибір та спростить процес купівлі.

Мета роботи: розробка вебзастосунку для забезпечення багатокористувацького доступу до каталогу та послуг із продажу картин.

Для досягнення мети необхідно вирішити наступні **завдання:**

– проаналізувати сферу онлайн-продажів творів мистецтва;

– проаналізувати програмні та інструментальні засоби для розробки інтернет-магазину;

– розробити логіку бази даних для зберігання інформації про товари, користувачів та замовлення;

– розробити логіку вебзастосунку, включаючи системи автентифікації та управління кошиком;

– виконати програмну реалізацію та тестування інтернет-магазину в різних сценаріях використання.

Об’єкт роботи: процес автоматизації продажу картин.

Предмет роботи: програмні засоби автоматизації продажу картин.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАНЬ

1.1 Аналіз предметної області та визначення об’єкта і предмета дослідження

Вебзастосунки для продажу творів мистецтва сьогодні відіграють важливу роль у сфері електронної комерції та культури. Це пов’язано з тим, що все більше процесів експонування та купівлі картин переходить в онлайн-середовище. Розвиток інформаційних технологій зробив онлайн-галереї зручними інструментами, що забезпечують доступ до мистецтва всім бажаючим незалежно від їх місця проживання.

Мистецький ринок постійно трансформується: з’являються нові таланти, а попит зберігається як на класичний живопис, так і на сучасні цифрові роботи. У цих умовах вебзастосунки дозволяють швидко знаходити необхідні полотна, спрощують їх порівняння та надають детальну інформацію про техніку виконання, автора та історію створення.

Важливою особливістю сучасних онлайн-галерей є їх орієнтація на користувача та забезпечення високого рівня візуальної естетики. Це досягається за рахунок використання адаптивного дизайну, високоякісних зображень та швидкої обробки запитів [1]. Сучасні системи продажу картин повинні враховувати такі фактори:

– безпека обробки персональних даних та платежів;
– надійність зберігання інформації про транзакції;
– масштабованість системи для додавання нових колекцій;
– підтримка високої якості відображення медіа-контенту.

Особливу роль відіграє якість інтерфейсу, адже мінімалістичний дизайн дозволяє не відволікати увагу від головного об’єкта – картини. Користувачі очікують швидкого доступу до каталогу, зручної навігації та додаткових можливостей, таких як створення власних списків бажаного або отримання персоналізованих рекомендацій. Таким чином, розробка вебзастосунку продажу

картин є актуальною задачею, що потребує поєднання сучасних IT-підходів із вимогами мистецької сфери.

Саме тому об’єктом дослідження є процес організації онлайн-продажу картин у вебсередовищі, а предметом дослідження є методи та програмні засоби розробки вебзастосунку, зокрема: реалізація каталогу, управління кошиком та списком бажаного, аутентифікація користувачів та обробка замовлень.

1.2 Аналіз сучасного стану програмних рішень

На сучасному ринку існує велика кількість вебзастосунків для продажу картин, які можна поділити на онлайн-галереї та платформи, що спеціалізуються на розміщенні і продажу художніх творів від різних авторів. Для аналізу використовувались такі критерії:

– функціональність;
– архітектура;
– переваги та недоліки.

1.2.1 Аналіз платформи Artsy

Artsy (табл. 1.1, рис. 1.1) – глобально технологічна екосистема, що визначає стандарти для всього арт-ринку [2]. В основі її функціонування лежить унікальна інтелектуальна надбудова – «The Art Genome Project». Це науково-технологічна розробка, яка використовує понад тисячу характеристик для опису кожної картини, включаючи художній стиль, історичний період, техніку виконання та навіть емоційне забарвлення.

З точки зору функціональної структури, Artsy реалізує складну гібридну модель. Вона одночасно обслуговує три типи суб’єктів: приватних колекціонерів, професійні галереї та аукціонні будинки. Вебзастосунок інтегрований із базами даних найбільших світових виставок, що дозволяє користувачу віртуально відвідувати події в реальному часі. Особлива увага в архітектурі платформи приділена пошуковій системі: завдяки глибокому тегуванню, користувач отримує персоналізовані рекомендації, які з кожним кліком стають точнішими. Крім

комерції, Artsy виконує роль медіа-ресурсу, публікуючи аналітику, що робить її незамінним інструментом для інвесторів у мистецтво.

Таблиця 1.1 – Характеристика платформи Artsy.net

Назва	Artsy.net – найбільша у світі платформа для пошуку та збору мистецтва.
Розробник	Artsy, Inc.
Архітектура	Вебзастосунок (клієнт-сервер).
Мова реалізації	HTML/CSS/JavaScript.
Перелік функцій, характеристик	1) обширний каталог робіт; 2) актуальні освітні статті; 3) інтеграція з провідними світовими аукціонами; 4) фільтрація за напрямками мистецтва.
Переваги	1) величезна база даних; 2) наявність освітнього контенту; 3) найвищий рівень довіри на ринку онлайн-продажів; 4) персоналізований підбір творів за стилем і технікою.
Недоліки	1) складність інтерфейсу для початківців; 2) висока конкуренція серед митців; 3) високі комісійні для галерей-партнерів.

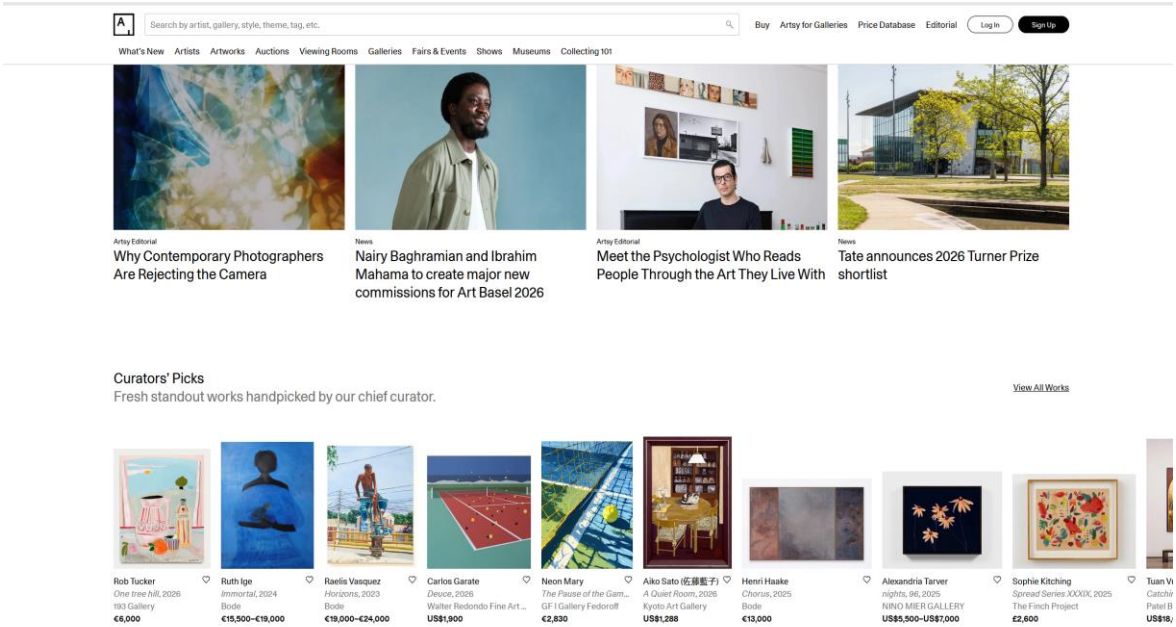

Рисунок 1.1 – Інтерфейс вебзастосунку Artsy

Ось так представлено вигляд вебзастосунку онлайн-платформи Artsy.

1.2.2 Аналіз маркетплейсу Saatchi Art

Saatchi Art (табл. 1.2, рис. 1.2) – представляє собою найбільш масштабований приклад маркетплейсу, який повністю змінив парадигму взаємодії художника та покупця [3]. Основна ідеологічна відмінність цього вебзастосунку – відмова від класичного «галерейного фільтра». Платформа надає можливість кожному митцю, незалежно від його статусу, створити професійне портфоліо та вийти на міжнародний рівень.

Архітектурно Saatchi Art є надзвичайно складним механізмом, орієнтованим на автоматизацію бізнес-процесів. Ключовим модулем системи є інтегрований логістичний вузол, який автоматично розраховує вартість доставки, митні збори та страхування для 140 країн світу. Це знімає технічний бар'єр із художника, дозволяючи йому зосередитися на творчості. Ще одним інноваційним рішенням є впровадження інструментів доповненої реальності, що дозволяє покупцеві через вебінтерфейс масштабувати картину та бачити її реальний вигляд у власному інтер’єрі. Це суттєво підвищує коефіцієнт конверсії, долаючи головний страх онлайн-покупки мистецтва – невідповідність розмірів та кольорів.

Таблиця 1.2 – Маркетплейс Saatchi Art

Назва	Saatchi Art – провідна онлайн-галерея з продажу оригінального мистецтва.
Розробник	Leaf Group.
Архітектура	Вебзастосунок (клієнт-сервер).
Мова реалізації	HTML/CSS/JavaScript.
Перелік функцій, характеристик	1) примірка картини в інтерв'ю через браузер; 2) автоматизація логістики: платформа сама готує документи для митниці та доставки; 3) продаж лімітованих принтів; 4) персональний консалтинг; 5) перегляд творів мистецтва у доповненій реальності.

Кінець таблиці 1.2

Переваги	1) відсутність ексклюзивності (художник може продавати роботи і в інших місцях); 2) величезна аудиторія покупців із 140+ країн світу; 3) професійна кураторська підтримка нових імен; 4) зручні тематичні добірки.
Недоліки	1) висока конкуренція (мільйони робіт у базі даних); 2) комісія платформи складає до 35-40%.

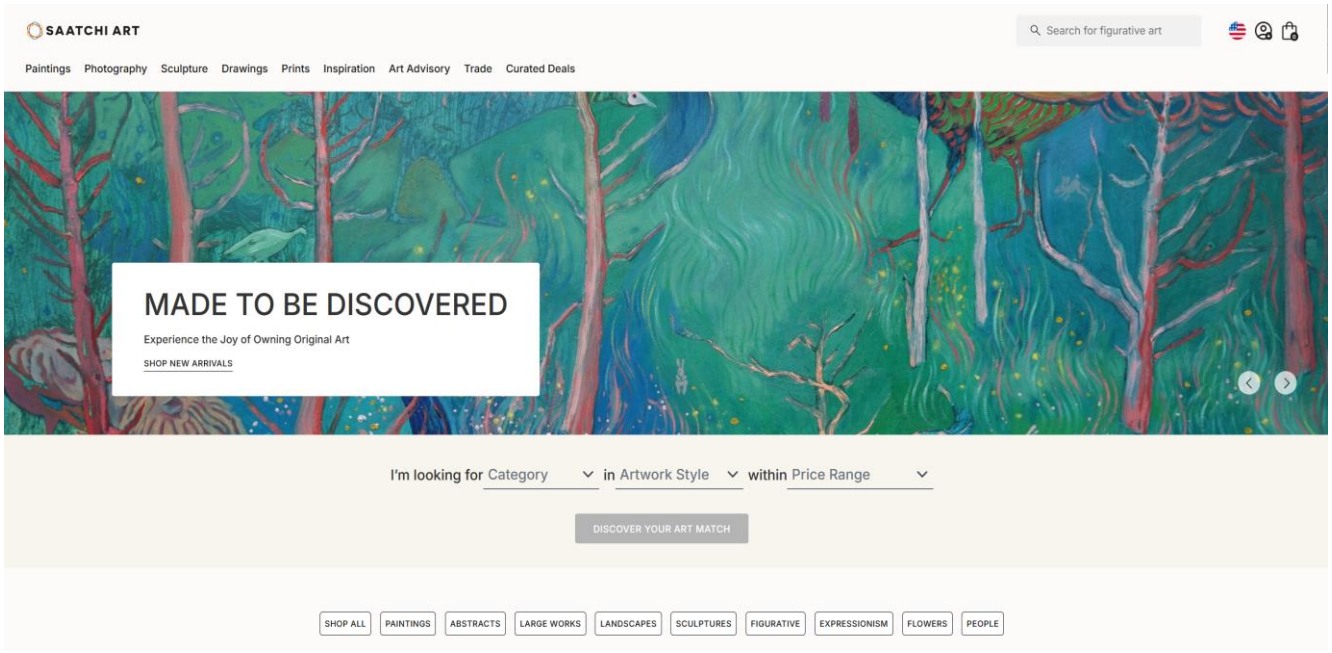

Рисунок 1.2 – Інтерфейс вебзастосунку Saatchi Art

Такий вигляд у аналогічного вебзастосунку Saatchi Art.

1.2.3 Аналіз вебзастосунку Phaidon

Phaidon (табл. 1.3, рис. 1.3) – сьогодні є унікальним прикладом того, як класичне видавництво зі світовим іменем успішно трансформувалося у цифрову платформу, що диктує тренди в індустрії мистецтва та дизайну [4]. Після інтеграції маркетплейсу Artspace у свою екосистему, Phaidon створив безшовний простір, де комерція нерозривно пов'язана з інтелектуальним контекстом. Це не просто інструмент для покупки картин, а елітарне середовище, що базується на майже віковій експертизі видавництва у сфері візуальної культури.

З технічної та концептуальної точок зору, вебзастосунок Phaidon реалізує модель «Content-First Commerce». Це означає, що процес вибору твору мистецтва завжди супроводжується глибоким аналізом: біографією автора, історією створення та критичними оглядами з архівів Phaidon. Такий підхід вирішує головну проблему онлайн-ринку – дефіцит фізичного контакту та довіри до об'єкта.

Архітектура платформи Phaidon фокусується на тиражному мистецтві та ексклюзивних колабораціях. Це дозволяє залучати аудиторію, яка прагне інвестувати у мистецтво, але шукає гарантії автентичності, які надає бренд Phaidon. Дизайн системи є ідеальни взірцем «тихої розкоші»: мінімалістична сітка, вишукана типографіка та відсутність зайвих елементів створюють ефект перебування у реальній галереї.

Таблиця 1.3 – Платформа Phaidon

Назва	Phaidon – елітарний маркетплейс тиражного та сучасного мистецтва.
Розробник	Phaidon Press.
Архітектура	Вебзастосунок (клієнт-сервер).
Мова реалізації	HTML/CSS/JavaScript.
Перелік функцій, характеристик	1) продаж ексклюзивних видань та лімітованих серій; 2) тісна інтеграція з базою знань Phaidon (історія мистецтв); 3) керівництва для колекціонерів; 4) закриті продажі для VIP-користувачів.
Переваги	1) висока якість відбору (тільки перевірені галереї та митці); 2) престижність бренду, пов’язаного з Phaidon; 3) зручний інтерфейс для покупки предметів дизайну та книг; 4) продаж книг разом з іншими творами мистецтва.
Недоліки	1) менший обсяг картин у порівнянні з іншими платформами; 2) орієнтація на дорогі сегменти ринку.

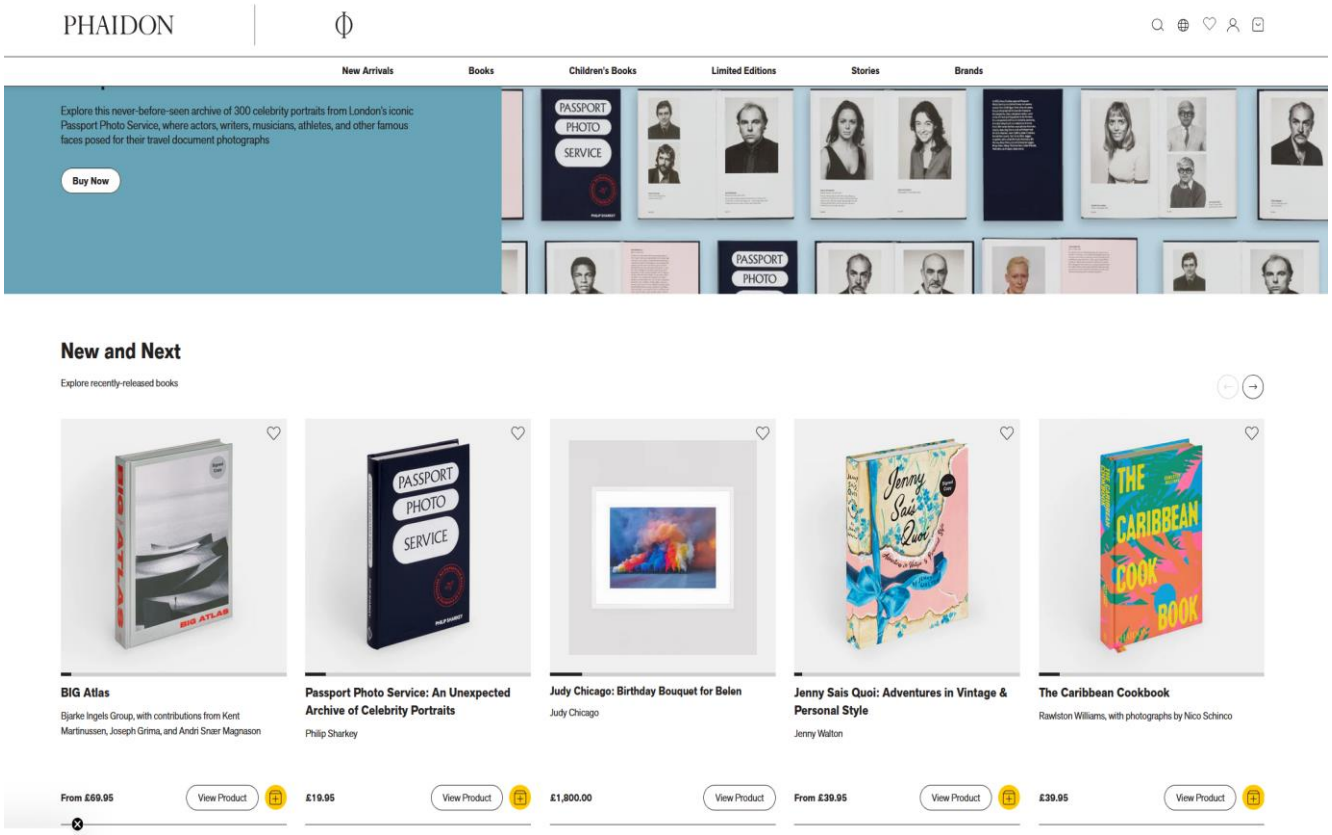

Рисунок 1.3 – Інтерфейс вебзастосунку Phaidon

Таким чином представлено вебзастосунок Phaidon.

1.2.4 Аналіз платформи мережі галерей Tate

Tate (табл. 1.4, рис. 1.4) – еталон того, як державна культурна інституція може трансформуватися у сучасний цифровий хаб [5]. На відміну від попередніх комерційних платформ, пріоритетом Tate є збереження, каталогізація та популяризація національного надбання. Система керування контентом цього ресурсу є однією з найкращих у галузі, оскільки вона оперує метаданими десятків тисяч об’єктів, кожен з яких має детальну історію реставрації, виставок та критичних відгуків.

Платформа реалізує концепцію відкритого музею. Окрім електронної комерції (продаж квитків, сувенірів та принтів), вебзастосунок пропонує величезний масив безкоштовного інтерактивного контенту: відео-майстеркласи, 360-градусні тури залами галерей та спеціалізовані дитячі розділи (Tate Kids) з елементами гейміфікації. Вебзастосунок орієнтований на широку аудиторію,

включаючи як відвідувачів музею, так і користувачів з усього світу, які цікавляться мистецтвом.

Таблиця 1.4 – Платформа мережі галерей Tate

Назва	Tate Gallery – цифрова платформа мережі галерей Tate.
Розробник	Tate Digital Team.
Архітектура	Платформа електронної комерції (клієнт-сервер).
Мова реалізації	HTML/CSS/JavaScript.
Перелік функцій, характеристик	1) присутній інтерактивний гейміфікований розділ для дітей; 2) цифровий архів метаданих (найбільший у Британії) ; 3) віртуальні 360-тури та онлайн-курси з мистецтва; 4) пряма інтеграція з квитковими системами та музейною крамницею.
Переваги	1) потужний освітній та науковий ресурс; 2) величезна база оцифрованих творів (70,000+ об'єктів); 3) безкоштовний доступ до більшості контенту.
Недоліки	1) це не маркетплейс в звичайному розумінні (продаються товари з музейного магазину, квитки та принти); 2) надмірна кількість розділів.

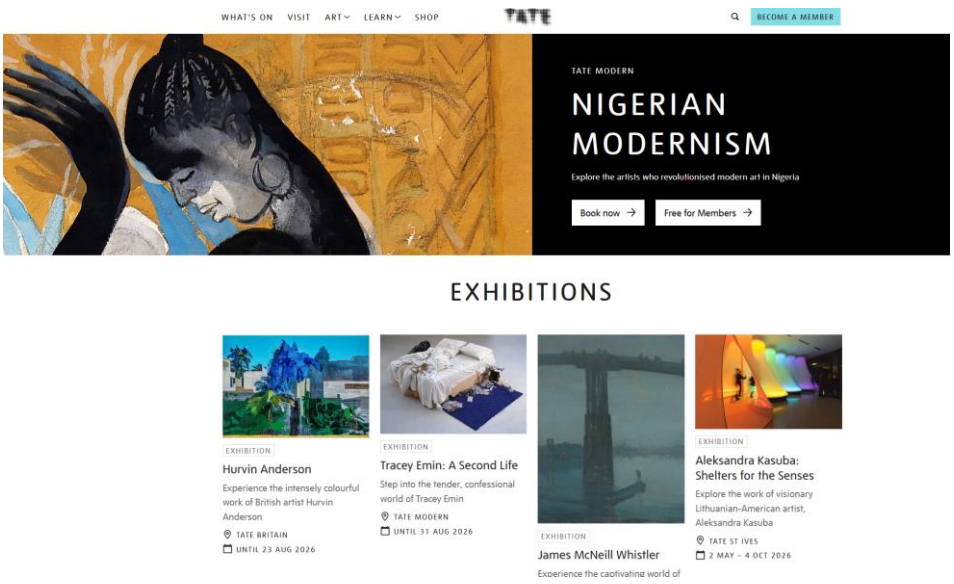

Рисунок 1.4 – Інтерфейс платформи Tate

Даний вигляд має платформа мережі галерей Tate.

