

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Інформаційно-аналітична система для продажу та
покупки творів мистецтва»**

Виконав студент групи ІПЗм-23-2
ЛІНЕЦЬКИЙ Богдан Ігорович

Керівник роботи:
ЛЕВКІВСЬКИЙ Віталій Леонідович

Рецензент:
ЧИЖМОТРЯ Олексій Володимирович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Завідувач кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«01» жовтня 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ЛІНЕЦЬКИЙ Богдан Ігорович**

Керівник роботи: **ЛЕВКІВСЬКИЙ Віталій Леонідович**

Тема роботи: **«Інформаційно-аналітична система для продажу та покупки творів мистецтва»**,

затверджена Наказом закладу вищої освіти від **«01» жовтня 2024 р. № 520/с**

Вихідні дані для роботи: об'єктом дослідження є процес продажу та купівлі творів мистецтва, спілкування між учасниками та можливість аналізу їх діяльності. Предметом дослідження є використання веборієнтованих технологій для автоматизації процесу продажу та купівлі творів мистецтва, спілкування між учасниками та можливість аналізу їх діяльності.

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Левківський В. Л.	01.10.2024р.	01.10.2024р.
2	Левківський В. Л.	01.10.2024р.	01.10.2024р.
3	Левківський В. Л.	01.10.2024р.	01.10.2024р.

РЕФЕРАТ

Кваліфікаційна робота освітнього ступеня «магістр» присвячена автоматизації процесу комунікації любителів мистецтва, можливості публікації художніх творів та функціоналу для їх купівлі і продажу.

Кваліфікаційна робота складається з програмної системи та пояснювальної записки. Пояснювальна записка до кваліфікаційної роботи містить 65 сторінок, 31 ілюстрацію та 5 таблиць. Робота містить 22 літературних посилання. Додатки викладені на 42 сторінках.

В пояснювальній записці до кваліфікаційної роботи визначені головні завдання для розробки системи, проаналізовано аналоги розробленої системи. Наведено загальну структуру програмного комплексу, опис алгоритмів взаємодії окремих модулів системи, сценарії роботи програмного комплексу. Також наводяться результати тестування системи.

КЛЮЧОВІ СЛОВА: МИСТЕЦТВО, СТАТИСТИКА, ЗАПИС, АНАЛІЗ ДАНИХ, САЙТ, СИСТЕМА, ПОКУПКА, ПРОДАЖ.

ABSTRACT

The qualification work of the educational degree "master" is dedicated to the automation of the communication process of art lovers, the possibility of publishing artistic works and functionality for their purchase and sale.

The qualification project consists of a software system and an explanatory note. The explanatory note comprises 65 pages, 31 illustrations, and 5 tables. The project includes 22 references, with appendices presented over 42 pages.

The explanatory note outlines the primary objectives for the system's development, analyzes existing analogs of the developed system, and presents the overall structure of the software complex. It describes the algorithms for interaction between individual system modules and provides scenarios for the operation of the software complex. Additionally, the results of system testing are presented.

KEYWORDS: ART, STATISTICS, RECORD, DATA ANALYSIS, WEBSITE, SYSTEM, BUYING, SELLING.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ СИСТЕМИ	9
1.1. АНАЛІЗ АНАЛОГІВ ПРОГРАМНОГО ПРОДУКТУ.....	9
1.2. МЕТОДИ ВІЗУАЛІЗАЦІЇ АНАЛІТИЧНИХ ДАНИХ	12
1.3. ПОСТАНОВКА ЗАДАЧІ	16
1.4. ВИБІР АРХІТЕКТУРИ СИСТЕМИ.....	17
1.5. ОБҐРУНТУВАННЯ ВИБОРУ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ТА ВИМОГИ ДО АПАРАТНОГО ЗАБЕЗПЕЧЕННЯ.....	19
Висновки до першого розділу	21
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНО-АНАЛІТИЧНОЇ СИСТЕМИ	23
2.1. ВИЗНАЧЕННЯ ВАРІАНТІВ ВИКОРИСТАННЯ ТА СТРУКТУРИ СИСТЕМИ.....	23
2.2. РОЗРОБКА БАЗИ ДАНИХ СИСТЕМИ.....	26
2.3. ПРАКТИЧНА РЕАЛІЗАЦІЯ АНАЛІТИЧНОЇ СКЛАДОВОЇ СИСТЕМИ.....	36
2.4. ПРОГРАМНА РЕАЛІЗАЦІЯ ОСНОВНИХ АЛГОРИТМІВ СИСТЕМИ	40
Висновки до другого розділу	48
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ	49
3.1. ПОРЯДОК ВСТАНОВЛЕННЯ ТА НАЛАШТУВАННЯ СИСТЕМИ.	49
3.2. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ.	50
3.3. ТЕСТУВАННЯ РОБОТИ СИСТЕМИ.	59
Висновки до третього розділу	62
ВИСНОВКИ.....	63
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	64
ДОДАТКИ.....	66

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CSS	(англ. Cascading Style Sheets) – спеціальна мова, що використовується для опису зовнішнього вигляду сторінок, побудованих за допомогою мову розмітки даних.
HTML	(англ. Hypertext Markup Language) – мова гіпертекстової розмітки вебсторінок
SQL	(англ. Structured Query Language) – декларативна мова програмування, що застосовується для взаємодії користувача з базами даних
VS	Visual Studio
ПЗ	програмне забезпечення

ВСТУП

Актуальність теми. Створення інформаційно-аналітичної системи для продажу та покупки творів мистецтва зумовлена стрімким розвитком ринку мистецтва, цифровізацією економіки та зміною споживчих тенденцій. Сучасний ринок мистецтва виходить на новий рівень завдяки інтеграції сучасних технологій, що дозволяють здійснювати купівлю-продаж творів не лише в офлайн, але й через онлайн-платформи. Це особливо важливо в умовах глобалізації, коли потенційні покупці й продавці знаходяться в різних куточках світу.

Розробка інформаційно-аналітичної системи для продажу та покупки творів мистецтва відповідає потребам сучасного ринку, адже така система не тільки спрощує процес пошуку й придбання робіт, але й надає прозору інформацію про походження, вартість і стан об'єкта. Це сприяє підвищенню довіри між учасниками ринку, зниженню ризиків шахрайства та створенню сприятливих умов для нових покупців і художників.

Окрім того, важливим аспектом є впровадження аналітичних інструментів, які допомагають оцінити тенденції ринку, спрогнозувати зміни у вартості робіт та ідентифікувати перспективних художників. Система також дозволяє зберігати інформацію про транзакції, створюючи цифрові архіви, що є важливим для забезпечення спадковості культурних цінностей.

Таким чином, інформаційно-аналітична система стає не лише інструментом торгівлі, але й важливим ресурсом для розвитку ринку мистецтва, створюючи умови для його демократизації, глобалізації та подальшого зростання.

Метою кваліфікаційної роботи є розробка інформаційно-аналітичної системи для забезпечення комунікації любителів мистецтва, можливості публікації художніх творів та функціоналу для їх купівлі, продажу і аналізу діяльності.

Встановлена мета обумовлює наступні завдання: визначення архітектури та узагальненої структури системи, обґрунтування та вибір засобів реалізації системи, розробка інформаційного, логічного та математичного забезпечення системи.

Об'єктом дослідження є процес продажу та купівлі творів мистецтва, спілкування між учасниками та можливість аналізу їх діяльності.

Предметом дослідження є використання веборієнтованих технологій для автоматизації процесу продажу та купівлі творів мистецтва, спілкування між учасниками та можливість аналізу їх діяльності.

Інформаційна база дослідження складається з літературних джерел з питань розробки веборієнтованих додатків; дані провідних сучасних платформ в області мистецтва.

Методи дослідження. Під час аналізу інформаційних джерел в процесі дослідження було використано абстрагування, аналіз та формалізацію отриманих результатів.

Практичне значення результатів дослідження полягає в розробці програмного продукту, який забезпечує комунікацію для любителів творчості, покупку, продаж творів мистецтва та аналіз діяльності. Програмний продукт поєднує у собі переваги схожих платформ та мінімізує їхні недоліки.

В рамках розробки кваліфікаційної роботи були опубліковані тези:

Лінецький Б. І., Концидайло А. М., Левківський В. Л. Сучасні підходи до тестування інформаційно-аналітичних систем. Тези доповідей VII Всеукраїнської науково-технічної конференції «Комп'ютерні технології: інновації, проблеми, рішення», м. Житомир, 2 – 3 грудня 2024 р. Житомир: Житомирська політехніка, 2024. С. 135-136.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ СИСТЕМИ

1.1. Аналіз аналогів програмного продукту

У сучасному світі цифрових технологій ринок мистецтва зазнає значних змін, зокрема через впровадження онлайн-платформ для купівлі та продажу творів мистецтва. Платформи, які об'єднують художників та покупців, стали важливим інструментом для розвитку сучасного мистецтва, а також для формування нових механізмів комерціалізації творчості. Оскільки існує багато подібних онлайн-сервісів, важливо провести детальний аналіз трьох платформ, двох закордонних та однієї української щоб вивчити їхні функціональні можливості, переваги та недоліки. Це дасть змогу не лише оцінити поточний стан таких платформ, але й виявити найкращі практики для розробки власної інформаційно-аналітичної системи, спрямованої на покращення взаємодії між художниками, колекціонерами та поціновувачами мистецтва.

У даному розділі буде здійснено аналіз декількох платформ, що займаються продажем творів мистецтва онлайн, зокрема Artfinder, KyivGallery та Artnet. Кожен з цих ресурсів представляє собою успішний приклад платформи для комерціалізації творів мистецтва, проте має свої характерні риси та орієнтований на різні сегменти аудиторії. Завдяки аналізу їхніх функцій, принципів роботи та конкурентних переваг можна визначити ключові аспекти, які мають бути враховані при розробці подібного продукту, а також виявити можливості для удосконалення існуючих рішень [1].

Розглянемо кожну з цих платформ за ключовими аспектами: основні, функції, цільова аудиторія, конкурентні переваги, недоліки, відмінності.

Платформа **Artfinder** позиціонує себе як маркетплейс на продаж творів мистецтва. Місія онлайн-майданчика – зробити мистецтво доступним для поціновувачів прекрасного та створити умови для розвитку творчого потенціалу незалежних майстрів. На початкових етапах становлення проекту засновник

Artfinder Спенсер Хаймен зумів отримати фінансування від венчурних фондів Wellington Partners та Graylock+Northzone. Зараз на онлайн-платформі виставляють свої роботи понад 7 тисяч авторів зі 108 країн світу.

Основними функціями додатку є прямий продаж творів мистецтва від авторів робіт, тобто художники напряму продають свої твори покупцям, колекціонери мають змогу переглядати добірки та рекомендації саме для них по певному стилю чи жанру, кожен художник має власний профіль з біографією, історією робіт та відгуками покупців, що створює відчуття персоналізованої взаємодії, покупці мають змогу отримати консультації щодо вибору робіт художників, платформа пропонує різноманітні цінові варіанти, а також підписки, що відкривають додаткові переваги для покупців.

Artfinder орієнтований на міжнародну аудиторію — як на новачків, так і на досвідчених колекціонерів, які прагнуть безпосередньо підтримувати художників, купуючи твори напряму. Багато робіт доступні за порівняно низькими цінами, що робить платформу привабливою для молодих колекціонерів.

Перевагами системи є інтернаціональне охоплення, тобто будь-який художник світу має змогу працювати на даній платформі, а також різна цінова категорія є привабливою для будь-якого поціновувача творів мистецтва.

Головними недоліками системи є відсутність детальної аналітики, а саме платформа не надає таких глибоких аналітичних інструментів щодо ринку та цін, як наприклад Artnet. Конкуренція між художниками з різних країн може бути високою, що ускладнює виділення окремих митців [2].

Система **KyivGallery** – провідна галерея живопису, графіки та скульптури в Україні. Засновник проекту Роман Палій поставив собі за мету об'єднати талановитих українських авторів та поціновувачів прекрасного на одній платформі для розвитку вітчизняного арт-ринку. KyivGallery стала першою онлайн-галереєю в Україні. На сайті виставляються оригінальні витвори

мистецтва професійних художників та скульпторів. На даний момент із онлайн-галереєю співпрацює понад 160 авторів.

Основними функціями системи являється спеціалізація на українських художниках, що створює унікальну нішу на ринку, також платформа пропонує статті та інтерв'ю українських художників, які збагачують культурне виховання українців. KyivGallery допомагає колекціонерам у виборі творів мистецтва, так само, як і в попередньому аналозі Artfinder. Прямий продаж творів мистецтва здійснюється без посередників.

Система KyivGallery в основному націлена на внутрішній ринок України, але і міжнародні покупці цікавляться українським мистецтвом. Платформа створена для поціновувачів української культури, а також для іноземних колекціонерів, які прагнуть придбати оригінальні роботи з українським колоритом.

Основними перевагами платформи є вузька спеціалізація, яка робить її привабливою для тих, хто шукає автентичне українське мистецтво, освітні матеріали та інтерв'ю з митцями створюють унікальний контент для всіх поціновувачів мистецтва, а також KyivGallery надає особливий сервіс для колекціонерів, що допомагає їм обирати роботи відповідно до індивідуальних запитів.

Головними недоліками можна вважати обмежений асортимент у глобальному контексті, а також платформа не має аналітичних інструментів для відстеження ринкових тенденцій і цінової історії [3].

Платформа **Artnet** – це міжнародна онлайн-платформа для торгівлі предметами мистецтва, керована компанією Artnet Worldwide Corporation. Засновник проекту П'єр Серне ще 1989 року розробив програмне забезпечення з метою оцінки ринкової вартості творів мистецтва, але повністю реалізувати його потенціал вдалося лише нульових. У 2014 році на сайті Artnet з'явився цілодобовий портал новин Artnet News для висвітлення знакових подій в арт-індустрії.

Основний напрямок діяльності платформи – послуги у сфері забезпечення користувачів інформацією про онлайн-аукціони. Починаючи з 1985 року, Artnet зібрав інформацію про більш ніж 10 млн результатів торгів від 1700 аукціонних будинків. Крім сервісу збору інформації про продаж на сайті діє платформа, яка поєднує галереї, дилерів та колекціонерів з усього світу. У базі даних сайту знаходяться роботи 35 тисяч художників та скульпторів.

Основними функціями системи є одна з найбільших баз даних аукціонних продажів, що дозволяє розуміти ринкові ціни, ще одна відмінність від попередніх, це те що є онлайн-аукціони. Розділ Artnet News містить аналітичні матеріали, новини про сучасне мистецтво та тенденції на ринку, а також є професійні звіти з оцінкою вартості.

Платформа Artnet орієнтована на професійних колекціонерів, інвесторів, галереї та арт-дилерів, які хочуть мати доступ до глибокої аналітики та точної інформації про ринок мистецтва. Це система для досвідчених користувачів, які потребують комплексних інструментів для аналізу.

Перевагами являється потужна база даних - це те чим дуже виділяється дана платформа, також Artnet News надає актуальну інформацію про сучасне мистецтво, що робить платформу не тільки торговим інструментом, але й інформаційним ресурсом. Artnet є міжнародною платформою, що працює з покупцями та продавцями з усього світу.

Головними недоліками є висока вартість підписки та послуг на Artnet, що робить платформу менш доступною для новачків, а також велику кількість функцій може бути перепорою для звичайних користувачів[4].

1.2. Методи візуалізації аналітичних даних

Інформація оточує нас усюди: повідомлення в месенджерах, пости в соціальних мережах, відеоролики на YouTube, фотографії на телефоні, музика по дорозі на роботу, цінники в магазинах і т. д. Швидкість обміну і доступ до інформації значно змінилися. Наукові відкриття, технологічні прориви в різних

галузях, підвищення рівня освіченості населення – все це позитивно позначається на розвитку суспільства в цілому.

Людський мозок дуже пластичний, і він добре засвоює і обробляє інформацію, навіть якщо її багато. Проте обсяг інформації, що надходить, в кілька разів перевищує можливості її засвоєння. Через це багато людей перебувають у стані так званого «інформаційного перевантаження». Виникають труднощі розуміння проблеми і прийняття рішень, причиною яких є надлишок інформації.

Дослідження показують, що упродовж дня людина забуває близько 80 % всього побаченого і почутого. Це дуже великий обсяг непотрібної інформації, яку герой сера Артура Конан Дойла Шерлок Холмс дуже лаконічно назвав «мотлохом». Однак існують різні техніки і методики кращого засвоєння та запам'ятовування інформації. Одним із способів ефективного навчання якраз є візуалізація. Це зображення числової і текстової інформації у вигляді графіків, діаграм, структурних схем, таблиць, малюнків, карт і т. д.

Численні дослідження підтверджують, те що 90 % інформації людина сприймає через зір, 70 % сенсорних рецепторів знаходяться в очах, близько половини нейронів головного мозку людини задіяні в обробці візуальної інформації, на 17 % вище продуктивність людини, що працює з візуальною інформацією, на 4,5 % краще згадуються деталі візуальної інформації, в 60 000 разів швидше сприймається візуальна інформація порівняно з текстовою, людина запам'ятовує 10 % із почутого, 20 % – з побаченого і 80 % – із побаченого і зробленого.

Сьогодні, у добу комп'ютерних технологій і величезної кількості інформації, без перетворення цифрових даних у зображення, яке буде зрозуміле людині, вже не обійтися. 60 % успіху візуалізації залежить від вибору типу графіка, 30 % – від його правильного використання та 10 % – від його правильного оформлення [5].

Задачі візуалізації дозволяють охоплювати великі обсяги інформації, наче стискаючи її, роблячи компактною. Так само, вона дозволяє зробити доступнішим сприйняття складної інформації, пошвидшуючи порівняння величин і полегшуючи виявлення паттернів в даних.

Важливою рисою візуалізованих даних є їхня переконливість, тому дуже важливо уникати викривлення інформації в процесі візуалізації. Зокрема, в хорошій візуалізації наочність загальної картини не заважає сприйняттю деталей.

Головне, що вирізняє візуалізацію як інфопродукт від використання візуалізації як техніки аналізу — чітке повідомлення. Якщо в даних нема нічого такого, що хотілося би в них показати сприймачеві, жодна візуалізація не зможе зробити їх цікавими.

Надалі потрібно визначитись з технікою візуалізації, ми розглянемо можливо й найпростіші, але й найвикористовуваніші [6].

Гістограма показує розподіл значень в множині об'єктів. На одній осі, як правило, горизонтальній, ми відкладаємо значення, а на вертикальній — показуємо, скільком об'єктам у множині воно відповідає. Оскільки кількість об'єктів — натуральне число, гістограми завжди будують від нуля (рис. 1.1).

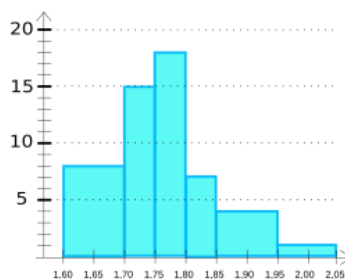


Рис. 1.1. Гістограма

Стовпчасті діаграми: у звичайній стовпчастій діаграмі, призначеній для визначення кількох значень, значуща лише одна вісь, а вздовж іншої розташовано стовпчики зручної для сприйняття ширини, висота яких кодує значення. Стовпчасті діаграми часто будують не від нуля. Складнішим різновидом стовпчастої діаграми є Стовпчаста діаграма зі змінною шириною

стовпчиків, де ширина стовпчиків не є фіксованою, а кодує другу змінну. Проблемою такої діаграми можна назвати те, що якщо співвідношення за одною змінною оцінювані легко, то другу ми сприймаємо опосередковано площею стовпчиків. Відповідно, якщо обидві змінних дають спільний ефект, цей тип діаграми годиться, а якщо змінні і їхній вплив незалежні, то ні (рис. 1.2).

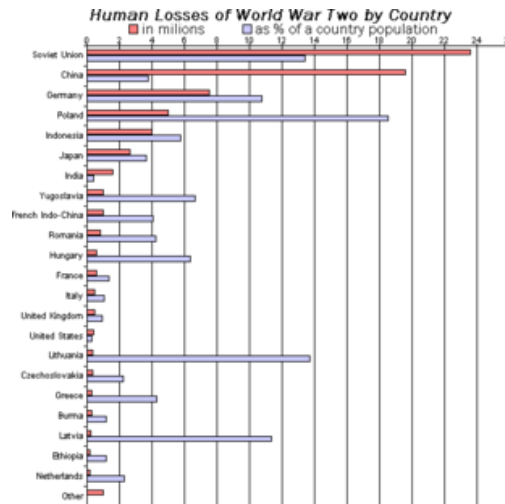


Рис. 1.2. Стовпчаста діаграма

Лінійні діаграми, діаграми розсіювання і бульбашкові діаграми: знайома з шкільного курсу алгебри декартова система координат, є одним із найпотужніших інструментів візуалізації даних. Просте розміщення точок даних в координатній сітці дає діаграму розсіювання, яка, будучи зручним інструментом пошуку кореляції під час аналізу, може бути адекватним засобом візуалізації даних, в яких треба показати як тенденцію, так і типові відхилення від неї (рис.1.3).

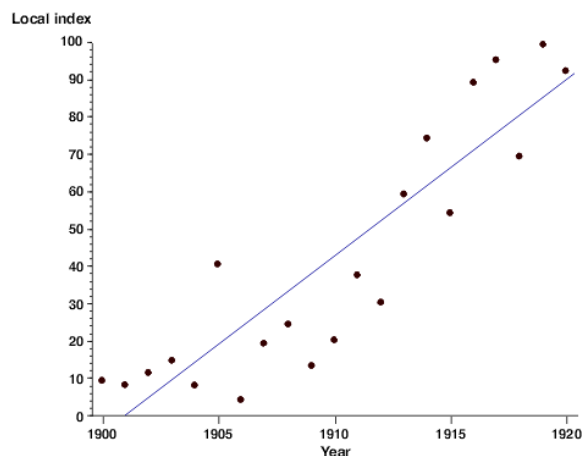


Рис. 1.3. Лінійна діаграма та діаграма розсіювання

Колові діаграми: аналогічно до поділу на частини стовпчиків в стосовій діаграмі, кола можна ділити на сектори, щоби показати частки у цілому. Така «тортова» діаграма добре передає співвідношення кількох значних часток, але стає неефективною, якщо часток дуже багато. Тоді варто об'єднати дрібні частки в групу «інші» (рис. 1.4).



Рис. 1.4. Колова діаграма

1.3. Постановка задачі

Головною метою кваліфікаційної роботи є розробка вебсайту для продажу та покупки творів мистецтва. Розроблювана система повинна ввібрати у себе, найкращі практики створення сайтів та оминати недоліки вже існуючих систем.

Сервіс дозволить реєструватись та авторизуватись на сайті, художникам-завантажувати свої роботи та виставляти ціну на їхні твори, покупцям-купувати картини, спілкуватись, підписуватись відмічати вподобайками та спілкуватися в чаті, а також користуватися системою аналітики.

Функції додатку повинні поступово розширюватися разом з етапами розробки. Серед основних можливостей можна виділити такі: можливість створення профілю, можливість здійснювати покупки, можливість налаштування профілю, можливість застосовування фільтрів (стиль, предмет картини, ціна), можливість спілкуватись в чаті, можливість перегляду майбутніх подій, перегляд статистики продажу картин за період та можливість підписки на інших користувачів.

Основні етапи розробки проекту включають в себе налаштування середовища, проектування та створення бази даних, проектування та розробка

ядра системи, в якому зосереджена бізнес логіка, розробка модулю аналітики, розробка інтерфесу сайту, створення процесу забезпечення безпеки доступу за допомогою автентифікації та авторизації, демонстрація проекту та його здача.

1.4. Вибір архітектури системи

Для розробки даної системи можна використати наступні шаблони:

- Model – View – Controller (модель – представлення – контролер);
- Model – View – Presenter (модель – представлення – пред'явник);
- Service – oriented architecture (сервісно – орієнтована архітектура);
- Model – View – View Model (модель – представлення – модель представлення);
- Microservices (мікросервісна).