Проведений порівняльний аналіз існуючих рішень у сфері онлайн-продажів творів мистецтва виявив суттєві відмінності між вузькоспеціалізованими галерейними платформами та відкритими глобальними маркетплейсами.

Встановлено, що професійні галерейні ресурси та видавничі доми орієнтовані на експонування обмеженого кола визнаних митців і забезпечують повний цикл супроводу клієнта: від надання експертної мистецтвознавчої консультації до видачі сертифікатів автентичності. Їхніми ключовими перевагами є високий рівень довіри, гарантована оригінальність робіт, ретельний кураторський відбір та персоналізований підхід до колекціонера. Проте вони мають обмеження у вигляді вузької спеціалізації на конкретних стилях чи авторах, а також високого цінового порогу, що робить їх менш доступними для масового споживача.

Розглядаючи глобальні арт-маркетплейси, можна визначити, що вони надають значно ширші можливості для користувачів, включаючи доступ до величезного масиву робіт різних технік, жанрів та цінових категорій – від оригінальних полотен до тиражних принтів. Ключовими перевагами таких платформ є їх масштабність, гнучкі інструменти пошуку на основі складних алгоритмів рекомендацій та впровадження інноваційних сервісів. Однак такі масштабні ресурси часто стикаються з проблемою перенасиченості контентом, що створює труднощі у виборі для покупця та підвищує ризики появи робіт низької художньої якості або сумнівної автентичності через вільний доступ для незалежних авторів.

1.3 Аналіз структурних і функціональних особливостей об’єкта дослідження

1.3.1 Структура системи

Вебзастосунок з продажу картин реалізується на основі клієнт-серверної архітектури та складається з таких основних компонентів:

– клієнтська частина (інтерфейс користувача);

– серверна частина (бізнес-логіка);
– база даних (зберігання інформації).

Клієнтська частина застосунку розроблена на базі стандартів HTML5, CSS3 та JavaScript, виступає в ролі інтерактивного інтерфейсу, що забезпечує візуалізацію даних та обробку подій користувача в середовищі веббраузера. Серверна частина виконує роль центрального вузла обробки інформації: вона відповідає за реалізацію бізнес-логіки, менеджмент вхідних запитів та безпечну комунікацію з рівнем даних. Реляційна база даних забезпечує персистентність (постійність) зберігання інформації, структуруючи дані про профілі користувачів, параметри художніх творів, стан кошика та історію оформлених замовлень.

1.3.2 Учасники системи

Зазвичай у системі виділяють такі ролі користувачів:

– покупець – здійснює пошук картин, переглядає інформацію та офрмлює замовлення;
– продавець (художник) – розміщує оголошення про продаж власної картини, редагує їх;
– адміністратор – редагує оголошення, керує користувачами та контентом, забезпечує стабільну роботу системи.

1.3.3 Функціональні можливості системи

Після проведення аналізу визначено основні функції, які характерні для подібних вебзастосунків:

1) для покупця:
   – реєстрація та авторизація;
   – перегляд каталогу товарів;
   – пошук та фільтрація;
   – перегляд детальної інформації;
   – додавання до обраного;
   – оформлення замовлення;

– вибір способу оплати.

2) для продавця:

– подання заявки на створення власного оголошення;

– редагування та видалення власних оголошень;

– завантаження фотографій картини.

3) для адміністратора:

– управління усіма користувачами;

– коригування опублікованих оголошень;

– контроль роботи системи.

1.4 Постановка задачі

На основі проведеного аналізу предметної області, особливостей ринку електронної комерції та існуючих програмних рішень встановлено, що сучасні вебзастосунки для продажу картин мають ряд функціональних та концептуальних обмежень.

Спеціалізовані галерейні вебзастосунки забезпечують високий рівень експертності, гарантують автентичність робіт та надають глибокий мистецтвознавчий контекст, проте вони зазвичай мають обмежений асортимент, орієнтовані на вузьке коло авторів і відзначаються високим ціновим порогом. З іншого боку, глобальні арт-маркетплейси пропонують широку базу творів різних жанрів та цінових категорій, але часто не можуть забезпечити належний рівень перевірки якості контенту та не надають індивідуального супроводу покупця.

У зв’язку з цим виникає необхідність розробки вебзастосунку, який би поєднував переваги професійних систем (довіра, якісний контент, експертність) із функціональними можливостями сучасних інтернет-магазинів (зручність пошуку, доступність, інтерактивність).

Тому метою даної роботи є розробка вебзастосунку для автоматизації процесу продажу картин, що забезпечує багатокористувацький доступ до каталогу художніх творів і надає зручні цифрові інструменти для їх пошуку, формування списків бажаного та безпечного придбання онлайн.

Висновки до розділу 1

У першому розділі проведено комплексний аналіз предметної області вебзастосунків для реалізації творів мистецтва, що дозволило визначити ключові особливості та тенденції розвитку цифрового арт-ринку. Встановлено, що сучасні інформаційні системи відіграють критичну роль у процесах експонування та продажу картин, забезпечуючи поціновувачам мистецтва зручний доступ до візуального контенту та спрощуючи механізми взаємодії між художниками (галереями) та покупцями.

Визначення об’єкту та предмету дослідження дало змогу чітко окреслити межі розроблюваної системи, виділити ключові бізнес-процеси, що потребують автоматизації, та визначити основні стратегічні напрямки її програмної реалізації.

Під час порівняльного аналізу існуючих програмних рішень встановлено, що спеціалізовані галерейні вебзастосунки та видавничі платформи забезпечують високу надійність і якісний кураторський відбір, проте часто обмежені вузьким колом авторів та високою вартістю робіт. Глобальні арт-маркетплейси, навпаки, надають необмежений вибір творів різних жанрів, проте мають ризики, пов’язані з перевіркою художньої якості та автентичності представлених об’єктів.

На основі проведеного аналізу предметної області та технологічного досвіду світових лідерів ринку сформульовано задачу кваліфікаційної роботи, яка полягає у розробці вебзастосунку для автоматизації процесу продажу картин. Майбутня система має поєднати в собі зручність користувацького інтерфейсу, надійність серверної логіки та ефективність управління базою даних.

2 МОДЕЛЮВАННЯ ОБ'ЄКТУ ТА ПРЕДМЕТУ

2.1 Аналіз сучасного стану інструментарію реалізації вебзастосунків

Вебзастосунки продажу картин працюють із великими масивами медіа-контенту, базами даних художніх творів та складними пошуковими запитами, що вимагає застосування сучасних архітектурних підходів. Розробка такого вебзастосунку потребує використання технологій, що забезпечують ефективну обробку даних, візуальну естетику інтерфейсу, масштабованість та надійний захист інформації.

Розглядаючи сучасні наукові дослідження, можна визначити, що вебзастосунки у сфері арт-комерції повинні базуватися на масштабованих архітектурах, які дозволяють обробляти значні обсяги графічних даних та забезпечують високу продуктивність. Одним із найбільш розповсюджених підходів є клієнт-серверна архітектура (рис. 2.1), яка забезпечує чіткий розподіл функцій між інтерфейсом користувача (Frontend) та серверною частиною (Backend).

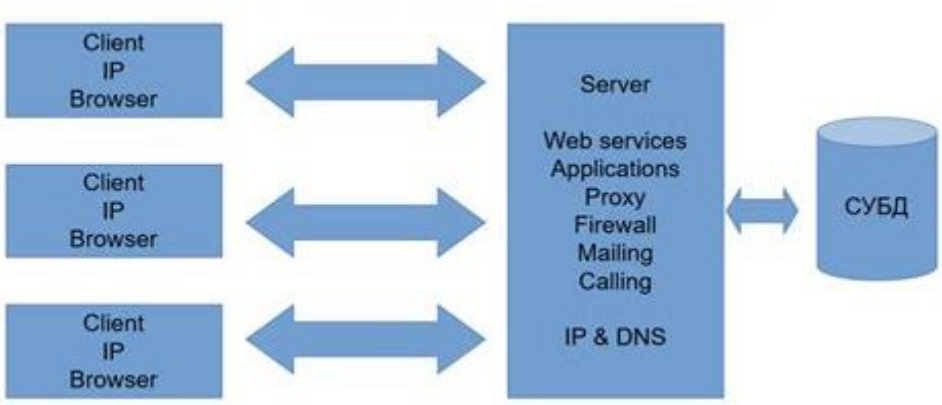

Рисунок 2.1 – Клієнт-серверна архітектура

Також у переглянутих наукових роботах досліджується використання мікросервісної та монолітної архітектур (рис. 2.2). Мікросервісна архітектура дозволяє розділити вебзастосунок на незалежні компоненти. Наприклад, модуль аутентифікації, каталог картин, платіжний шлюз, що можуть масштабуватися окремо. Монолітна архітектура пропонує простіший підхід, що є більш доцільним

для початкових етапів розробки або систем середнього масштабу [6, 7]. Хоча мікросервісний підхід підвищує гнучкість системи, він потребує складнішої реалізації та налагодження зв’язків між сервісами [8].

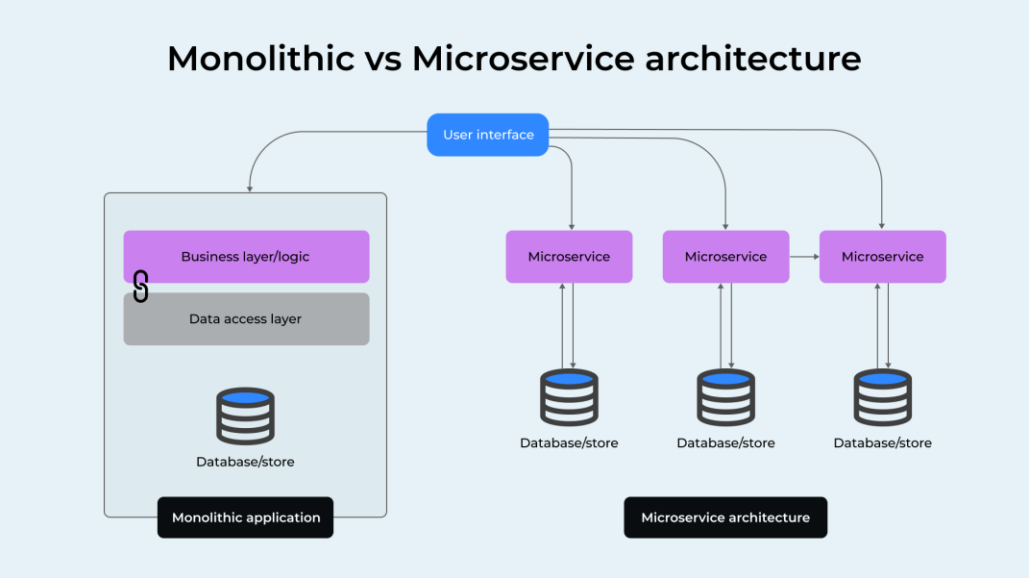

Рисунок 2.2 – Монолітна та мікросервісна архітектури

Для створення користувацького інтерфейсу вебзастосунків використовуються технології HTML5, CSS3 та JavaScript (рис. 2.3), які формують структуру, візуальне оформлення та інтерактивність вебсторінок [9].

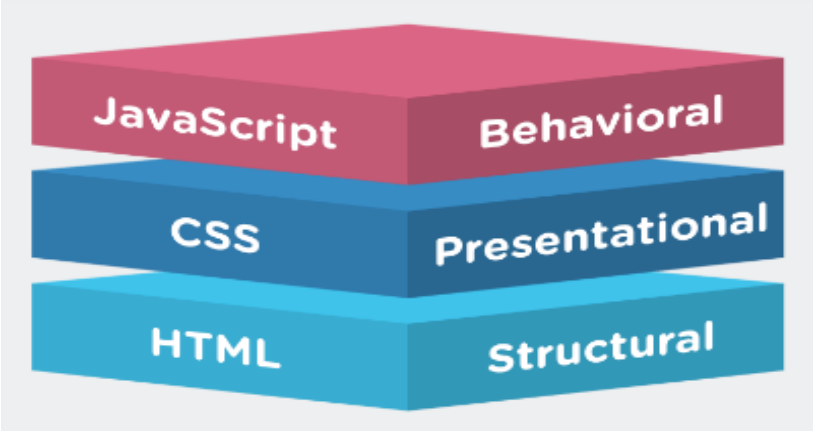

Рисунок 2.3 – HTML5, JavaScript та CSS3 [10]

Серед сучасних JavaScript-фреймворків найбільшу увагу приділяють Angular, React та Vue (табл. 2.1), які дозволяють реалізовувати односторінкові застосунки (SPA). Згідно з дослідженнями використання цих фреймворків дозволяє зменшити навантаження на сервер, забезпечити швидке оновлення інтерфейсу без

перезавантаження сторінки, підвищити продуктивність вебзастосунків та покращити взаємодію з користувачем [11].

Таблиця 2.1 – Angular, React та Vue

Характеристика	Angular	React	Vue
Підхід до розробки	На основі TypeScript	Все на JavaScript	На основі JavaScript та HTML
Коли обирати	Для розробки гібридних застосунків, або вебзастосунків	Для розробки SPA та мобільних застосунків для різних платформ	Для розробки розширених SPA
Використання	Створення великомасштабних, багатофункціональних застосунків, або застосунків корпоративного рівня	Створення сучасних вебзастосунків та застосунків для Android та iOS	Створення вебзастосунків та односторінкових застосунків
Зручність	Використання структурного фреймворку	Забезпечення гнучкого середовища розробки	Зосередженість на розділені завдань
Модель	На основі архітектури MVC (Model – View – Controller)	На основі Virtual DOM (Document object model)	На основі Virtual DOM (Document object model)
Написано на	TypeScript	JavaScript	JavaScript
Мова	Рекомендується використання TypeScript	Рекомендується використання JSX (JavaScript XML)	Шаблони HTML та JavaScript
Можливість повторного використання	Так	Ні, лише CSS	Так, HTML і CSS
Масштабованість	Модульна сруктура розробки	Компонентний підхід	Підхід на основі шаблонів

Серверна частина вебзастосунку може реалізовуватися на базі платформ Node.js, Java Spring, ASP.NET (табл. 2.2). Використання сучасних серверних платформ забезпечує високу швидкість обробки асинхронних запитів, що важливо для динамічного завантаження великої кількості зображень та обробки замовлень у реальному часі [12, 13].

Таблиця 2.2 – Node.js, Java Spring та ASP.NET

Характеристика	Node.js	Java Spring	ASP.NET
Мова програмування	JavaScript / TypeScript	Java	C#, F#
Архітектура	Однопотокова, асинхронна	Багатопотокова, класична обробка запитів	Багатопотокова
Продуктивність	Стабільно для великої кількості одночасних підключень	Висока надійність та стабільність	Дуже висока швидкість виконання, оптимізована під сервери Microsoft, висока продуктивність
Екосистема	Великий вибір модулів (npm), швидкий старт, слабке типування	Сувора типізація, корпоративні стандарти, велика кількість готових рішень	Тісна інтеграція з інструментами Microsoft (Visual Studio, Azure)
Використання	Стартапи, вебзастосунки в реальному часі (Real-time)	Великі корпоративні системи, банки, страхові компанії, промисловість	Корпоративні системи, інтеграція з екосистемою Windows, потужні вебзастосунки

Також доволі важливим елементом сучасної розробки є використання систем контролю версій. Git (рис. 2.4) дозволяє зберігати історію змін проєкту, забезпечує можливість командної розробки та гарантує стабільність програмного продукту шляхом керування різними гілками коду.

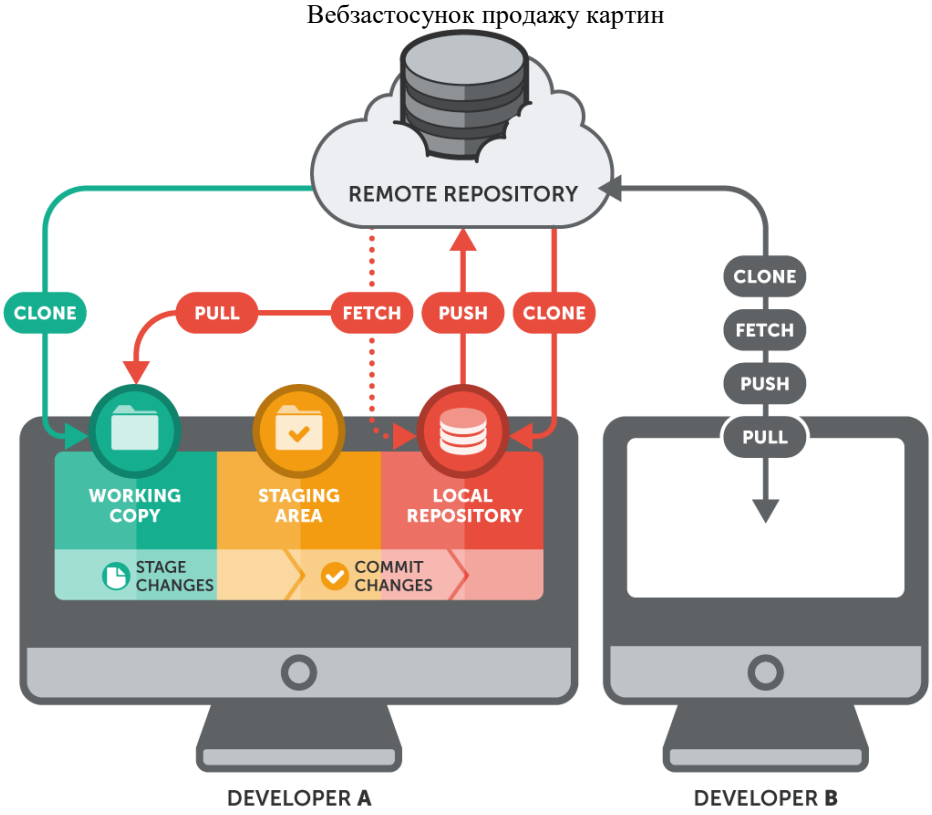

Рисунок 2.4 – Схема роботи системи контролю версій Git [14]

Для постійного зберігання даних у вебзастосунках найчастіше використовуються реляційні бази даних, такі як: MySQL, PostgreSQL, Microsoft SQL Server та Oracle (табл). Вони забезпечують цілісність інформації про товар та його детальну інформацію, підтримку транзакцій під час оформлення покупок та ефективне виконання складних аналітичних запитів [15].

Таблиця 2.3 – MySQL, PostgreSQL та Microsoft SQL Server

Характеристика	MySQL	PostgreSQL	SQL Server
Розробник	Oracle Corporation	Глобальна група розробників PostgreSQL	Microsoft
Ліцензія	З відкритим вихідним кодом (GPL), комерційна	З відкритим вихідним кодом (ліцензія PostgreSQL)	Власницька версія (з безкоштовною Express-версією)

Кінець таблиці 2.3

Підтримка операційних систем	Кросплатформний	Кросплатформний	Windows, Linux (обмежена підтримка)
Відповідність ACID	Так (з рушієм InnoDB)	Повністю сумісний	Повністю сумісний
Відповідність стандартам	Помірна	Висока (близько до стандартів SQL)	Висока
Продуктивність	Швидка для робочих навантажень з великим обсягом читання даних	Краще для складних запитів та записів	Оптимізована для корпоративних навантажень
Підтримка JSON	Так	Розширена підтримка JSON та індексації	Так
Процедурна мова	SQL, обмежені збережені процедури	PL/pgSQL, PL/Python, PL/Perl, тощо	T-SQL
Підтримка	Комерційна підтримка від Oracle	Комерційна підтримка (EDB)	Сильна комерційна підтримка від Microsoft
Інструменти графічного інтерфейсу	MySQL Workbench, phpMyAdmin	pgAdmin, Dbeaver	SQL Server Management Studio
Безпека	Добра, SSL, дозволи користувачів	Сильна, захист на рівні рядків, SSL	Сильна, інтегрована з Windows AD
Варіанти використання	Вебзастосунки, CMS	Аналітика, геопросторові застосунки, фінанси	Корпоративні застосунки, бізнес-аналітика, великомасштабні бази даних

Проаналізувавши стан інструментарію можна сказати, що для розробки вебзастосунку продажу картин доцільно обрати клієнт-серверну архітектуру,

сучасні серверні платформи та реляційні бази даних (наприклад Angular, Node.js та MySQL) для ефективної реалізації функціональних можливостей системи.

2.2 Методи та моделі розробки вебзастосунку

Процес розробки вебзастосунку продажу картин засновується на використанні сучасних методів програмної інженерії. Це дозволяє забезпечити високу якість, надійність та масштабованість програмного забезпечення.

Одним із найбільш ефективних підходів є ітеративно-інкрементна модель розробки. Вона передбачає створення системи поетапно з поступовим додаванням нових функціональних можливостей та вдосконаленням вже існуючих. Такий підхід дозволяє гнучко реагувати на зміни вимог та мінімізувати ризики при розробці складних систем електронної комерції.

Також значна увага приділяється концепції вебінженерії. Вона передбачає системний підхід до життєвого циклу вебзастосунків і включає детальний аналіз вимог, проєктування, реалізацію та тестування [16]. Використання цієї методології дозволяє підвищити стабільність програмного продукту та забезпечити його максимальну відповідність потребам цільової аудиторії колекціонерів та покупців мистецтва.

Розробка системи ґрунтується на об’єктно-орієнтованому підході, що дозволяє представити предметну область у вигляді сукупності взаємопов’язаних об’єктів, таких як: користувач, картина, категорія та замовлення. Це забезпечує модульність системи, можливість повторного використання коду та значно спрощує подальший супровід продукту. [17]

При побудові архітектури системи можуть застосовуватися різні парадигми. У сучасних вебзастосунках базовою є клієнт-серверна архітектура, яка визначає принципи взаємодії між клієнтською та серверною частинами. Водночас внутрішня організація серверної частин може реалізовуватись за допомогою монолітної або мікросервісної архітектури. Для масштабних систем доцільним є використання мікросервісного підходу, що дозволяє незалежно масштабувати окремі компоненти та має такі переваги: покращена доступність, відмовостійкість, горизонтальна

масштабованість і велика гнучкість розробки. Однак, для систем, що не мають тисяч одночасно працюючих користувачів та здатні до ефективного вертикального масштабування, перехід від монолітної архітектури до мікросервісної може не принести тих переваг, про які повідомляють провідні світові компанії. Тому для невеликих та середніх проєктів ефективним залишається використання монолітної архітектури [6, 7]. Також пропонується використовувати гібридний метод масштабування, заснований на поєднанні двох підходів: мікросервісний для високонавантажених модулів (наприклад, сервіс обробки та збереження зображень картин у високій роздільній здатності) та монолітний підхід для менш навантажених частин базової бізнес-логіки [8].

Для концептуального моделювання предметної області та структури системи застосовуються стандартизовані нотації, такі як UML та ER-моделювання. Вони дають змогу візуалізувати архітектуру, сценарії взаємодії користувачів із платформою, структуру реляційної бази даних та алгоритми оформлення замовлень.

Для забезпечення безперебійної роботи та високої якості програмного забезпечення застосовуються різні рівні тестування:

– модульне тестування;
– інтеграційне тестування;
– функціональне тестування.

Таким чином, розглянуті сучасні методи та моделі розробки дозволяють спроєктувати та реалізувати ефективний, безпечний та масштабований вебзастосунок продажу картин.

2.3 Специфікація вимог до програмного забезпечення

1. Призначення та межі проєкту

1.1 Призначення системи

Програмне забезпечення являє собою вебзастосунок, головним призначенням якого є забезпечення багатокористувацького доступу до

електронного каталогу творів мистецтва. Система створена для автоматизації онлайн-продажу картин.

1.2 Погодження, що ухвалені в програмній документації

Програмний продукт реалізується як сучасний вебзастосунок на базі клієнт-серверної архітектури з використанням реляційної системи управління базами даних. Доступ користувачів до функціональних можливостей системи забезпечується за допомогою стандартних веббраузерів, що гарантує кросплатформенність рішення.

1.3 Межі проєкту ПЗ

– реалізація вебверсії з можливістю адаптації під мобільні та десктопні платформи;

– інтеграція зовнішніх програмних інтерфейсів (API) для оптимізації роботи системи;

– забезпечення повного функціоналу користувача в межах єдиної програмної платформи.

2. Загальний опис

2.1 Сфера застосування

Система призначена для використання компанією-дистриб'ютором з метою організації централізованого онлайн-продажу картин та предметів образотворчого мистецтва. Вебзастосунок виступає основним інструментом цифрової комерції, забезпечуючи представлення наявного асортименту товарів широкому колу покупців та автоматизацію процесу обробки замовлень.

2.2 Характеристики користувачів

– клієнт – здійснює пошук картин у каталозі, переглядає інформацію формує списки бажаного (якщо зареєстрований) та здійснює придбання обраних товарів онлайн (якщо зареєстрований);

– дистриб'ютор – забезпечує централізоване представлення асортименту картин, управління статусами замовлень та надання актуальної інформації про наявність художніх робіт у режимі реального часу.

2.3 Загальна структура системи

Система складається з:

– адаптивного вебінтерфейсу користувача;

– серверної логіки;

– реляційної бази даних.

2.4 Загальні обмеження

– доступ через Інтернет;

– підтримка сучасних браузерів;

– кількість одночасних користувачів яку дозволяють можливості сервера.

3. Функції системи

3.1 Реєстрація користувача

Функція дозволяє створити обліковий запис. Вхідна інформація – ім’я користувача, електронна пошта та пароль. Вихідна інформація – вхід/створення облікового запису. Функціональні вимоги: перевірка унікальності email; збереження даних користувача; збереження стану логування користувача в системі.

3.2 Перегляд каталогу картин

Функція відображає список наявних у системі картин. Вхідна інформація – звернення користувача до розділу каталогу. Вихідна інформація – повний перелік наявних картин. Функціональні вимоги: візуалізація карток товарів із назвою, ціною та зображенням кожної роботи.

3.3 Керування списком бажаного

Функція дозволяє користувачеві зберігати обрані художні твори для подальшого перегляду. Вхідна інформація – ID картини та запит на додавання або видалення. Вихідна інформація – оновлений персональний список обраних картин. Функціональні вимоги: додавання картини до списку; видалення зі списку; відображення переліку вподобаних картин на окремій сторінці.

3.4 Оформлення заяви на покупку

Функція створює заяву на купівлю. Вхідна інформація – обрана картина; дані користувача (прізвище, місце проживання, інформація про банківську картку).

Вихідна інформація – повідомлення про створення та збереження заяви. Функціональні вимоги: створення заяви; збереження стану заяви в базі даних.

3.5 Адміністрування каталогу

Функція додавання, редагування та видалення картин у будь-який час. Вхідна інформація – дані картини. Вихідна інформація – оновлення поточного списку картин, актуалізований стан бази даних. Функціональні вимоги: створення картки товару; редагування інформації про лот; видалення застарілої інформації про картини.

4. Вимоги до інформаційного забезпечення

4.1 Джерела і зміст вхідної інформації

– облікові дані користувачів (ім’я, адреса електронної пошти, пароль);
– дані для оформлення замовлення (повна контактна інформація);
– дані про художні твори (назва картини, ціна, зображення).

4.2 Нормативно-довідкова інформація

– назви картин;
– перелік художніх стилів;
– цінові категорії товарів.

4.3 Організація збереження інформації

Інформація зберігається у реляційній базі даних. Регулярно виконується резервне копіювання для запобігання втрати даних.

5. Вимоги до технічного забезпечення

– сервер з підтримкою вебсерверного ПЗ;
– клієнтський пристрій з підтримкою запуску веббраузерів;
– доступ до мережі Інтернет.

6. Вимоги до програмного забезпечення

6.1 Архітектура системи

Клієнт-серверна архітектура.

6.2 Системне програмне забезпечення

Вебзастосунок реалізовується за допомогою фреймворку Angular. Серверна логіка виконана на платформі Node.js. В якості бази даних системи виступає Microsoft SQL Server.

6.3 Мережеве ПЗ

Використання протоколу HTTPS.

6.4 ПЗ ведення бази даних

Система керування базами даних Microsoft SQL Server.

6.5 Мови та технології

Frontend: HTML, CSS, Angular, JavaScript, Bootstrap, TypeScript. Backend: Node.js, JavaScript. База даних: система керування базами даних Microsoft SQL Server.

7. Вимоги до зовнішніх інтерфейсів

7.1 Інтерфейс користувача

Графічний вебінтерфейс із адаптивним та мінімалістичним дизайном.

7.2 Апаратний інтерфейс

Сучасний пристрій з доступом до веббраузера через Інтернет.

7.3 Програмний інтерфейс

REST API для обміну даними.

7.4 Комунікаційний протокол

HTTPS.

8. Властивості програмного забезпечення

8.1 Доступність

Система доступна цілодобово (24/7), за винятком часу проведення технічних робіт або аварійних ситуацій на стороні серверного обладнання чи хостинг-провайдера. Вебзастосунок має коректно працювати на різних типах пристроїв (ПК, ноутбуки, мобільні девайси) та забезпечувати доступ до інформації через веббраузер без необхідності встановлення додаткового програмного забезпечення.

8.2 Супроводжуваність

Програмне забезпечення повинно бути зручним для супроводу та оновлення.

8.3 Переносимість

Програмна система має бути інфраструктурно незалежною, що забезпечує можливість її міграції та розгортання в різних серверних середовищах або на сторонніх хостинг-майданчиках без необхідності модифікації вихідного коду.

8.4 Продуктивність

Програмне забезпечення повинно забезпечувати високу швидкість відгуку інтерфейсу та стабільну обробку запитів при нормальному навантаженні.

8.5 Надійність

Система повинна забезпечувати стабільну та коректну роботу протягом тривалого часу. Надійність реалізується через регулярне резервне копіювання бази даних для гарантування цілісності інформації.

8.6 Безпека

– шифрування даних;

– автентифікація користувачів;

– захист від SQL-ін’єкцій.

Комплексна реалізація зазначених заходів безпеки дозволяє мінімізувати ризики зовнішніх загроз та гарантувати високий рівень захищеності інформаційних ресурсів системи в межах архітектури клієнт-серверної взаємодії.

9. Інші вимоги

Система повинна:

– мати можливість подальшого розширення функціоналу;

– відповідати вимогам чинних стандартів та рекомендацій щодо розробки вебзастосунків;

– мати адаптивний інтерфейс, який буде коректно відображатися на різних розширеннях екранів.

Висновки до розділу 2

У другому розділі проведено аналіз методологічних підходів та інструментальних засобів розробки вебзастосунків для електронної дистрибуції художніх творів. Визначено, що поєднання принципів об’єктно-орієнтованого програмування та клієнт-серверної архітектури є оптимальним для створення

масштабованої системи з ефективним управлінням реляційними базами даних. Детально розглянуті технологічні рішення доводять свою здатність забезпечити надійну обробку клієнтських запитів та високу швидкість обміну інформацією між сервером і клієнтом.

На основі аналізу предметної області сформовано специфікацію функціональних та нефункціональних вимог. Основну увагу приділено забезпеченню цілісності даних при класифікації картин за стилями, швидкодії інтерфейсу та стабільності системи під навантаженням. Обґрунтовано необхідність адаптивності програмного рішення для коректної роботи на різних типах пристроїв. Детальне опрацювання цих аспектів гарантує створення інтуїтивно зрозумілого середовища для взаємодії з каталогом, кошиком та списком бажаного. Такий підхід зводить до мінімуму технічні ризики і створює максимально комфортний досвід для майбутніх покупців мистецтва.

Отримані результати проведеного аналізу та формалізовані вимоги є фундаментальною теоретичною та практичною базою для подальшого етапу проектування. Вони дозволяють перейти до побудови UML-діаграм та безпосередньої програмної реалізації вебзастосунку, гарантуючи відповідність розробки встановленим стандартам якості та потребам користувачів.

3 ПРОЄКТУВАННЯ ТА КОНСТРУЮВАННЯ ВЕБЗАСТОСУНКУ

Проєктування програмного забезпечення є одним із найголовніших та найважливіших етапів життєвого циклу розробки будь-якого IT-продукту. Цей процес виступає критичною сполучною ланкою між етапом збору та аналізу вимог і безпосереднім написанням програмного коду. Саме на етапі проєктування визначається глобальна архітектурна структура майбутнього вебзастосунку, його базовий функціонал, здатність до безперебійної роботи під навантаженням, а також потенціал для безперервного розвитку та розширення в майбутньому. Процес проєктування передбачає розробку чіткого, логічно обґрунтованого плану дій, вибір оптимальних методів представлення даних, застосування передових стилів і шаблонів архітектури, а також визначення стратегій вирішення складних алгоритмічних завдань.

Глибоке та деталізоване проєктування програмного забезпечення виконується з метою досягнення таких ключових цілей:

– обґрунтування вибору технологічного стеку та інструментальних засобів розробки, які найкращим чином відповідають специфіці поставленого завдання;

– формування деталізованої архітектурної моделі, що дає можливість попередньо оцінити структурну складність розробки, необхідні обчислювальні ресурси та орієнтовні терміни реалізації проєкту;

– мінімізація технічних ризиків шляхом виявлення потенційних логічних конфліктів та архітектурних вразливостей ще до початку написання коду, що дозволяє уникнути критичних помилок і виключає виконання зайвих або дублюючих дій;

– створення вичерпної технічної документації, яка гарантує однозначне уявлення про можливості майбутнього продукту, його інтерфейси та логіку взаємодії внутрішніх компонентів;

– забезпечення високого рівня масштабованості, що дозволяє заздалегідь продумати, як саме IT-продукт адаптуватиметься до збільшення кількості

користувачів або обсягу даних у базі, та як найефективніше інтегрувати новий функціонал без руйнування існуючої архітектури.

Відсутність або недостатня увага до етапу проєктування часто призводить до виникнення необхідності в масштабному рефакторингу коду на пізніх етапах розробки. Тому детальне моделювання є обов'язковою умовою для створення надійного програмного продукту.

Проєктування сучасного вебзастосунку виконується з використанням загальноприйнятих інженерних підходів та стандартизованих методів моделювання програмного забезпечення. У цьому процесі застосовуються функціональні, структурні, інформаційні та поведінкові моделі, які в комплексі дозволяють всебічно описати роботу системи. Для візуального представлення логіки роботи системи найчастіше використовуються діаграми уніфікованої мови моделювання. Вони поділяються на структурні та поведінкові. Їх комплексне використання дозволяє наочно змоделювати кроки взаємодії користувачів із системою, проаналізувати життєвий цикл об'єктів та логіку виконання основних функцій вебзастосунку, таких як навігація каталогом чи оформлення замовлень.

Окрема увага на етапі проєктування приділяється розробці інформаційної моделі системи. За допомогою сутнісно-зв'язкових діаграм формується фундамент для побудови реляційної бази даних. Процес моделювання даних включає визначення ключових сутностей, їхніх атрибутів та типів зв'язків між ними, що забезпечує нормалізацію даних та гарантує їхню цілісність у середовищі Microsoft SQL Server.

Загалом, поточний етап охоплює проєктування оптимальної структури бази даних, побудову відмовостійкої клієнт-серверної архітектури, визначення протоколів взаємозв’язків між frontend та backend-модулями, а також розробку основних алгоритмів обробки інформації. Проведений комплекс робіт з моделювання дає можливість всебічно оцінити ефективність запропонованих архітектурних рішень та формує міцну, науково обґрунтовану основу для подальшого конструювання програмного забезпечення.

3.1 Архітектура та стек технологій

Архітектура сучасних вебзастосунків у сфері електронної комерції будується за фундаментальним принципом клієнт-серверної взаємодії. Цей підхід забезпечує чітке структурне розділення між клієнтською частиною, яка відповідає за візуальне представлення даних, серверною логікою та системою збереження інформації. Розподілена архітектура дозволяє досягти високого рівня масштабованості, спрощує подальший супровід програмного коду та гарантує ефективну обробку значного масиву паралельних запитів користувачів.

Базовим механізмом інтеграції між компонентами системи виступає архітектурний стиль REST API. Завдяки цьому протоколу клієнтський застосунок генерує та надсилає стандартизовані HTTP-запити до серверного середовища, отримуючи структуровані відповіді у форматі JSON. Використання такого формату обміну даними забезпечує повну незалежність розробки фронтенд і бекенд частин, оптимізує процеси маршрутизації та створює надійне підґрунтя для горизонтального масштабування вебзастосунку в майбутньому.

Клієнтська частина системи виконує роль основного інтерфейсу взаємодії користувача з платформою, забезпечуючи формування візуального простору, обробку інтерактивних подій та динамічне відображення контенту. Для побудови високоефективного фронтенду застосовується комплекс сучасних вебтехнологій:

– HTML5 – для створення структури вебсторінок та опису елементів інтерфейсу;

– CSS3 – для оформлення зовнішнього вигляду вебзастосунку, створення адаптивного дизайну та реалізації анімацій;

– JavaScript – використовується для підтримки роботи окремих бібліотек та взаємодії браузера з вебсторінкою;

– Angular – фреймворк для створення односторінкового застосунку (SPA). Він дозволяє реалізувати компонентну архітектуру, маршрутизацію, двосторонню взаємодію з даними та оновлення інтерфейсу без перезавантаження сторінки [18];

– TypeScript – як основна мова програмування клієнтської частини, що забезпечує типізацію, покращує читабельність коду та спрощує супровід великих проєктів.

Окрема увага приділяється адаптивності інтерфейсу клієнтської частини. Гнучка верстка дозволяє системі автоматично підлаштовуватися під роздільну здатність екрана, що гарантує коректне відображення візуальних елементів каталогу на персональних комп'ютерах, планшетах та мобільних пристроях.

Серверна частина системи реалізується на базі програмної платформи Node.js, що використовує асинхронну, подійно-орієнтовану модель обробки запитів. Такий вибір забезпечує високу пропускну здатність бекенду при виконанні складної бізнес-логіки. Серверна частина відповідає за вирішення наступних завдань:

– авторизацію та автентифікацію користувачів;

– обробку інформаційного масиву щодо художніх творів;

– адміністрування каталогу картин, включаючи створення, редагування та видалення відповідних карток товарів;

– валідацію та фіксацію заявок на придбання об'єктів мистецтва;

– виконання стандартизованих операцій при взаємодії з реляційною базою даних;

– генерацію коректних відповідей у межах налаштованих маршрутів REST API.

У якості системи керування базами даних обрано рішення Microsoft SQL Server. Вона гарантує надійне збереження інформації, підтримує механізми транзакційності та забезпечує сувору цілісність даних на рівні табличних зв'язків. У структурі бази даних централізовано зберігаються:

– дані користувачів;

– інформація про картини;

– історія замовлень;

– допоміжна службова інформація для конфігурації вебзастосунку.

Загальний алгоритм взаємодії між компонентами системи має чітку логічну послідовність. Спочатку користувач через адаптивний вебінтерфейс ініціює певну дію, зокрема перегляд галереї картин або підтвердження замовлення. Далі Angular-застосунок формує відповідний HTTP-запит та спрямовує його до REST API серверної частини. Платформа Node.js [17] перехоплює запит, виконує необхідну бізнес-логіку, формує звернення до бази даних Microsoft SQL Server та після отримання результату генерує відповідь клієнту у форматі JSON. На фінальному етапі отримані дані оброблюються frontend та відображаються користувачу без необхідності оновлення сторінки.

Для забезпечення безпеки застосунку використовуються механізми:

– автентифікації та авторизації користувачів;
– валідації введених даних;
– захисту від SQL-ін’єкцій;
– застосування сучасних протоколів для генерації токенів автентифікації.

Загалом, інтеграція обраного стеку технологій забезпечує високі показники продуктивності вебзастосунку, закладає міцний фундамент для зручного супроводу програмного коду та дозволяє реалізувати інтуїтивно зрозумілий адаптивний інтерфейс.

Перевагами обраної архітектури є:

– модульність та зручність супроводу системи;
– незалежність клієнтської та серверної частин;
– широкі можливості для горизонтального масштабування backend;
– висока швидкість реакції інтерфейсу на дії користувача;
– кросбраузерна сумісність та коректна робота на різних типах пристроїв;
– високий рівень захисту конфіденційних даних клієнтів.

Таким чином, спроєктована архітектура та узгоджені програмні компоненти гарантують ефективну реалізацію всіх заявлених функціональних можливостей вебзастосунку для продажу картин та створюють надійну технологічну базу для майбутньої модернізації системи.

3.2 Моделювання варіантів використання системи

Моделювання варіантів використання є невід'ємною складовою процесу проєктування, що дозволяє чітко формалізувати функціональні вимоги до програмного забезпечення. Вони трансформують загальні бізнес-вимоги у зрозумілі алгоритмічні сценарії, які забезпечують прозорий зв’язок між концепцією проєкту та її технічною реалізацією. Варіанти використання розробляються з метою точного визначення меж проєкту, формування вичерпного переліку системних функцій, створення критеріїв для майбутнього тестування продукту, а також для оптимізації процесу розробки шляхом декомпозиції завдань.

1) Коротка форма.

Процес ідентифікації користувача розпочинається, коли неавторизований відвідувач відкриває вебзастосунок та ініціює вхід через відповідний елемент інтерфейсу. Система відображає вікно, де особа обирає необхідну дію: створити новий обліковий запис або пройти автентифікацію в існуючому. Після заповнення форми та перевірки даних системою, статус відвідувача змінюється на авторизованого користувача, що відкриває доступ до розширених функцій застосунку.

Перегляд каталогу художніх творів є базовою функцією системи, доступною всім категоріям відвідувачів. Після завантаження головної сторінки формується візуальне представлення повного переліку наявних картин. Інтерфейс реалізовано у вигляді набору карток товарів, кожна з яких містить графічне зображення твору, його назву та актуальну вартість.

Управління списком бажаного дозволяє авторизованим клієнтам зберігати інформацію про картини, які їх зацікавили, без негайного додавання до кошика. Користувач може легко додавати нові лоти до свого персонального списку або видаляти ті, що втратили актуальність.

Процес оформлення замовлення ініціюється виключно авторизованим користувачем після додавання обраних картин до кошика та переходу до етапу підтвердження покупки. Користувач заповнює форму оформлення транзакції, де

відбувається клієнтська та серверна валідація введених даних. При успішній валідації в базі даних створюється відповідний запис, а користувач отримує системне сповіщення в інтерфейсі про успішне збереження замовлення.

2) Поверхнева форма.

Реєстрація або авторизація користувача

Головний сценарій: процес розпочинається з моменту, коли відвідувач натискає на інтерактивний елемент авторизації у головному меню вебзастосунку. Клієнтська частина системи реагує на цю подію та відображає модальне вікно. Користувач вводить свої автентифікаційні дані, після чого підтверджує відправлення форми. Вебзастосунок формує стандартизований мережевий запит і передає ці дані на серверну частину системи. Сервер приймає запит, забезпечує безпечну обробку отриманої інформації та виконує звернення до бази даних для пошуку відповідного запису в системі користувачів. При успішному збігу облікових даних та верифікації безпеки, серверна частина генерує унікальний ідентифікатор сесії. Цей ключ передається назад на клієнтську частину, де фіксується стан поточного сеансу, а інтерфейс застосунку динамічно змінює статус користувача на авторизованого, відкриваючи доступ до розширеного функціоналу платформи.