Оптимальним для реалізації вебсайту буде використання архітектурного шаблону MVC (рис. 1.5).

MVC – це патерн проектування вебдодатків, який включає кілька дрібніших шаблонів. При використанні MVC на три окремі компоненти розділені модель даних програми, інтерфейс користувача і логіка взаємодії користувача з системою, завдяки чому модифікація одного з цих компонентів надає мінімальний вплив на інші або не має його зовсім [7].

Основна мета застосування MVC полягає у розподілі даних та бізнес-логіки від візуалізації. За рахунок такого поділу підвищується можливість повторного використання програмного коду: наприклад, додати уявлення даних будь-якого існуючого маршруту не тільки у вигляді HTML, а й у форматах JSON, XML, PDF, XLSX стає дуже просто і не потребує змін шару бізнес-логіки вихідного маршруту. Також спрощується і супровід програмного коду: внесення змін до зовнішнього вигляду, наприклад, не відбивається на бізнес-логіці, а зміни бізнес-логіки не торкаються візуалізації [8].

MVC Architecture Pattern

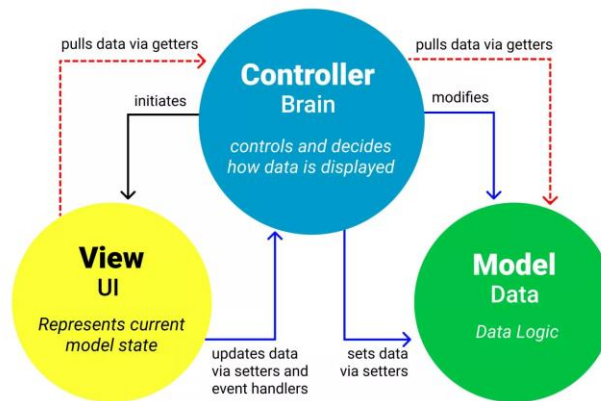


Рис. 1.5. MVC архітектура

Концепція MVC поділяє дані, представлення та обробку дій користувача на компоненти:

модель / Model – є об'єктною моделлю певної предметної області, включає дані та методи роботи з цими даними, реагує на запити з контролера, повертаючи дані та/або змінюючи свій стан. При цьому модель не містить у собі інформації про способи візуалізації даних або формати їх подання, а також не взаємодіє з користувачем безпосередньо;

подання / View – відповідає за відображення інформації (візуалізацію). Одні й самі дані можуть представлятися різними способами й у різних форматах. Наприклад, колекцію об'єктів за допомогою різних уявлень можна представити на рівні інтерфейсу користувача як в табличному вигляді, так і списком; на рівні API можна експортувати дані як у JSON, так і в XML або XSLX;

контролер / Controller – забезпечує зв'язок між користувачем та системою, використовує модель та уявлення для реалізації необхідної реакції на дії користувача. Як правило, на рівні контролера здійснюється фільтрація отриманих даних та авторизація – перевіряються права користувача на виконання дій або отримання інформації.

Основними перевагами використання MVC архітектури є: просте обслуговування коду, який легко розширювати та розвивати, розробку різних компонентів можна виконувати паралельно, уникнення складності в розробці

ПЗ, розділивши програму на три блоки (модель, подання та контролер), всі класи та об'єкти незалежні один від одного, тому їх можна тестувати окремо.

Таким чином, можна зробити висновок, що даний архітектурний шаблон забезпечує всі технологічні можливості для побудови системи, що дозволяє реалізувати поставлені завдання.

1.5. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Для реалізації веборієнтованої платформи обрано мову програмування C#, а саме технологію ASP.NET. Для розробки бази даних SQLite.

Мова C# – це багатопарадигмова об'єктно-орієнтована та компонентно-орієнтована мова програмування зі строгою типизацією, розроблена для платформи .NET Framework. Розроблена командою Microsoft Research під керівництвом Андерса Гейлсберга [9].

Синтаксис мови близький до мов C++ та Java. Деякі риси (наприклад, строга статична типизація), наближують її структуру до Delphi (Object Pascal).

Основні переваги мови C# включають.

Основними перевагами мови C# можна вважати підтримку об'єктно-орієнтованого програмування, що робить код легким для розширення та підтримки. Суворі типізація та типова система захищають від багатьох помилок. Завдяки .NET Core, C# підтримує розробку для Windows, macOS, Linux. Багате стандартне середовище має велику бібліотеку класів для роботи з базами даних, мережею та файловою системою. Сильна підтримка інструментів Visual Studio та інші інструменти забезпечують відлагодження, автодоповнення та інші можливості для зручного кодування.

Фреймворк ASP.NET, використовується для розробки вебдодатків. Сильні сторони ASP.NET це – висока продуктивність, а саме платформа використовує оптимізовані механізми кешування і збірки, що підвищує швидкодію додатків. Завдяки .NET Core можна створювати додатки для Windows, macOS, Linux.

Вбудована підтримка в Visual Studio забезпечує простоту написання та тестування коду. ASP.NET має потужні інструменти безпеки (аутентифікація, авторизація), що робить її надійним вибором для корпоративних додатків, а також легко обробляє великий обсяг трафіку, що робить її підходящою для великих проєктів.

Одним із основних інструментів при розробці системи та моделей класифікації є середовище розробки. На сьогоднішній день на ринку інформаційних технологій існує велика кількість різних засобів розробки додатків, у кожного з яких є свої переваги. Тому для вибору найкращого варіанту для розробки вебдодатку та модулю інтелектуального аналізу варто виділити наступні вимоги до інструментального засобу: робота з проєктами, пошук по файлах проєкту, підтримка Git, автоматичне доповнення коду, автоформатування та підсвічування синтаксису, приємний інтерфейс, підтримка плагінів, швидка робота, доступність [10].

Серед тих середовищ розробки, що задовольняють дані потреби C# розробників найбільшою популярністю користується: Visual Studio. Для роботи над розроблюваною системою було обрано редактор VS.

Вибір даного засобу обумовлений тим, що редактор має приємний інтерфейс, який водночас стислий і зрозумілий. У Visual Studio добре реалізована робота з гітом (робота зводиться до написання тексту комітів і натискання кнопки синхронізації). Підтримка розширень, є можливість додавання функцій під специфічні потреби розробника. Visual Studio розповсюджується безкоштовно і доступний у версіях для платформ Windows, Linux і OS X. Потужні інструменти налагодження включають функції відлагодження, тестування і моніторингу, що допомагають швидко знаходити і виправляти помилки.

В якості підходу реалізації системи управління базами даних обрано SQLite. SQLite дуже компактна і легка у використанні, не вимагає великих ресурсів системи і може працювати на практично всіх платформах, включаючи

мобільні пристрої та вбудовані системи. Висока продуктивність системи забезпечується за рахунок спрощення деяких стандартів, що використовуються в ній. SQLite інтегрується безпосередньо в додаток. Файл бази даних SQLite зберігається на диску, і додаток працює з ним безпосередньо. Низькі накладні витрати на адміністрування. Оскільки SQLite вбудовується в застосунок і не потребує окремого сервера, адміністрування бази даних зводиться до мінімуму. SQLite підтримує транзакції, що гарантує цілісність даних і забезпечує атомарність операцій. SQLite-база даних може бути перенесена з однієї платформи на іншу без зміни коду програми [11].

Важливим інструментальним засобом при розробці вебдодатку є також інструмент для розробки бази даних. В таких цілях найкраще використовувати офіційну систему для розробки та адміністрування – DB Browser for SQLite.

DB Browser for SQLite (DB4S) – це високоякісний візуальний інструмент з відкритим вихідним кодом для створення, проектування та редагування файлів бази даних, сумісних із SQLite. DB4S призначений для користувачів і розробників, які хочуть створювати, шукати та редагувати бази даних. DB4S використовує знайомий інтерфейс, подібний до електронних таблиць, і не потрібно вивчати складні команди SQL [12]. DB Browser for SQLite забезпечує усі необхідні для роботи можливості, а саме перегляд, додавання та видалення бази даних і таблиць, редагування даних у таблиці у візуальному режимі, а також дозволяє наочно представити модель бази даних в графічному вигляді.

Висновки до першого розділу

Проведений аналіз предметної області та поглиблене вивчення тематики дозволило визначити основні задачі, що необхідні для реалізації вебплатформи.

Огляд аналогів показав, що головними функціями майбутнього вебсайту має бути: можливість створення та редагування профіля продавця і покупця, можливість купівлі та продажу творів мистецтва, можливість спілкування

покупця та продавця через вбудований чат, отримання аналітики по даних системи.

Також виділено основні риси системи: простота у використанні (мінімалістичний і зручний інтерфейс), наявність статистики публікацій (купівлі, продажів).

Було визначено оптимальну структуру та MVC архітектуру додатку, що представлені у першому підрозділі, у тому числі на прикладі діаграми компонентів системи.

Для реалізації веборієнтованої інформаційно-аналітичної системи в даному розділі було проаналізовано та вирішено використовувати наступні засоби.

Мову програмування C# (для реалізації блогу). Фреймворк ASP.NET. Середовище розробки Visual Studio. Базу даних SQLite. Графічну оболонку для роботи з базою даних – DB Browser for SQLite.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНО-АНАЛІТИЧНОЇ СИСТЕМИ

2.1. Визначення варіантів використання та структури системи.

Система створюється для реалізації платформи загального користування по вузькому напрямку “мистецтво”, що надаватиме функціонал соціальної мережі з можливістю реєстрації, редагування профілю, можливістю публікацій робіт художників, їх продажу та купівлі, підписки на інших користувачів та спілкування між ними, а також аналізу статистичних даних у зручній та зрозумілій формі.

Основними бізнес вимогами є створення проекту з метою залучення художників (продавців), колекціонерів, поціновувачів мистецтва та звичайних користувачів (можливих покупців), до можливості заробітку, грошових вкладень, культурного збагачення та взагальному спілкування, як в соцмережі.

Можливості включають в себе надання аналізу купівлі та продажу, кількості лайків і статистичних даних системи.

Представлення проекту реалізовано у вигляді веборієнтованого додатку, що має зручний інтерфейс та необхідну інформацію по даній тематиці.

Вимоги користувачів означені можливістю створення профілю, можливістю зміни інформації профілю, можливістю додавання, редагування, видалення, перегляду картин для публікуючої сторони, спілкування в чаті, можливістю купівлі та продажу картин, можливістю перегляду статистики та результатів аналітики профілю користувача (зокрема аналіз настроїв коментаторів публікацій).

Відповідно до детального аналізу вимог користувачів можна побудувати діаграму варіантів використання веборієнтованої системи (рис. 2.1).

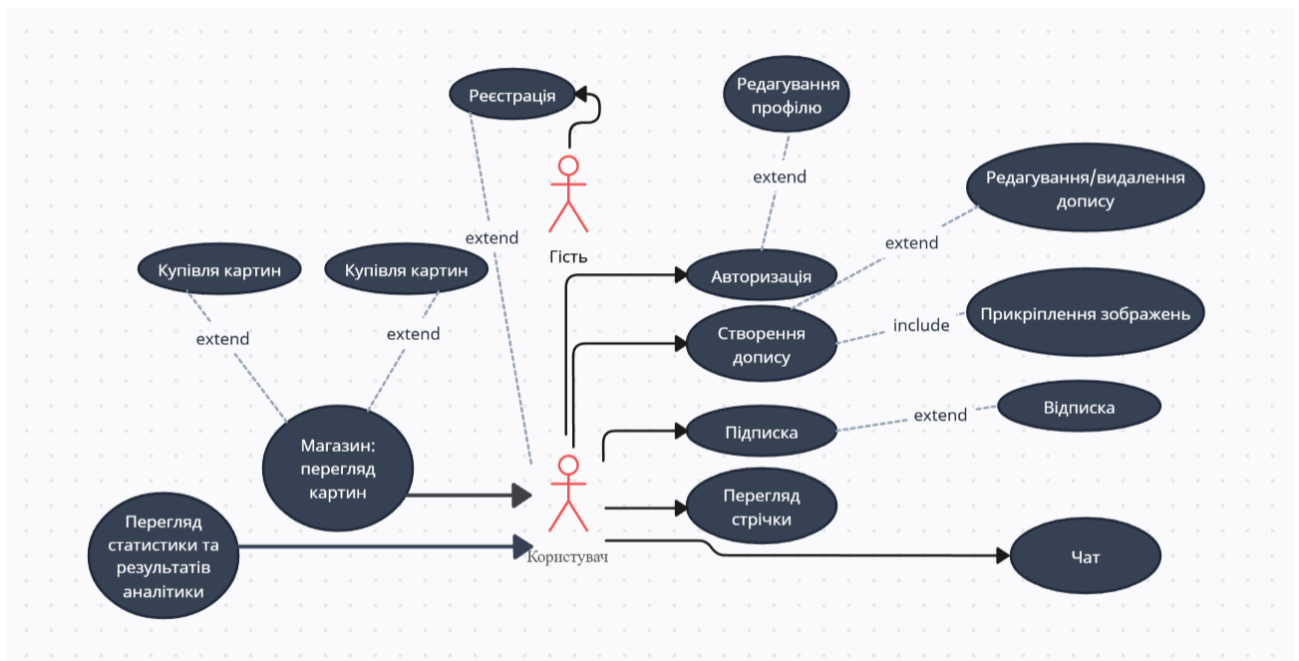


Рис. 2.1. Варіанти використання

Вхідними даними є продажі опублікованих картин художників, а також інформація стосовно них (вподобайки, коментарі, перегляди, категорії) заповнена користувачем. Організація вхідних і вихідних даних повинна відповідати інформаційній структурі виконуваних операцій. Введення даних повинно виконуватися з використанням відповідних форм, побудованих на основі візуальних компонентів.

Вихідними даними є стрічка публікацій, користувачів та дані статистики та аналітики.

Функціональними вимогами являється забезпечення всіх необхідних можливостей для загальних користувачів та надійне зберігання даних.

Не функціональні вимоги це – сприйняття, тобто час відповіді системи на запити не повинен перевищувати декілька секунд, програмний продукт повинен мати інтуїтивно-зрозумілий інтерфейс для забезпечення швидкої роботи користувача [13]. Надійність, а саме система повинна працювати без помилок та збоїв, а також система повинна підтримувати багатьох одночасно працюючих користувачів.

Для розробки та структуризації додатку були використані породжуючі та поведінкові патерни.

Породжуючі патерни (шаблони, що відповідають за зручне та безпечне створення нових об'єктів) [14].

Будівельник (builder) – це породжувальний патерн проектування, що дає змогу створювати складні об'єкти крок за кроком. Будівельник дає можливість використовувати один і той самий код будівництва для отримання різних відображень об'єктів.

Фабричний метод – це породжувальний патерн проектування, який визначає загальний інтерфейс для створення об'єктів у суперкласі, дозволяючи підкласам змінювати тип створюваних об'єктів.

Одинак – це породжувальний патерн проектування, який гарантує, що клас має лише один екземпляр, та надає глобальну точку доступу до нього.

Поведінкові патерни (патерни, що вирішують завдання ефективної та безпечної взаємодії між об'єктами програми).

Ланцюжок обов'язків або ж патерн Chain of Responsibility дозволяє передавати запит ланцюжком обробників, поки один з них не обробить запит. Незамінний при обробці запитів на сервері, де кожен обробник може передати запит наступному обробнику в ланцюжку: перевірка при авторизації на сайті, оброблення подій у GUI тощо [15].

Будівельник – об'єкт будується поступово через методи, що додають функціональність (наприклад, AddLogging, AddMvc), після чого викликається метод Build() для створення фінального продукту (WebApplication). Створення екземпляру додатку:

```
IConfigurationRoot configuration = new ConfigurationBuilder()
    .SetBasePath(AppDomain.CurrentDomain.BaseDirectory)
    .AddJsonFile("appsettings.json")
    .Build();

var builder = WebApplication.CreateBuilder(args);

// Services ...
builder.Services.AddLogging();
builder.Services.AddMvc();
```

Фабричний підхід для налаштування аутентифікації.

```
builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
    .AddCookie(options =>
    {
        options.LoginPath = "/Account/Login";
        options.LogoutPath = "/Account/Logout";
        options.ExpireTimeSpan = TimeSpan.FromMinutes(30);
        options.SlidingExpiration = true;
    });
```

Один екземпляр класу `HttpContextAccessor` створюється і використовується протягом усього життєвого циклу додатка.

```
builder.Services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();
```

Кожен компонент у ланцюжку обробляє запит або передає його далі.

Кожен елемент middleware може прийняти або модифікувати запит.

```
// Middlewares
app.UseRouting();
app.UseAuthentication();
app.UseAuthorization();
```

2.2. Розробка бази даних системи.

Невід’ємною частиною функціонування системи є обробка інформації, а саме створення, оновлення, видалення та вибірка даних. ASP.NET Core Identity надає платформу для керування обліковими записами користувачів та їх зберігання у програмах ASP.NET Core. Identity додається до проекту при виборі окремих облікових записів користувачів, як механізм автентифікації. За замовчуванням Identity використовує модель даних Entity Framework (EF) Core. Для реалізації цих задач було спроектовано базу даних (рис. 2.2).

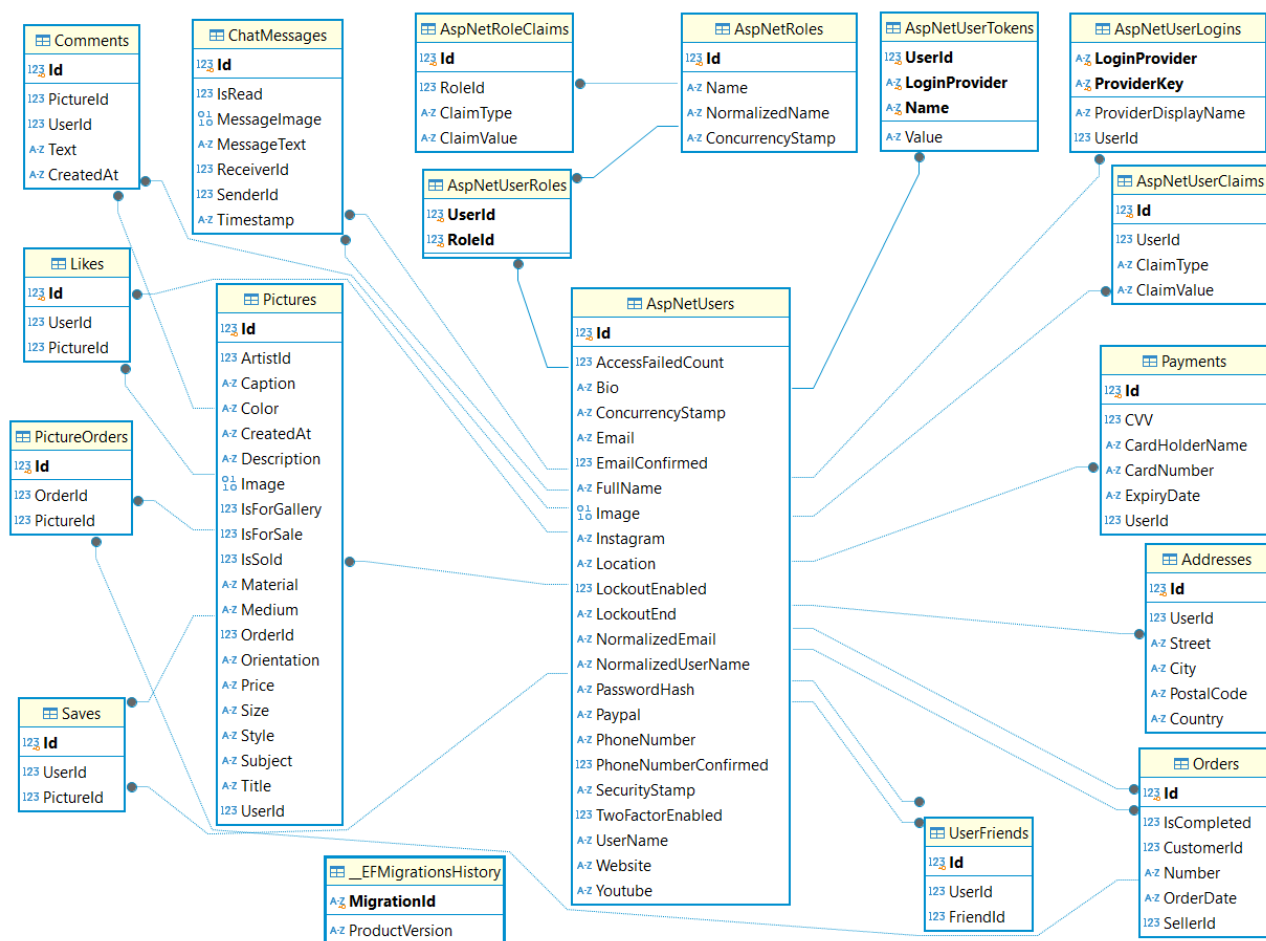


Рис. 2.2. Структура бази даних системи

База даних містить у собі наступні таблиці.

Шаблони таблиць ASP.NET: AspNetRoles, AspNetRoleClaims, AspNetUsers, AspNetUserClaims, AspNetUserLogins, AspNetUserRoles, AspNetUserTokens.

Головне призначення цих таблиць:

- Користувачі (AspNetUsers) – хто працює в системі.
- Ролі (AspNetRoles) – які права доступу вони мають.
- Клейми (AspNetUserClaims і AspNetRoleClaims) – додаткові атрибути та права.
- Зовнішня автентифікація (AspNetUserLogins) – інтеграція із зовнішніми сервісами.
- Токени (AspNetUserTokens) – механізм безпеки для автентифікації та роботи сесій.

Ці таблиці надають інфраструктуру для реалізації аутентифікації (підтвердження особи користувача) та авторизації (контроль доступу до ресурсів) у веб-додатку ASP.NET.

Власно створені таблиці: Addresses, Payments, Orders, PictureOrders, UserFriends, Likes, Comments, Pictures, ChatMessages, Saves.

Розглянемо детальніше структуру головних таблиць із зазначенням первинних та зовнішніх ключів, а також з описом призначення колонок та їх типу даних.

Таблиця «AspNetUsers» містить інформацію про користувачів системи (табл. 2.1).

Таблиця 2.1

Опис таблиці «AspNetUsers»

Назва колонки	Тип даних	NOT NULL	Призначення	Первинний ключ (PK)	Вторинний ключ (FK)
Id	INTEGER	Так	Унікальний ідентифікатор користувача	Так	Ні
AccessFailedCount	INTEGER	Так	Кількість невдалих спроб входу користувача	Ні	Ні
Bio	TEXT	Ні	Біографія користувача	Ні	Ні
ConcurrencyStamp	TEXT	Ні	Поле для захисту від одночасного редагування запису (відстеження версій)	Ні	Ні
Email	TEXT	Ні	Адреса електронної пошти користувача	Ні	Ні
EmailConfirmed	INTEGER	Так	Чи підтверджено електронну пошту (0 — ні, 1 — так)	Ні	Ні

FullName	TEXT	Так	Повне ім'я користувача	Hi	Hi
Image	BLOB	Hi	Фото користувача або аватар у вигляді бінарних даних	Hi	Hi
Instagram	TEXT	Hi	Лінк на Instagram профіль користувача	Hi	Hi
Location	TEXT	Hi	Місце розташування користувача	Hi	Hi
LockoutEnabled	INTEGER	Так	Чи активовано блокування облікового запису (0 — ні, 1 — так)	Hi	Hi
LockoutEnd	TEXT	Hi	Дата та час завершення блокування облікового запису	Hi	Hi
NormalizedEmail	TEXT	Hi	Унормована (для порівняння) версія email	Hi	Hi
NormalizedUserName	TEXT	Hi	Унормована (для порівняння) версія імені користувача	Hi	Hi
PasswordHash	TEXT	Hi	Хеш пароля користувача	Hi	Hi
Paypal	TEXT	Hi	Посилання на профіль користувача в PayPal	Hi	Hi
PhoneNumber	TEXT	Hi	Номер телефону користувача	Hi	Hi
PhoneNumberConfirmed	INTEGER	Так	Чи підтверджений номер	Hi	Hi

			телефону (0 — ні, 1 — так)		
SecurityStamp	TEXT	Ні	Токен для забезпечення додаткової безпеки	Ні	Ні
TwoFactorEnabled	INTEGER	Так	Чи ввімкнено двофакторну аутентифікацію (0 — ні, 1 — так)	Ні	Ні
UserName	TEXT	Ні	Логін користувача	Ні	Ні
Website	TEXT	Ні	Персональний вебсайт користувача	Ні	Ні

При використанні Entity Framework Core для створення бази даних, User буде використовувати таблицюAspNetUsers в базі даних, а властивості цього класу, такі як FullName, Paypal, Location та інші будуть додаватися як колонки в цій таблиці.