Альтернативні сценарії:

– якщо користувач припускається помилки при введенні пароля або електронної пошти, серверна частина після перевірки в базі даних повертає помилку автентифікації, а система відображає попереджувальне повідомлення про невідповідність даних;

– у випадку спроби входу під даними, які повністю відсутні в системі, платформа ідентифікує відсутність такого облікового запису та пропонує пройти процедуру реєстрації з внесенням нових даних до бази;

– за умови виникнення технічних збоїв або відсутності стабільного зв'язку з сервером бази даних, система перехоплює помилку та виводить інформаційне користувачу повідомлення з проханням спробувати виконати вхід пізніше.

Оформлення замовлення

Головний сценарій: авторизований користувач аналізує загальний каталог картин, додає необхідний лот до свого персонального кошика та переходить до спеціалізованого розділу підтвердження покупки. Система динамічно зчитує вміст кошика з бази даних та відображає форму для введення інформації про замовлення. Користувач заповнює необхідні поля, вказуючи дані для ідентифікації транзакції, та підтверджує намір здійснити покупку. Вебзастосунок проводить первинну перевірку правильності заповнення полів на стороні клієнта, а потім передає сформований пакет даних на серверну частину. Сервер виконує фінальну валідацію та відкриває транзакцію в базі даних. Програма одночасно створює новий головний запис у системі замовлень та деталізовані записи у суміжних таблицях, де фіксуються конкретна картина, її кількість та актуальна ціна. Паралельно система автоматично оновлює дані товарного каталогу, зменшуючи показник доступної кількості на складі на відповідну величину. Після успішного завершення транзакції кошик користувача очищається, а в інтерфейсі з'являється повідомлення про успішне збереження та реєстрацію замовлення.

Альтернативні сценарії:

– користувач намагається оформити замовити картину, її немає в наявності. Система блокує кнопку покупки;

– під час заповнення фінальної форми користувач вводить невалідні або неповні дані, серверна частина відхиляє запит, а інтерфейс підсвічує конкретні помилки у полях форми;

– під час заповнення форми відбувається збій. Система пропонує спробувати оформити замовлення пізніше;

Управління списком бажаного

Головний сценарій: користувач під час ознайомлення з художніми творами в загальному каталозі виявляє зацікавленість у конкретному лоті та натискає на відповідну графічну іконку збереження, розташовану безпосередньо на картці картини. Після успішного запису, сайт повертає підтвердження, а інтерфейс

вебзастосунку миттєво оновлює візуальний стан іконки, підтверджуючи користувачу успішне збереження твору мистецтва.

Альтернативні сценарії:

– користувач повторно натискає на іконку вже збереженої картини. У цьому випадку система інтерпретує дію як бажання видалити твір зі списку, товар видаляється;

– відбувається технічний збій. Система проводить повторну спробу додати товар до списку бажаного.

Варіанти використання системи демонструють взаємодію користувачів вебзастосунку із основними функціями системи, такими як: реєстрація/авторизація, перегляд каталогу картин, оформлення замовлень та керування товарами, додавання до списку бажаного.

На основі даних описаних в варіантах використання побудовано діаграму використання (рис. 3.1).

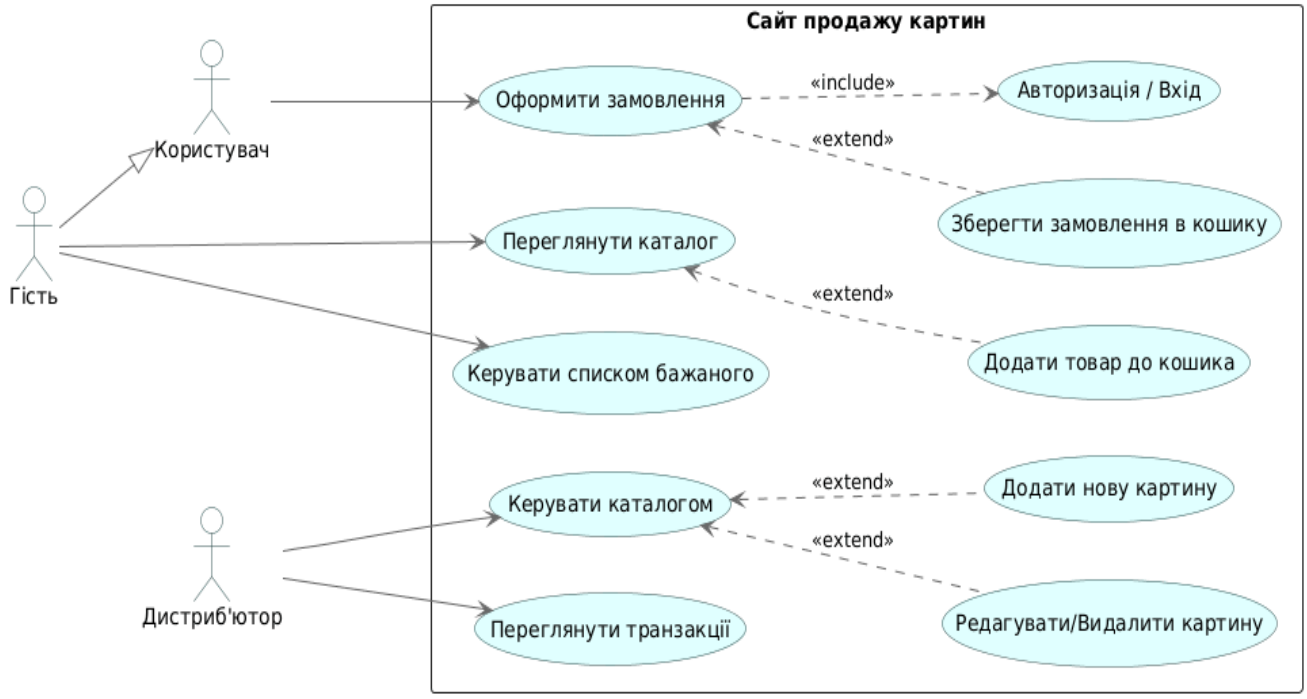

Рисунок 3.1 – Діаграма варіантів використання

Діаграма варіантів використання вебзастосунку продажу картин містить трьох головних акторів: гостя, покупця та дистриб’ютора.

3.3 Графічне представлення роботи програмного забезпечення

Для графічного представлення роботи системи використовуються UML-діаграми – це стандартизовані графічні схеми, які використовуються для візуалізації та проєктування програмних систем, процесів і структур даних. Вони дозволяють у повному обсязі візуалізувати функціональність, структуру, логіку та взаємодію компонентів програмного забезпечення що розроблюється.

Діаграми послідовності – це поведінкові діаграми, які демонструють як різні об’єкти і компоненти вебзастосунку взаємодіють між собою для виконання певних завдань. На цих діаграмах зображені актори, основні об’єкти та компоненти системи. Між ними визначаються повідомлення, що демонструють виклики методів або передачу даних (рис. 3.2 – 3.4).

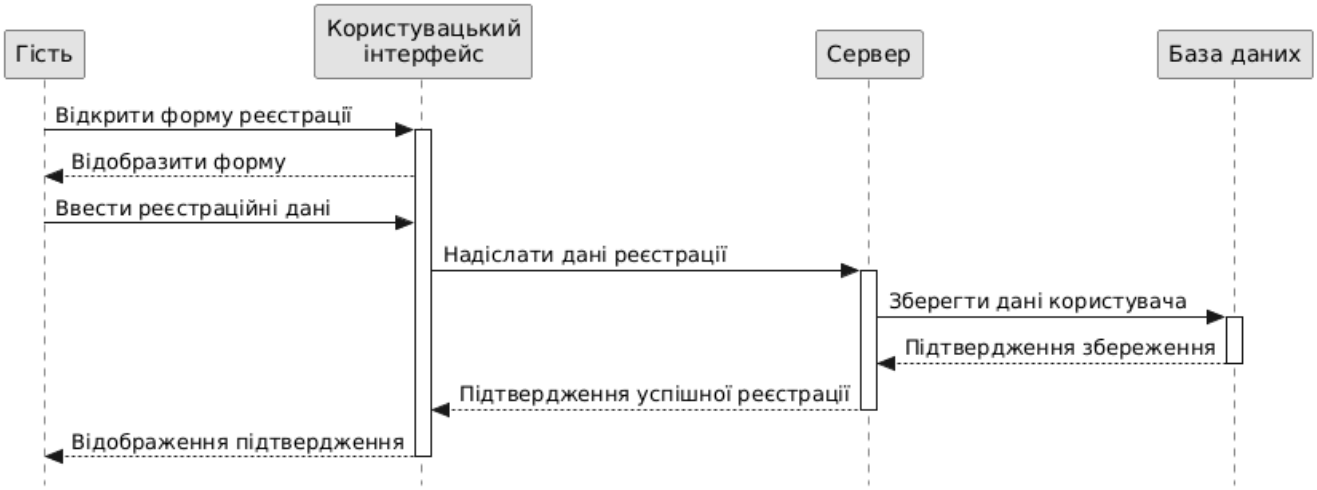

Рисунок 3.2 – Реєстрація користувача

На діаграмі зображено послідовність дій під час реєстрації користувача. Алгоритм роботи охоплює наступні етапи:

– неавторизований відвідувач (гість) ініціює процес, відкриваючи форму реєстрації через інтерфейс вебзастосунку;

– клієнтська частина системи динамічно відображає відповідну форму для введення облікових даних;

– після заповнення полів гість підтверджує дію, і клієнтський застосунок формує та відправляє мережевий запит із даними на серверну частину;

– сервер приймає запит та виконує звернення до бази даних для безпосереднього збереження інформації про нового користувача;

– після успішного виконання операції база даних повертає підтвердження серверу, який своєю чергою сигналізує клієнтській частині про успішне створення облікового запису;

– інтерфейс користувача оновлюється та виводить відповідне візуальне повідомлення про успішну реєстрацію.

Ця діаграма наочно демонструє наскрізну взаємодію між користувачем, клієнтським середовищем, сервером та сховищем даних, моделюючи один із найважливіших базових сценаріїв доступу до функціоналу вебзастосунку.

Рисунок 3.3 – Додавання товару до кошика

Дана діаграма зображає процес додавання товару до кошика користувача. Алгоритм роботи охоплює наступні етапи:

– покупець ініціює процес, обираючи картину та натискаючи відповідну кнопку додавання в інтерфейсі вебзастосунку;

– клієнтська частина формує запит і передає дані про обраний лот (ідентифікатор картини та кількість) на серверну частину системи;

– сервер виконує проміжне звернення до бази даних для перевірки актуального доступного залишку обраної картини;

– після підтвердження наявності товару, сервер ініціює запит до бази даних для безпосереднього збереження запису про товар у персональному кошику користувача;

– база даних повертає підтвердження успішного запису серверу, який своєю чергою передає успішну відповідь на клієнтську частину;

– інтерфейс користувача динамічно оновлюється, візуально сповіщаючи покупця про успішне додавання товару та оновлюючи лічильник кошика.

Ця діаграма демонструє транзакційну взаємодію покупця з системою, деталізуючи внутрішню логіку обробки даних під час виконання ключової операції процесу здійснення покупки.

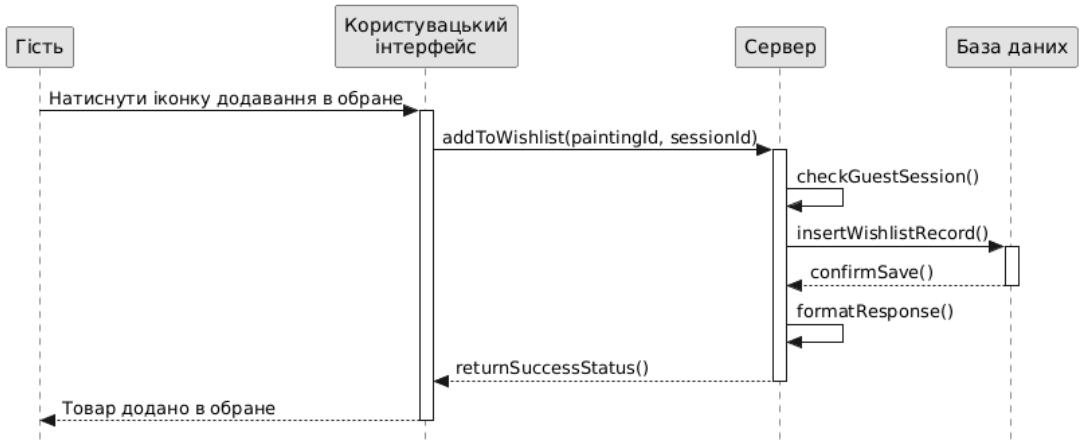

Рисунок 3.4 – Додавання товару до списку бажаного

Діаграма демонструє процес додавання обраного користувачем товару до свого списку бажаного. Алгоритм роботи охоплює наступні етапи:

– відвідувач сайту (гість або авторизований покупець) ініціює дію, натискаючи на відповідну графічну іконку збереження біля обраної картини в каталозі;

– клієнтська частина застосунку перехоплює цю подію та відправляє запит на серверну частину, передаючи ідентифікатор картини та дані поточного сеансу;

– сервер API приймає запит, перевіряє активну сесію відвідувача та формує відповідне звернення до бази даних;

– у базі даних виконується операція створення нового запису, який пов'язує обраний художній твір із поточним користувачем, після чого серверу повертається підтвердження успішної транзакції;

– сервер форматує відповідь та надсилає сигнал про успішне збереження назад до клієнтського застосунку;

– користувацький інтерфейс миттєво оновлюється, візуально змінюючи стан іконки, що слугує користувачеві підтвердженням успішного додавання лота.

Ця діаграма відображає механізм взаємодії клієнта з системою збереження персональних вподобань, підкреслюючи гнучкість системи, яка дозволяє працювати з цим функціоналом навіть без обов'язкової попередньої авторизації.

Діаграма компонентів - це одна з головних структурних діаграм уніфікованої мови моделювання. Вона візуалізує високорівневу архітектуру системи, відображаючи великі програмні модулі (компоненти) та зв’язки між ними. Дає змогу вичерпно описати, з яких підсистем складається вебзастосунок, за що відповідає кожен логічний блок та яким чином вони обмінюються інформацією.

Діаграма компонентів використовується для:

– проєктування надійної модульної архітектури програмного забезпечення на початкових етапах розробки;

– полегшення взаємодії і комунікації між розробниками, тестувальниками та аналітиками щодо загальної структури проєкту;

– документування архітектури, що допомагає програмістам швидко зрозуміти розподіл відповідальності між клієнтською, серверною частинами та базою даних під час супроводу системи.

Кожен елемент на такій діаграмі виступає окремим архітектурним вузлом і структурно описує наступні сутності:

– компоненти - фізичні або логічні частини системи, що виконують певну функцію;

– інтерфейси - точки доступу, через які компоненти надають свої послуги іншим частинам системи;

– залежності - зв'язки, які показують напрямок потоку даних та протоколи взаємодії.

Зв’язки між компонентами наочно демонструють маршрутизацію даних та логіку взаємодії між різними шарами застосунку. Разом усе це дозволяє чітко

визначити архітектуру системи, робить програмне забезпечення гнучким до масштабування та дозволяє грамотно розділити відповідальність між різними частинами системи (рис. 3.5).

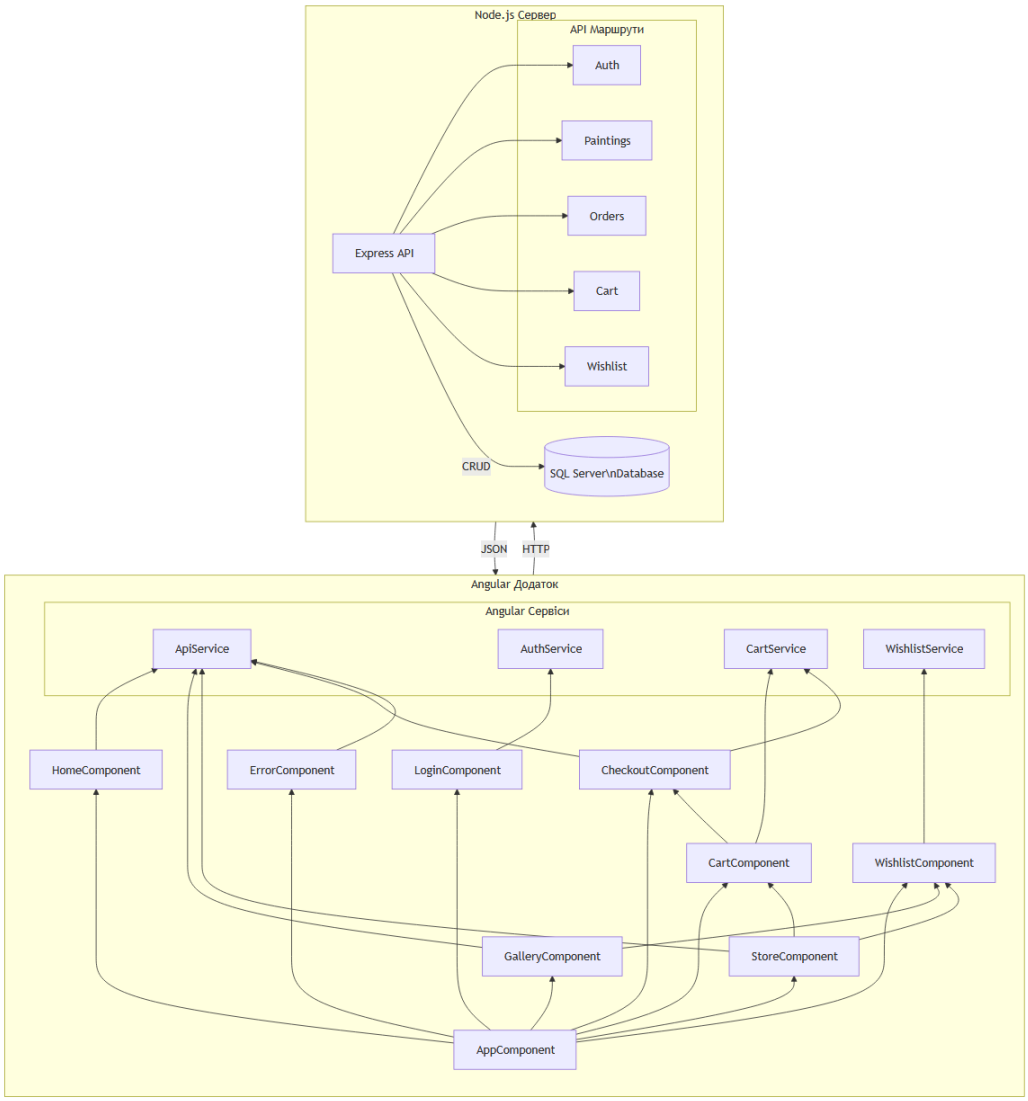

Рисунок 3.5 – Діаграма компонентів

На діаграмі зображено основні компоненти системи, які забезпечують правильне функціонування вебзастосунку продажу картин. Кожен з наведених архітектурних блоків відіграє важливу роль у системі та відповідає певному елементу технічної реалізації програмного забезпечення - від клієнтських сервісів Angular до серверного Express API та бази даних..

Діаграма пакетів - це структурна діаграма уніфікованої мови моделювання (UML), яка демонструє організацію елементів системи у вигляді логічних груп або

ж пакетів, а також визначає ієрархію та залежності між ними. Її традиційно використовують для декомпозиції великих і складних проєктів на менші, керовані підсистеми. Такий підхід значно полегшує процес розробки та подальшого супроводу програмного забезпечення. Пакет виступає як абстрактний контейнер для логічно пов'язаних елементів системи, таких як класи, компоненти, сервіси, маршрути чи інтерфейси.

Розробка таких діаграм переслідує одразу кілька важливих цілей. Насамперед це структурування архітектури системи. Розділення складної кодової бази на логічні модулі дозволяє впорядкувати структуру застосунку та чітко розмежувати зони відповідальності кожного елемента. Крім того, діаграма забезпечує наочну візуалізацію напрямків взаємодії між модулями. Завдяки цьому розробник швидко розуміє, які саме підсистеми вимагають наявності інших для своєї коректної роботи. Це безпосередньо сприяє спрощенню супроводу та масштабування проєкту.

Графічно на діаграмі пакетів відображають спеціальні елементи у формі папок або прямокутників із назвами пакетів, між якими проводять лінії залежностей. Зазвичай ці зв'язки позначають пунктирними стрілками. Вони наочно демонструють напрямок впливу: якщо один пакет використовує класи чи інтерфейси іншого, він стає залежним від нього. Отже, будь-які структурні зміни в базовому пакеті безпосередньо впливають на роботу всіх залежних від нього модулів.

У контексті проєктування сучасних вебзастосунків діаграма пакетів відіграє особливо важливу роль. Вона допомагає візуально розділити проєкт на рівні клієнтського інтерфейсу, серверної бізнес-логіки та механізмів доступу до бази даних (рис. 3.6).

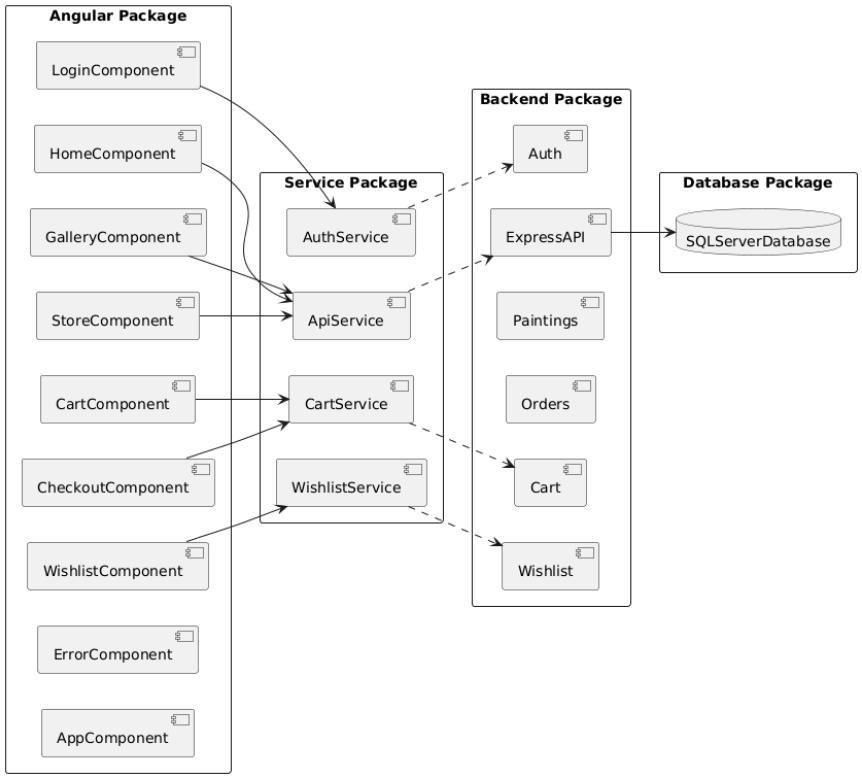

Рисунок 3.6 – Діаграма пакетів

Розроблена діаграма пакетів відображає логічну структуру клієнт-серверної частини вебзастосунку. Архітектура системи розділена на чотири основні пакети, які чітко розмежовують сфери відповідальності між інтерфейсом користувача, логікою обміну даними, серверним API та сховищем інформації: Angular Package, Service Package, Backend Package та Database Package.

Angular Package відповідає за рівень представлення і забезпечує безпосередню візуальну взаємодію користувача із системою. До його складу входять UI-компоненти застосунку: LoginComponent (для авторизації), HomeComponent (головна сторінка), GalleryComponent та StoreComponent (відображення каталогу), CartComponent та CheckoutComponent (робота з кошиком та оформленням замовлення), WishlistComponent (список бажаного), а також ErrorComponent та кореневий AppComponent.

Service Package виступає проміжним шаром бізнес-логіки на стороні клієнта. Він містить набір інжектованих сервісів (AuthService, ApiService, CartService, WishlistService), які відповідають за управління станом застосунку та формування запитів до сервера. Компоненти з пакета Angular Package мають пряму залежність

від цих сервісів. LoginComponent делегує логіку автентифікації до AuthService, а CartComponent взаємодіє з CartService.

Backend Package реалізує серверну частину застосунку. Він містить модулі та контролери для обробки запитів: Auth, ExpressAPI, Paintings, Orders, Cart та Wishlist. Пакет Service Package має безпосередню зовнішню залежність від Backend Package, оскільки клієнтські сервіси надсилають HTTP-запити до відповідних серверних ендпоінтів для отримання або збереження даних.

Database Package відповідає за рівень фізичного доступу до даних. Він містить компонент SQLServerDatabase, який інкапсулює логіку взаємодії з реляційною базою даних. Серверний пакет Backend Package залежить від Database Package, оскільки всі операції зі збереження транзакцій, пошуку картин та оновлення інформації про користувачів виконуються саме на рівні бази даних.

Висновки до розділу 3

У третьому розділі виконано проєктування вебзастосунку продажу картин з урахуванням усіх можливих сучасних вимог. Детально розглянуто архітектуру системи, сформовано моделі, що охоплюють бізнес-процеси та можливості технічної реалізіції.

Побудова UML-діаграм варіантів використання, послідовності, класів та пакетів дала можливість в повному обсязі описати структуру та принцип роботи програмного забезпечення. Використання таких моделей дозволяє візуалізувати поведінку системи, визначити взаємодії між компонентами та сформувати чітке уявлення про функціонування вебзастосунку.

У процесі проєктування обгрунтовано використання клієнт-серверної архітектури та сучасного стеку технологій, до якого входять HTML5, CSS3, Angular, JavaScript, TypeScript, Node.js та Microsoft SQL Server. Обрані технології забезпечують високу швидкість відгуку інтерфейсу, масштабованість системи та оптимізований обмін даними. Також розроблено структурну модель програмного забезпечення, яка дозволяє організувати взаємодію між основними компонентами системи, забезпечити обробку даних користувачів та роботу з каталогом картин.

4 РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

У даному розділі розглядається етап практичної реалізації розробленого вебзастосунку для електронної дистрибуції художніх творів. На основі результатів попереднього концептуального проєктування, побудованих інформаційних моделей та архітектурних схем здійснюється безпосереднє програмне втілення системи за допомогою передових інструментів веброзробки.

Основна увага приділяється процесу програмування клієнтської та серверної частин вебзастосунку, налаштуванню безпечної взаємодії між компонентами системи через прикладний програмний інтерфейс, а також інтеграції з реляційною базою даних. Під час розробки використано узгоджений стек технологій, що забезпечує створення інтуїтивно зрозумілого адаптивного інтерфейсу, гарантує високу швидкість обробки клієнтських запитів та надійне збереження транзакційної інформації.

Окремо розглядаються питання забезпечення модульності кодової бази, що гарантує зручність подальшого супроводу та масштабованість програмного забезпечення. Детально описуються технічні особливості реалізації ключових функціональних можливостей вебзастосунку, зокрема алгоритми роботи з каталогом картин, механізми управління персональним кошиком та списком бажаного, процеси оформлення замовлень, а також безпечна авторизація користувачів.

Завершальна частина розділу присвячена процесу верифікації та тестування розробленого програмного забезпечення. Наводяться застосовані підходи до перевірки функціоналу, описуються тестові сценарії для основних бізнес-процесів та аналізуються результати тестування, які підтверджують коректність роботи системи відповідно до висунутих технічних вимог.

4.1 Структура вебзастосунку

Структура розробленого вебзастосунку з продажу картин побудована за фундаментальним принципом суворого розділення клієнтської та серверної частин.

Такий архітектурний підхід забезпечує повну логічну та фізичну незалежність фронтенд і бекенд компонентів системи. Це суттєво спрощує супровід програмного коду, полегшує процес розширення функціоналу, а також дозволяє масштабувати окремі вузли вебзастосунку незалежно один від одного в умовах зростання навантаження.

Проєкт концептуально та фізично складається з двох основних ізольованих середовищ (рис. 4.1). Першим середовищем є клієнтська частина вебзастосунку, реалізована за допомогою фреймворку Angular. Другим середовищем виступає серверна частина застосунку, побудована на базі платформи Node.js із використанням інструментарію Express для ефективної маршрутизації.

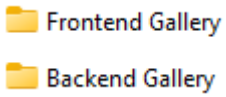

Рисунок 4.1 – Загальна структура проєкту

Frontend-частина розробленої системи відповідає за безпосередню реалізацію користувацького інтерфейсу, забезпечуючи високий рівень взаємодії між користувачем та програмним комплексом. До її основних завдань належать перехоплення та обробка подій взаємодії клієнта із системою, організація зручної та інтуїтивно зрозумілої навігації між сторінками застосунку, а також динамічне відображення і рендеринг візуального контенту в режимі реального часу.

Серверна частина, у свою чергу, виступає логічним ядром системи та забезпечує надійне виконання встановленої бізнес-логіки, сувору валідацію вхідних даних для захисту від вразливостей, оперативну обробку мережевих запитів, а також безпечне здійснення транзакцій у базі даних. Взаємодія між frontend та backend шарами застосунку здійснюється виключно через стандартизовані HTTP-запити архітектури REST API. Такий підхід забезпечує повну незалежність розробки та розгортання обох частин системи, а також дозволяє передавати структуровані пакети даних у полегшеному форматі JSON, що мінімізує навантаження на мережеві канали зв'язку та пришвидшує час відгуку.

Клієнтська частина системи реалізована у вигляді сучасного SPA-застосунку (Single Page Application) з використанням фреймворку Angular. Застосування такого архітектурного підходу дозволяє оновлювати окремі компоненти та частини сторінки асинхронно, без її повного перезавантаження з сервера. Це не тільки суттєво покращує швидкість роботи системи та оптимізує використання обчислювальних ресурсів, а й забезпечує максимальну зручність, плавність та високу ергономічність інтерфейсу для кінцевого користувача.

Загальна структура Angular-проєкту чітко розмежована і складається з набору конфігураційних файлів та головної робочої директорії src. (рис. 4.2).

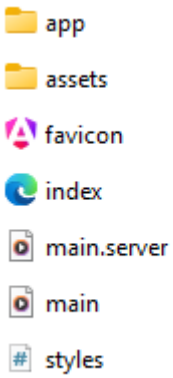

Рисунок 4.2 – Директорія src

Конфігураційні файли, які знаходяться у кореневому каталозі проєкту, відповідають за глобальні налаштування середовища розробки, автоматизоване керування залежностями і бібліотеками, а також визначають параметри фінальної збірки та оптимізації системи перед деплоєм. Директорія src виступає основною папкою клієнтської частини, де зосереджено весь вихідний код застосунку. У структурі цієї директорії критично важливе значення мають дві головні папки, які визначають архітектурну цілісність інтерфейсу. Папка assets використовується як сховище для всіх статичних ресурсів проєкту, включаючи графічні матеріали, зображення картин, шрифти та файли стилів оформлення. Папка app містить у собі модулі, сервіси та окремі компоненти, які описують безпосередню логіку поведінки, структуру та шаблони клієнтської частини вебзастосунку. (рис. 4.3).

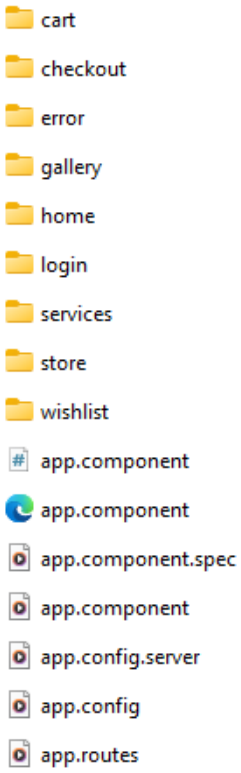

Рисунок 4.3 – Структура директорії app

Головним компонентом виступає app.component, який є точкою входу для користувацького інтерфейсу. Він слугує контейнером для всіх інших компонентів і завантажується найпершим під час запуску вебзастосунку.

Файл app.config є центральним файлом для налаштування глобальних залежностей (провайдерів) та сервісів. Він відповідає за налаштування HTTP-клієнта, маршрутизації та інших сервісів.

Файл app.routes є основним конфігураційним файлом в Angular, який містить масив маршрутів. Він визначає, який саме компонент має завантажуватися в браузері при переході за певною URL-адресою, і таким чином реалізує односторінкову навігацію.

Також у структурі директорії app присутні такі папки:

– cart - зберігає файли моделей даних, які використовуються для роботи кошика;

– checkout - містить дані про обробку оформлення замовлення;

– error - у цій папці знаходяться дані про сторінку з помилкою;

– gallery - містить логіку та перевірку даних реєстрації;

– home - описує функціонування списку бажаного;

– login - зберігає файли логіки роботи сторінки входу;

– services - містить сервіси, які забезпечують взаємодію між компонентами та серверною частиною застосунку;

– store - в даній папці можна знайти файли, які відповідають за роботу з комерційною частиною вебзастосунку;

– wishlist - має дані про компонентну частину списку бажаного та його логіку роботи.

Такий компонентний підхід Angular дозволяє розділити інтерфейс на незалежні функціональні частини, що спрощує повторне використання коду, тестування та подальший супровід програмного забезпечення.

Серверна частина вебзастосунку реалізована за допомогою середовища Node.js. Вона забезпечує обробку мережевих HTTP-запитів, виконання серверних процесів, реалізацію бізнес-логіки та надійну взаємодію з базою даних (рис. 4.4).

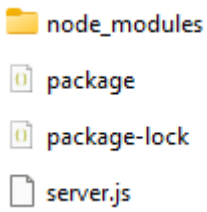

Рисунок 4.4 – Структура backend частини

У цій директорії розміщені конфігураційні файли, зокрема package та package-lock, які містять інформацію про підключені бібліотеки та залежності, необхідні для роботи серверної частини вебзастосунку. Також тут задаються налаштування підключення до бази даних, параметри середовища розробки та інші важливі службові налаштування серверної частини, тоді як у папці node_modules зберігаються безпосередньо самі завантажені пакети.

Файл server.js є основним файлом запуску Node.js-застосунку. Він відповідає за глобальну конфігурацію серверної частини, налаштування сервісів, ініціалізацію маршрутизації, підключення контролерів, застосування політик безпеки та безпосередній запуск вебсервера.

4.2 Структура бази даних

База даних виступає фундаментальною інформаційною основою розробленого вебзастосунку, являючи собою чітко структуроване та логічно організоване сховище. Її головне архітектурне завдання полягає у надійному зберіганні, глибокій систематизації та забезпеченні швидкого безперебійного доступу до всієї динамічної інформації платформи. Цей масив даних охоплює детальні відомості про користувачів, каталог художніх творів, історію формування замовлень, персональні списки бажаного та інші ключові сутності предметної області, необхідні для коректної роботи системи.

Як основне технологічне середовище для керування інформацією обрано Microsoft SQL Server. Рішення інтегрувати саме Microsoft SQL Server у проєкт зумовлене специфічними перевагами цієї платформи. Вона гарантує найвищий корпоративний рівень захисту конфіденційної інформації, демонструє виняткову продуктивність під час одночасної обробки великої кількості транзакцій, а також містить потужні вбудовані механізми високої доступності і надійного резервного копіювання даних (рис. 4.5).

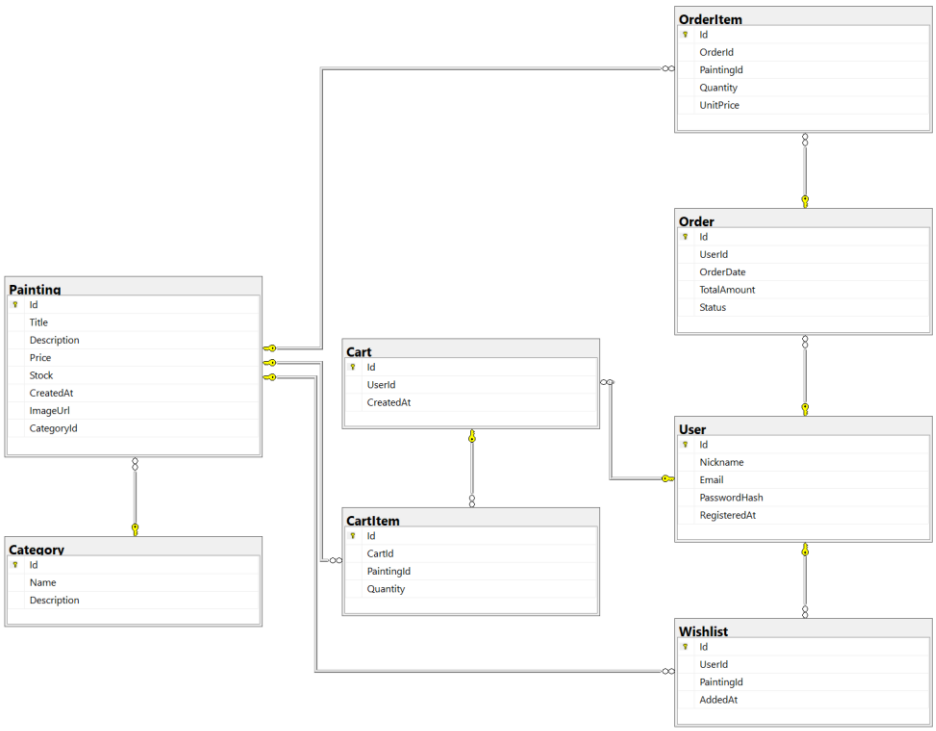

Рисунок 4.5 – Схема БД

Розроблена логічна модель бази даних відображає архітектуру зберігання інформації та зв'язки між основними сутностями предметної області вебзастосунку. На наведеній схемі представлено ключові таблиці, які спільно забезпечують повноцінне функціонування. Зокрема, структура охоплює блоки для управління каталогом картин і категоріями, обліковими записами користувачів, процесами формування кошика, оформлення замовлень та збереженням персональних списків бажаного.

Усі елементи бази даних інтегровані між собою за допомогою класичних реляційних зв'язків, переважно типу «один до багатьох», що реалізуються через сувору систему первинних та зовнішніх ключів. Така нормалізована структура дозволяє уникнути надмірності та дублювання даних, гарантує їхню посилальну цілісність під час виконання транзакцій, а також забезпечує високу швидкість обробки пошукових запитів.

У розробленому вебзастосунку продажу картин база даних містить вісім таблиць, кожна з яких відповідає за зберігання та обробку даних певних сутностей у системі (рис. 4.6). Усі таблиці бази даних використовують кодування UTF-8, яке є стандартом для операційних систем і підтримує коректне відображення символів будь-якої мови світу.

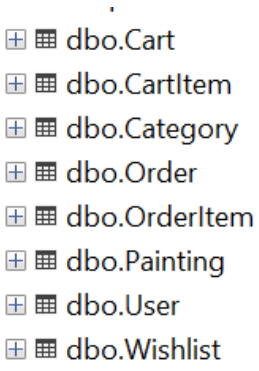

Рисунок 4.6 – Таблиці бази даних

Опис таблиць:

– User – зберігає реєстраційні дані користувачів;
– Category – перелік тематичних категорій картин;
– Painting – описує кожну картину та її наявний запас (Stock);

– Cart – представляє кошик конкретного користувача;
– CartItem – зв’язує картину із кошиком та фіксує кількість товару;
– Wishlist – зберігає список обраного для кожного користувача;
– Order – містить інформацію про сформовану покупку;
– OrderItem – зберігає деталі кожного замовлення: які картини, за якою ціною, та яка їх кількість в наявності.

Розроблена архітектура реляційної бази даних виступає надійним фундаментом для реалізації ключових функціональних можливостей платформи. Вона забезпечує структуроване зберігання каталогу картин, ведення персональних кошиків та списків бажаного, а також реєстрацію покупок клієнтів. Практичне впровадження такої структури дозволяє гарантувати високу безпеку та цілісність збереження інформації, відмінну швидкодію системи, а також створює необхідні умови для подальшого масштабування програмного комплексу.

4.3 Опис користувацького інтерфейсу

Для забезпечення максимально зручного та інтуїтивно зрозумілого використання вебзастосунку з продажу картин підготовлено детальний опис роботи з клієнтським інтерфейсом. У даному підрозділі розкрито основні можливості та функціонал системи, логіку взаємодії користувача з графічними елементами, а також ключові особливості навігації по каталогу.

При ініціалізації вебзастосунку в браузері відвідувач одразу потрапляє на головну сторінку «Home», яка візуально презентує платформу та слугує стартовою точкою для подальшої роботи з художніми творами (рис. 4.6).

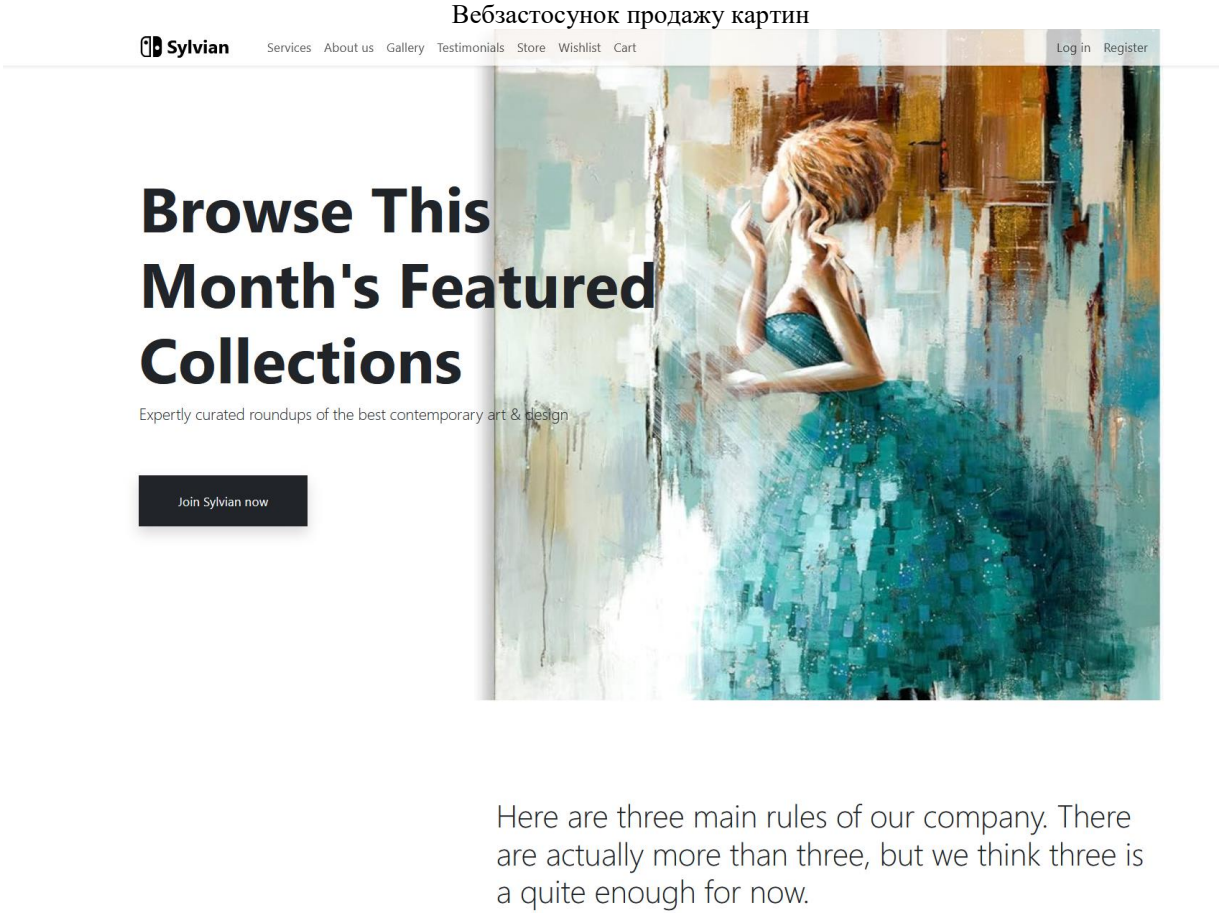

Рисунок 4.7 – Сторінка «Home»

У верхній частині сторінки розташована навігаційна панель, яка є одним із найважливіших елементів інтерфейсу та безпосередньо впливає на користувацький досвід і загальну ефективність сайту. Цей елемент забезпечує зручне, швидке та інтуїтивно зрозуміле переміщення відвідувачів між різними сторінками і функціональними розділами вебзастосунку.