Програмна реалізація створення таблиці AspNetUsers:

```
public class User : IdentityUser<int>
{
    public string FullName { get; set; }
    public string? Paypal { get; set; }
    public string? Location { get; set; }
    public string? Website { get; set; }
    public string? Instagram { get; set; }
    public string? Youtube { get; set; }
    public string? Bio { get; set; }
    public List<Picture>? Pictures { get; set; }
    public List<Payment>? Payments { get; set; }
    public List<Address>? Addresses { get; set; }
    public List<Order>? Orders { get; set; }
    public List<Like>? Likes { get; set; }
    public List<Save>? Saves { get; set; }
    public List<Comment>? Comments { get; set; }
    public byte[]? Image { get; set; }
    public List<UserFriend>? UserFriends { get; set; }
    public virtual List<ChatMessage> ChatMessages { get; set; }
    public User()
    {
        Pictures = new List<Picture>();
        Payments = new List<Payment>();
        Addresses = new List<Address>();
        Orders = new List<Order>();
        UserFriends = new List<UserFriend>();
        Likes = new List<Like>();
        Saves = new List<Save>();
        Comments = new List<Comment>();
        ChatMessages = new List<ChatMessage>();
    }
}
```

Таблиця «Pictures» містить інформацію про картини (табл. 2.2).

Таблиця 2.2

Опис таблиці «Pictures»

Назва колонки	Тип даних	NOT NULL	Призначення	Первинний ключ (РК)	Вторинний ключ (FK)
Id	INTEGER	Так	Унікальний ідентифікатор зображення.	Так	Ні
ArtistId	INTEGER	Ні	Ідентифікатор художника, пов'язаний з таблицею художників (якщо існує).	Ні	Так
Caption	TEXT	Ні	Підпис до зображення.	Ні	Ні
Color	TEXT	Ні	Основний колір або кольори зображення.	Ні	Ні
CreatedAt	TEXT	Так	Дата створення зображення.	Ні	Ні
Description	TEXT	Ні	Опис зображення.	Ні	Ні
Image	BLOB	Ні	Бінарні дані зображення (наприклад, файл).	Ні	Ні
IsForGallery	INTEGER	Так	Чи призначене зображення для галереї (0 — ні, 1 — так).	Ні	Ні
IsForSale	INTEGER	Так	Чи доступне зображення для продажу (0 — ні, 1 — так).	Ні	Ні
IsSold	INTEGER	Так	Чи було продано зображення (0 — ні, 1 — так).	Ні	Ні
Material	TEXT	Ні	Матеріал, на якому створене зображення.	Ні	Ні
Medium	TEXT	Ні	Середовище або техніка, використана для створення.	Ні	Ні
OrderId	INTEGER	Ні	Ідентифікатор замовлення, пов'язаний з таблицею замовлень (якщо існує).	Ні	Так
Orientation	TEXT	Ні	Орієнтація зображення.	Ні	Ні
Price	TEXT	Ні	Ціна зображення (у вигляді тексту для забезпечення валюти).	Ні	Ні
Size	TEXT	Ні	Розмір зображення (наприклад, 40х60 см).	Ні	Ні
Style	TEXT	Ні	Художній стиль зображення.	Ні	Ні
Subject	TEXT	Ні	Тема або суб'єкт зображення.	Ні	Ні
Title	TEXT	Ні	Назва зображення.	Ні	Ні
UserId	INTEGER	Так	Ідентифікатор користувача, пов'язаний з таблицею AspNetUsers(Id) .	Ні	Так

Так як в системі присутні, як твори мистецтва для продажу, так і просто зображення для наповнення профілю, дана таблиця використовується для зберігання інформації про зображення, а саме інформація для організації продажу та купівлі, відгуків та взаємодії користувачів з цим контентом.

Програмна реалізація створення таблиці Pictures:

```
public class Picture
{
    public int Id { get; set; }

    [StringLength(100)]
    public string? Title { get; set; } = "No title";

    [StringLength(300)]
    public string? Description { get; set; } = "No description";

    [StringLength(300)]
    public string? Caption { get; set; }

    [StringLength(100)]
    public string? Style { get; set; } = "Other";

    [StringLength(100)]
    public string? Subject { get; set; } = "Other";

    [StringLength(100)]
    public string? Medium { get; set; } = "Other";

    [StringLength(100)]
    public string? Material { get; set; } = "Other";

    [Range(0.01, double.MaxValue, ErrorMessage = "Price must be a positive value.")]
    public decimal? Price { get; set; }
    public string? Size { get; set; } = "100x100 inches";

    [StringLength(20)]
    public string? Orientation { get; set; } = "Other";

    [StringLength(50)]

    public string? Color { get; set; } = "Other";
    public int UserId { get; set; }
    public virtual User User { get; set; }
    public byte[]? Image { get; set; }
    public int? OrderId { get; set; }
    public virtual List<PictureOrder> PictureOrders { get; set; } = new List<PictureOrder>();
    public virtual List<Like> Likes { get; set; } = new List<Like>();
    public virtual List<Save> Saves { get; set; } = new List<Save>();
    public virtual List<Comment> Comments { get; set; } = new List<Comment>();
    public bool IsForSale { get; set; } = false;
    public bool IsSold { get; set; } = false;
    public bool IsForGallery { get; set; } = false;
    public DateTime CreatedAt { get; set; } = DateTime.Now;
}
```

Таблиця «Payments» містить інформацію про платіжну систему (табл. 2.3).

Таблиця. 2.3

Опис таблиці «Payments»

Назва колонки	Тип даних	NOT NULL	Призначення	Первинний ключ (PK)	Вторинний ключ (FK)
Id	INTEGER	Так	Унікальний ідентифікатор платежу	Так	Ні
CVV	INTEGER	Так	Код безпеки платіжної картки	Ні	Ні
CardHolderName	TEXT	Так	Ім'я власника картки	Ні	Ні
CardNumber	TEXT	Так	Номер платіжної картки	Ні	Ні
ExpiryDate	TEXT	Так	Термін дії картки	Ні	Ні
UserId	INTEGER	Так	Ідентифікатор користувача, що здійснив платіж	Ні	Так (посилання на Id таблиціAspNetUsers)

Дана таблиця зберігає платіжну інформацію, тобто дані про власника карти, CVV – код, кінцеву дату дії картки, номер карти та посилання на таблицю AspNetUsers з ідентифікатором користувача, який здійснив платіж.

Програмна реалізація створення таблиці «Payments»:

```
[Table("Payments")]
public class Payment
{
    [Key]
    public int Id { get; set; }

    [Required]
    [CreditCard]
    public string CardNumber { get; set; }

    [Required]
    [ForeignKey("User")]
    public int UserId { get; set; }

    [Required]
    public string CardHolderName { get; set; }

    [Required]
    [DataType(DataType.Date)]
    public DateTime ExpiryDate { get; set; }

    [Required]
    [Range(100, 999, ErrorMessage = "CVV must be a 3-digit number.")]
    public int CVV { get; set; }
    public virtual User User { get; set; }
}
```

Таблиця «ChatMessages» містить інформацію про платіжну систему (табл. 2.4).

Таблиця. 2.4

Опис таблиці «ChatMessages»

Назва колонки	Тип даних	NOT NULL	Призначення	Первинний ключ (PK)	Вторинний ключ (FK)
Id	INTEGER	Так	Унікальний ідентифікатор повідомлення	Так	Ні
IsRead	INTEGER	Так	Чи було прочитано повідомлення (0 — ні, 1 — так)	Ні	Ні
MessageImage	BLOB	Ні	Зображення, прикріплене до повідомлення	Ні	Ні
MessageText	TEXT	Ні	Текст повідомлення	Ні	Ні
ReceiverId	INTEGER	Так	Ідентифікатор користувача, який отримав повідомлення	Ні	Так (посилання на Id таблиціAspNetUsers)
SenderId	INTEGER	Так	Ідентифікатор користувача, який надіслав повідомлення	Ні	Так (посилання на Id таблиціAspNetUsers)
Timestamp	TEXT	Так	Час відправлення повідомлення	Ні	Ні

В даній таблиці зберігається інформація про історію ведення обміну повідомленнями, ким і кому відправленні з полем для збереження часу, як текстових так і мультимедійних повідомлень.

Програмна реалізація створення таблиці «ChatMessages»:

```
public class ChatMessage
{
    public int Id { get; set; }

    public int SenderId { get; set; }
    public virtual User Sender { get; set; }

    public int ReceiverId { get; set; }
    public virtual User Receiver { get; set; }

    public string? MessageText { get; set; }
    public byte[]? MessageImage { get; set; }
    public DateTime Timestamp { get; set; }
    public bool IsRead { get; set; }
}
```

Таблиця «Orders» містить інформацію про статус платежу (табл. 2.5).

Таблиця. 2.5

Опис таблиці «Orders»

Назва колонки	Тип даних	NOT NULL	Призначення	Первинний ключ (PK)	Вторинний ключ (FK)
Id	INTEGER	Так	Унікальний ідентифікатор замовлення	Так	Ні
IsCompleted	INTEGER	Так	Статус виконання замовлення (0 — ні, 1 — так)	Ні	Ні
CustomerId	INTEGER	Так	Ідентифікатор користувача, який здійснив замовлення	Ні	Так (посилання на Id таблиціAspNetUsers)
Number	TEXT	Так	Номер замовлення	Ні	Ні
OrderDate	TEXT	Так	Дата створення замовлення	Ні	Ні
SellerId	INTEGER	Так	Ідентифікатор користувача, який продав товар	Ні	Так (посилання на Id таблиціAspNetUsers)

Дана таблицю зберігає інформацію про замовлення у системі. Вона забезпечує зв'язок між покупцем та продавцем і має інформацію про номер замовлення та дату замовлення, а також статус виконання оплати.

Програмна реалізація створення таблиці «Orders»:

```
[Table("Orders")]
public class Order
{
    [Key]
    public int Id { get; set; }

    [ForeignKey("Customer")]
    public int CustomerId { get; set; }

    [ForeignKey("Seller")]
    public int SellerId { get; set; }

    [Required]
    [MaxLength(50)]
    public string Number { get; set; }
    public bool IsCompleted { get; set; } = false;

    [Required]
    public DateTime OrderDate { get; set; }
    public virtual User Customer { get; set; }
    public virtual User Seller { get; set; }
    public virtual ICollection<PictureOrder> PictureOrders { get; set; } = new
    List<PictureOrder>();
}
```

2.3. Практична реалізація аналітичної складової системи

Проаналізувавши в попередньому підрозділі структуру даних системи, можемо перейти до реалізації аналітичної складової системи. В цілому аналітичні дані можна умовно розділити на три складових: загальна статистика, статистика покупок, статистика продажів. Для цього потрібно реалізувати ряд методів, які будуть графічно відображати стан основних показників активностей користувачів.

Розглянемо загальну статистику, метод який його описує називається `AllStatistics`, він збирає та обробляє дані за сімома напрямками: відношення продажів до покупок картин; ТОП 10 користувачів по вподобайкам; ТОП 10 користувачів за продажами; найбільш популярних картин по стилю; найбільш популярних картин по предмету зображення на картині; найбільш популярних картин за різновидом фарб; найбільш популярних картин за матеріалом на яких вона написана.

Розглянемо фрагмент програмного коду, який реалізує відношення продажів до покупок картин:

```
var currentUserId = User.FindFirstValue(ClaimTypes.NameIdentifier);
if (currentUserId == null)
{
    return RedirectToAction("Login", "Account");
}

var userId = int.Parse(currentUserId);
var expensesPerMonth = await _context.PictureOrders
    .Where(po => po.Order.CustomerId == userId && po.Order.IsCompleted &&
    po.Picture.IsSold) // Перевіряємо, що замовлення виконано, належить користувачу
    і картина продана
    .Include(po => po.Picture) // Включаємо Picture для доступу до ціни
    .Include(po => po.Order) // Включаємо Order для доступу до OrderDate
    .GroupBy(po => new { po.Order.OrderDate.Year, po.Order.OrderDate.Month }) //
    Групуємо за роком і місяцем дати замовлення
    .Select(g => new
    {
        Year = g.Key.Year,
        Month = g.Key.Month,
        TotalExpenses = g.Sum(po => (double)po.Picture.Price) // Сума вартості
        витрат користувача на картини у кожному місяці
    })
    .OrderBy(g => g.Year)
    .ThenBy(g => g.Month)
    .ToListAsync();

var salesPerMonth = await _context.PictureOrders
```

```

        .Where(po => po.Picture.UserId == userId && po.Picture.IsSold &&
        po.Order.IsCompleted) // Перевіряємо, що картина продана і замовлення виконано
        .Include(po => po.Picture) // Включаємо Picture для доступу до ціни
        .Include(po => po.Order) // Включаємо Order для доступу до OrderDate
        .GroupBy(po => new { po.Order.OrderDate.Year, po.Order.OrderDate.Month }) //
        Групуємо за роком і місяцем дати замовлення
        .Select(g => new
        {
            Year = g.Key.Year,
            Month = g.Key.Month,
            TotalAmount = g.Sum(po => (double)po.Picture.Price) // Сума вартості
            проданих картин у кожному місяці
        })
        .OrderBy(g => g.Year)
        .ThenBy(g => g.Month)
        .ToListAsync();

        // Передаємо результат у ViewBag
        ViewBag.SalesPerMonth = salesPerMonth;

```

Даний метод шукає користувача в базі даних, знаходить куплені та продані картини і передає інформацію через властивість ViewBag для візуалізації у вигляді кругової діаграми.

Метод ТОП 10 користувачів за вподобайками реалізований наступним чином:

```

        var topUsersByLikes = await _context.Likes
        .GroupBy(l => l.Picture.UserId) // Групуємо по UserId картин
        .Select(g => new
        {
            UserId = g.Key, // UserId власника картини
            UserName = g.FirstOrDefault().Picture.User.FullName, // Ім'я користувача
            LikesCount = g.Count() // Кількість лайків
        })
        .OrderByDescending(g => g.LikesCount) // Сортуємо по лайках
        .Take(10) // Топ 10
        .ToListAsync();

        // Передаємо результат у ViewBag
        ViewBag.TopUsersByLikes = topUsersByLikes;

```

Даний фрагмент коду відповідає за отримання інформації про вподобайки з бази даних, групування та сортування по користувачам.

Реалізація методів: ТОП 10 користувачів за продажами, найбільш популярних картин по стилю, найбільш популярних картин по предмету зображення на картині, найбільш популярних картин за різновидом фарб та найбільш популярних картин за матеріалом на яких вона написана, виконують аналогічні дії, пошук яких здійснюється за іншими критеріями.

Статистика покупок виконується на основі напрямку купівлі творів мистецтва по різних категоріях та складається з трьох функцій: загальна сума

купівлі картин авторизованим користувачем по місяцях, за період; загальна кількість куплених картин авторизованим користувачем по місяцях, за період; ТОП 10 користувачів, в яких авторизований користувач здійснював купівлю.

Даний фрагмент коду відповідає за зіставлення даних по місяцях, купівлі картин авторизованим користувачем, за обраний період та була виконана наступним чином:

```
var expensesPerMonth = await _context.PictureOrders
    .Where(po => po.Order.CustomerId == userId && po.Order.IsCompleted &&
        po.Picture.IsSold) // Перевіряємо, що замовлення виконано, належить користувачу
        і картина продана
    .Include(po => po.Picture) // Включаємо Picture для доступу до ціни
    .Include(po => po.Order) // Включаємо Order для доступу до OrderDate
    .GroupBy(po => new { po.Order.OrderDate.Year, po.Order.OrderDate.Month }) //
    Групуємо за роком і місяцем дати замовлення
    .Select(g => new
    {
        Year = g.Key.Year,
        Month = g.Key.Month,
        TotalExpenses = g.Sum(po => (double)po.Picture.Price) // Сума вартості
        витрат користувача на картини у кожному місяці
    })
    .OrderBy(g => g.Year)
    .ThenBy(g => g.Month)
    .ToListAsync();
```

Метод виконує обробку, загальної кількості куплених авторизованим користувачем картин та зображує у вигляді графіку по місяцях за період. Даний підхід важливий для аналізу витрат за встановлений відрізок часу.

Реалізація методу загальної кількості куплених авторизованим користувачем картин, виглядає схожим чином, де за вхідні дані, взято кількість куплених мистецьких творів.

```
var pictureCountPerMonth = await _context.PictureOrders
    .Where(po => po.Order.CustomerId == userId && po.Order.IsCompleted &&
        po.Picture.IsSold) // Check that the order is completed, belongs to the user,
        and the picture is sold
    .Include(po => po.Picture) // Include Picture for access to its data
    .Include(po => po.Order) // Include Order for access to OrderDate
    .GroupBy(po => new { po.Order.OrderDate.Year, po.Order.OrderDate.Month }) //
    Group by year and month of the order date
    .Select(g => new
    {
        Year = g.Key.Year,
        Month = g.Key.Month,
        PictureCount = g.Count() // Count the number of pictures in each month
    })
    .OrderBy(g => g.Year)
    .ThenBy(g => g.Month)
    .ToListAsync();

ViewBag.PictureCountPerMonth = pictureCountPerMonth;
```

Метод ТОП 10 користувачів, в яких авторизований користувач здійснював купівлю, відображає найбільш популярних художників за кількістю проданих робіт, для даної людини. Програмна реалізація даного методу:

```
var topUsersByPurchased = await _context.PictureOrders
    .Where(po => po.Order.CustomerId == userId && po.Picture.IsSold &&
        po.Order.IsCompleted) // Перевіряємо, що картина продана і замовлення виконано
    .GroupBy(po => po.Picture.UserId) // Групуємо по UserId картин
    .Select(g => new
    {
        UserId = g.Key, // UserId власника картини
        UserName = g.FirstOrDefault().Picture.User.FullName, // Ім'я користувача
        PurchaseCount = g.Count() // Кількість проданих картин
    })
    .OrderByDescending(g => g.PurchaseCount) // Сортуюмо по кількості проданих картин
    .Take(10) // Топ 10
    .ToListAsync();

// Передаємо результат у ViewBag
ViewBag.TopUsersByPurchased = topUsersByPurchased;
```

Статистика продажів виконується на основі напрямку продажів творів мистецтва авторизованим користувачем по різних категоріях та складається з шести функцій: продажі робіт в грошовому еквіваленті, продаж картин по кількості штук, кількість проданих картин за стилем, кількість продажів по предмету зображення на картині, кількість продажів картин за різновидом фарб та матеріалом на яких вона написана.

Розглянемо метод куплених картин певного користувача. Він є одним з найважливіших даних для аналізу, щоб приймати економічно вигідні дії в майбутньому:

```
var userId = int.Parse(currentUserId);
var picturesPerMonth = await _context.PictureOrders
    .Where(po => po.Picture.UserId == userId && po.Picture.IsSold &&
        po.Order.IsCompleted) // Перевіряємо, що картина продана і замовлення виконано
    .Include(po => po.Order) // Включаємо Order для доступу до OrderDate
    .GroupBy(po => new { po.Order.OrderDate.Year, po.Order.OrderDate.Month }) //
    .Select(g => new
    {
        Year = g.Key.Year,
        Month = g.Key.Month,
        Count = g.Count() // Підраховуємо кількість картин, проданих у кожному
        місяці
    })
    .OrderBy(g => g.Year)
    .ThenBy(g => g.Month)
    .ToListAsync();

var salesPerMonth = await _context.PictureOrders
```

```

.Where(po => po.Picture.UserId == userId && po.Picture.IsSold &&
po.Order.IsCompleted) // Перевіряємо, що картина продана і замовлення виконано
.Include(po => po.Picture) // Включаємо Picture для доступу до ціни
.Include(po => po.Order) // Включаємо Order для доступу до OrderDate
.GroupBy(po => new { po.Order.OrderDate.Year, po.Order.OrderDate.Month }) //
Групуємо за роком і місяцем дати замовлення
.Select(g => new
{
    Year = g.Key.Year,
    Month = g.Key.Month,
    TotalAmount = g.Sum(po => (double)po.Picture.Price) // Сума вартості
    проданих картин у кожному місяці
})
.OrderBy(g => g.Year)
.ThenBy(g => g.Month)
.ToListAsync();

// Передаємо результат у ViewBag
ViewBag.SalesPerMonth = salesPerMonth;

```

Даний фрагмент програмного коду відображає рівень доходу художника. Це виконано для візуального відображення рівня успішності, в грошовому еквіваленті.

Проаналізувавши всю вищезгадану інформацію ми можемо бачити в якому стилі чим та на чому намальовані картини користуються найбільшим попитом, та як найефективніше розвивати продажі на своїй сторінці.

2.4. Програмна реалізація основних алгоритмів системи

Зважаючи на базову структуру MVC і поставлені для розробки задачі, сформовано файлову структуру проекту (рис. 2.3).

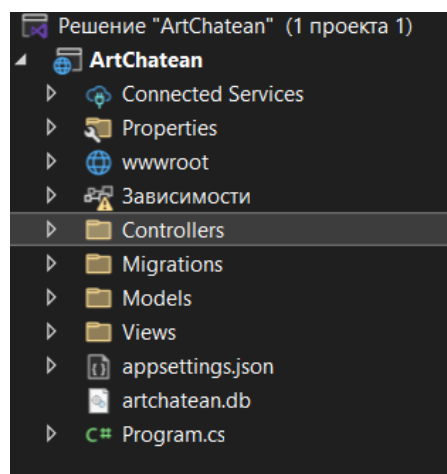


Рис. 2.3. Файлова структура проекту

Файл Program.cs це точка в ходу в додаток:

```
IConfigurationRoot configuration = new ConfigurationBuilder()
    .SetBasePath(AppDomain.CurrentDomain.BaseDirectory)
    .AddJsonFile("appsettings.json")
    .Build();

var builder = WebApplication.CreateBuilder(args);

// Services ...
builder.Services.AddLogging();
builder.Services.AddMvc();
builder.Services.AddControllersWithViews();
builder.Services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();
builder.Services.AddDbContext<ArtDbContext>(options =>
{
    options.UseSqlite(configuration.GetConnectionString("Default"));
});
builder.Services.AddIdentity<User, IdentityRole<int>>(options =>
{
    options.SignIn.RequireConfirmedAccount = false;
    options.SignIn.RequireConfirmedEmail = false;
    options.SignIn.RequireConfirmedPhoneNumber = false;
    options.Password.RequiredLength = 3;
    options.Password.RequiredUniqueChars = 0;
    options.Password.RequireDigit = false;
    options.Password.RequireNonAlphanumeric = false;
    options.Password.RequireLowercase = false;
    options.Password.RequireUppercase = false;
})
.AddRoles<IdentityRole<int>>()
.AddEntityFrameworkStores<ArtDbContext>()
.AddDefaultTokenProviders();

builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
    .AddCookie(options =>
    {
        options.LoginPath = "/Account/Login";
        options.LogoutPath = "/Account/Logout";
        options.ExpireTimeSpan = TimeSpan.FromMinutes(30);
        options.SlidingExpiration = true;
    });

builder.Services.AddSignalR(options =>
{
    options.MaximumReceiveMessageSize = 1024 * 1024 * 5; // 5 MB,
});

//
builder.Services.AddTransient<IEmailServiceMailKit, EmailServiceMailKit>();

var app = builder.Build();

// Middlewares
app.UseRouting();
app.UseAuthentication();
app.UseAuthorization();
app.MapControllers();
app.UseStaticFiles();

app.MapHub<ChatHub>("/chathub");
app.MapControllerRoute(
```

```

        name: "default",
        pattern: "{controller=Main}/{action=Main}/{id?}"
    };

app.UseStaticFiles();

app.Run();

```

Файл appsettings.json, містить налаштування ASP.NET Core.

```

"Logging": {
  "LogLevel": {
    "Default": "Information",
    "Microsoft.AspNetCore": "Warning"
  }
}

```

Даний код налаштовує логування. Це дозволяє зменшити шум у логах, обмежуючи записування менш важливих повідомлень від системних компонентів.

```
"AllowedHosts": "*",
```

Даний параметр відповідає за дозволи хостам для обробки запитів, в даному випадку це дозволено всім.

```

"ConnectionStrings": {
  "Default": "Data Source=artchatean.db"
},

```

Відповідає за підключення до бази даних SQLite.

```

"EmailSettings": {
  "SmtpServer": "smtp.gmail.com",
  "Port": 587,
  "EnableSsl": true,
  "Username": "bodya8294094@gmail.com",
  "Password": "k c k y o v w i s r h x q y l a",
  "From": "bodya8294094@gmail.com"
}

```

Даний розділ відповідає за налаштування SMTP-сервера. В даному випадку використовується пошта gmail.