У правій частині навігаційної панелі розміщені елементи доступу до облікового запису. Інтерфейс передбачає дві окремі кнопки: «Log in» та «Register». Кнопка «Log in» призначена для авторизації вже існуючих клієнтів системи шляхом введення своїх даних. В свою чергу кнопка «Register» використовується для реєстрації нових користувачів, які вперше відвідали платформу та бажають створити персональний профіль (рис. 4.7).

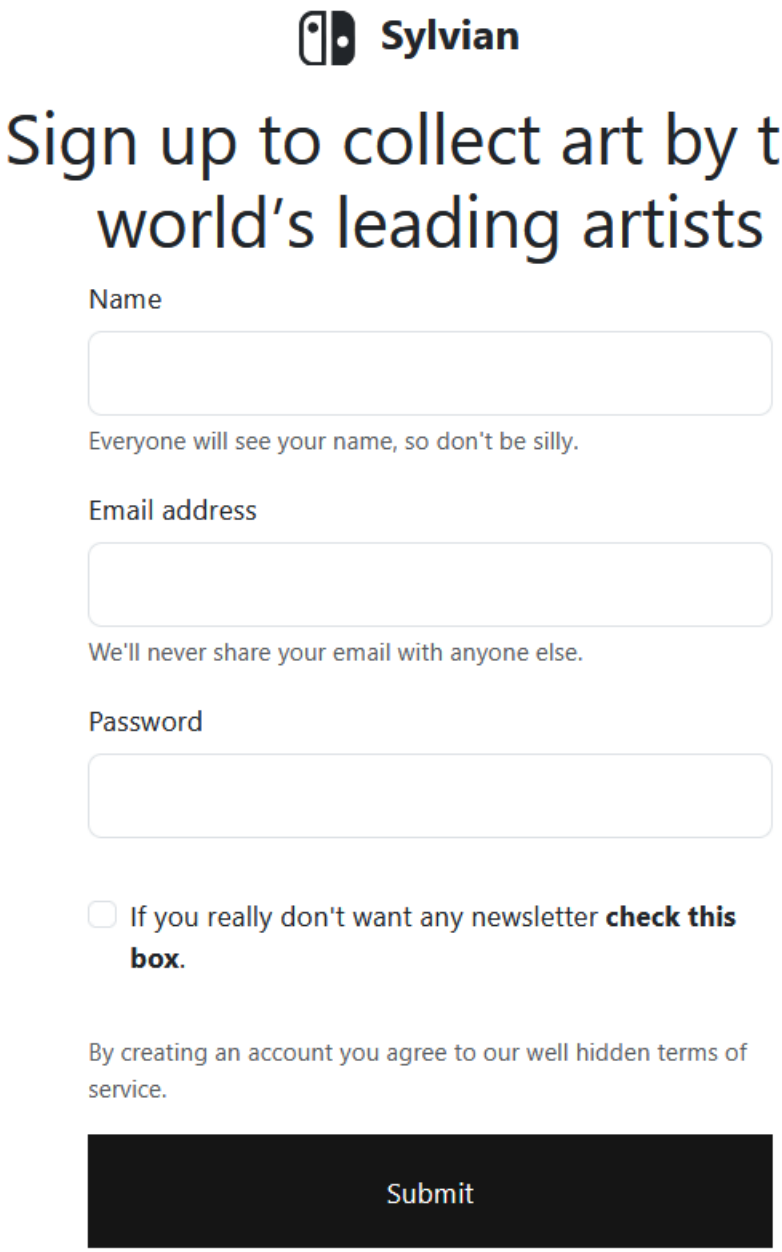

Рисунок 4.8 – Форма реєстрації

Для створення нового облікового запису користувачу необхідно заповнити реєстраційну форму, послідовно вказавши своє ім'я, актуальну адресу електронної пошти та надійний пароль. Додатково інтерфейс форми містить опціональний чекбокс для керування підпискою на інформаційну розсилку. Після заповнення полів та натискання кнопки «Submit», система ініціює відправку введених даних на

сервер. Там відбувається валідація інформації та перевірка унікальності електронної пошти, після чого створюється новий запис у базі даних, і користувач успішно реєструється на платформі (рис. 4.9).

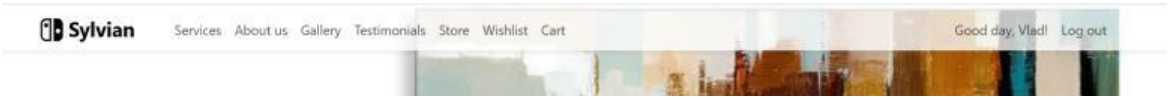

Рисунок 4.9 – Створений профілю користувача

Головна сторінка «Home»

Знаходячись на головній сторінці, користувач має змогу візуально ознайомитися з ключовою концепцією платформи та переглянути представлені на сайті картини. Цей розділ слугує своєрідною естетичною візитівкою вебзастосунку. В інформаційному блоці представлено короткий опис діяльності дистриб'ютора, який безпосередньо займається пошуком, колекціонуванням та продажем унікальних художніх робіт (рис. 4.10 – 4.11).

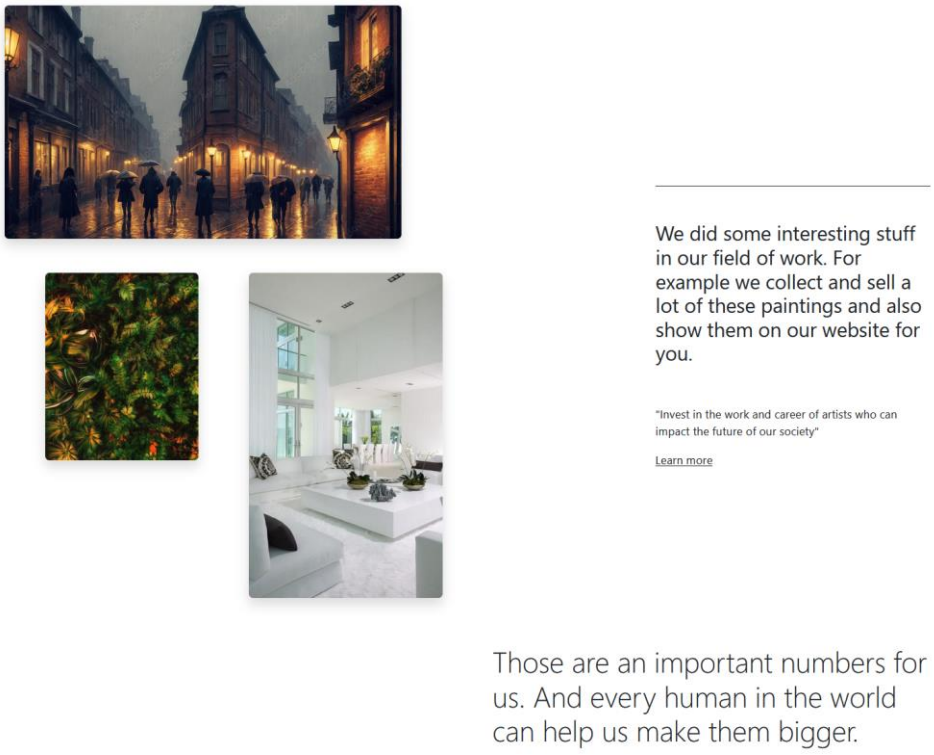

Рисунок 4.10 – Інформаційна панель

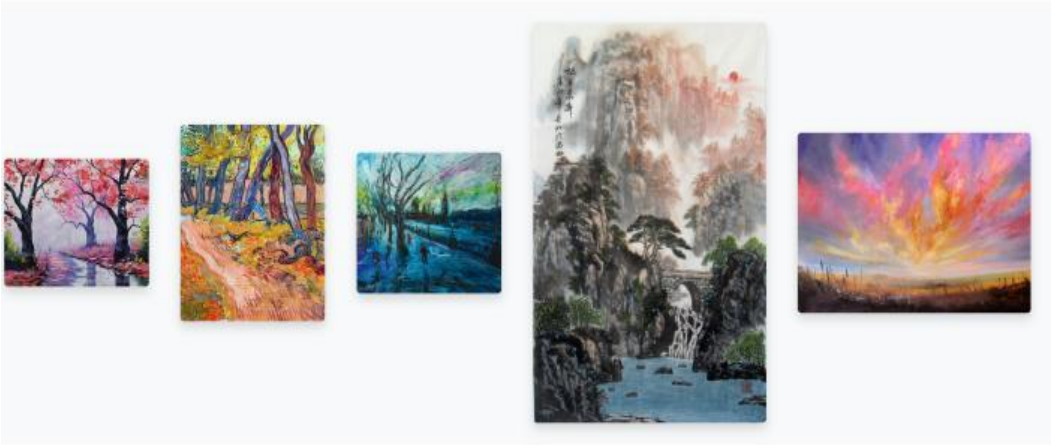

Рисунок 4.11 – Карусель картин

У нижній частині сторінки розміщено ключові інформаційні блоки, які забезпечують відвідувача додатковими відомостями та зручними засобами навігації. Зокрема, у цій зоні представлено детальну інформацію про доступні міжнародні сервіси доставки, ексклюзивність запропонованих картин, а також роз'яснення щодо роботи служби онлайн-підтримки. Якщо під час використання вебзастосунку у клієнта виникнуть запитання чи проблеми, він може скористатися контактною інформацією. Крім того, для оптимізації користувацького досвіду в цій області згруповано швидкі посилання на основні категорії мистецтва, інформаційні розділи про платформу та службові сторінки системи (рис. 4.12).

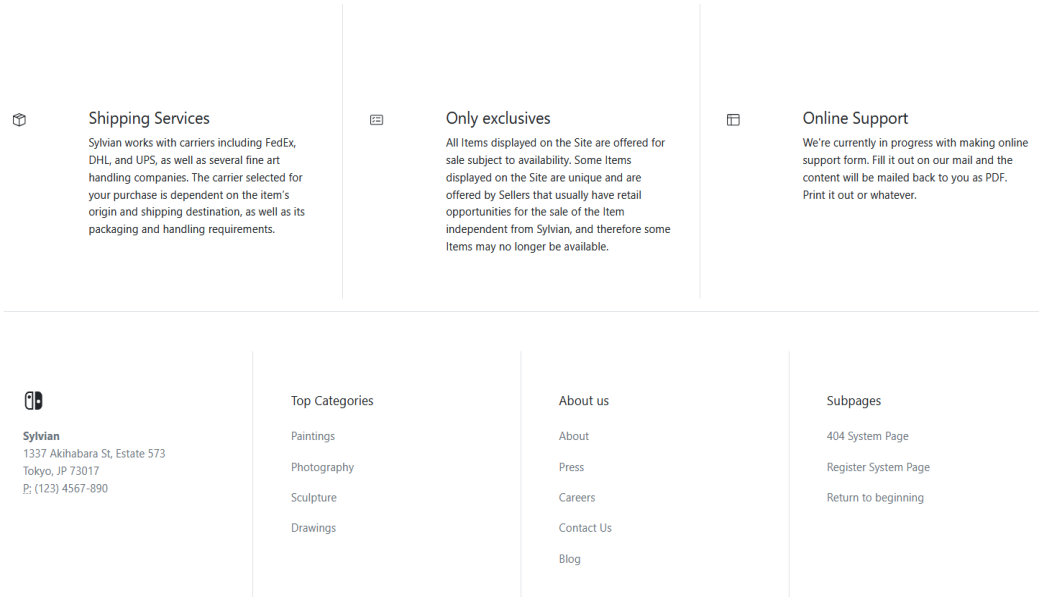

Рисунок 4.12 – Контактна інформація

Сторінка «Error»

У випадку, якщо користувач намагається перейти за неіснуючою URL-адресою, або запитувана сторінка була переміщена чи видалена, система автоматично маршрутизує його на спеціальну сторінку обробки помилок (помилка 404). Інтерфейс цієї сторінки містить відповідне текстове повідомлення, яке інформує клієнта про відсутність шуканого ресурсу. Для збереження безперервності користувацького досвіду та забезпечення зручної навігації, на сторінці помилки передбачено інтерактивне посилання, яке дозволяє відвідувачу швидко повернутися на головну сторінку платформи (рис. 4.13).

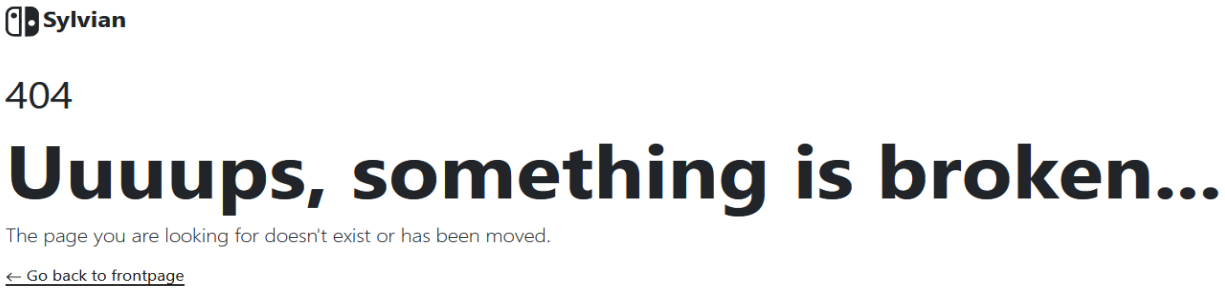

Рисунок 4.13 – Сторінка «Error»

Сторінка «Wishlist»

Сторінка «Wishlist» призначена для збереження художніх творів, які найбільше зацікавили користувача, з метою їх подальшого детального перегляду. Інтерфейс сторінки організовано у вигляді зручного та візуально привабливого каталогу. На кожній картці відображається мініатюра відповідної картини, її унікальна назва та актуальна вартість. Для забезпечення зручного керування збереженими позиціями під кожним твором мистецтва передбачена кнопка видалення. Вона дозволяє клієнту миттєво прибрати неактуальну картину зі свого персонального списку, тим самим підтримуючи його в актуальному стані (рис. 4.14).

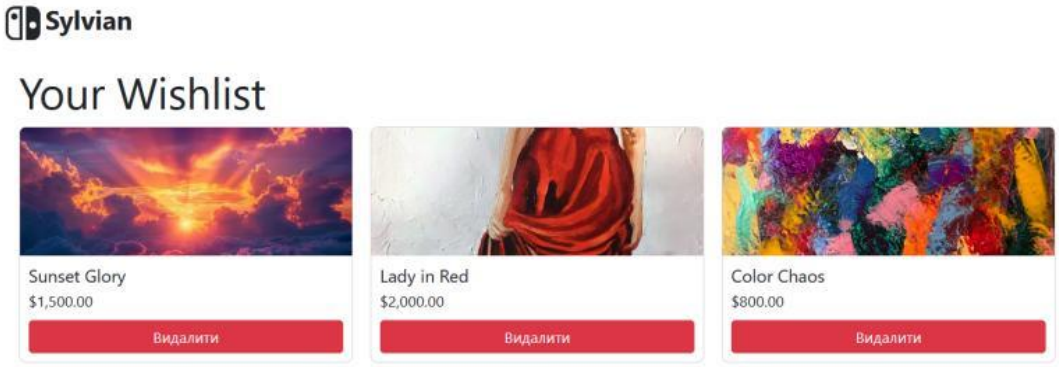

Рисунок 4.14 – Сторінка «Wishlist»

Сторінка «Store»

Розділ «Store» слугує основним каталогом вебзастосунку, де представлений весь асортимент картин. Інтерфейс сторінки реалізовано у вигляді зручної сітки карток товарів, що дозволяє користувачу швидко переглядати пропозиції.

Для кожної картини система виводить її назву, вартість та актуальний залишок на складі. Якщо товар є в наявності, користувач може вказати потрібну кількість і додати його до кошика. У випадку, коли всі екземпляри розпродано, система автоматично блокує функцію покупки для цієї позиції.

Також прямо з каталогу реалізовано швидку взаємодію зі списком бажаного: будь-яку картину можна зберегти для подальшого перегляду або видалити зі списку натисканням на відповідну іконку поруч із кнопкою замовлення (рис. 4.15).

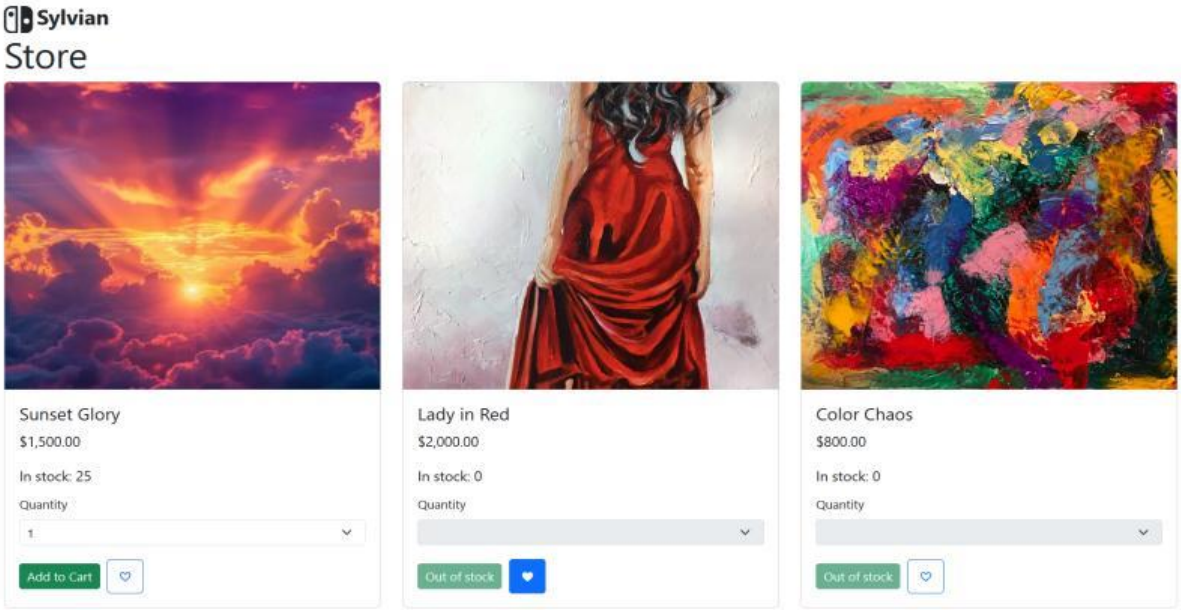

Рисунок 4.15 – Сторінка «Store»

Сторінка кошика є проміжним етапом між вибором картин та безпосереднім оформленням покупки. Для кожної позиції виводиться мініатюра, назва, базова ціна та проміжна вартість, яка розраховується з огляду на вказану кількість.

Користувач має змогу керувати своїм замовленням прямо на цій сторінці. Інтерфейс дозволяє легко змінювати кількість примірників певної картини або ж повністю видалити позицію з кошика. Застосунок динамічно реагує на дії користувача та миттєво перераховує всі показники.

У нижній частині екрана система автоматично підбиває підсумки, відображаючи загальну кількість обраних товарів та фінальну суму до сплати. Переконавшись у правильності сформованого кошика, клієнт може перейти до завершального етапу введення даних та оплати, ініціювавши процес оформлення замовлення (рис. 4.15).

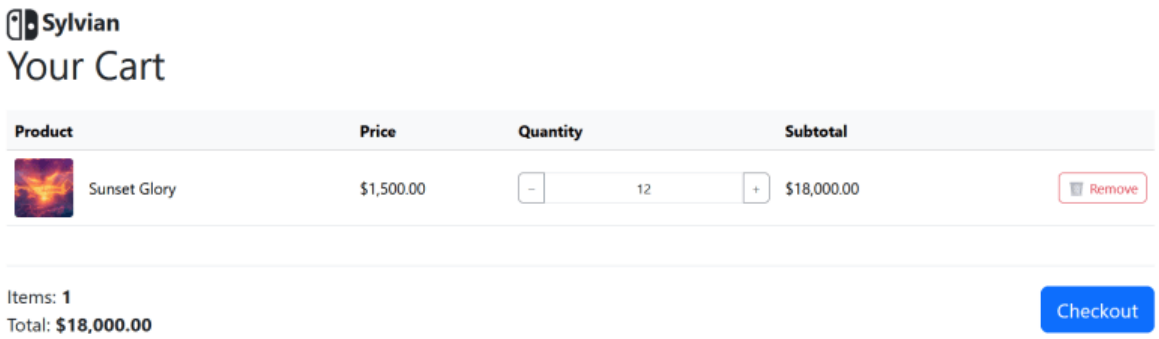

Рисунок 4.15 – Вигляд кошику користувача

Якщо натиснути на кнопку «Checkout», користувач перейде на сторінку оформлення замовлення (рис. 4.16).

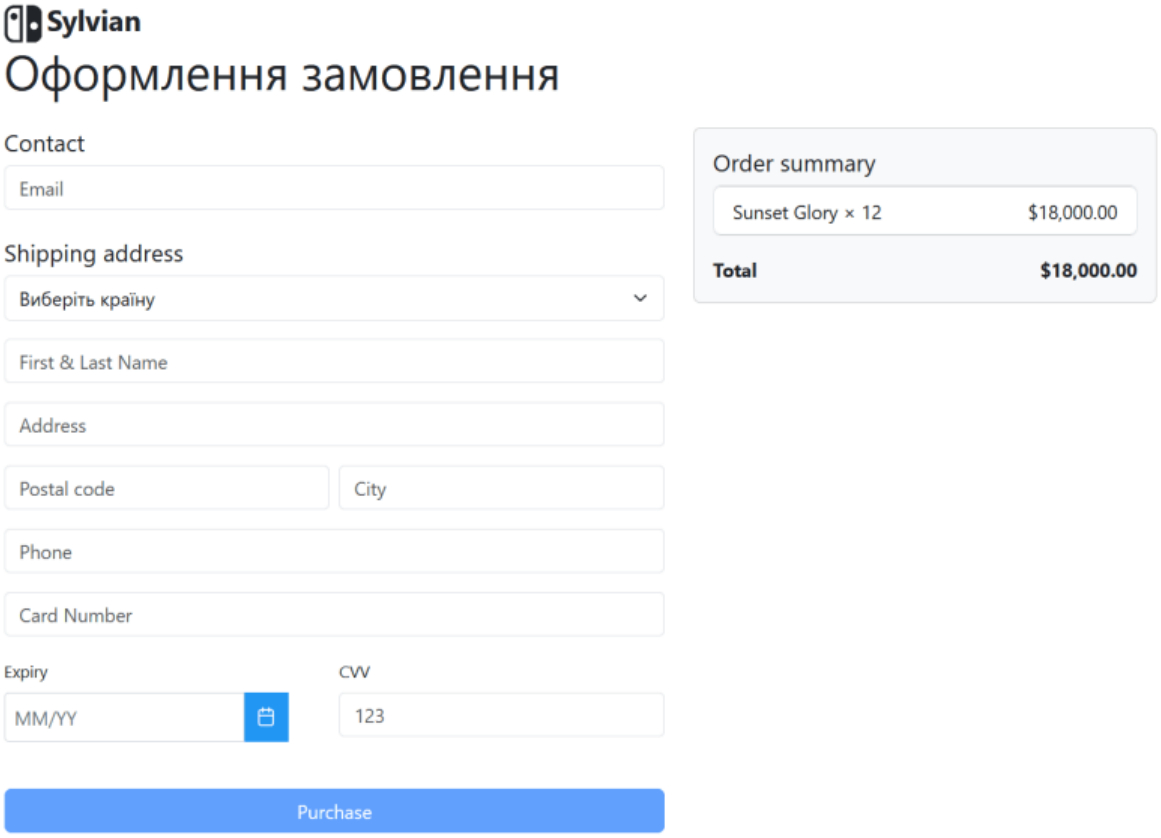

Рисунок 4.16 – Форма оформлення замовлення

Для остаточного оформлення покупки користувач переходить на сторінку чекауту. Тут необхідно заповнити форму з контактними даними та деталями доставки: вказати електронну пошту, ім'я, обрати країну, прописати повну адресу з поштовим індексом та залишити контактний телефон. Оскільки застосунок підтримує онлайн-оплату, на цій же сторінці передбачено блок для безпечного введення реквізитів банківської картки.

У правій частині екрана паралельно виводиться короткий підсумок замовлення. Це дозволяє клієнту ще раз перевірити обрані картини, їх кількість та загальну суму до сплати. Після коректного заповнення всіх необхідних полів користувач натискає кнопку «Purchase», після чого система фіксує транзакцію, зберігає замовлення в базі даних і починає його опрацювання.

Таким чином, розроблена клієнтська частина забезпечує максимально зручний і логічний процес взаємодії з платформою. Вебзастосунок дозволяє відвідувачам легко переглядати каталог мистецтва, зберігати вподобані роботи,

вільно керувати своїм кошиком та швидко оформлювати покупки безпосередньо у дистриб'ютора.

4.4 Функціональне тестування програмного забезпечення

Після завершення етапу розробки проведено комплексне тестування системи. Його головною метою стала перевірка коректності роботи всіх залучених модулів, надійності взаємодії між клієнтською та серверною частинами, а також загальної стабільності вебзастосунку під час виконання типових операцій.

Для перевірки логіки роботи платформи застосовувалося ручне функціональне тестування. У процесі успішно пройдено та валідовано ключові сценарії взаємодії клієнта з системою: створення нового облікового запису та подальшу авторизацію, а також коректну роботу з персональним списком бажаного. Окрему увагу приділено комерційній частині: перевірялася логіка зміни статусів товару у разі його відсутності на складі, правильність динамічного підрахунку вартості в кошику та безпомилковість повного циклу оформлення замовлення і збереження його в базі даних.

Таблиця 4.1 – Тестування функціоналу вебзастосунку

№	Назва тестування	Кроки виконання	Очікуваний результат	Примітка
1	Реєстрація користувача	Заповнити реєстраційну форму	Дані зберігаються в базі даних, створюється новий обліковий запис, користувач входить у систему	Виконано успішно
2	Авторизація користувача	Заповнити форму авторизації	Система перевіряє дані, при збігу відкривається доступ до профілю – інакше виникає помилка	Виконано успішно
3	Додавання картини до списку бажаного	Натиснути відповідну іконку на картці товару в каталозі	Картина миттєво з'являється у розділі «Wishlist» для подальшого перегляду, користувач отримує повідомлення на екрані	Виконано успішно

Кінець таблиці 4.1

4	Додавання товару до кошика	Обрати бажану кількість товару та підтвердити	Картина додається до кошика, застосунок автоматично перераховує загальну вартість. Загальна кількість доступного товару змінюється	Виконано успішно
5	Перевірка ліміту товарів	Спроба додати до кошика розпродану картину	Кнопка покупки змінюється на неактивну, можливість придбати дану картину повністю блокується	Виконано успішно
6	Оформлення замовлення	Заповнити форму доставки, реквізити та підтвердити покупку	Система перевіряє інформацію – транзакція фіксується, нове замовлення зберігається в базі даних	Виконано успішно
7	Обробка неіснуючої сторінки	Спроба переходу за некоректною URL-адресою	Відкривається сторінка помилки 404 з інтерактивним посиланням для повернення на головну	Виконано успішно
8	Вихід із системи	Натиснути кнопку «Log out»	Поточна сесія користувача закінчується, виконується вихід з облікового запису	Виконано успішно

За результатами проведених тестів система підтвердила повністю коректну роботу всіх основних функцій вебзастосунку. Під час перевірки платформа продемонструвала безпомилкове виконання сценаріїв реєстрації, управління списком бажаного та кошиком, а також фінального оформлення замовлення. Комплексний підхід до тестування також довів стійкість програмного продукту до виняткових ситуацій. Система коректно обробляє виняткові ситуації, забезпечуючи стабільний користувацький досвід. Отримані показники свідчать про стабільну та швидку взаємодію клієнтської і серверної частин, надійну обробку транзакцій базою даних та доводять готовність вебзастосунку до повноцінної експлуатації в реальних умовах.

Висновки до розділу 4

У четвертому розділі детально розкриваються всі етапи практичної реалізації платформи для продажу картин. Текст описує архітектуру розробленої системи, яка базується на стабільній взаємодії клієнтської частини та продуктивного сервера на базі Node.js. Окрема увага присвячена проєктуванню інформаційної структури під управлінням системи Microsoft SQL Server. Створена нормалізована реляційна модель забезпечує безпечне збереження даних про користувачів, каталог художніх творів та транзакції, гарантуючи при цьому високу швидкодію.

Окрім внутрішніх архітектурних рішень, у розділі представлено розгорнутий опис графічного інтерфейсу, який одночасно слугує наочним керівництвом користувача. Розроблений дизайн фокусується на загальній зручності та візуальній естетиці. Він дозволяє відвідувачам легко переглядати асортимент картин, вільно керувати персональним списком бажаного та динамічно формувати кошик. Логіка інтерфейсу інтуїтивно підводить клієнта до етапу чекауту, роблячи процес купівлі прозорим та швидким.

Завершальним етапом роботи над системою стало комплексне функціональне тестування програмного продукту. Результати практичної перевірки підтвердили стабільність модулів, відсутність критичних помилок під час виконання клієнтських запитів та повну відповідність реалізованого функціоналу початковим технічним вимогам. Розроблений вебзастосунок успішно реалізує бізнес-модель єдиного дистриб'ютора, надає надійний цифровий інструмент для продажу картин онлайн і повністю готовий до фінального розгортання на робочому сервері.

ВИСНОВКИ

В процесі виконання кваліфікаційної бакалаврської роботи успішно досягнуто головної поставленої мети: розроблено сучасний вебзастосунок для забезпечення багатокористувацького доступу до каталогу та послуг із продажу картин. Дослідження детально охопило об’єкт роботи, а саме процес дистанційного представлення та продажу товарів у мережі Інтернет. Водночас предметом практичного вивчення стали методи та програмні засоби для автоматизації продажу картин. Комплексний підхід до виконання завдань дозволив створити повноцінну платформу за моделлю єдиного дистриб'ютора, яка повністю відповідає актуальним вимогам електронної комерції.

Для досягнення мети послідовно вирішено низку важливих завдань. Насамперед глибоко проаналізовано сферу онлайн-продажів творів мистецтва. Вивчення цієї предметної області дало змогу сформувати чітку бізнес-модель системи, визначити базовий функціонал та розробити логіку взаємодії майбутніх клієнтів із програмним продуктом. Оцінка існуючих рішень допомогла спроєктувати власний варіант застосунку, який фокусується на зручній навігації, мінімалізмі та естетичній привабливості.

Наступним кроком успішно проаналізовано сучасні програмні та інструментальні засоби для розробки інтернет-магазину. Дослідження актуальних парадигм веброзробки допомогло обґрунтувати вибір оптимального технологічного стека та закріпити принципи побудови надійних клієнт-серверних рішень. Складена специфікація технічних вимог гарантує високу продуктивність застосунку, де серверну логіку побудовано на базі середовища Node.js, а клієнтську частину реалізовано за допомогою гнучкого компонентного підходу.

Важливим етапом стало інженерне проєктування системи, під час якого розроблено логіку бази даних для надійного зберігання інформації про товари, користувачів та замовлення. Побудовано структурні діаграми для візуалізації внутрішніх процесів та створено нормалізовану реляційну схему під управлінням Microsoft SQL Server. Такий підхід повністю виключає дублювання інформації,

підтримує посилальну цілісність та створює міцний інформаційний фундамент платформи.

У рамках практичної частини успішно розроблено логіку вебзастосунку, включаючи системи автентифікації клієнтів, управління списком бажаного та віртуальним кошиком. Як підсумок, виконано безпосередню програмну реалізацію та комплексне тестування інтернет-магазину в різних сценаріях використання. Результати перевірки доводять абсолютну працездатність усіх модулів: система коректно обробляє реєстрацію, блокує додавання розпроданих картин та надійно фіксує кожне оформлене замовлення.

Виконання кваліфікаційної бакалаврської роботи дозволило не лише систематизувати теоретичні знання, але й отримати готовий до експлуатації продукт. Створена платформа забезпечує зручний та безпечний процес купівлі мистецтва онлайн. Розроблена архітектура має значний технічний потенціал для подальшого масштабування і впровадження новітніх інструментів аналітики.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Pikh I., Merenych Y. Semantic models for web application design. Computer Systems and Information Technologies, 2025, (2), 20–26. DOI: 10.31891/csit-2025-2-2.

2. Artsy – Discover, Buy, and Sell Fine Art. Artsy. URL: https://www.artsy.net/ (Accessed: 21.04.2026).

3. Artwork: Buy Original Art Online, Paintings & More. Saatchi Art. URL: https://www.saatchiart.com/ (Accessed: 21.04.2026).

4. Phaidon Gallery. Phaidon. URL: https://www.phaidon.com/ (Accessed: 23.04.2026).

5. Tate Gallery. Tate. URL: https://www.tate.org.uk/ (Accessed: 23.04.2026).

6. Blinowski G., Ojdowska A., Przybyłek A. Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation // IEEE Access, 2022. Vol. 10, pp. 20357-20374. DOI: 10.1109/ACCESS.2022.3152803.

7. Yakovenko V., Ulianovska Y., Yakovenko T. Analysing features of e-commerce systems architecture // Economics of enterprises. Macroeconomics. Economic Cybernetics: Reports on Research Projects, 2022. Vol. 1 No. 4(63). DOI: 10.15587/2706-5448.2022.253932.

8. Stepanov O., Klym H. Features of the implementation of micro-interfaces in information systems // Advances in Cyber-Physical Systems. 2024. Vol. 9 No. 1. P. 54–60. DOI: 10.23939/acps2024.01.054.

9. HTML5. HTML Living Standard. URL: https://html.spec.whatwg.org/multipage/ (Accessed: 27.04.2026).

10. Difference between HTML, CSS and JavaScript. Medium. URL: https://medium.com/@Bharat2044/difference-between-html-css-and-javascript-51a977ac3ed1 (Accessed: 25.04.2026).

11. Неділько О., Сачук В. Порівняння JavaScript-фреймворків React, Angular і Vue.js у дослідженні продуктивності та масштабованості веб-додатків. Herald of Khmelnytskyi National University, 2025. Technical Sciences, 359(6.2), 231-234. DOI: 10.31891/2307-5732-2025-359-103.

12. Пех П., Янковський Б. Особливості створення веб-додатків з використанням C# ASP.NET Core MVC. Комп’ютерно-інтегровані технології: освіта, наука, виробництво, 2025, (59), 253-257. DOI: 10.36910/6775-2524-0560-2024-57-32.

13. Node.js Documentation. Node.js – Run JavaScript Everywhere. URL: https://nodejs.org/docs/latest/api/ (Accessed: 21.04.2026).

14. Getting started with Git. Medium. URL: https://medium.com/@sri.ashish91/getting-started-with-git-42829d31255d (accessed: 25.04.2026).

15. Воротнікова З. Огляд баз даних при розробці програмного забезпечення для різних операційних систем. Вісник Приазовського Державного Технічного Університету, 2025. Серія: Технічні науки, (51), 19–31. DOI: 10.31498/2225-6733.51.2025.344596.

16. Козуб Г., Сурма Ю., Артеменко О. Роль вебкомпонентів у побудові сучасних інтерфейсів: переваги та обмеження. Вісник Херсонського національного технічного університету, 2025. Том 2 № 2(93). DOI: 10.35546/kntu2078-4481.2025.2.2.21.

17. Дзюрбан Е., Яшина О. Метод оцінки об’єктно-орієнтованих програмних систем на основі аналізу зміни вимог до програмної системи. Herald of Khmelnytskyi National University, 2022. Technical Sciences, 315 № 6(1), 77-81. DOI: 10.31891/2307-5732-2022-315-6-77-81.

18. Angular. URL: https://angular.dev/reference/configs/file-structure (accessed: 27.04.2026).

ДОДАТОК А

Лістинг коду сторінки оформлення замовлення

HTML сторінки

```
<div class="container my-5">
    <a class="navbar-brand pe-4 fs-4" href="/">
            <svg
              xmlns="http://www.w3.org/2000/svg"
              width="32"
              height="32"
              fill="currentColor"
              class="bi bi-nintendo-switch"
              viewBox="0 0 16 16"
            >
              <path
                d="M9.34 8.005c0-4.38.01-7.972.023-7.982C9.373.01 10.036 0 10.831
0c1.153 0 1.51.01 1.743.05 1.73.298 3.045 1.6 3.373 3.326.046.242.053.809.053 4.61 0
4.06.005 4.537-.123 4.976-.022.076-.048.15-.08.242a4.14 4.14 0 0 1-3.426 2.767c-
.317.033-2.889.046-2.978.013-.05-.02-.053-.752-.053-7.979m4.675.269a1.62 1.62 0 0 0-
1.113-1.034 1.61 1.61 0 0 0-1.938 1.073 1.9 1.9 0 0 0-.014.935 1.63 1.63 0 0 0 1.952
1.107c.51-.136.908-.504 1.11-1.028.11-.285.113-.742.003-1.053M3.71 3.317c-.208.04-
.526.199-.695.348-.348.301-.52.729-.494
1.232.013.262.03.332.136.544.155.321.39.556.712.715.222.11.278.123.567.133.261.01.354
0 .53-.06.719-.242 1.153-.94 1.03-1.656-.142-.852-.95-1.422-1.786-1.256"
              />
              <path
                d="M3.425.053a4.14 4.14 0 0 0-3.28 3.015C0 3.628-.01 3.956.005
8.3c.01 3.99.014 4.082.08 4.39.368 1.66 1.548 2.844 3.224 3.235.22.05.497.06 2.29.07
1.856.012 2.048.009 2.097-.04.05-.05.053-.69.053-7.94 0-5.374-.01-7.906-.033-7.952-
.033-.06-.09-.063-2.03-.06-1.578.004-2.052.014-2.26.05Zm3 14.665-1.35-.016c-1.242-
.013-1.375-.02-1.623-.083a2.81 2.81 0 0 1-2.08-2.167c-.074-.335-.074-8.579-.004-
8.907a2.85 2.85 0 0 1 1.716-2.05c.438-.176.64-.196 2.058-.2l1.282-.003v13.426Z"
              />
            </svg>
            <span class="ms-1 fw-bolder">Sylvian</span>
          </a>
  <h1 class="mb-4">Оформлення замовлення</h1>
  <div class="row gx-4">

    <div class="col-md-7">
      <form [formGroup]="form" (ngSubmit)="submit()">

        <h5>Contact</h5>
        <div class="mb-3">
          <input type="email"
                 formControlName="contactEmail"
                 class="form-control"
                 placeholder="Email">
          <div *ngIf="f['contactEmail'].touched && f['contactEmail'].invalid"
```

```
       class="text-danger small">
    <div *ngIf="f['contactEmail'].errors?.['required']">
      Email обов’язковий
    </div>
    <div *ngIf="f['contactEmail'].errors?.['email']">
      Невірний формат
    </div>
  </div>
</div>


<h5 class="mt-4">Shipping address</h5>
<div class="mb-3">
  <select formControlName="country" class="form-select">
    <option value="">Виберіть країну</option>
    <option>Ukraine</option>
    <option>Poland</option>
    <option>USA</option>
  </select>
  <div *ngIf="f['country'].touched && f['country'].invalid"
       class="text-danger small">
    Країну обов’язково вибрати
  </div>
</div>
<div class="mb-3">
  <input type="text"
         formControlName="fullName"
         class="form-control"
         placeholder="First & Last Name">
  <div *ngIf="f['fullName'].touched && f['fullName'].invalid"
       class="text-danger small">
    Ім’я обов’язкове
  </div>
</div>
<div class="mb-3">
  <input type="text"
         formControlName="address"
         class="form-control"
         placeholder="Address">
  <div *ngIf="f['address'].touched && f['address'].invalid"
       class="text-danger small">
    Адреса обов’язкова
  </div>
</div>
<div class="row g-2">
  <div class="col">
    <input type="text"
           formControlName="postalCode"
           class="form-control"
           placeholder="Postal code">
    <div *ngIf="f['postalCode'].touched && f['postalCode'].invalid"
```

```
                class="text-danger small">
              Поштовий індекс обов’язковий
            </div>
          </div>
          <div class="col">
            <input type="text"
                   formControlName="city"
                   class="form-control"
                   placeholder="City">
            <div *ngIf="f['city'].touched && f['city'].invalid"
                 class="text-danger small">
              Місто обов’язкове
            </div>
          </div>
        </div>
        <div class="mb-3 mt-3">
  <input
    type="tel"
    formControlName="phone"
    class="form-control"
    placeholder="Phone"
    maxlength="15"
    inputmode="tel"
  />
  <div *ngIf="f['phone'].touched && f['phone'].invalid" class="text-danger small">
    <div *ngIf="f['phone'].errors?.['required']">Phone обов’язковий</div>
    <div *ngIf="f['phone'].errors?.['pattern']">Лише цифри, пробіли, +, -, ()</div>
    <div *ngIf="f['phone'].errors?.['maxlength']">Максимум 15 символів</div>
  </div>
</div>

<div class="mb-3">
  <input
    type="text"
    formControlName="cardNumber"
    class="form-control"
    placeholder="Card Number"
    maxlength="16"
    inputmode="numeric"
  />
  <div *ngIf="f['cardNumber'].touched && f['cardNumber'].invalid" class="text-danger
small">
    <div *ngIf="f['cardNumber'].errors?.['required']">Card number обов’язковий</div>
    <div *ngIf="f['cardNumber'].errors?.['pattern']">Лише цифри</div>
    <div *ngIf="f['cardNumber'].errors?.['minlength'] ||
f['cardNumber'].errors?.['maxlength']">
      Має бути рівно 16 цифр
    </div>
  </div>
</div>
```

```
  <div class="row g-2 mb-3">

  <div class="col-md-6 form-group">
    <label for="expiry" class="form-label small">Expiry</label>
    <p-calendar
  id="expiry"
   formControlName="expiry"
  view="month"
  dateFormat="mm/yy"
  placeholder="MM/YY"
  [showIcon]="true"
 ></p-calendar>
    <div *ngIf="f['expiry'].touched && f['expiry'].invalid" class="text-danger small
mt-1">
      <div *ngIf="f['expiry'].errors?.['required']">Виберіть місяць та рік</div>
    </div>
  </div>


    <div class="col">
      <label for="cvv" class="form-label small">CVV</label>
      <input
        id="cvv"
        type="text"
        formControlName="cvv"
        class="form-control"
        maxlength="3"
        inputmode="numeric"
        placeholder="123"
      />
      <div *ngIf="f['cvv'].touched && f['cvv'].invalid" class="text-danger small">
        <div *ngIf="f['cvv'].errors?.['required']">CVV обов’язковий</div>
        <div *ngIf="f['cvv'].errors?.['pattern']">Лише цифри</div>
        <div *ngIf="f['cvv'].errors?.['minlength'] ||
f['cvv'].errors?.['maxlength']">
          Має бути 3 цифри
        </div>
      </div>
    </div>
  </div>

        <button
  type="submit"
  class="btn btn-primary w-100 mt-4"
  [disabled]="form.invalid || items.length === 0 || loading"
>
  {{ loading ? 'Оформлення...' : 'Purchase' }}
</button>
      </form>
    </div>
```

```
    <div class="col-md-5">
      <div class="card p-3">
        <h5>Order summary</h5>
        <ul class="list-group mb-3">
          <li class="list-group-item d-flex justify-content-between align-items-
center"
              *ngFor="let i of items">
            <span>{{ i.Title }} × {{ i.Quantity }}</span>
            <span>{{ (i.Price * i.Quantity) | currency }}</span>
          </li>
        </ul>
        <div class="d-flex justify-content-between">
          <strong>Total</strong>
          <strong>{{ total | currency }}</strong>
        </div>
      </div>
    </div>
  </div>
</div>
```