В кореневій папці проекту також зберігаються базові класи Model, View, Controller. Розглянемо декілька основних класів. Перший з них це AccountController, він відповідає за аунтефікацію, авторизацію, взаємодію з базою даних та роботи з ролями. Метод [HttpPost] Login перевіряє чи користувач зареєстрований, у разі невдачі рандомно змінює картинку. Перевіряє чи e-mail користувача був підтверджений, якщо ні то просить зайти на пошту та перейти

за посиланням. Також перевіряє чи співпадає логін та пароль, призначає роль Admin, якщо це не було виконано раніше та перенаправляє на головну сторінку сайту.

```
[HttpPost]
public async Task<IActionResult> Login([FromForm] LoginForm form)
{
    // Генерація випадкової картинки
    var count = _context.Pictures.Count();
    var randomIndex = new Random().Next(0, count);
    var randomPicture = _context.Pictures.Skip(randomIndex).FirstOrDefault();
    ViewBag.RandomPicture = randomPicture;

    // Знаходження користувача за email
    var user = await _userManager.FindByEmailAsync(form.Email);
    if (user == null)
    {
        ModelState.AddModelError(nameof(form.Email), "User not found!");
        return View(form);
    }
    // Перевірка, чи підтверджено електронну пошту
    if (!await _userManager.IsEmailConfirmedAsync(user))
    {
        ModelState.AddModelError(nameof(form.Email), "Please confirm your email
before logging in!");
        return View(form);
    }

    // Перевірка пароля і логін
    var result = await _signInManager.PasswordSignInAsync(user.UserName,
form.Password, false, false);
    if (!result.Succeeded)
    {
        ModelState.AddModelError(nameof(form.Email), "Invalid password!");
        return View(form);
    }

    if (form.RememberMe)
    {
        var authProperties = new AuthenticationProperties
        {
            IsPersistent = true,
            ExpiresUtc = DateTimeOffset.UtcNow.AddDays(30)
        };
        await _signInManager.SignInAsync(user, authProperties);
    }
    else
    {
        await _signInManager.SignInAsync(user, isPersistent: false);
    }

    // Створення ролі Admin, якщо вона не існує
    var adminRole = await _roleManager.FindByNameAsync("Admin");
    if (adminRole == null)
    {
        await _roleManager.CreateAsync(new IdentityRole<int>() { Name = "Admin"
});
    }

    // Додавання користувача до ролі Admin, якщо він ще не є в цій ролі
    if (!await _userManager.IsInRoleAsync(user, "Admin"))
```

```

    {
        await _userManager.AddToRoleAsync(user, "Admin");
    }

    // Перенаправлення на головну сторінку
    return Redirect("/Main/Main");
}

```

Наступний клас – це PaymentController, він відповідає за платіжну систему, в нашому випадку це LiqPay. Дане API дозволяє розробникам безкоштовно тестувати та в майбутньому користуватися їхньою платіжною системою. Реалізація даного класу є важливою складовою системи тому що наближає умови використання до реальних.

```

public class PaymentController : Controller
{
    private readonly ArtDbContext _context;
    private readonly string _publicKey = "";
    private readonly string _privateKey = "";

    public PaymentController(ArtDbContext context)
    {
        _context = context;
    }
    [HttpPost]
    public async Task<IActionResult> CheckPayment([FromBody] int orderId)
    {
        var liqpay = new LiqPayClient(_publicKey, _privateKey);

        // Підготовка параметрів запиту
        var requestParams = new LiqPayRequest
        {
            Version = 3, //Версия Api
            Action = LiqPay.SDK.Dto.Enums.LiqPayRequestAction.Status,
            OrderId = orderId.ToString(),
        };

        // Викликаємо асинхронний метод для отримання статусу платежу
        var response = await liqpay.RequestAsync("request", requestParams);
        Console.WriteLine(response.Status);
        if (response != null)
        {
            // Повертаємо статус як JSON
            return Json(new { status = response.Status.ToString() });
        }
        else
        {
            // Повертаємо повідомлення про помилку
            return Json(new { status = "No response from LiqPay." });
        }
    }
}

```

Класу ChatController, який був розроблений для спілкування в нашій системі. Це важлива складова проекту. Крім можливості спілкування, в чат

також приходять повідомлення про купівлю та продаж картин. Він реалізований наступним чином:

```
private readonly ArtDbContext _context;
private readonly IHubContext<ChatHub> _hubContext;

public ChatController(ArtDbContext context, IHubContext<ChatHub> hubContext)
{
    _context = context;
    _hubContext = hubContext;
}

// Метод для відображення чату між користувачами
[HttpGet]
public async Task<ActionResult> MainChat(int? receiverId)
{
    var userIdstr = User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (userIdstr == null)
    {
        return RedirectToAction("Login", "Account");
    }

    var userId = int.Parse(userIdstr);

    // БЛОК ЯКИЙ ФІЛЬТРУЄ І ГРУПУЄ КОРИСТУВАЧІВ ДЛЯ ОТРИМАННЯ НЕОБХДНИХ ДАННИХ

    // Отримуємо ID друзів з останніми повідомленнями
    var friendIdsWithLastMessage = await _context.ChatMessages
        .Where(m => m.SenderId == userId || m.ReceiverId == userId)
        .GroupBy(m => m.SenderId == userId ? m.ReceiverId : m.SenderId)
        .Select(g => new
        {
            FriendId = g.Key, // ID друга
            LastMessageTime = g.OrderByDescending(m =>
                m.Timestamp).Select(m => m.Timestamp).FirstOrDefault(), // для побудови списку
            LastMessage = g.OrderByDescending(m => m.Timestamp).Select(m
                => m.MessageText).FirstOrDefault() // для додавання під ПІБ користувача
        })
        .ToListAsync();

    // Отримуємо друзів за ID
    var friends = await _context.Users
        .Where(u => friendIdsWithLastMessage.Select(f =>
            f.FriendId).Contains(u.Id))
        .Select(user => new
        {
            Id = user.Id,
            FullName = user.FullName,
            Image = user.Image
        })
        .ToListAsync();

    // Отримуємо кількість непрочитаних повідомлень для кожного друга
    var friendsWithUnreadMessages = await _context.ChatMessages
        .Where(m => m.ReceiverId == userId && !m.IsRead)
        .GroupBy(m => m.SenderId)
        .Select(g => new
        {
            FriendId = g.Key,
            UnreadCount = g.Key == receiverId ? 0 : g.Count() // для
                відображення кількості непрочитаних повідомлень
        })
        .ToListAsync();
}
```

```

        .ToListAsync();

        // Об'єднуємо у кінцевий список для передачі
        var friendsWithMessages = friends
            .Join(friendIdsWithLastMessage,
                user => user.Id,
                msg => msg.FriendId,
                (user, msg) => new
                {
                    user.Id,
                    user.FullName,
                    user.Image,
                    LastMessageTime = (DateTime?)msg.LastMessageTime,
                    LastMessage = msg.LastMessage,
                    UnreadCount = friendsWithUnreadMessages
                        .FirstOrDefault(f => f.FriendId == user.Id)?.UnreadCount
                })
            ?? 0 // Кількість непрочитаних повідомлень для усіх у списку
        .OrderByDescending(f => f.LastMessageTime)
        .ToListAsync();

```

Цей фрагмент коду виконує ряд дій, потрібних для передачі інформації в базу даних, а надалі за допомогою бібліотеки ASP.NET – SignalR, створюється двостороння комунікація для спілкування в реальному часі, яка не потребує оновлення сторінки додатку для відображення повідомлень. Це було реалізовано наступним чином:

```

public class ChatHub : Hub
{
    private readonly ArtDbContext _context;

    public ChatHub(ArtDbContext context)
    {
        _context = context;
    }

    public async Task SendMessage(string userId, string message)
    {
        await Clients.All.SendAsync("ReceiveMessage", userId, message);
    }

    public async Task SendPrivateMessage(string senderId, string receiverId,
        string message)
    {
        var chatMessage = new ChatMessage
        {
            SenderId = int.Parse(senderId),
            ReceiverId = int.Parse(receiverId),
            MessageText = message,
            Timestamp = DateTime.Now,
            IsRead = false
        };

        // Додаємо повідомлення до бази даних
        _context.ChatMessages.Add(chatMessage);
        await _context.SaveChangesAsync();

        await Clients.User(receiverId).SendAsync("ReceivePrivateMessage",
            message, senderId);
    }
}

```

Клас `MainController` – це основний клас, який відповідає за обробку запитів, працює з профілями, публікаціями, пошуком користувачів, зміною паролів, додаванням коментарів та вподобайок, додаванням картин до стрічки та іншими. Розглянемо декілька методів цього класу.

Метод `GetUser` виконує перевірку аунтефікації, завантажує інформацію з бази даних та повертає JSON-об'єкт з даними користувача.

```
public async Task<IActionResult> GetUser()
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (userId == null)
    {
        return RedirectToAction("Login", "Account");
    }

    var user = await _context.Users
        .FirstOrDefaultAsync(u => u.Id == int.Parse(userId));

    string base64Image = user.Image != null
        ? Convert.ToBase64String(user.Image)
        : null;

    // Повертаємо дані користувача
    return Json(new
    {
        Id = user.Id,
        Name = user.FullName,
        AvatarBase64 = base64Image
    });
}
```

Метод `Main` завантажує інформацію про користувача, далі його сімох друзів для подальшого відображення на головній сторінці, завантажує публікації друзів та сортує їх по даті публікації, підвантажує дані з вподобайками та коментарями.

```
public async Task<IActionResult> Main()
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (userId == null)
    {
        return RedirectToAction("Login", "Account");
    }

    ViewBag.UserId = int.Parse(userId);
    ViewBag.UserName = (await _context.Users
        .Where(u => u.Id == int.Parse(userId))
        .FirstOrDefaultAsync())?.FullName;
    var friends = await _context.UserFriends
        .Include(uf => uf.Friend)
        .Where(uf => uf.UserId == int.Parse(userId)).Take(7)
        .ToListAsync();
    ViewBag.Friends = friends;
}
```

```

var friendIds = await _context.UserFriends
    .Where(uf => uf.UserId == int.Parse(userId))
    .Select(uf => uf.FriendId)
    .ToListAsync();

var friendPictures = await _context.Pictures
    .Where(p => friendIds.Contains(p.User.Id))
    .OrderByDescending(p => p.CreatedAt)
    .Select(p => new
    {
        p.Id,
        p.Image,
        p.Title,
        p.CreatedAt,
        p.Caption,
        LikeCount = p.Likes.Count(),
        CommentCount = p.Comments.Count(),
        UserImage = p.User.Image,
        UserFullName = p.User.FullName,
        UserId = p.User.Id,
        IsLikedByCurrentUser = p.Likes.Any(l => l.UserId ==
int.Parse(userId)),
        IsSavedByCurrentUser = p.Saves.Any(s => s.UserId ==
int.Parse(userId)),
        Comments = p.Comments.Select(c => new
        {
            c.Id,
            c.UserId,
            c.Text,
            c.CreatedAt,
            UserCommentFullName = c.User.FullName,
            UserCommentId = c.User.Id
        }).ToList()
    }).ToList()
    .Take(10).ToListAsync();

return View(friendPictures);}

```

Висновки до другого розділу

Відповідно до детального аналізу вимог користувачів у даному розділі було визначено варіанти використання системи та побудовано діаграму. Було розглянуто застосовані при розробці архітектури патерни проектування. Також розроблено структуру бази даних для збереження усієї необхідної інформації для повноцінної роботи системи.

Було визначено головні напрямки для розробки статистично-аналітичних даних системи та реалізовано у вигляді діаграм. Наведено детальний розгляд основних аспектів реалізації вебсайту, зокрема описано принципи роботи основних компонентів системи та файлову структуру додатку.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ

3.1. Порядок встановлення та налаштування системи.

Система розроблена у вигляді вебсайту. Сторінки сайтів – це набір текстових файлів, розмічених мовою HTML. Ці файли розуміються і обробляються браузером та виводяться на засіб відображення користувача (монітор, екран, принтер або синтезатор мови). Тому, перегляд сторінок в Інтернеті не потребує надпотужної комп'ютерної системи чи оптоволоконного каналу зв'язку [22].

Проте, оскільки частина контенту генерується за допомогою скриптів мови програмування JavaScript, а сценарії JavaScript надсилаються сервером для виконання на боці клієнта (тобто саме на комп'ютері користувача). Та через те що дописи можуть включати у себе мультимедійні матеріали, все-таки можна виділити деякі вимоги до апаратної частини та інтернет-з'єднання для комфортного перегляду вебсайту в браузері.

Мінімальні вимоги до комп'ютера: процесор 1,5 ГГц, оперативна пам'ять 512 Мб, вільне місце на диску 350 Мб, операційна система Windows 7 або вище / Mac OS X 10.6 або вище / Linux (Ubuntu 10.04 + / Debian 6 + / OpenSuSE 11.3 + / Fedora Linux 14).

Дисплей (монітор, екран) повинен мати розподільну здатність щонайменше 1024 точки (пікселі) по горизонталі.

Підключення до мережі Інтернет що забезпечує прийом даних близько 1 Мегабіт на секунду (при роботі з текстами такої швидкості цілком достатньо).

Таким чином розроблена система не вимагає спеціалізованого апаратного забезпечення, додаткових налаштувань та засобів для розгортання окрім стандартних для усіх сучасних браузерів.

Деплой, або розгортання, у програмуванні являє собою процес перетворення розробленого програмного коду на працюючий продукт, доступний для використання кінцевими користувачами. Тобто, програмний код,

який до цього моменту перебував у різних стадіях розроблення (як на комп'ютері розробника, так і на тестових серверах), тепер готовий до запуску на робочому сервері або пристрої. Розглянемо діаграму розгортання системи (рис. 3.1).

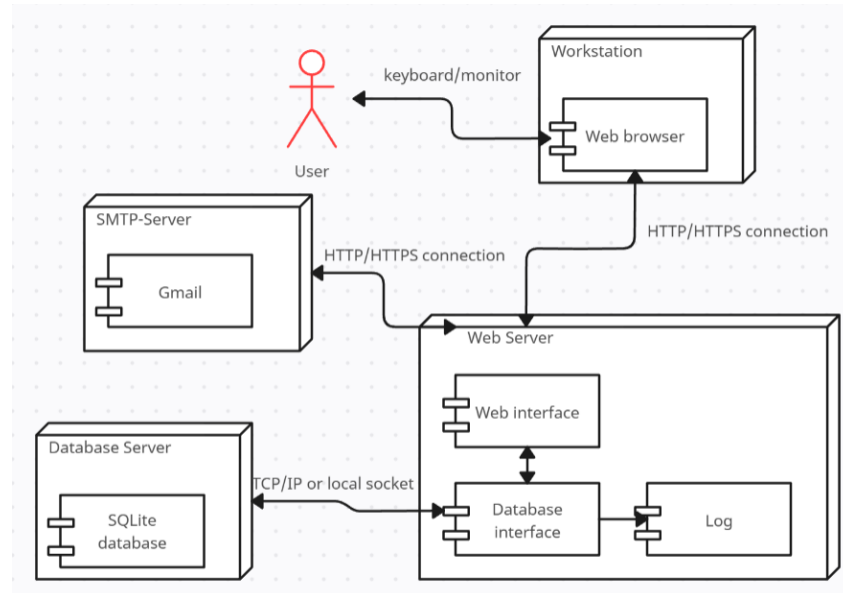


Рис. 3.1. Схема розгортання системи

Схема відображає користувача та 4 основні компоненти системи: робоча станція, вебсервер, який приймає запити від користувача, сервер бази даних та сервер SMTP, який відповідає за передачу повідомлень про купівлю/продаж та підтвердження електронної пошти при реєстрації клієнта.

3.2. Інтерфейс та порядок роботи з системою.

Після запуску системи користувач опиняється на сторінці входу в додаток, (рис. 3.2). Використано два поля Email та Password, чекбокс із запам'ятовуванням користувача на протязі 30 днів та можливість, при втраті пароллю відновити його. Для доступу до системи потрібно пройти реєстрацію, яка зображена на (рис. 3.3).

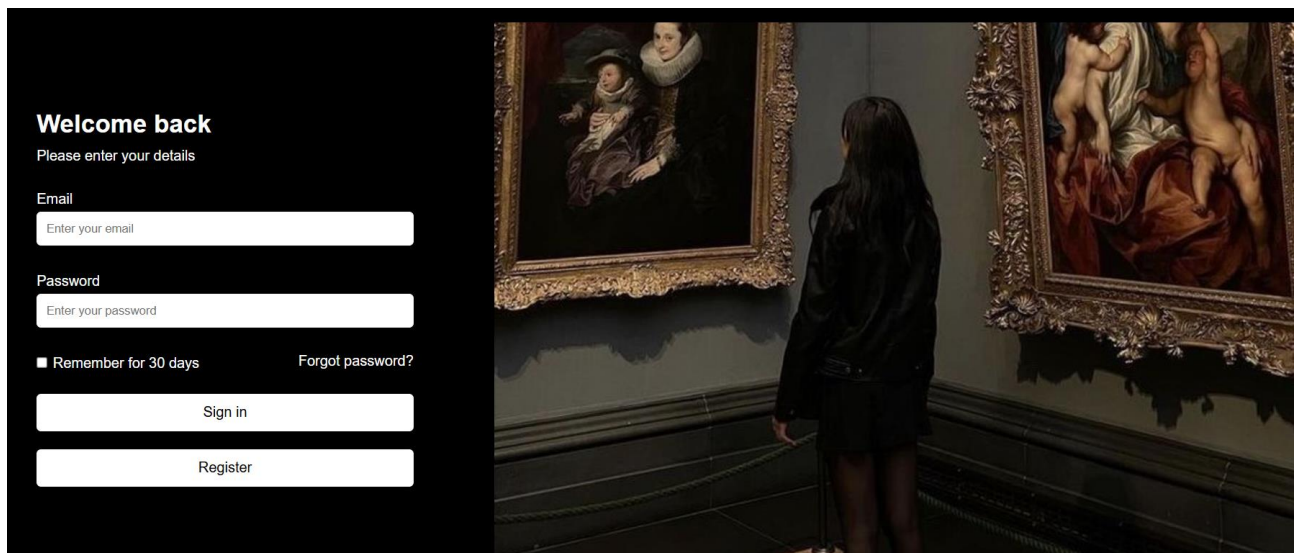


Рис. 3.2. Форма входу в систему

Після заповнення всіх полів форми реєстрації, на пошту прийде лист для підтвердження облікового запису, потрібно перейти за посиланням.

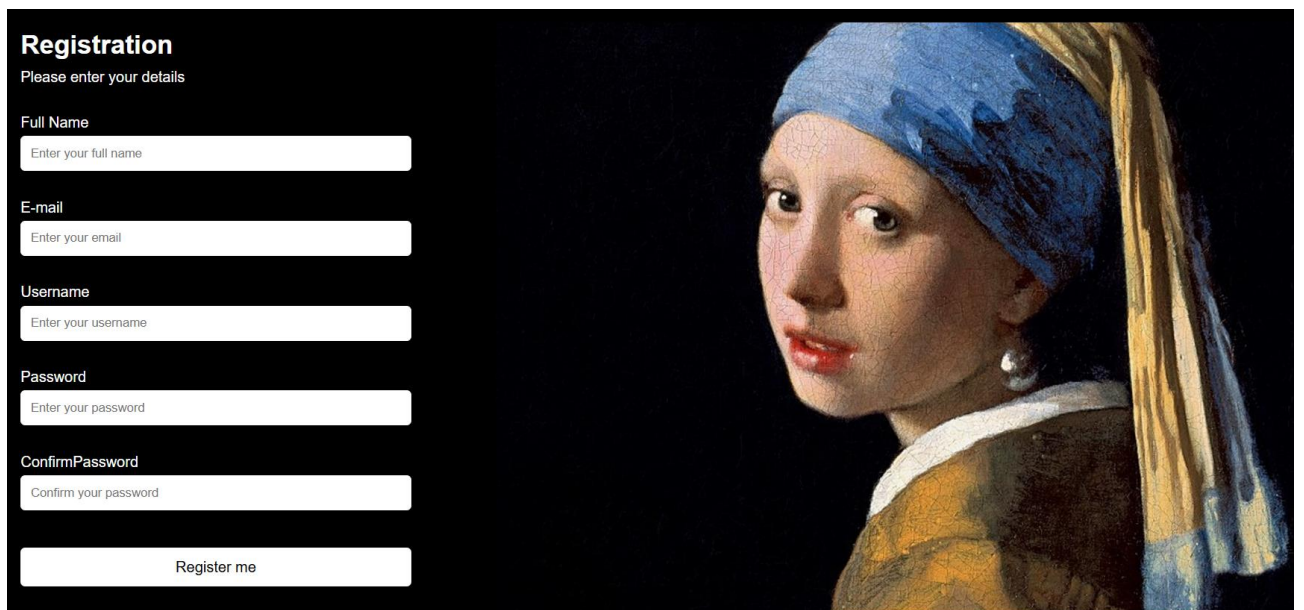


Рис. 3.3. Форма реєстрації користувача

У випадку коли користувач має активну сесію або він здійснив успішну вхід в систему, він автоматично буде перенаправлений на головну сторінку (рис. 3.4). На головній сторінці розміщені сім друзів з їхніми фото, всі публікації системи в порядку від нових до старіших. Розміщено 10 публікацій та є кнопка для завантаження наступних 10.



Рис. 3.4 Головна сторінка системи

В шапці профілю розміщено меню з посиланнями на основні функції сайту. Магазин – посилання на сторінку магазину, де є форма через, яку можна додавати картини на продаж з повним описом (автор, розмір, матеріал на якому намальовано та чим намальовано, ціна та інше) (рис. 3.5), також є кнопка «Sold pictures», де відображена інформації, які картини були продані (рис. 3.6).

Рис 3.5. Додавання картини до магазину

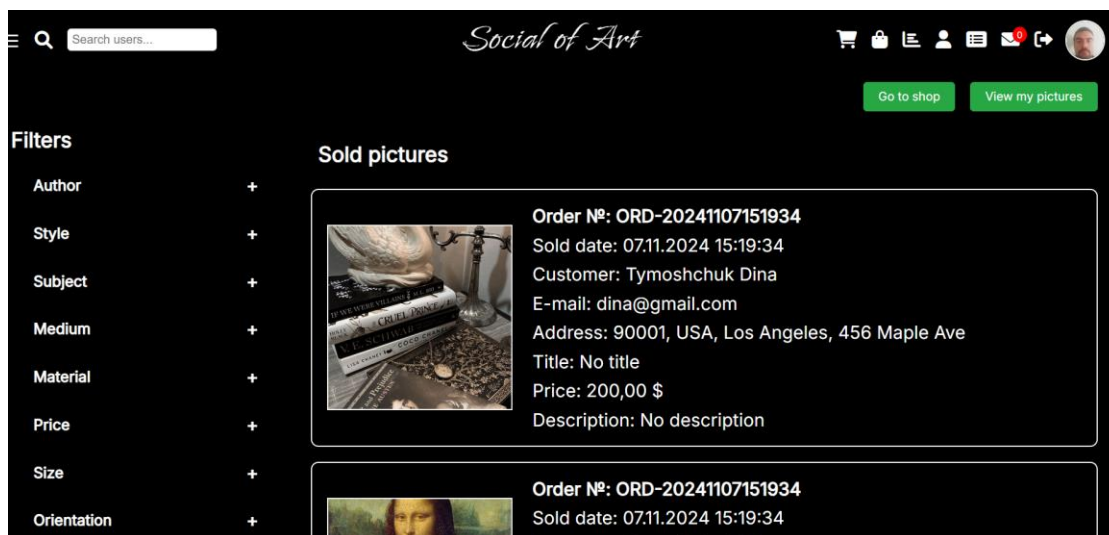


Рис. 3.6. Перегляд проданих картин

Мої замовлення – посилання на сторінку із можливістю перегляду активних та виконаних замовлень, адресою, з реквізитами для оплати. Сторінка з виконаними замовленнями виглядає наступним чином (рис. 3.7).

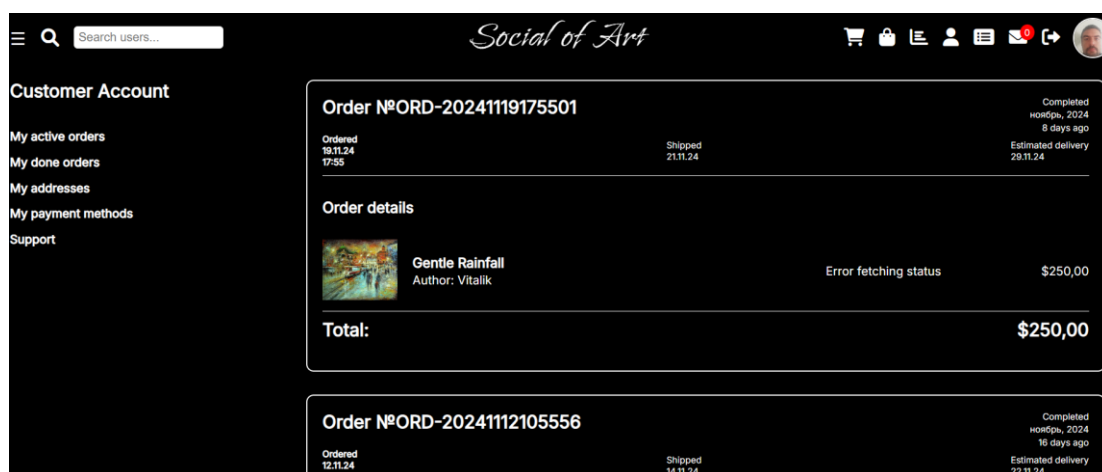


Рис. 3.7. Перегляд куплених картин

Також є можливість роботи з аналітикою, де відображено дані у вигляді графіків, діаграм для аналізу даних. Аналітика розділена на три окремі сторінки: загальна статистика, статистика покупок, статистика продажів.

В загальній статистиці можемо бачити відображення наступних даних: відношення продажів картин до покупок (рис. 3.8); ТОП 10 користувачів по вподобайкам (рис. 3.9); ТОП 10 користувачів за продажами; найбільш популярні картини по стилю; найбільш популярні картини по змісту зображення; найбільш

популярні картини за різновидом фарб; найбільш популярні картини за матеріалом на яких вона написана.

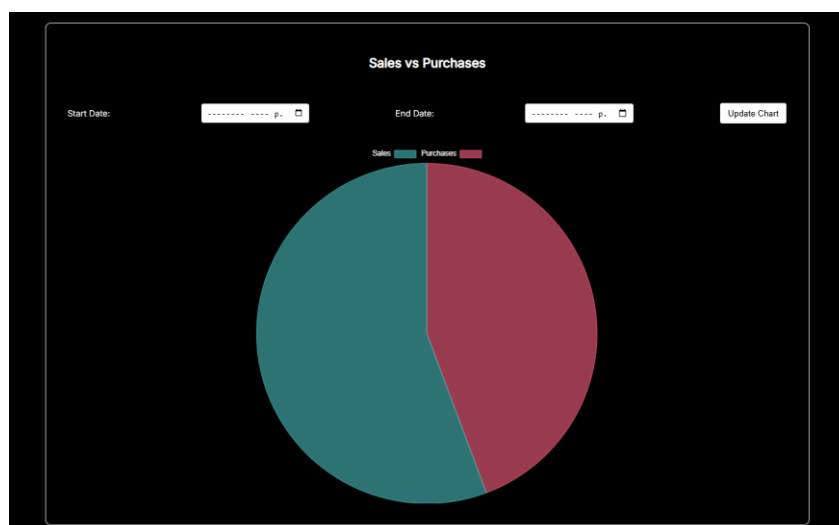


Рис. 3.8. Діаграма відношення продажів картин до покупок

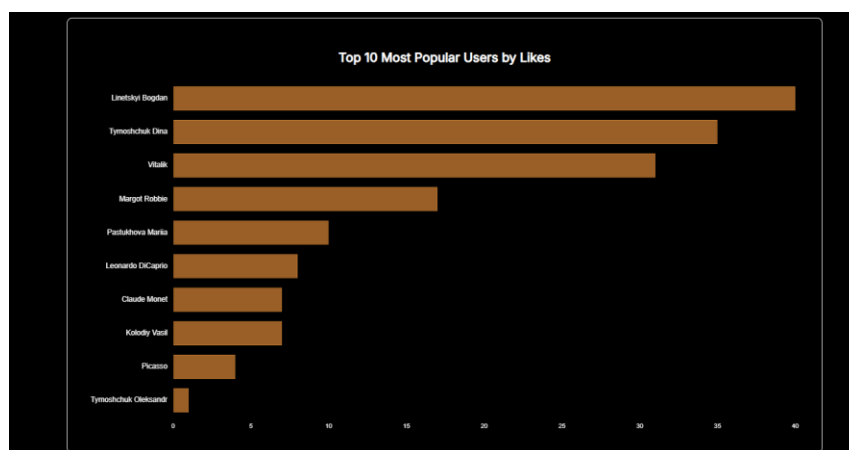


Рис. 3.9. Графік ТОП 10 користувачів за вподобайками

Статистика покупок виконується на основі напрямку купівлі творів мистецтва по різних категоріях та складається з трьох функцій: загальна сума купівлі картин авторизованим користувачем по місяцях, за період (рис. 3.10); загальна кількість куплених картин авторизованим користувачем по місяцях, за період (рис. 3.11); ТОП 10 користувачів, в яких авторизований користувач здійснював купівлю (рис. 3.12).

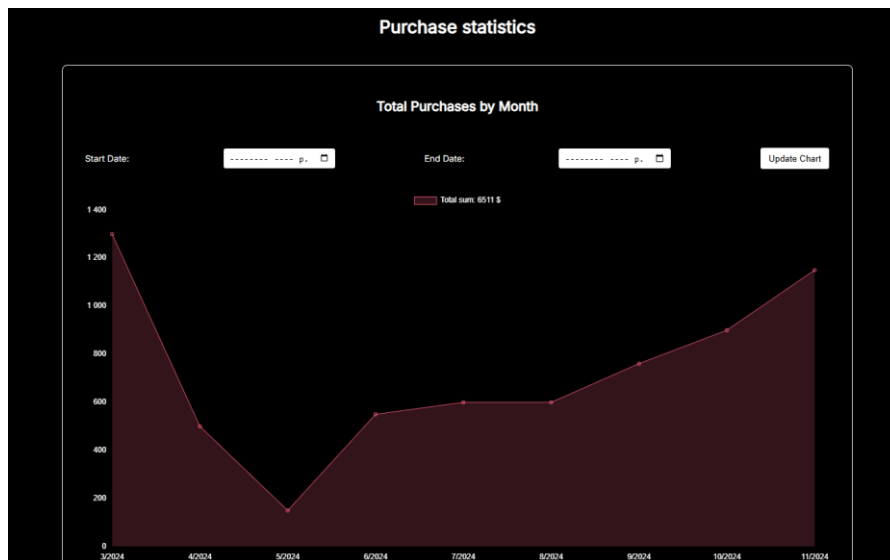


Рис. 3.10. Графік купівлі картин по місяцях

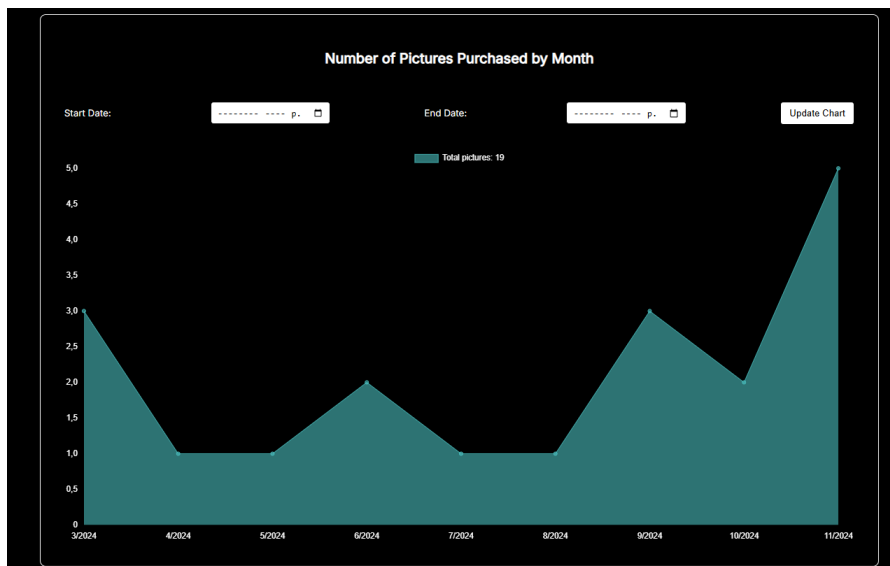


Рис. 3.10. Графік загальної кількості куплених картин

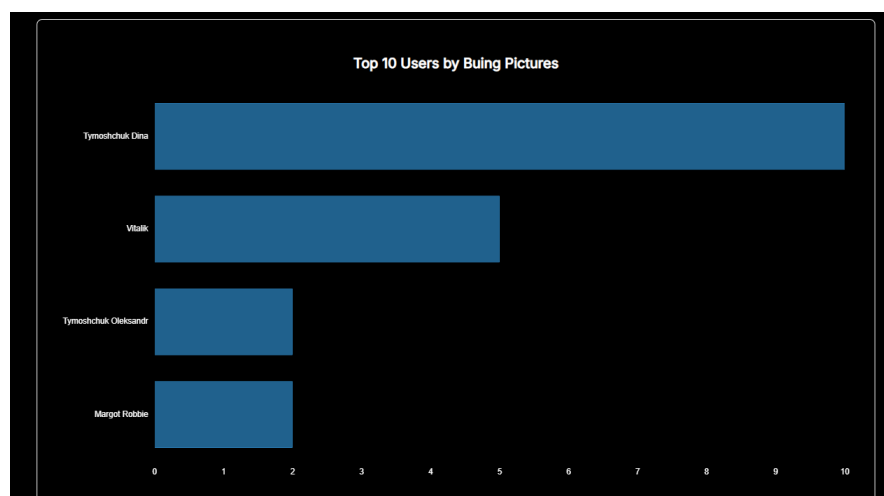


Рис. 3.11. Графік ТОП 10 художників за продажем картин користувача

Статистика продажів виконується, на основі напрямку продажів творів мистецтва авторизованим користувачем по різних складовим та складається з шести категорій для : продажі робіт в грошовому еквіваленті (рис 3.12), продаж картин по кількості штук, кількість проданих картин за стилем (рис 3.13), кількість продажів по предмету зображення на картині, кількість продажів картин за різновидом фарб та матеріалом на яких вона написана.

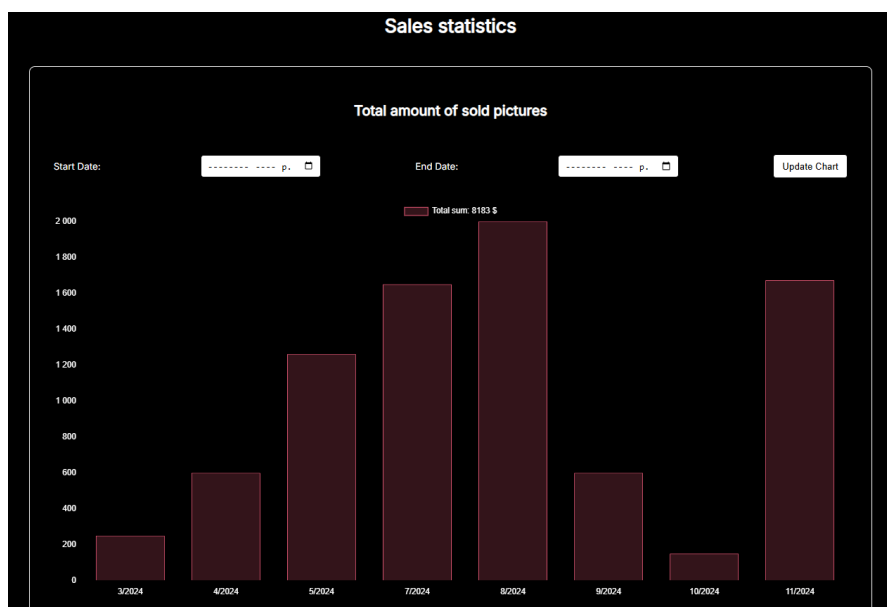


Рис 3.12. Графік продажу робіт в грошовому еквіваленті

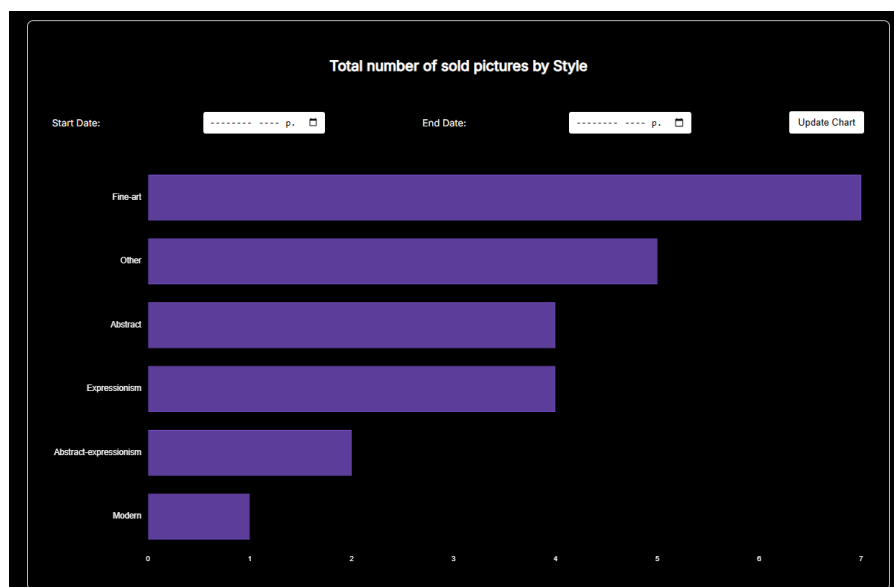


Рис 3.13. Графік кількості проданих картин за стилем

Налаштування профілю — за цим посиланням знаходяться кнопки редагування загальної інформацію про користувача, кнопка створення посту,

інформація загальної кількості вподобайок постів та підписників. Також розміщена стрічка з усіма картинами, які Ви додавали з можливістю видалення з продажу та взагалі із системи (рис 3.14).

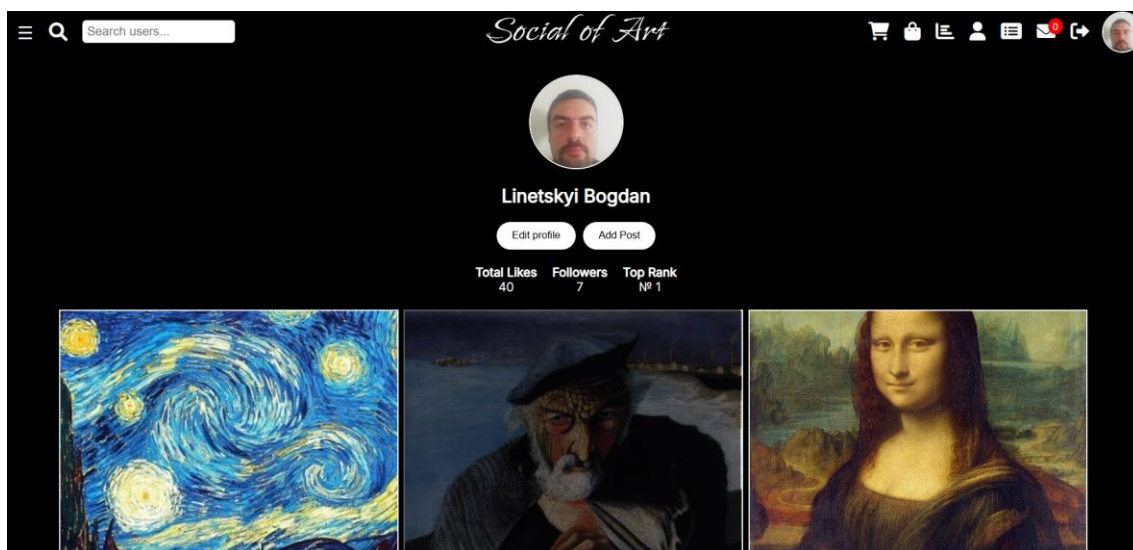


Рис. 3.14. Налаштування профілю користувача

Чат – посилання на сторінку для спілкування з друзями , покупцями та продавцями. Є можливість прикріплювати файли та емоджі, крім цього приходять повідомлення про купівлю картин (рис. 3.15).

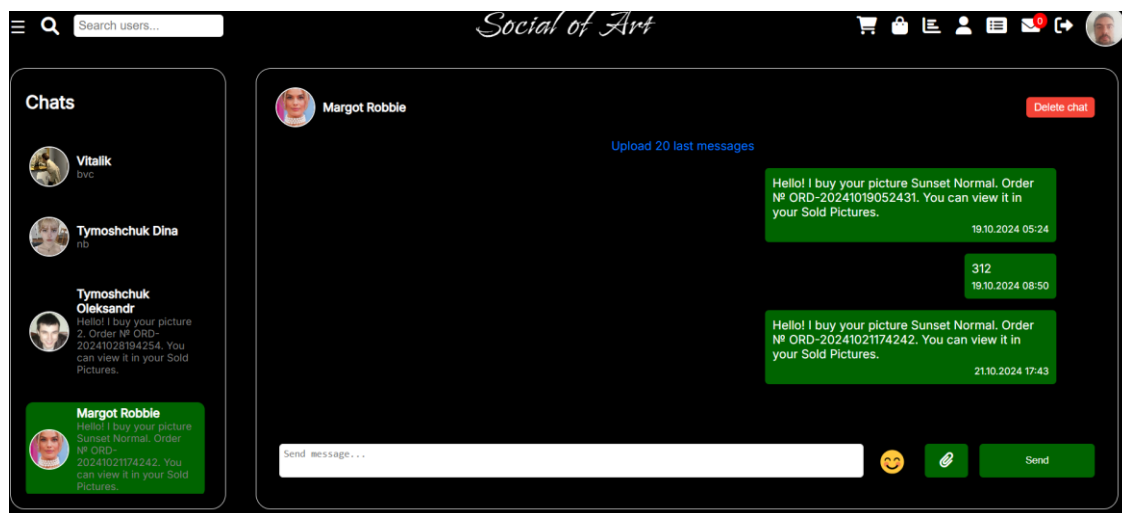


Рис. 3.15 Чат

Вкладка моя активність, сформована на основі вподобайок постів та відмічених картин з магазину. Вподобайки постів показують всі відмічені нами зображення та картини (рис 3.16). Натиснувши на зображення, виконується

перехід у стрічку того друга, хто є автором даної публікації, де є можливість залишити коментар та переглянути всі публікації користувача (рис. 3.17).

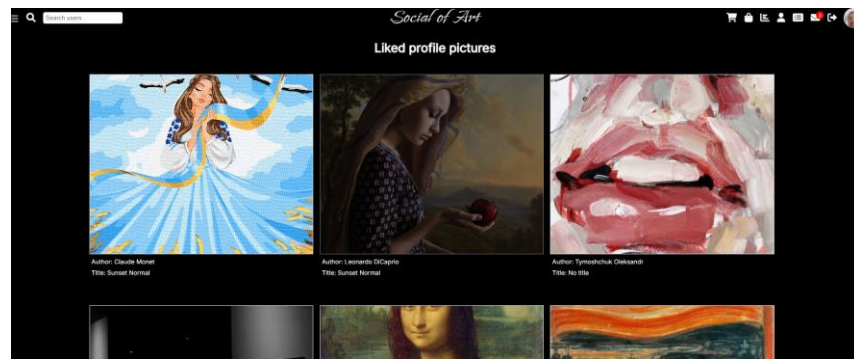


Рис. 3.16. Відмічені вподобайками картини

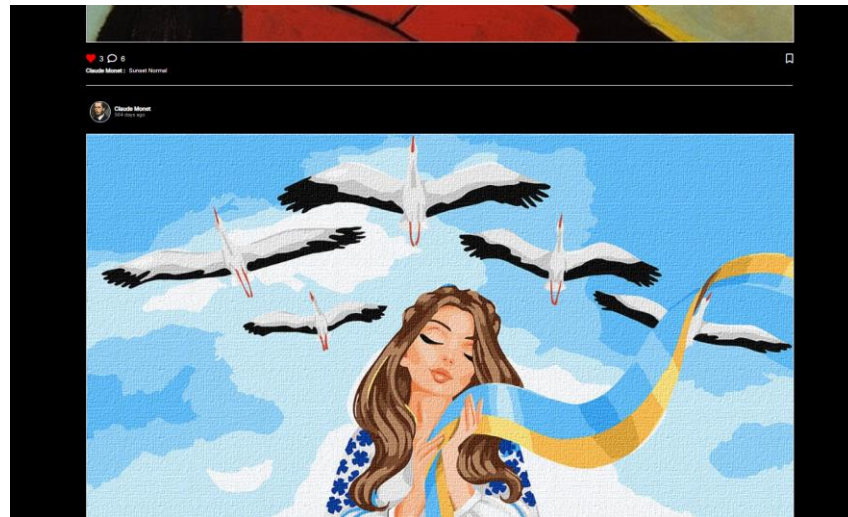


Рис. 3.17. Перехід на стрічку публікації

Вподобайки картин із магазину, показують всі відмічені лайками твори, які можливо придбати (рис. 3.18).

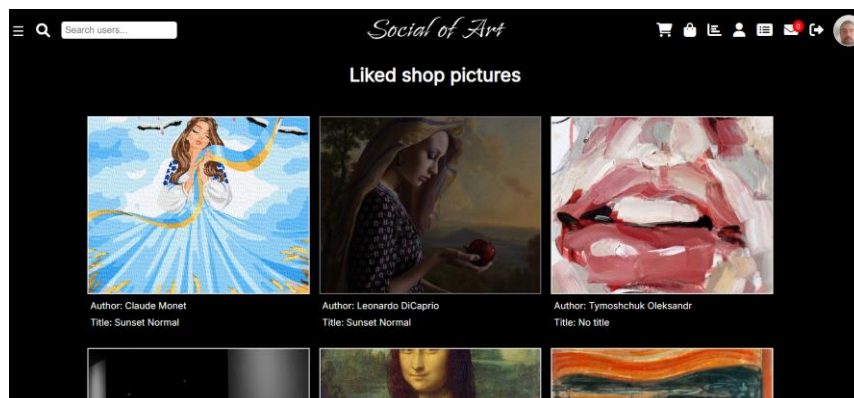


Рис. 3.18. Відмічені лайком картини

При натисканні по зображенню картини, відбувається перехід на сторінку, де відображається ціна картини, є кнопка додавання до моїх замовлень та знизу є стрічка інших творів автора (рис 3.19).