TypeScript частина

```
import { Component, OnInit } from '@angular/core';
import { CommonModule } from '@angular/common';
import { ReactiveFormsModule, FormBuilder, FormGroup, Validators } from
'@angular/forms';
import { Router } from '@angular/router';
import { ApiService } from '../services/api.service';
import { ToastrService } from 'ngx-toastr';
import { BsDatepickerModule, BsDatepickerConfig } from 'ngx-bootstrap/datepicker';
import { CalendarModule } from 'primeng/calendar';

@Component({
  selector: 'app-checkout',
  standalone: true,
  imports: [
    CommonModule,
    ReactiveFormsModule,
    BsDatepickerModule,
    CalendarModule
  ],
  templateUrl: './checkout.component.html',
  styleUrls: ['./checkout.component.css']
})
export class CheckoutComponent implements OnInit {
  form!: FormGroup;
  items: any[] = [];
  loading = false;
```

```
  datePickerConfig: Partial<BsDatepickerConfig>;

  constructor(
    private fb: FormBuilder,
    public api: ApiService,
    private toastr: ToastrService,
    private router: Router,
    private bsConfig: BsDatepickerConfig
  ) {

    this.datePickerConfig = {
      ...this.bsConfig,
      minMode: 'month',
      dateInputFormat: 'MM/YYYY',
      showWeekNumbers: false
    };
  }

  ngOnInit() {

    this.api.getCart().subscribe({
      next: data => this.items = data,
      error: () => this.toastr.error('Не вдалося завантажити кошик', 'Помилка')
    });


    this.form = this.fb.group({
      contactEmail: [ '', [Validators.required, Validators.email] ],
      fullName:     [ '', Validators.required ],
      address:      [ '', Validators.required ],
      city:         [ '', Validators.required ],
      postalCode:   [ '', Validators.required ],
      country:      [ '', Validators.required ],
      phone: [
        '',
        [
          Validators.required,
          Validators.pattern(/^[0-9\-\+\s\(\)]+$/),
          Validators.maxLength(15)
        ]
      ],
      cardNumber: [
        '',
        [
          Validators.required,
          Validators.pattern(/^\d+$/),
          Validators.minLength(16),
          Validators.maxLength(16)
        ]
      ],
      expiry: [
```

```
      null,
      [ Validators.required ]
    ],
    cvv: [
      '',
      [
        Validators.required,
        Validators.pattern(/^\d+$/),
        Validators.minLength(3),
        Validators.maxLength(3)
      ]
    ]
  });
}

get f() {
  return this.form.controls;
}

get total(): number {
  return this.items.reduce((sum, i) => sum + i.Price * i.Quantity, 0);
}

submit() {
if (this.form.invalid || this.items.length === 0) {
  this.form.markAllAsTouched();
  return;
}

const expDate: Date = this.form.value.expiry;
const month = (expDate.getMonth() + 1).toString().padStart(2, '0');
const year = expDate.getFullYear();
const expiryString = `${year}-${month}`;

const orderItems = this.items.map(i => ({
  paintingId: i.PaintingId,
  quantity: i.Quantity
}));

const payload = {
  items: orderItems,
  payment: {
    cardNumber: this.form.value.cardNumber,
    expiry: expiryString,
    cvv: this.form.value.cvv
  },
  shipping: {
    email: this.form.value.contactEmail,
    name: this.form.value.fullName,
    address: this.form.value.address,
    city: this.form.value.city,
```

```
      postalCode: this.form.value.postalCode,
      country: this.form.value.country,
      phone: this.form.value.phone
    }
  };

  this.loading = true;


  this.api.createOrder(payload.items).subscribe({
    next: async () => {
      this.toastr.success('Замовлення успішно оформлено!', 'Success');

      try {

        await this.api.clearCart().toPromise();


        this.items = [];


        this.router.navigate(['/']);
      } catch (error) {
        this.toastr.error('Помилка очищення кошика', 'Error');
        console.error(error);
      } finally {
        this.loading = false;
      }
    },
    error: (err) => {

      if (err.status === 401) {
        this.toastr.error('Сесія закінчилась. Будь ласка, увійдіть знову', 'Помилка
авторизації');
        this.api.logout();
        this.router.navigate(['/login']);
      } else {
        this.toastr.error(err.error?.message || 'Помилка оформлення', 'Error');
      }
      this.loading = false;
    }
  });
}
}
```

ДОДАТОК Б

Лістинг коду серверної складової

JavaScript

```
const express = require('express');
const sql = require('mssql/msnodesqlv8');
const cors = require('cors');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');

const app = express();
app.use(cors());
app.use(express.json());

const JWT_SECRET = 'Froggie';
const TOKEN_EXPIRES = '2h';

const dbConfig = {
  driver: 'msnodesqlv8',
  connectionString:
    'Driver={ODBC Driver 17 for SQL
Server};Server=S1LVERASH\\SQLEXPRESS;Database=GallerySite;Trusted_Connection=Yes;Trus
tServerCertificate=Yes;',
  requestTimeout: 300000
};

// ==================== АВТЕНТИФІКАЦІЯ ====================

app.post('/register', async (req, res) => {
  const { nickname, email, password } = req.body;
  if (!nickname || !email || !password) {
    return res.status(400).json({ message: 'Усі поля обов’язкові' });
  }

  try {
    const pool = await sql.connect(dbConfig);


    const exists = await pool.request()
      .input('email', sql.NVarChar, email)
      .query('SELECT 1 FROM [User] WHERE Email = @email');

    if (exists.recordset.length) {
      return res.status(409).json({ message: 'Email уже зареєстровано' });
    }

    const hash = await bcrypt.hash(password, 10);
    await pool.request()
      .input('nickname', sql.NVarChar, nickname)
```

```
      .input('email',    sql.NVarChar, email)
      .input('hash',     sql.NVarChar, hash)
      .query(`
        INSERT INTO [User](Nickname, Email, PasswordHash)
        VALUES (@nickname, @email, @hash)
      `);

    res.status(201).json({ message: 'Реєстрація успішна' });
  } catch (err) {
    console.error(err);

    if (err.originalError?.number === 2627) {
      return res.status(409).json({ message: 'Email уже зареєстровано' });
    }
    res.status(500).json({ message: 'Server error' });
  }
});

app.post('/login', async (req, res) => {
  const { email, password } = req.body;
  if (!email || !password) {
    return res.status(400).json({ message: 'Усі поля обов’язкові' });
  }

  try {
    const pool = await sql.connect(dbConfig);
    const result = await pool.request()
      .input('email', sql.NVarChar, email)
      .query('SELECT Id, Nickname, PasswordHash FROM [User] WHERE Email = @email');

    if (!result.recordset.length) {
      return res.status(401).json({ message: 'Неправильний email або пароль' });
    }

    const user = result.recordset[0];
    const match = await bcrypt.compare(password, user.PasswordHash);
    if (!match) {
      return res.status(401).json({ message: 'Неправильний email або пароль' });
    }


    const payload = { userId: user.Id, nickname: user.Nickname };
    const token = jwt.sign(payload, JWT_SECRET, { expiresIn: TOKEN_EXPIRES });


    res.json({ token, nickname: user.Nickname });
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Server error' });
  }
});
```

```
function auth(req, res, next) {
  const auth = req.headers.authorization;
  if (!auth) return res.status(401).send('No token');

  const [scheme, token] = auth.split(' ');
  if (scheme !== 'Bearer' || !token) return res.status(401).send('Bad token');

  try {
    const payload = jwt.verify(token, JWT_SECRET);
    req.userId = payload.userId;
    next();
  } catch (err) {
    console.error('JWT verification error:', err.message);


    let errorMessage = 'Invalid token';
    if (err.name === 'TokenExpiredError') {
      errorMessage = 'Token expired';
    } else if (err.name === 'JsonWebTokenError') {
      errorMessage = 'Malformed token';
    }

    res.status(401).send(errorMessage);
  }
}


// ==================== КАРТИНИ, КАТЕГОРІЇ ====================

app.get('/paintings', async (req, res) => {
  try {
    const pool = await sql.connect(dbConfig);
    const result = await pool.request().query('SELECT * FROM Painting');
    res.json(result.recordset);
  } catch (err) {
    res.status(500).send(err.message);
  }
});

app.post('/paintings', async (req, res) => {
  const { title, description, price, stock, imageUrl, categoryId } = req.body;
  try {
    const pool = await sql.connect(dbConfig);
    await pool.request()
      .input('t', sql.NVarChar, title)
      .input('d', sql.NVarChar, description)
      .input('p', sql.Decimal(10, 2), price)
      .input('s', sql.Int, stock)
      .input('img', sql.NVarChar, imageUrl)
      .input('cid', sql.Int, categoryId)
```

```
      .query(`
        INSERT INTO Painting(Title, Description, Price, Stock, ImageUrl, CategoryId,
CreatedAt)
        VALUES (@t, @d, @p, @s, @img, @cid, GETDATE())`);
    res.status(201).send('Створено');
  } catch (err) {
    res.status(500).send(err.message);
  }
});

app.get('/categories', async (req, res) => {
  try {
    const pool = await sql.connect(dbConfig);
    const result = await pool.request().query('SELECT * FROM Category');
    res.json(result.recordset);
  } catch (err) {
    res.status(500).send(err.message);
  }
});

// ==================== ЗАМОВЛЕННЯ ====================

app.post('/orders', auth, async (req, res) => {
  const { items } = req.body;
  let tx;
  try {
    const pool = await sql.connect(dbConfig);
    tx = pool.transaction();
    await tx.begin();


    for (const item of items) {
      const stockRes = await tx.request()
        .input('pId', sql.Int, item.paintingId)
        .query(`
          SELECT Stock
          FROM Painting WITH (UPDLOCK, ROWLOCK)
          WHERE Id = @pId;
        `);

      const stock = stockRes.recordset[0]?.Stock ?? 0;

      if (item.quantity > stock) {
        await tx.rollback();
        return res.status(400).json({
          message: `Only ${stock} items left in stock for painting
${item.paintingId}`
        });
      }
    }
```

```
    const insertOrder = await tx.request()
      .input('userId', sql.Int, req.userId)
      .query(`
        INSERT INTO [Order](UserId, OrderDate, TotalAmount, Status)
        VALUES (@userId, GETDATE(), 0, 'new');
        SELECT SCOPE_IDENTITY() AS orderId;
      `);
    const orderId = insertOrder.recordset[0].orderId;


    for (const it of items) {

      const priceRes = await tx.request()
        .input('pId', sql.Int, it.paintingId)
        .query('SELECT Price FROM Painting WHERE Id = @pId;');
      const unitPrice = priceRes.recordset[0]?.Price ?? 0;


      await tx.request()
        .input('oId', sql.Int, orderId)
        .input('pId', sql.Int, it.paintingId)
        .input('qty', sql.Int, it.quantity)
        .input('up', sql.Decimal(10,2), unitPrice)
        .query(`
          INSERT INTO OrderItem(OrderId, PaintingId, Quantity, UnitPrice)
          VALUES (@oId, @pId, @qty, @up);
        `);
    }


    await tx.request()
      .input('oId', sql.Int, orderId)
      .query(`
        UPDATE [Order]
        SET TotalAmount = dbo.fn_CalcOrderTotal(@oId)
        WHERE Id = @oId;
      `);

    await tx.commit();
    res.status(201).json({ orderId });

  } catch (err) {
    console.error(err);
    if (tx) {
      try { await tx.rollback(); } catch {}
    }


    if (err.message.includes('Недостатній запас')) {
```

```
      return res.status(400).json({ message: 'Insufficient stock for one of the
paintings' });
    }

    res.status(500).json({ message: 'Server error' });
  }
});

app.get('/my-orders', auth, async (req, res) => {
  try {
    const pool = await sql.connect(dbConfig);
    const result = await pool.request()
      .input('userId', sql.Int, req.userId)
      .query('SELECT * FROM [Order] WHERE UserId = @userId;');
    res.json(result.recordset);
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Server error' });
  }
});

// ==================== КОШИК ====================

app.post('/cart', auth, async (req, res) => {
  const { paintingId, quantity } = req.body;
  let tx;
  try {
    const pool = await sql.connect(dbConfig);
    tx = pool.transaction();
    await tx.begin();


    const pRes = await tx.request()
      .input('pId', sql.Int, paintingId)
      .query(`
        SELECT Stock
        FROM Painting WITH (UPDLOCK, ROWLOCK)
        WHERE Id = @pId;
      `);
    if (!pRes.recordset.length) {
      await tx.rollback();
      return res.status(404).json({ message: 'Painting not found' });
    }
    const stock = pRes.recordset[0].Stock;


    const cRes = await tx.request()
      .input('uId', sql.Int, req.userId)
      .query('SELECT Id FROM Cart WHERE UserId = @uId;');
    let cartId;
    if (cRes.recordset.length) {
```

```
  cartId = cRes.recordset[0].Id;
} else {
  const nc = await tx.request()
    .input('uId2', sql.Int, req.userId)
    .query(`
      INSERT INTO Cart(UserId, CreatedAt)
      VALUES (@uId2, GETDATE());
      SELECT SCOPE_IDENTITY() AS cartId;
    `);
  cartId = nc.recordset[0].cartId;
}


const ciRes = await tx.request()
  .input('cId', sql.Int, cartId)
  .input('pId2', sql.Int, paintingId)
  .query(`
    SELECT Id, Quantity
    FROM CartItem
    WHERE CartId = @cId AND PaintingId = @pId2;
  `);
const existing = ciRes.recordset[0];
const existingQty = existing ? existing.Quantity : 0;


if (quantity < 0) {
  await tx.rollback();
  return res.status(400).json({ message: 'Quantity cannot be negative' });
}


const newQty = quantity;
const diff = newQty - existingQty;

// Перевірити запас
if (newQty > stock) {
  await tx.rollback();
  return res.status(400).json({ message: `Only ${stock} items left in stock` });
}


if (existing) {
  if (newQty === 0) {
    await tx.request()
      .input('ciId', sql.Int, existing.Id)
      .query(`DELETE FROM CartItem WHERE Id = @ciId;`);
  } else {
    await tx.request()
      .input('ciId2', sql.Int, existing.Id)
      .input('newQ',  sql.Int, newQty)
      .query(`
```

```
            UPDATE CartItem
            SET Quantity = @newQ
            WHERE Id = @ciId2;
          `);
      }
    } else if (newQty > 0) {
      await tx.request()
        .input('cId2', sql.Int, cartId)
        .input('pId4', sql.Int, paintingId)
        .input('q2',   sql.Int, newQty)
        .query(`
          INSERT INTO CartItem(CartId, PaintingId, Quantity)
          VALUES (@cId2, @pId4, @q2);
        `);
    }

    await tx.commit();
    res.json({ message: 'Cart updated' });

  } catch (err) {
    console.error(err);
    if (tx) {
      try { await tx.rollback(); } catch {}
    }
    res.status(500).json({ message: 'Server error' });
  }
});
app.get('/cart', auth, async (req, res) => {
  try {
    const pool = await sql.connect(dbConfig);
    const result = await pool.request()
      .input('u', sql.Int, req.userId)
      .query(`
        SELECT
          ci.Id              AS CartItemId,
          ci.Quantity,
          p.Id               AS PaintingId,
          p.Title,
          p.Price,
          p.Stock            AS remainingStock,
          (ci.Quantity + p.Stock) AS maxQty,
          p.ImageUrl
        FROM Cart c
        JOIN CartItem ci ON ci.CartId = c.Id
        JOIN Painting p  ON p.Id      = ci.PaintingId
        WHERE c.UserId = @u;
      `);
    return res.json(result.recordset);
  } catch (err) {
    console.error(err);
    return res.status(500).json({ message: 'Server error' });
```

```
  }
});

app.delete('/cart/:id', auth, async (req, res) => {
  const cartItemId = parseInt(req.params.id, 10);
  if (isNaN(cartItemId)) {
    return res.status(400).json({ message: 'Невірний ID позиції' });
  }
  try {
    const pool = await sql.connect(dbConfig);
    const ciRes = await pool.request()
      .input('ci', sql.Int, cartItemId)
      .input('u', sql.Int, req.userId)
      .query(`
        SELECT ci.Quantity, ci.PaintingId
        FROM CartItem ci
        JOIN Cart c ON c.Id = ci.CartId
        WHERE ci.Id = @ci AND c.UserId = @u;
      `);
    if (!ciRes.recordset.length) {
      return res.status(404).json({ message: 'Позиція не знайдена' });
    }
    const { Quantity, PaintingId } = ciRes.recordset[0];

    await pool.request()
      .input('ci', sql.Int, cartItemId)
      .query('DELETE FROM CartItem WHERE Id = @ci');

    await pool.request()
      .input('p', sql.Int, PaintingId)
      .input('d', sql.Int, Quantity)
      .query('UPDATE Painting SET Stock = Stock + @d WHERE Id = @p');

    return res.json({ message: 'Позицію видалено' });
  } catch (err) {
    console.error(err);
    return res.status(500).json({ message: 'Server error' });
  }
});

app.delete('/cart', auth, async (req, res) => {
  try {
    const pool = await sql.connect(dbConfig);


    const cartRes = await pool.request()
      .input('u', sql.Int, req.userId)
      .query('SELECT Id FROM Cart WHERE UserId = @u');

    if (!cartRes.recordset.length) {
      return res.status(404).json({ message: 'Кошик не знайдено' });
```

```
    }

    const cartId = cartRes.recordset[0].Id;


    await pool.request()
      .input('c', sql.Int, cartId)
      .query('DELETE FROM CartItem WHERE CartId = @c');

    return res.json({ message: 'Кошик очищено' });
  } catch (err) {
    console.error(err);
    return res.status(500).json({ message: 'Server error' });
  }
});
// ==================== WISHLIST ====================

app.post('/wishlist', auth, async (req, res) => {
  const { paintingId } = req.body;
  try {
    const pool = await sql.connect(dbConfig);
    const result = await pool.request()
      .input('u', sql.Int, req.userId)
      .input('p', sql.Int, paintingId)
      .query(`
        INSERT INTO Wishlist (UserId, PaintingId)
        VALUES (@u, @p);
        SELECT SCOPE_IDENTITY() AS id;
      `);

    const wishlistRowId = result.recordset[0].id;
    res.status(201).json({ id: wishlistRowId });
  } catch (err) {
    console.error(err);

    if (err.originalError?.number === 2627) {
      return res.status(409).json({ message: 'Already in wishlist' });
    }
    res.status(500).json({ message: 'Server error' });
  }
});

app.get('/wishlist', auth, async (req, res) => {
  try {
    const pool = await sql.connect(dbConfig);
    const result = await pool.request()
      .input('u', sql.Int, req.userId)
      .query(`
        SELECT
          w.Id             AS wishlistId,
          p.Id             AS paintingId,
```

```
          p.Title,
          p.Description,
          p.Price,
          p.Stock,
          p.ImageUrl,
          p.CategoryId,
          p.CreatedAt
        FROM Wishlist w
        JOIN Painting p ON p.Id = w.PaintingId
        WHERE w.UserId = @u
      `);
    res.json(result.recordset);
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Server error' });
  }
});

app.delete('/wishlist/painting/:paintingId', auth, async (req, res) => {
  const paintingId = parseInt(req.params.paintingId, 10);
  if (isNaN(paintingId)) {
    return res.status(400).json({ message: 'Invalid paintingId' });
  }
  try {
    const pool = await sql.connect(dbConfig);
    const result = await pool.request()
      .input('p', sql.Int, paintingId)
      .input('u', sql.Int, req.userId)
      .query(`
        DELETE FROM Wishlist
        WHERE PaintingId = @p AND UserId = @u;
      `);
    if (result.rowsAffected[0] === 0) {
      return res.status(404).json({ message: 'Not found in wishlist' });
    }
    res.json({ message: 'Removed from wishlist' });
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Server error' });
  }
});

app.post('/wishlist/painting/:paintingId', auth, async (req, res) => {
  const paintingId = parseInt(req.params.paintingId, 10);
  if (isNaN(paintingId)) {
    return res.status(400).json({ message: 'Invalid paintingId' });
  }
  try {
    const pool = await sql.connect(dbConfig);
    await pool.request()
      .input('p', sql.Int, paintingId)
```

```
      .input('u', sql.Int, req.userId)
      .query(`
        INSERT INTO Wishlist (UserId, PaintingId)
        VALUES (@u, @p);
      `);
    res.status(201).json({ message: 'Added to wishlist' });
  } catch (err) {
    if (err.originalError?.number === 2627) {
      return res.status(409).json({ message: 'Already in wishlist' });
    }
    console.error(err);
    res.status(500).json({ message: 'Server error' });
  }
});
const PORT = 3000;
```