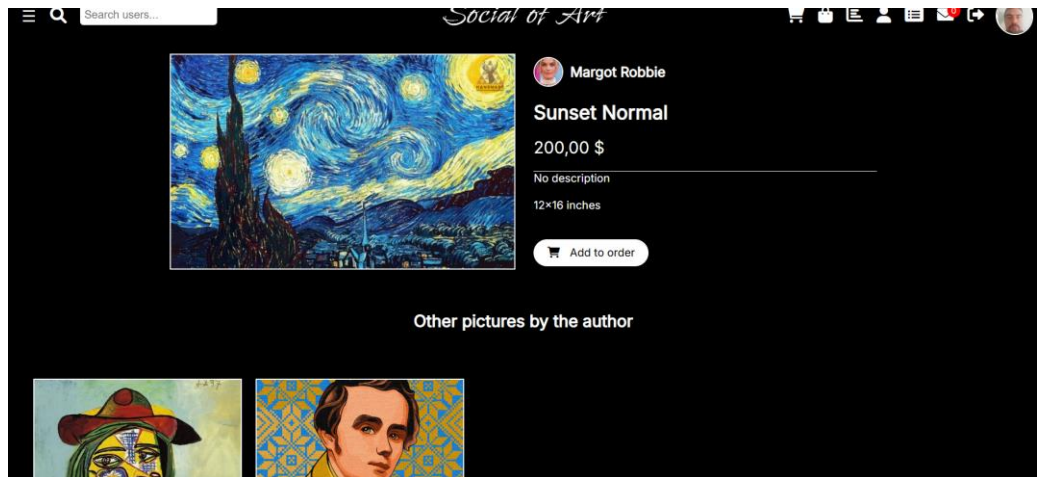


Рис. 3.19. Відмічена вподобайкою картину з магазину

В шапці профілю також є поле для пошуку користувачів та кнопка виходу з поточного профілю.

3.3. Тестування роботи системи.

Тестування, як заключний етап розробки додатку, грає життєво важливу роль в процесі створення високоякісного програмного забезпечення. Для перевірки основних показників розробленої системи проведено тестування сайту за допомогою інструменту Google Lighthouse.

Google Lighthouse — це інструмент аудиту з відкритим вихідним кодом, який допомагає розробникам підвищити продуктивність і доступність своїх веб-проектів. Його можна запустити на будь-якій веб-сторінці, загальнодоступній або яка вимагає аутентифікації. Google Lighthouse здійснює перевірку продуктивності, доступності та відповідності стандартам і найкращим практикам.

Для перевірки обрано головну сторінку додатку, оскільки вона відвідується користувачами найчастіше та містить найбільше контенту.

Результат тестування головної сторінки показав досить непогані результати (рис. 3.20): продуктивність – 81/100, доступність – 92/100, відповідність найкращим практикам – 74/100.

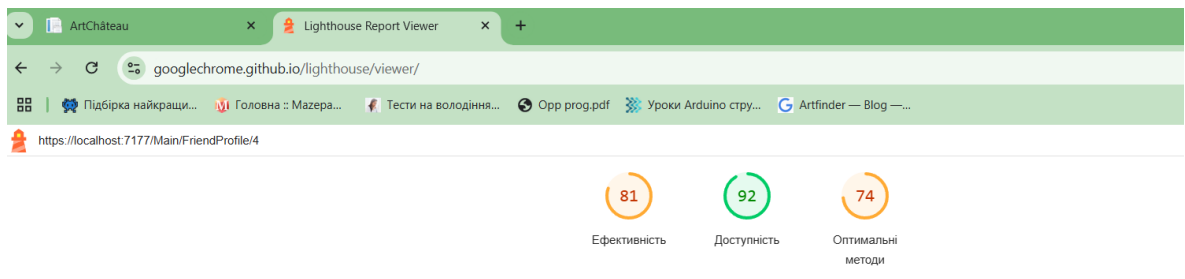


Рис. 3.20. Результат перевірки в Google Lighthouse

Розглянемо кожну метрику детальніше.

Продуктивність – результат аналізу швидкості завантаження сайту. На цю оцінку впливають наступні показники (рис. 3.21).

First Contentful Paint: вимірює час за який перший текст/зображення відобразилося на екрані.

Total Blocking Time: час блокування введення в мілісекундах, коли відвідувач нічого не може робити на сайті.

Speed Index: показує, наскільки швидкий вміст сторінки відображається візуально.

Largest Contentful Paint: показує час до відображення найбільшого та видимого вмісту на екрані.

Cumulative Layout Shift: підсумовує бали від кожного зсуву на сторінці, що відбувається під час завантаження сторінки.

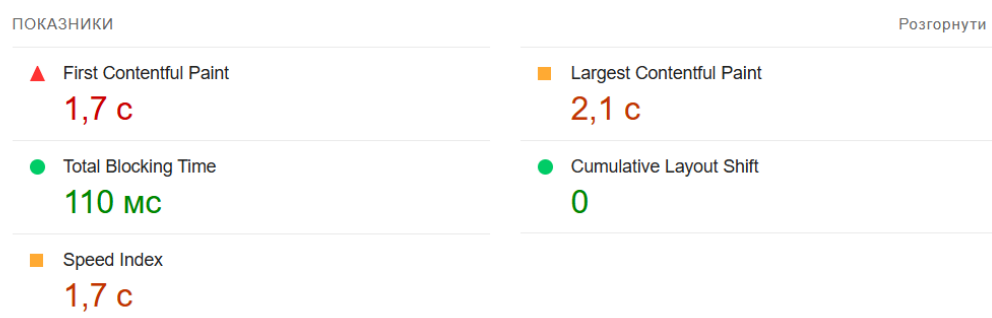


Рис. 3.21. Результат перевірки продуктивності

Перевірка доступності показує, чи всі користувачі можуть отримувати доступ до контенту і ефективно переміщатися по сайту. Ця оцінка залежить від зрозумілості та сприймання контенту, можливості керувати інтерфейсом і пересуватися вмістом без допомоги миші (рис. 3.22).

●	<code>[aria-hidden="true"]</code> немає в документі <code><body></code>	▼
●	Кнопки мають доступну для зчитування назву	▼
●	Зображення мають атрибути <code>[alt]</code>	▼
●	Параметр <code>[user-scalable="no"]</code> не використовується в елементі <code><meta name="viewport"></code> , а атрибут <code>[maximum-scale]</code> має значення не менше 5.	▼
●	Документ містить елемент <code><title></code>	▼
●	Елемент <code><html></code> містить атрибут <code>[lang]</code>	▼
●	Елемент <code><html></code> має дійсне значення атрибута <code>[lang]</code>	▼
●	Елементи форми мають пов'язані мітки	▼
●	Посилання мають назви, які можна розпізнати	▼
●	Області дотику мають достатній розмір і інтервали.	▼
●	Елементи заголовка розташовані послідовно в порядку спадання	▼
●	Елементи зображення не мають атрибутів <code>[alt]</code> , які є зайвим текстом.	▼

Рис. 3.22. Результат перевірки доступності

Перевірка оптимальних методів – показує вразливі сторони системи, набір рекомендацій оптимізації роботи вебсайту, у тому числі помилки браузера в консолі, зображення «піксельної» роздільної здатності та програму HTTPS (рис. 3.23).

ВИКОНАНІ ПЕРЕВІРКИ (11)		Сховати
●	Використовує протокол HTTPS	▼
●	Уникає сторонніх файлів cookie	▼
●	Дає користувачам змогу вставляти контент із буфера в поля введення	▼
●	Уникає надсилення запитів на доступ до геолокації під час завантаження сторінки	▼
●	Уникає надсилення запитів на показ сповіщень під час завантаження сторінки	▼
●	Показує зображення правильного формату	▼
●	Показує зображення з правильною роздільною здатністю	▼
●	Має тег <code><meta name="viewport"></code> з атрибутами <code>width</code> або <code>initial-scale</code>	▼
●	Сторінка має елемент HTML <code>doctype</code>	▼
●	Належно визначає набір символів	▼
●	Сторінка містить дійсні карти джерел	▼

Рис. 3.23. Результат перевірки найкращих практик

Висновки до третього розділу

В даному розділі було наведено порядок встановлення та налаштування додатку. Було детально зображено структуру інтерфейсу додатку та принципи взаємодії з ним. Покроково пояснено (з відповідними зображеннями) як використовувати основні функції системи.

Описано результати тестування додатку за допомогою інструменту Google Lighthouse. Перевірка основних характеристик показала непогані результати: продуктивність – 81/100, доступність – 92/100, відповідність найкращим практикам – 74/100. З’ясовано, що система працює коректно та виконує усі поставлені задачі.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було спроектовано та реалізовано, систему для продажу та купівлі творів мистецтва з модулем аналізу.

В роботі було проведено аналіз предметної області та поглиблене вивчення тематики, що дозволило визначити основні задачі, які необхідні для реалізації вебплатформи. Головними рисами та функціями додатку є: забезпечення спілкування в чаті, проведення операцій продажу та купівлі творів мистецтва, робота з аналітикою.

Обрано оптимальну архітектуру для реалізації системи – MVC, підібрано необхідний та достатній набір програмних продуктів для реалізації поставлених задач.

Було проаналізовано функціональні та нефункціональні вимоги до системи, розроблено структуру бази даних для збереження усієї необхідної інформації для повноцінної роботи системи.

Розглянуто алгоритми основних процесів системи та наведено детальний розгляд головних аспектів реалізації вебсистеми.

Також було описано порядок встановлення і налаштування системи. Наведено покроковий приклад роботи з системою та проведено її тестування.

В результаті проведеної роботи було вирішено поставлену задачу та розроблено інформаційно-аналітичну систему для купівлі та продажу творів мистецтва. Реалізований програмний комплекс готовий до використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ТОП-14 світових онлайн-галерей та арт-маркетплейсів для колекціонерів сучасного мистецтва [Електронний ресурс] – 2024 – Режим доступу: <https://kyiv.gallery/statii/top-14-svitovyh-onlain-galerei-ta-art-marketpleisiv-dlia-kolektsioneriv-suchasnogo-mystetstva>
2. Artfinder [Електронний ресурс] – 2024 – Режим доступу: <https://www.artfinder.com>
3. KyivGallery [Електронний ресурс] – 2024 – Режим доступу: <https://kyiv.gallery/>
4. Artnet [Електронний ресурс] – 2024 – Режим доступу: <https://www.artnet.com>
5. Муляр В. П. Візуалізація даних та інфографіка. Навчальний посібник [Електронний ресурс] – 2020 – Режим доступу: <https://evnuir.vnu.edu.ua/bitstream/123456789/21255/1/infohrafika.pdf>
6. Відкритий посібник з відкритих даних [Електронний ресурс] – 2019 – Режим доступу: <https://socialdata.org.ua/manual/manual5/>
7. Побудова динамічних веб-застосунків за допомогою MVC [Електронний ресурс] – 2020 – Режим доступу: <https://blog.ithillel.ua/articles/building-dynamic-web-applications-using-mvc>
8. Матеріали десятої міжнародної наукової конференції студентів та молодих вчених [Електронний ресурс] – 2021 – Режим доступу: <https://sci.ldubgd.edu.ua/bitstream/123456789/7584/1/%D0%9C%D0%A22020.pdf>
9. Коноваленко І. В. Програмування мовою C# 6.0, видавництво: “Тернопіль, ТНТУ” – 2016 – 227с.
10. Адам Фрімен, ASP.NET CORE 3 з прикладами на C# для професіоналів – 2021 – 1086с.
11. Що таке SQLite, для чого і як його використовувати [Електронний ресурс] – 2018. – Режим доступу: <https://foxminded.ua/prohramuvannia-baz-danykh-na-sqlite/>

12. DB Browser for SQLite [Електронний ресурс] – 2020 – Режим доступу: <https://www.onworks.net/uk/software/app-db-browser-for-sqlite>
13. Що таке веб-аналітика, а також сервіси та системи для її проведення [Електронний ресурс] – 2021. – Режим доступу: <https://wezom.com.ua/ua/blog/instrumenty-veb-analitiki>
14. Патерни [Електронний ресурс] – 2018. – Режим доступу: <https://refactoring.guru/uk/design-patterns/builder>
15. Що таке патерни [Електронний ресурс] – 2024. – Режим доступу: <https://itvdn.com.ua/blog/article/what-are-design-patterns-in-programming>
16. Data Analytics vs Data Science: у чому різниця? [Електронний ресурс] – 2023 – Режим доступу: <https://web-academy.ua/blog/stati/data-analytics-vs-data-science>
17. Бізнес-аналітика та статистика: як працює і чим допоможе компанії [Електронний ресурс] – 2023. – Режим доступу: <https://litera.com.ua/2024/04/27/biznes-analytika-ta-statystyka-yak-pratsyuie-i-chym-dopomozhe-kompaniyi/>
18. LINQ. Огляд [Електронний ресурс] – 2020. – Режим доступу: <https://devzone.org.ua/post/linq-ohliad>
19. Умови та Правила надання послуг в системі LiqPay [Електронний ресурс] – 2024. – Режим доступу: <https://www.liqpay.ua/information/terms>
20. Кращі сервіси для веб-аналітики [Електронний ресурс] – 2022. – Режим доступу: <https://www.ukraine.com.ua/uk/blog/web-analytics/luchshie-servisi-dlya-veb-analitiki.html>.
21. Матеріали десятої міжнародної наукової конференції студентів та молодих вчених [Електронний ресурс] – 2020. – Режим доступу: <https://sci.ldubgd.edu.ua/bitstream/123456789/7584/1/%D0%9C%D0%A22020.pdf>.
22. Що таке deploy і навіщо він потрібен [Електронний ресурс] – 2023. – Режим доступу: <https://foxminded.ua/deploy-tse/>

ДОДАТКИ

Лістинг StatisticController.cs

```

using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using System.Security.Claims;
using ArtChateau.Models;
using ArtChateau.Models.Forms;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Mvc.ApplicationModels;
using Microsoft.AspNetCore.Mvc.RazorPages;
using static Microsoft.EntityFrameworkCore.DbLoggerCategory;
using static System.Collections.Specialized.BitVector32;
using System.Linq;

namespace ArtChateau.Controllers;

public class StatisticController : Controller
{
    private readonly ArtDbContext _context;
    public StatisticController(ArtDbContext context)
    {
        _context = context;
    }
    [HttpGet]
    public async Task<IActionResult> OrderStatistics()
    {
        var currentUserId = User.FindFirstValue(ClaimTypes.NameIdentifier);
        if (currentUserId == null)
        {
            return RedirectToAction("Login", "Account");
        }
        var userId = int.Parse(currentUserId);

        var expensesPerMonth = await _context.PictureOrders
            .Where(po => po.Order.CustomerId == userId && po.Order.IsCompleted &&
                po.Picture.IsSold) // Перевіряємо, що замовлення виконано, належить користувачу і картина
                продана
            .Include(po => po.Picture) // Включаємо Picture для доступу до ціни
            .Include(po => po.Order) // Включаємо Order для доступу до OrderDate
            .GroupBy(po => new { po.Order.OrderDate.Year, po.Order.OrderDate.Month }) // Групуємо
            за роком і місяцем дати замовлення
            .Select(g => new
            {
                Year = g.Key.Year,
                Month = g.Key.Month,
                TotalExpenses = g.Sum(po => (double)po.Picture.Price) // Сума вартості витрат
                користувача на картини у кожному місяці
            })
            .OrderBy(g => g.Year)
            .ThenBy(g => g.Month)
            .ToListAsync();

        var pictureCountPerMonth = await _context.PictureOrders
            .Where(po => po.Order.CustomerId == userId && po.Order.IsCompleted &&
                po.Picture.IsSold)
            .Include(po => po.Picture)
            .Include(po => po.Order)
            .GroupBy(po => new { po.Order.OrderDate.Year, po.Order.OrderDate.Month })
            .Select(g => new
            {
                Year = g.Key.Year,
                Month = g.Key.Month,
                PictureCount = g.Count()
            })
            .ToListAsync();
    }
}

```

```

    })
    .OrderBy(g => g.Year)
    .ThenBy(g => g.Month)
    .ToListAsync();

    ViewBag.PictureCountPerMonth = pictureCountPerMonth;

    var topUsersByPurchased = await _context.PictureOrders
        .Where(po => po.Order.CustomerId == userId && po.Picture.IsSold &&
            po.Order.IsCompleted) // Перевіряємо, що картина продана і замовлення виконано
        .GroupBy(po => po.Picture.UserId) // Групуємо по UserId картин
        .Select(g => new
        {
            UserId = g.Key, // UserId власника картини
            UserName = g.FirstOrDefault().Picture.User.FullName, // Ім'я користувача
            PurchaseCount = g.Count() // Кількість проданих картин
        })
        .OrderByDescending(g => g.PurchaseCount) // Сортуюмо по кількості проданих картин
        .Take(10) // Топ 10
        .ToListAsync();

    // Передаємо результат у ViewBag
    ViewBag.TopUsersByPurchased = topUsersByPurchased;

    return View(expensesPerMonth);
}
[HttpGet]
public async Task<IActionResult> AllStatistics()
{
    var currentUserId = User.FindFirstValue(ClaimTypes.NameIdentifier);
    if (currentUserId == null)
    {
        return RedirectToAction("Login", "Account");
    }

    var userId = int.Parse(currentUserId);
    var expensesPerMonth = await _context.PictureOrders
        .Where(po => po.Order.CustomerId == userId && po.Order.IsCompleted &&
            po.Picture.IsSold) // Перевіряємо, що замовлення виконано, належить користувачу і картина
            продана
        .Include(po => po.Picture) // Включаємо Picture для доступу до ціни
        .Include(po => po.Order) // Включаємо Order для доступу до OrderDate
        .GroupBy(po => new { po.Order.OrderDate.Year, po.Order.OrderDate.Month }) // Групуємо
        за роком і місяцем дати замовлення
        .Select(g => new
        {
            Year = g.Key.Year,
            Month = g.Key.Month,
            TotalExpenses = g.Sum(po => (double)po.Picture.Price) // Сума вартості витрат
            користувача на картини у кожному місяці
        })
        .OrderBy(g => g.Year)
        .ThenBy(g => g.Month)
        .ToListAsync();

    var salesPerMonth = await _context.PictureOrders
        .Where(po => po.Picture.UserId == userId && po.Picture.IsSold &&
            po.Order.IsCompleted) // Перевіряємо, що картина продана і замовлення виконано
        .Include(po => po.Picture) // Включаємо Picture для доступу до ціни
        .Include(po => po.Order) // Включаємо Order для доступу до OrderDate
        .GroupBy(po => new { po.Order.OrderDate.Year, po.Order.OrderDate.Month }) //
        Групуємо за роком і місяцем дати замовлення
        .Select(g => new
        {
            Year = g.Key.Year,
            Month = g.Key.Month,

```

```

        TotalAmount = g.Sum(po => (double)po.Picture.Price) // Сума вартості проданих
картин у кожному місяці
    })
    .OrderBy(g => g.Year)
    .ThenBy(g => g.Month)
    .ToListAsync();

    // Передаємо результат у ViewBag
    ViewBag.SalesPerMonth = salesPerMonth;

    var topUsersByLikes = await _context.Likes
    .GroupBy(l => l.Picture.UserId) // Групуємо по UserId картин
    .Select(g => new
    {
        UserId = g.Key, // UserId власника картини
        UserName = g.FirstOrDefault().Picture.User.FullName, // Ім'я користувача
        LikesCount = g.Count() // Кількість лайків
    })
    .OrderByDescending(g => g.LikesCount) // Сортуюмо по лайках
    .Take(10) // Топ 10
    .ToListAsync();

    // Передаємо результат у ViewBag
    ViewBag.TopUsersByLikes = topUsersByLikes;

    var topUsersBySales = await _context.PictureOrders
    .Where(po => po.Picture.IsSold && po.Order.IsCompleted) // Перевіряємо, що картина
продана і замовлення виконано
    .GroupBy(po => po.Picture.UserId) // Групуємо по UserId картин
    .Select(g => new
    {
        UserId = g.Key, // UserId власника картини
        UserName = g.FirstOrDefault().Picture.User.FullName, // Ім'я користувача
        SoldCount = g.Count() // Кількість проданих картин
    })
    .OrderByDescending(g => g.SoldCount) // Сортуюмо по кількості проданих картин
    .Take(10) // Топ 10
    .ToListAsync();

    // Передаємо результат у ViewBag
    ViewBag.TopUsersBySales = topUsersBySales;

    var picturesByStyle = await _context.PictureOrders
    .Where(po => po.Picture.IsSold && po.Order.IsCompleted) // Перевірка, що картина
продана і замовлення виконано
    .Include(po => po.Picture) // Включаємо Picture для доступу до стилю
    .Include(po => po.Order) // Включаємо Order для доступу до дати замовлення
    .GroupBy(po => new { po.Picture.Style, po.Order.OrderDate.Year,
po.Order.OrderDate.Month }) // Групуємо за стилем, роком і місяцем
    .Select(g => new
    {
        Style = g.Key.Style, // Назва стилю
        Year = g.Key.Year, // Рік
        Month = g.Key.Month, // Місяць
        Count = g.Count() // Кількість картин, проданих у кожному стилі за місяць
    })
    .OrderBy(g => g.Year)
    .ThenBy(g => g.Month)
    .ThenBy(g => g.Style)
    .ToListAsync();

    ViewBag.PicturesByStyle = picturesByStyle;

    var picturesBySubject = await _context.PictureOrders
    .Where(po => po.Picture.IsSold && po.Order.IsCompleted) // Перевірка, що картина
продана і замовлення виконано
    .Include(po => po.Picture) // Включаємо Picture для доступу до Subject

```

```

        .Include(po => po.Order) // Включаємо Order для доступу до дати замовлення
        .GroupBy(po => new { po.Picture.Subject, po.Order.OrderDate.Year,
        po.Order.OrderDate.Month }) // Групуємо за Subject, роком і місяцем
        .Select(g => new
        {
            Subject = g.Key.Subject, // Назва Subject
            Year = g.Key.Year,        // Рік
            Month = g.Key.Month,      // Місяць
            Count = g.Count()         // Кількість картин, проданих по Subject за місяць
        })
        .OrderBy(g => g.Year)
        .ThenBy(g => g.Month)
        .ThenBy(g => g.Subject)
        .ToListAsync();

        ViewBag.PicturesBySubject = picturesBySubject;

        var picturesByMedium = await _context.PictureOrders
        .Where(po => po.Picture.IsSold && po.Order.IsCompleted) // Перевірка, що картина
        продана і замовлення виконано
        .Include(po => po.Picture) // Включаємо Picture для доступу до Medium
        .Include(po => po.Order)   // Включаємо Order для доступу до дати замовлення
        .GroupBy(po => new { po.Picture.Medium, po.Order.OrderDate.Year,
        po.Order.OrderDate.Month }) // Групуємо за Medium, роком і місяцем
        .Select(g => new
        {
            Medium = g.Key.Medium, // Назва Medium
            Year = g.Key.Year,      // Рік
            Month = g.Key.Month,    // Місяць
            Count = g.Count()       // Кількість картин, проданих по Medium за місяць
        })
        .OrderBy(g => g.Year)
        .ThenBy(g => g.Month)
        .ThenBy(g => g.Medium)
        .ToListAsync();

        ViewBag.PicturesByMedium = picturesByMedium;

        var picturesByMaterial = await _context.PictureOrders
        .Where(po => po.Picture.IsSold && po.Order.IsCompleted) // Перевірка, що картина
        продана і замовлення виконано
        .Include(po => po.Picture) // Включаємо Picture для доступу до Material
        .Include(po => po.Order)   // Включаємо Order для доступу до дати замовлення
        .GroupBy(po => new { po.Picture.Material, po.Order.OrderDate.Year,
        po.Order.OrderDate.Month }) // Групуємо за Material, роком і місяцем
        .Select(g => new
        {
            Material = g.Key.Material, // Назва Material
            Year = g.Key.Year,          // Рік
            Month = g.Key.Month,        // Місяць
            Count = g.Count()           // Кількість картин, проданих по Material за місяць
        })
        .OrderBy(g => g.Year)
        .ThenBy(g => g.Month)
        .ThenBy(g => g.Material)
        .ToListAsync();

        ViewBag.PicturesByMaterial = picturesByMaterial;

        return View(expensesPerMonth);
    }
    [HttpGet]
    public async Task<ActionResult> ShopStatistics()
    {
        var currentUserId = User.FindFirstValue(ClaimTypes.NameIdentifier);
        if (currentUserId == null)
        {

```

```

        return RedirectToAction("Login", "Account");
    }

    var userId = int.Parse(currentUserId);
    var picturesPerMonth = await _context.PictureOrders
        .Where(po => po.Picture.UserId == userId && po.Picture.IsSold &&
        po.Order.IsCompleted) // Перевіряємо, що картина продана і замовлення виконано
        .Include(po => po.Order) // Включаємо Order для доступу до OrderDate
        .GroupBy(po => new { po.Order.OrderDate.Year, po.Order.OrderDate.Month }) //
        Групуємо за роком і місяцем дати замовлення
        .Select(g => new
        {
            Year = g.Key.Year,
            Month = g.Key.Month,
            Count = g.Count() // Підраховуємо кількість картин, проданих у кожному місяці
        })
        .OrderBy(g => g.Year)
        .ThenBy(g => g.Month)
        .ToListAsync();

    var salesPerMonth = await _context.PictureOrders
        .Where(po => po.Picture.UserId == userId && po.Picture.IsSold &&
        po.Order.IsCompleted) // Перевіряємо, що картина продана і замовлення виконано
        .Include(po => po.Picture) // Включаємо Picture для доступу до ціни
        .Include(po => po.Order) // Включаємо Order для доступу до OrderDate
        .GroupBy(po => new { po.Order.OrderDate.Year, po.Order.OrderDate.Month }) //
        Групуємо за роком і місяцем дати замовлення
        .Select(g => new
        {
            Year = g.Key.Year,
            Month = g.Key.Month,
            TotalAmount = g.Sum(po => (double)po.Picture.Price) // Сума вартості проданих
            картин у кожному місяці
        })
        .OrderBy(g => g.Year)
        .ThenBy(g => g.Month)
        .ToListAsync();

    // Передаємо результат у ViewBag
    ViewBag.SalesPerMonth = salesPerMonth;

    var picturesByStyle = await _context.PictureOrders
        .Where(po => po.Picture.UserId == userId && po.Picture.IsSold && po.Order.IsCompleted)
    // Перевірка, що картина продана і замовлення виконано
        .Include(po => po.Picture) // Включаємо Picture для доступу до стилю
        .Include(po => po.Order) // Включаємо Order для доступу до дати замовлення
        .GroupBy(po => new { po.Picture.Style, po.Order.OrderDate.Year,
        po.Order.OrderDate.Month }) // Групуємо за стилем, роком і місяцем
        .Select(g => new
        {
            Style = g.Key.Style, // Назва стилю
            Year = g.Key.Year, // Рік
            Month = g.Key.Month, // Місяць
            Count = g.Count() // Кількість картин, проданих у кожному стилі за місяць
        })
        .OrderBy(g => g.Year)
        .ThenBy(g => g.Month)
        .ThenBy(g => g.Style)
        .ToListAsync();

    ViewBag.PicturesByStyle = picturesByStyle;

    var picturesBySubject = await _context.PictureOrders
        .Where(po => po.Picture.UserId == userId && po.Picture.IsSold && po.Order.IsCompleted)
    // Перевірка, що картина продана і замовлення виконано
        .Include(po => po.Picture) // Включаємо Picture для доступу до Subject
        .Include(po => po.Order) // Включаємо Order для доступу до дати замовлення

```

```

        .GroupBy(po => new { po.Picture.Subject, po.Order.OrderDate.Year,
        po.Order.OrderDate.Month }) // Групуємо за Subject, роком і місяцем
        .Select(g => new
        {
            Subject = g.Key.Subject, // Назва Subject
            Year = g.Key.Year,        // Рік
            Month = g.Key.Month,      // Місяць
            Count = g.Count()         // Кількість картин, проданих по Subject за місяць
        })
        .OrderBy(g => g.Year)
        .ThenBy(g => g.Month)
        .ThenBy(g => g.Subject)
        .ToListAsync();

        ViewBag.PicturesBySubject = picturesBySubject;

        var picturesByMedium = await _context.PictureOrders
        .Where(po => po.Picture.UserId == userId && po.Picture.IsSold && po.Order.IsCompleted)
        // Перевірка, що картина продана і замовлення виконано
        .Include(po => po.Picture) // Включаємо Picture для доступу до Medium
        .Include(po => po.Order)    // Включаємо Order для доступу до дати замовлення
        .GroupBy(po => new { po.Picture.Medium, po.Order.OrderDate.Year,
        po.Order.OrderDate.Month }) // Групуємо за Medium, роком і місяцем
        .Select(g => new
        {
            Medium = g.Key.Medium, // Назва Medium
            Year = g.Key.Year,      // Рік
            Month = g.Key.Month,    // Місяць
            Count = g.Count()       // Кількість картин, проданих по Medium за місяць
        })
        .OrderBy(g => g.Year)
        .ThenBy(g => g.Month)
        .ThenBy(g => g.Medium)
        .ToListAsync();

        ViewBag.PicturesByMedium = picturesByMedium;

        var picturesByMaterial = await _context.PictureOrders
        .Where(po => po.Picture.UserId == userId && po.Picture.IsSold && po.Order.IsCompleted)
        // Перевірка, що картина продана і замовлення виконано
        .Include(po => po.Picture) // Включаємо Picture для доступу до Material
        .Include(po => po.Order)    // Включаємо Order для доступу до дати замовлення
        .GroupBy(po => new { po.Picture.Material, po.Order.OrderDate.Year,
        po.Order.OrderDate.Month }) // Групуємо за Material, роком і місяцем
        .Select(g => new
        {
            Material = g.Key.Material, // Назва Material
            Year = g.Key.Year,          // Рік
            Month = g.Key.Month,        // Місяць
            Count = g.Count()           // Кількість картин, проданих по Material за місяць
        })
        .OrderBy(g => g.Year)
        .ThenBy(g => g.Month)
        .ThenBy(g => g.Material)
        .ToListAsync();

        ViewBag.PicturesByMaterial = picturesByMaterial;

        return View(picturesPerMonth);
    }
}

```

Лістинг ChatController.cs

```
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using System.Security.Claims;
using ArtChatean.Models;
using ArtChatean.Models.Forms;
using Microsoft.EntityFrameworkCore;
using static System.Collections.Specialized.BitVector32;
using static System.Runtime.InteropServices.JavaScript.JSType;
using Microsoft.AspNetCore.SignalR;
using ArtChatean.Models.Services;

namespace ArtChatean.Controllers;

[Authorize]
public class ChatController : Controller
{
    private readonly ArtDbContext _context;
    private readonly IHubContext<ChatHub> _hubContext;

    public ChatController(ArtDbContext context, IHubContext<ChatHub> hubContext)
    {
        _context = context;
        _hubContext = hubContext;
    }

    // Метод для відображення чату між користувачами
    [HttpGet]
    public async Task<ActionResult> MainChat(int? receiverId)
    {
        var userIdstr = User.FindFirstValue(ClaimTypes.NameIdentifier);

        if (userIdstr == null)
        {
            return RedirectToAction("Login", "Account");
        }

        var userId = int.Parse(userIdstr);

        // БЛОК ЯКИЙ ФІЛЬТРУЄ І ГРУПУЄ КОРИСТУВАЧІВ ДЛЯ ОТРИМАННЯ НЕОБХІДНИХ ДАННИХ

        // Отримуємо ID друзів з останніми повідомленнями
        var friendIdsWithLastMessage = await _context.ChatMessages
            .Where(m => m.SenderId == userId || m.ReceiverId == userId)
            .GroupBy(m => m.SenderId == userId ? m.ReceiverId : m.SenderId)
            .Select(g => new
            {
                FriendId = g.Key, // ID друга
                LastMessageTime = g.OrderByDescending(m => m.Timestamp).Select(m =>
                    m.Timestamp).FirstOrDefault(), // для побудови списку
                LastMessage = g.OrderByDescending(m => m.Timestamp).Select(m =>
                    m.MessageText).FirstOrDefault() // для додавання під ПІБ користувача
            })
            .ToListAsync();

        // Отримуємо друзів за ID
        var friends = await _context.Users
            .Where(u => friendIdsWithLastMessage.Select(f => f.FriendId).Contains(u.Id))
            .Select(user => new
            {
                Id = user.Id,
                FullName = user.FullName,
                Image = user.Image
            })
            .ToListAsync();
    }
}
```

```

        .ToListAsync();

// Отримуємо кількість непрочитаних повідомлень для кожного друга
var friendsWithUnreadMessages = await _context.ChatMessages
    .Where(m => m.ReceiverId == userId && !m.IsRead)
    .GroupBy(m => m.SenderId)
    .Select(g => new
    {
        FriendId = g.Key,
        UnreadCount = g.Key == receiverId ? 0 : g.Count() // для
        відображення кількості непрочитаних повідомлень
    })
    .ToListAsync();

// Об'єднуємо у кінцевий список для передачі
var friendsWithMessages = friends
    .Join(friendIdsWithLastMessage,
        user => user.Id,
        msg => msg.FriendId,
        (user, msg) => new
        {
            user.Id,
            user.FullName,
            user.Image,
            LastMessageTime = (DateTime?)msg.LastMessageTime,
            LastMessage = msg.LastMessage,
            UnreadCount = friendsWithUnreadMessages
                .FirstOrDefault(f => f.FriendId == user.Id)?.UnreadCount ?? 0 //
            Кількість непрочитаних повідомлень для усіх у списку
        })
    .OrderByDescending(f => f.LastMessageTime)
    .ToList();

// Блок для можливості написати користувачу, якого немає у чатах, відкриває чат і
// поміщає його на верх списку
var isReceiverInList = friendsWithMessages.Any(f => f.Id == receiverId);
if (!isReceiverInList)
{
    var receiver = await _context.Users
        .Where(u => u.Id == receiverId)
        .Select(user => new
        {
            user.Id,
            user.FullName,
            user.Image
        })
        .FirstOrDefaultAsync();

    if (receiver != null)
    {
        var newFriendWithMessage = new
        {
            receiver.Id,
            receiver.FullName,
            receiver.Image,
            LastMessageTime = (DateTime?)null,
            LastMessage = "No messages yet",
            UnreadCount = 0
        };
        friendsWithMessages.Insert(0, newFriendWithMessage);
    }
}

// Передача списка користувачів, з яким спілкувався Юзер
ViewBag.Friends = friendsWithMessages;

// Блок для завантаження чату першого користувача у чаті

```

```

        if (!receiverId.HasValue)
        {
            var firstFriend = friendsWithMessages.FirstOrDefault();

            if (firstFriend != null)
            {
                receiverId = firstFriend.Id;
            }
        }

        // Передача повідомлень для Partial в Model
        var messages = await _context.ChatMessages
            .Where(m => (m.SenderId == userId && m.ReceiverId == receiverId)
                || (m.SenderId == receiverId && m.ReceiverId == userId))
            .OrderByDescending(m => m.Timestamp) // Спочатку сортуємо за останніми
            .Take(20) // Обмежуємо вибірку 10 повідомленнями
            .OrderBy(m => m.Timestamp) // Повертаємо до початкового порядку
            .ToListAsync();

        // Створюємо копію повідомлень для передачі в View
        var originalMessages = messages.Select(m => new ChatMessage
        {
            Id = m.Id,
            SenderId = m.SenderId,
            ReceiverId = m.ReceiverId,
            MessageText = m.MessageText,
            Timestamp = m.Timestamp,
            IsRead = m.IsRead,
            MessageImage = m.MessageImage // Зберігаємо початковий стан
        }).ToList();

        // Передача Id відправника і отримувача
        ViewBag.ReceiverId = receiverId;
        ViewBag.SenderId = userId;

        // При переході в чат, помітити всі повідомлення, як отримані
        var unreadMessages = messages.Where(m => m.ReceiverId == userId &&
!m.IsRead).ToList();
        if (unreadMessages.Any())
        {
            unreadMessages.ForEach(m => m.IsRead = true);
            await _context.SaveChangesAsync();
        }

        return View(originalMessages);
    }

    [HttpPost]
    public async Task<IActionResult> GetUnreadMessagesCount()
    {
        var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

        if (userId == null)
        {
            return RedirectToAction("Login", "Account");
        }

        int unreadMessagesCount = await _context.ChatMessages
            .Where(m => m.ReceiverId == int.Parse(userId) && !m.IsRead)
            .CountAsync();

        return Json(new { UnreadCount = unreadMessagesCount });
    }

    [HttpPost]
    public async Task<IActionResult> GetUnreadMessages([FromBody] int receiverId)
    {
        Console.WriteLine($"Received ReceiverId: {receiverId}");
        var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

```

```

        if (userId == null)
        {
            return RedirectToAction("Login", "Account");
        }

        // Позначаємо повідомлення від цього ReceiverId як прочитані
        var messagesFromReceiver = await _context.ChatMessages
            .Where(m => m.ReceiverId == int.Parse(userId) && m.SenderId == receiverId &&
!m.IsRead)
            .ToListAsync();

        if (messagesFromReceiver.Any())
        {
            foreach (var message in messagesFromReceiver)
            {
                message.IsRead = true;
            }

            await _context.SaveChangesAsync();
        }

        // Підраховуємо кількість непрочитаних повідомлень від усіх інших користувачів
        int unreadMessagesCount = await _context.ChatMessages
            .Where(m => m.ReceiverId == int.Parse(userId) && !m.IsRead)
            .CountAsync();

        return Json(new { UnreadCount = unreadMessagesCount });
    }

    [HttpPost]
    public IActionResult GetSenderInfo([FromBody] int senderId)
    {
        var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

        if (userId == null)
        {
            return RedirectToAction("Login", "Account");
        }

        // Шукаємо користувача в контексті за його Id
        var user = _context.Users.FirstOrDefault(u => u.Id == senderId);

        // Перевіряємо, чи знайдено користувача
        if (user == null)
        {
            return NotFound(new { message = "Користувач не знайдений" });
        }

        // Готуємо дані для відповіді
        var senderInfo = new
        {
            fullName = $"{user.FullName}", // ПІБ користувача
            image = Convert.ToBase64String(user.Image) // Перетворення зображення на base64
(якщо воно зберігається як байтовий масив)
        };

        // Повертаємо інформацію про користувача у форматі JSON
        return Ok(senderInfo);
    }

    [HttpPost]
    public async Task<IActionResult> UpdateChatPartial([FromBody] UpdateChatRequest
request)
    {
        var userIdstr = User.FindFirstValue(ClaimTypes.NameIdentifier);

        if (userIdstr == null)
        {

```

```

        return RedirectToAction("Login", "Account");
    }

    var userId = int.Parse(userIdstr);
    int? receiverId = request.ReceiverId;
    int count = request.Count;
    // БЛОК ЯКИЙ ФІЛЬТРУЄ І ГРУПУЄ КОРИСТУВАЧІВ ДЛЯ ОТРИМАННЯ НЕОБХІДНИХ ДАННИХ

    // Отримуємо ID друзів з останніми повідомленнями
    var friendIdsWithLastMessage = await _context.ChatMessages
        .Where(m => m.SenderId == userId || m.ReceiverId == userId)
        .GroupBy(m => m.SenderId == userId ? m.ReceiverId : m.SenderId)
        .Select(g => new
        {
            FriendId = g.Key, // ID друга
            LastMessageTime = g.OrderByDescending(m => m.Timestamp).Select(m =>
m.Timestamp).FirstOrDefault(), // для побудови списку
            LastMessage = g.OrderByDescending(m => m.Timestamp).Select(m =>
m.MessageText).FirstOrDefault() // для додавання під ПІБ користувача
        })
        .ToListAsync();

    // Отримуємо друзів за ID
    var friends = await _context.Users
        .Where(u => friendIdsWithLastMessage.Select(f => f.FriendId).Contains(u.Id))
        .Select(user => new
        {
            Id = user.Id,
            FullName = user.FullName,
            Image = user.Image
        })
        .ToListAsync();

    // Отримуємо кількість непрочитаних повідомлень для кожного друга
    var friendsWithUnreadMessages = await _context.ChatMessages
        .Where(m => m.ReceiverId == userId && !m.IsRead)
        .GroupBy(m => m.SenderId)
        .Select(g => new
        {
            FriendId = g.Key,
            UnreadCount = g.Key == receiverId ? 0 : g.Count() // для
відображення кількості непрочитаних повідомлень
        })
        .ToListAsync();

    // Об'єднуємо у кінцевий список для передачі
    var friendsWithMessages = friends
        .Join(friendIdsWithLastMessage,
            user => user.Id,
            msg => msg.FriendId,
            (user, msg) => new
            {
                user.Id,
                user.FullName,
                user.Image,
                LastMessageTime = (DateTime?)msg.LastMessageTime,
                LastMessage = msg.LastMessage,
                UnreadCount = friendsWithUnreadMessages
                    .FirstOrDefault(f => f.FriendId == user.Id)?.UnreadCount ?? 0 //
Кількість непрочитаних повідомлень для усіх у списку
            })
        .OrderByDescending(f => f.LastMessageTime)
        .ToList();

    // Блок для можливості написати користувачу, якого немає у чатах, відкриває чат і
    поміщає його на верх списку
    var isReceiverInList = friendsWithMessages.Any(f => f.Id == receiverId);

```

```

if (!isReceiverInList)
{
    var receiver = await _context.Users
        .Where(u => u.Id == receiverId)
        .Select(user => new
        {
            user.Id,
            user.FullName,
            user.Image
        })
        .FirstOrDefaultAsync();

    if (receiver != null)
    {
        var newFriendWithMessage = new
        {
            receiver.Id,
            receiver.FullName,
            receiver.Image,
            LastMessageTime = (DateTime?)null,
            LastMessage = "No messages yet",
            UnreadCount = 0
        };
        friendsWithMessages.Insert(0, newFriendWithMessage);
    }
}

// Передача списку користувачів, з яким спілкувався юзер
ViewBag.Friends = friendsWithMessages;

// Блок для завантаження чату першого користувача у чаті
if (!receiverId.HasValue)
{
    var firstFriend = friendsWithMessages.FirstOrDefault();

    if (firstFriend != null)
    {
        receiverId = firstFriend.Id;
    }
}

// Передача повідомлень для Partial в Model
var messages = await _context.ChatMessages
    .Where(m => (m.SenderId == userId && m.ReceiverId == receiverId)
        || (m.SenderId == receiverId && m.ReceiverId == userId))
    .OrderByDescending(m => m.Timestamp) // Спочатку сортуємо за останніми
    .Take(count) // Обмежуємо вибірку 10 повідомленнями
    .OrderBy(m => m.Timestamp) // Повертаємо до початкового порядку
    .ToListAsync();

// Створюємо копію повідомлень для передачі в View
var originalMessages = messages.Select(m => new ChatMessage
{
    Id = m.Id,
    SenderId = m.SenderId,
    ReceiverId = m.ReceiverId,
    MessageText = m.MessageText,
    Timestamp = m.Timestamp,
    IsRead = m.IsRead,
    MessageImage = m.MessageImage // Зберігаємо початковий стан
}).ToList();

// Передача Id відправника і отримувача
ViewBag.ReceiverId = receiverId;
ViewBag.SenderId = userId;

// При переході в чат, помітити всі повідомлення, як отримані

```

```

        var unreadMessages = messages.Where(m => m.ReceiverId == userId &&
!m.IsRead).ToList();
        if (unreadMessages.Any())
        {
            unreadMessages.ForEach(m => m.IsRead = true);
            await _context.SaveChangesAsync();
        }
        return PartialView("_ChatWithUser", messages);
    }

    [HttpPost]
    public IActionResult DeleteChat([FromBody] int friendId)
    {
        var userIdstr = User.FindFirstValue(ClaimTypes.NameIdentifier);

        if (userIdstr == null)
        {
            return RedirectToAction("Login", "Account");
        }

        var userId = int.Parse(userIdstr);

        // Знаходимо всі повідомлення між поточним користувачем і другом
        var messagesToDelete = _context.ChatMessages
            .Where(m => (m.SenderId == friendId && m.ReceiverId == userId) ||
                (m.SenderId == userId && m.ReceiverId == friendId))
            .ToList();

        if (messagesToDelete.Count == 0)
        {
            return NotFound("No chat messages found.");
        }

        // Видаляємо знайдені повідомлення
        _context.ChatMessages.RemoveRange(messagesToDelete);
        _context.SaveChanges();
        return Ok();
    }
}

```

Лістинг PaymentController.cs

```

using LiqPay.SDK.Dto;
using LiqPay.SDK;
using Microsoft.AspNetCore.Mvc;
using ArtChatean.Models;
using ArtChatean.Models.Forms;
using Microsoft.EntityFrameworkCore;
using System.Security.Cryptography.X509Certificates;
using Newtonsoft.Json;
using System.Text;
using LiqPay.SDK.Dto.Enums;
using static System.Runtime.InteropServices.JavaScript.JSType;
using Org.BouncyCastle.Asn1.X9;

public class PaymentController : Controller
{
    private readonly ArtDbContext _context;
    private readonly string _publicKey = "sandbox_i42953124367";
    private readonly string _privateKey =
"sandbox_GVIkr7ERzdyQfT6PiBWTZrwhzasXuuHvLCj0Fj09";

    public PaymentController(ArtDbContext context)
    {
        _context = context;
    }

    [HttpPost]

```

```

public async Task<IActionResult> CheckPayment([FromBody] int orderId)
{
    //Console.WriteLine("CALLBACK");
    var liqpay = new LiqPayClient(_publicKey, _privateKey);

    // Підготовка параметрів запиту
    var requestParams = new LiqPayRequest
    {
        Version = 3, //Версія Api
        Action = LiqPay.SDK.Dto.Enums.LiqPayRequestAction.Status, //Transaction type
        OrderId = orderId.ToString(), //Unique order number on your website
    };

    // Викликаємо асинхронний метод для отримання статусу платежу
    var response = await liqpay.RequestAsync("request", requestParams);
    Console.WriteLine(response.Status);
    if (response != null)
    {
        // Повертаємо статус як JSON
        return Json(new { status = response.Status.ToString() });
    }
    else
    {
        // Повертаємо повідомлення про помилку
        return Json(new { status = "No response from LiqPay." });
    }
}
}

```

Лістинг ShopController.cs

```

using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using System.Security.Claims;
using ArtChateau.Models;
using ArtChateau.Models.Forms;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Mvc.ApplicationModels;
using Microsoft.AspNetCore.Mvc.RazorPages;
using static Microsoft.EntityFrameworkCore.DbLoggerCategory;
using static System.Collections.Specialized.BitVector32;
using System.Linq;

namespace ArtChateau.Controllers;

public class ShopController : Controller
{
    private readonly ArtDbContext _context;
    private const int PageSize = 15;
    public ShopController(ArtDbContext context)
    {
        _context = context;
    }
    [HttpGet]
    public async Task<IActionResult> Main(string section)
    {
        ViewBag.SelectedSection = section ?? "PictureGallery";

        section = ViewBag.SelectedSection;

        var currentUserId = User.FindFirstValue(ClaimTypes.NameIdentifier);
        if (currentUserId == null)
        {
            return RedirectToAction("Login", "Account");
        }
    }
}

```

```

    }
    ViewBag.CurrentUserId = int.Parse(currentUserId);

    var pictures = GetFilteredPictures(null, int.Parse(currentUserId),
section).Include(p => p.PictureOrders).ThenInclude(po => po.Order).ThenInclude(o =>
o.Customer).ThenInclude(c => c.Addresses).Include(p => p.User).ToList();

    ViewBag.CurrentPage = 1;
    currentUserId = User.FindFirstValue(ClaimTypes.NameIdentifier);
    if (currentUserId == null)
    {
        return RedirectToAction("Login", "Account");
    }

    return View(pictures);
}
[HttpPost]
public async Task<IActionResult> Filter([FromBody] FilterModel? filters)
{
    var currentUserId = User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (currentUserId == null)
    {
        return RedirectToAction("Login", "Account");
    }

    ViewBag.CurrentUserId = int.Parse(currentUserId);

    var filteredPictures = GetFilteredPictures(filters, int.Parse(currentUserId),
"PictureGallery").ToList();

    currentUserId = User.FindFirstValue(ClaimTypes.NameIdentifier);
    if (currentUserId == null)
    {
        return RedirectToAction("Login", "Account");
    }

    ViewBag.CurrentPage = filters.Page;

    return PartialView("_PictureGallery", filteredPictures);
}
private IQueryable<Picture> GetFilteredPictures(FilterModel? filters, int
currentUserId, string section)
{
    var query = _context.Pictures.Include(p => p.Likes)
        .Include(p => p.PictureOrders).ThenInclude(o => o.Order)
        .Include(p => p.User)
        .Where(p => p.IsForSale && !p.IsSold && p.UserId != currentUserId)
        .AsQueryable();

    ViewBag.ValueForFilters = query.ToList();

    if (section == "PictureGallery")
    {
        if (filters != null)
        {
            if (filters.Style != null && filters.Style.Any())
            {
                query = query.Where(p => filters.Style.Contains(p.Style));
            }
            if (filters.Subject != null && filters.Subject.Any())
            {
                query = query.Where(p => filters.Subject.Contains(p.Subject));
            }
            if (filters.Medium != null && filters.Medium.Any())
            {
                query = query.Where(p => filters.Medium.Contains(p.Medium));
            }
        }
    }
}

```

```

    }
    if (filters.Material != null && filters.Material.Any())
    {
        query = query.Where(p => filters.Material.Contains(p.Material));
    }
    if (filters.Price != null && filters.Price.Count == 2)
    {
        decimal priceFrom = filters.Price[0] ?? 0;
        decimal priceTo = filters.Price[1] ?? decimal.MaxValue;

        query = query.Where(p => p.Price.HasValue && p.Price.Value >= priceFrom
&& p.Price.Value <= priceTo);
    }
    if (filters.Size != null && filters.Size.Any())
    {
        query = query.Where(p => filters.Size.Contains(p.Size));
    }
    if (filters.Orientation != null && filters.Orientation.Any())
    {
        query = query.Where(p => filters.Orientation.Contains(p.Orientation));
    }
    if (filters.Color != null && filters.Color.Any())
    {
        query = query.Where(p => filters.Color.Contains(p.Color));
    }
    if (!string.IsNullOrEmpty(filters.UserName))
    {
        query = query.Where(p => p.User.FullName != null &&
p.User.FullName.Contains(filters.UserName));
    }
    }

    ViewBag.TotalPages = (int)Math.Ceiling(query.Count() / (double)PageSize);

    if (filters == null)
    {
        return query.Skip(0).Take(PageSize);
    }

    int page = (filters.Page ?? 1) - 1;

    return query.Skip(page * PageSize).Take(PageSize);
}
else
{
    query = _context.Pictures.Where(p => p.UserId == currentUserId &&
p.IsForSale).AsQueryable();
    return query;
}
}
[HttpGet]
public async Task<IActionResult> Picture(int id)
{
    var currentUserId = User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (currentUserId == null)
    {
        return RedirectToAction("Login", "Account");
    }

    ViewBag.currentUserid = int.Parse(currentUserid);

    var picture = await _context.Pictures
        .Include(p => p.User)
        .Include(po => po.PictureOrders)
        .ThenInclude(o => o.Order)
        .Include(p => p.User.Pictures)

```

```

        .FirstOrDefaultAsync(p => p.Id == id);

    if (picture == null)
    {
        return NotFound();
    }

    return View(picture);
}
[HttpPost]
public async Task<IActionResult> Like([FromBody] int id)
{
    var userIdstr = User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (userIdstr == null)
    {
        return RedirectToAction("Login", "Account");
    }
    var userId = int.Parse(userIdstr);

    var picture = await _context.Pictures
        .Include(p => p.Likes)
        .FirstOrDefaultAsync(p => p.Id == id);

    if (picture == null)
    {
        return NotFound(new { success = false, message = "Picture not found." });
    }

    var existingLike = picture.Likes.FirstOrDefault(l => l.UserId == userId);

    if (existingLike != null)
    {
        _context.Likes.Remove(existingLike);
    }
    else
    {
        var newLike = new Like { UserId = userId, PictureId = id };
        _context.Likes.Add(newLike);
    }

    await _context.SaveChangesAsync();

    var likeCount = picture.Likes.Count;

    return Ok(new { success = true, likeCount });
}
[HttpPost]
public async Task<IActionResult> AddPicture(Picture model, IFormFile? Image)
{
    var currentUserId = User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (currentUserId == null)
    {
        return RedirectToAction("Login", "Account");
    }
    var user = await _context.Users.FindAsync(model.UserId);

    if (user == null)
    {
        return NotFound("User not found");
    }

    var newPicture = new Picture
    {
        Title = model.Title,
        Description = model.Description,

```

```

        Caption = model.Caption,
        Style = model.Style,
        Subject = model.Subject,
        Medium = model.Medium,
        Material = model.Material,
        Price = model.Price,
        Size = model.Size,
        Orientation = model.Orientation,
        Color = model.Color,
        UserId = model.UserId,
        User = user,
        IsForSale = true,
        IsForGallery = model.IsForGallery
    };

    if (Image != null && Image.Length > 0)
    {
        using (var memoryStream = new MemoryStream())
        {
            await Image.CopyToAsync(memoryStream);
            newPicture.Image = memoryStream.ToArray();
        }
    }

    _context.Pictures.Add(newPicture);
    await _context.SaveChangesAsync();

    return Redirect("/Shop/Main?section=AddEditPicture");
}
[HttpPost]
public async Task<IActionResult> EditPicture(Picture model, IFormFile? Image)
{
    var currentUserId = User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (currentUserId == null)
    {
        return RedirectToAction("Login", "Account");
    }
    var existingPicture = await _context.Pictures.FindAsync(model.Id);

    if (existingPicture == null)
    {
        return NotFound("Picture not found");
    }

    existingPicture.Title = model.Title;
    existingPicture.Description = model.Description;
    existingPicture.Caption = model.Caption;
    existingPicture.Style = model.Style;
    existingPicture.Subject = model.Subject;
    existingPicture.Medium = model.Medium;
    existingPicture.Material = model.Material;
    existingPicture.Price = model.Price;
    existingPicture.Size = model.Size;
    existingPicture.Orientation = model.Orientation;
    existingPicture.Color = model.Color;
    existingPicture.IsForSale = true;
    existingPicture.IsForGallery = model.IsForGallery;

    if (Image != null && Image.Length > 0)
    {
        using (var memoryStream = new MemoryStream())
        {
            await Image.CopyToAsync(memoryStream);
            existingPicture.Image = memoryStream.ToArray();
        }
    }
}

```

```

        _context.Pictures.Update(existingPicture);
        await _context.SaveChangesAsync();

        return Redirect($"/Shop/Main?section=AddEditPicture");
    }
    [HttpPost]
    public async Task<IActionResult> DeletePicture([FromBody] int id)
    {
        var currentUserId = User.FindFirstValue(ClaimTypes.NameIdentifier);

        if (currentUserId == null)
        {
            return RedirectToAction("Login", "Account");
        }
        var picture = await _context.Pictures.FindAsync(id);

        if (picture == null)
        {
            return NotFound();
        }

        _context.Pictures.Remove(picture);
        await _context.SaveChangesAsync();
        return Ok();
    }
}

```

Лістинг OrderController.cs

```

using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using System.Security.Claims;
using ArtChateau.Models;
using ArtChateau.Models.Forms;
using Microsoft.EntityFrameworkCore;
using static System.Collections.Specialized.BitVector32;
using static System.Runtime.InteropServices.JavaScript.JSType;
using LiqPay.SDK.Dto;
using LiqPay.SDK;
using MailKit.Search;

namespace ArtChateau.Controllers;

public class OrderController : Controller
{
    private readonly ArtDbContext _context;
    private readonly IEmailServiceMailKit _emailServiceMailKit;
    private readonly string _publicKey = "sandbox_i42953124367";
    private readonly string _privateKey =
"sandbox_GVIkr7ERzdyQfT6PiBWTZrwhzasXuuHvLCj0Fj09";
    public OrderController(ArtDbContext context, IEmailServiceMailKit emailServiceMailKit)
    {
        _context = context;
        _emailServiceMailKit = emailServiceMailKit;
    }

    [HttpGet]
    public async Task<IActionResult> Main(string section)
    {
        ViewBag.SelectedSection = section ?? "ActiveOrders";
        var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

        if (userId == null)
        {

```

```

        return RedirectToAction("Login", "Account");
    }

    var userIdInt = int.Parse(userId);
    var user = await _context.Users
        .Include(u => u.Addresses)
        .Include(u => u.Payments)
        .Include(u => u.Orders.Where(o => o.CustomerId == userIdInt))
        .ThenInclude(o => o.PictureOrders)
        .ThenInclude(po => po.Picture).ThenInclude(u => u.User)
        .FirstOrDefaultAsync(u => u.Id == int.Parse(userId));

    if (user.Orders == null)
    {
        return NotFound();
    }
    return View(user);
}

[HttpGet]
public async Task<IActionResult> Checkout()
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (userId == null)
    {
        return RedirectToAction("Login", "Account");
    }

    var userIdInt = int.Parse(userId);

    var user = await _context.Users
        .Include(u => u.Addresses)
        .Include(u => u.Payments)
        .Include(u => u.Orders
            .Where(o => o.CustomerId == userIdInt && o.IsCompleted == false))
        .ThenInclude(o => o.PictureOrders)
        .ThenInclude(po => po.Picture)
        .ThenInclude(p => p.User)
        .FirstOrDefaultAsync(u => u.Id == int.Parse(userId));

    if (user.Orders == null)
    {
        return NotFound();
    }

    var httpContextAccessor =
        HttpContext.RequestServices.GetService(typeof(IHttpContextAccessor)) as
        IHttpContextAccessor;

    var order = user.Orders.FirstOrDefault();

    var liqpay = new LiqPayClient(_publicKey, _privateKey);

    string baseUrl = HttpContext.Request.Scheme + "://" +
        httpContextAccessor.HttpContext.Request.Host.Value;

    //Payment parameters
    var paymentParams = new LiqPayRequest
    {
        Version = 3, //Версия Api
        Action = LiqPay.SDK.Dto.Enums.LiqPayRequestAction.Pay, //Transaction type
        Amount = (double)order.PictureOrders.Sum(po => po.Picture.Price), //Payment
        amount
        Sandbox = "1",
        Currency = "USD", //Payment currency
    }

```

```

        Description = "Оплата тестового продукта - #" + order.Number, //Payment
Description
        OrderId = order.Id.ToString(), //Unique order number on your website
        ResultUrl = baseUrl + "/Order?section=DoneOrders", //Link to successful payment
page
    };

    //We get a generated form with Data, Signature and a button for payment
    var formCode = liqpay.CNBForm(paymentParams);

    ViewBag.FormCode = formCode;
    Console.WriteLine(formCode);
    return View(user);
}

[HttpGet]
public async Task<IActionResult> Finish()
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (userId == null)
    {
        return RedirectToAction("Login", "Account");
    }

    var userIdInt = int.Parse(userId);

    var user = await _context.Users
        .Include(u => u.Addresses)
        .Include(u => u.Payments)
        .Include(u => u.Orders
            .Where(o => o.CustomerId == userIdInt))
            .ThenInclude(o => o.PictureOrders)
            .ThenInclude(po => po.Picture)
            .ThenInclude(p => p.User)
            .FirstOrDefaultAsync(u => u.Id ==
int.Parse(userId));

    if (user.Orders == null)
    {
        return NotFound();
    }

    return View(user);
}

[HttpPost]
public async Task<IActionResult> DeletePictureOrder([FromBody] int pictureOrderId)
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (userId == null)
    {
        return RedirectToAction("Login", "Account");
    }
    try
    {
        var pictureOrder = await _context.PictureOrders
            .SingleOrDefaultAsync(po => po.Id == pictureOrderId);
        if (pictureOrder != null)
        {
            var order = await _context.Orders.FindAsync(pictureOrder.OrderId);
            _context.PictureOrders.Remove(pictureOrder);
            await _context.SaveChangesAsync();

            var remainingPictures = _context.PictureOrders.Where(po => po.OrderId ==
order.Id).Any();

```

```

        if (!remainingPictures)
        {
            _context.Orders.Remove(order);
            await _context.SaveChangesAsync();
        }

        return Ok();
    }
    return NotFound("Picture order not found.");
}
catch (Exception ex)
{
    return StatusCode(500, "An error occurred while deleting the picture order.");
}
}
[HttpPost]
public async Task<IActionResult> CompleteOrder([FromBody] int id)
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (userId == null)
    {
        return RedirectToAction("Login", "Account");
    }
    var order = await _context.Orders
        .Include(o => o.PictureOrders)
        .ThenInclude(po => po.Picture)
        .ThenInclude(p => p.User)
        .FirstOrDefaultAsync(o => o.Id == id);

    if (order == null)
    {
        return NotFound();
    }

    bool hasConflict = false;

    foreach (var pictureOrder in order.PictureOrders)
    {
        if (pictureOrder.Picture.IsSold)
        {
            hasConflict = true;
        }
    }

    if (hasConflict)
    {
        return Conflict("Some pictures were already sold.");
    }

    foreach (var pictureOrder in order.PictureOrders)
    {
        pictureOrder.Picture.IsSold = true;
    }

    order.IsCompleted = true;

    await _context.SaveChangesAsync();

    foreach (var pictureOrder in order.PictureOrders)
    {
        var sellerEmail = pictureOrder.Picture.User.Email;
        var sellerName = pictureOrder.Picture.User.FullName;
        var orderNumder = pictureOrder.Order.Number;

        if (!string.IsNullOrEmpty(sellerEmail))

```

```

        {
            var emailSubject = "Artwork Sold";
            var emailBody = $"Dear {sellerName}, your artwork
{pictureOrder.Picture.Title} has been sold. Order № {orderNumber}. You can view it in your
Sold Pictures.";

            // Використання MailKit для відправки
            await _emailServiceMailKit.SendEmailAsync(
                sellerEmail,
                emailSubject,
                emailBody
            );
        }
    }

    return Ok();
}

[HttpPost]
public async Task<IActionResult> AddToOrder([FromBody] int id)
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (userId == null)
    {
        return RedirectToAction("Login", "Account");
    }
    var user = await _context.Users
        .Include(u => u.Orders)
        .FirstOrDefaultAsync(u => u.Id == int.Parse(userId));

    if (user == null)
    {
        return NotFound();
    }

    var picture = await _context.Pictures.FindAsync(id);
    if (picture == null)
    {
        return NotFound();
    }

    var order = user.Orders.FirstOrDefault(o => !o.IsCompleted);
    if (order == null)
    {
        order = new Order
        {
            CustomerId = user.Id,
            OrderDate = DateTime.Now,
            IsCompleted = false,
            Number = "ORD-" + DateTime.Now.ToString("yyyyMMddHHmmss"),
            SellerId = picture.UserId
        };
        _context.Orders.Add(order);
        await _context.SaveChangesAsync();
    }

    var pictureOrder = new PictureOrder
    {
        OrderId = order.Id,
        PictureId = picture.Id
    };

    _context.PictureOrders.Add(pictureOrder);
    await _context.SaveChangesAsync();

    return Ok();
}

```

```

[HttpPost]
public async Task<IActionResult> SaveAddress(Address address)
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (userId == null)
    {
        return RedirectToAction("Login", "Account");
    }

    var userIdInt = int.Parse(userId);

    var user = await _context.Users.Include(u => u.Addresses).FirstOrDefaultAsync(u =>
u.Id == userIdInt);

    if (user != null)
    {
        var existingAddress = user.Addresses.FirstOrDefault(a => a.Id == address.Id);
        if (existingAddress != null)
        {
            existingAddress.Country = address.Country;
            existingAddress.PostalCode = address.PostalCode;
            existingAddress.City = address.City;
            existingAddress.Street = address.Street;
        }
        else
        {
            var newAddress = new Address
            {
                Country = address.Country,
                PostalCode = address.PostalCode,
                City = address.City,
                Street = address.Street,
                User = user
            };
            user.Addresses.Add(newAddress);
        }
        await _context.SaveChangesAsync();
    }
    else
    {
        TempData["Message"] = "User not found.";
        return Redirect("/Order/Main?section=PaymentMethods");
    }

    TempData["Message"] = "Address saved successfully!";

    return Redirect("/Order/Main?section=Addresses");
}

[HttpPost]
public async Task<IActionResult> SavePayment(Payment payment)
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (userId == null)
    {
        return RedirectToAction("Login", "Account");
    }

    var userIdInt = int.Parse(userId);

    var user = await _context.Users.Include(u => u.Payments)
        .FirstOrDefaultAsync(u => u.Id == userIdInt);

    if (user != null)
    {
        var existingPayment = user.Payments.FirstOrDefault(p => p.Id == payment.Id);
    }
}

```

```

        if (existingPayment != null)
        {
            existingPayment.CardNumber = payment.CardNumber;
            existingPayment.CardHolderName = payment.CardHolderName;
            existingPayment.ExpiryDate = payment.ExpiryDate;
            existingPayment.CVV = payment.CVV;
        }
        else
        {
            var newPayment = new Payment
            {
                CardNumber = payment.CardNumber,
                CardHolderName = payment.CardHolderName,
                ExpiryDate = payment.ExpiryDate,
                CVV = payment.CVV
            };
            user.Payments.Add(newPayment);
        }
        await _context.SaveChangesAsync();
    }
    else
    {
        TempData["Message"] = "User not found.";
        return Redirect("/Order/Main?section=PaymentMethods");
    }
    TempData["Message"] = "Payment method saved successfully!";
    return Redirect("/Order/Main?section=PaymentMethods");
}
}

```

Лістинг AccountController.cs

```

using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using System.Security.Claims;
using ArtChateau.Models;
using ArtChateau.Models.Forms;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Authentication.Cookies;

namespace ArtChateau.Controllers;
public class AccountController : Controller
{
    private readonly UserManager<User> _userManager;
    private readonly SignInManager<User> _signInManager;
    private readonly RoleManager<IdentityRole<int>> _roleManager;
    private readonly ArtDbContext _context;
    private readonly IEmailServiceMailKit _emailServiceMailKit;

    public AccountController(UserManager<User> userManager, SignInManager<User>
signInManager, RoleManager<IdentityRole<int>> roleManager, ArtDbContext context,
IEmailServiceMailKit emailServiceMailKit)
    {
        _roleManager = roleManager;
        _userManager = userManager;
        _signInManager = signInManager;
        _context = context;
        _emailServiceMailKit = emailServiceMailKit;
    }

    [HttpGet]
    public async Task<IActionResult> Login()
    {
        var count = await _context.Pictures.CountAsync();
        var randomIndex = new Random().Next(0, count);
    }
}

```

```

        var randomPicture = await
_context.Pictures.Skip(randomIndex).FirstOrDefaultAsync();
        ViewBag.RandomPicture = randomPicture;
        return View(new LoginForm());
    }

    [HttpGet]
    public async Task<IActionResult> Register()
    {
        var count = await _context.Pictures.CountAsync();
        var randomIndex = new Random().Next(0, count);
        var randomPicture = await
_context.Pictures.Skip(randomIndex).FirstOrDefaultAsync();
        ViewBag.RandomPicture = randomPicture;
        return View(new RegisterForm());
    }

    [HttpPost]
    public async Task<IActionResult> Login([FromForm] LoginForm form)
    {
        // Генерація випадкової картинки
        var count = _context.Pictures.Count();
        var randomIndex = new Random().Next(0, count);
        var randomPicture = _context.Pictures.Skip(randomIndex).FirstOrDefault();
        ViewBag.RandomPicture = randomPicture;

        // Знаходження користувача за email
        var user = await _userManager.FindByEmailAsync(form.Email);
        if (user == null)
        {
            ModelState.AddModelError(nameof(form.Email), "User not found!");
            return View(form);
        }
        // Перевірка, чи підтверджено електронну пошту
        if (!await _userManager.IsEmailConfirmedAsync(user))
        {
            ModelState.AddModelError(nameof(form.Email), "Please confirm your email before
logging in!");
            return View(form);
        }

        // Перевірка пароля і логін
        var result = await _signInManager.PasswordSignInAsync(user.UserName, form.Password,
false, false);
        if (!result.Succeeded)
        {
            ModelState.AddModelError(nameof(form.Email), "Invalid password!");
            return View(form);
        }

        if (form.RememberMe)
        {
            var authProperties = new AuthenticationProperties
            {
                IsPersistent = true,
                ExpiresUtc = DateTimeOffset.UtcNow.AddDays(30)
            };
            await _signInManager.SignInAsync(user, authProperties);
        }
        else
        {
            await _signInManager.SignInAsync(user, isPersistent: false);
        }

        // Створення ролі Admin, якщо вона не існує
        var adminRole = await _roleManager.FindByNameAsync("Admin");
        if (adminRole == null)

```

```

    {
        await _roleManager.CreateAsync(new IdentityRole<int>() { Name = "Admin" });
    }

    // Додавання користувача до ролі Admin, якщо він ще не є в цій ролі
    if (!await _userManager.IsInRoleAsync(user, "Admin"))
    {
        await _userManager.AddToRoleAsync(user, "Admin");
    }

    // Перенаправлення на головну сторінку
    return Redirect("/Main/Main");
}

[HttpPost]
public async Task<IActionResult> Register([FromForm] RegisterForm form)
{
    var count = _context.Pictures.Count();
    var randomIndex = new Random().Next(0, count);
    var randomPicture = _context.Pictures.Skip(randomIndex).FirstOrDefault();
    ViewBag.RandomPicture = randomPicture;

    if (!ModelState.IsValid)
    {
        return View(form);
    }

    var user = await _userManager.FindByEmailAsync(form.Email);

    if (user != null)
    {
        ModelState.AddModelError(nameof(form.Email), "Зазначений користувач уже існує!");
        return View(form);
    }

    user = new User
    {
        Email = form.Email,
        FullName = form.FullName,
        UserName = form.UserName
    };

    var result = await _userManager.CreateAsync(user, form.Password);

    if (!result.Succeeded)
    {
        ModelState.AddModelError(nameof(form.Email),
            string.Join(";", result.Errors.ToList().Select(e => e.Description))
        );
        return View(form);
    }

    var token = await _userManager.GenerateEmailConfirmationTokenAsync(user);

    // Створення посилання для підтвердження
    var confirmationLink = Url.Action("ConfirmEmail", "Account", new { userId =
user.Id, token = token }, Request.Scheme);

    // Відправка листа з підтвердженням через MailKit
    await _emailServiceMailKit.SendEmailAsync(
        user.Email,
        "Email Confirmation",
        $"Please confirm your email by clicking on the link: {confirmationLink}."
    );
    TempData["ConfirmationMessage"] = "To complete your registration, please confirm
your email. A confirmation email has been sent to your email address.";
}

```

```

        return RedirectToAction("Login", "Account");
    }

    [HttpGet]
    public async Task<IActionResult> ConfirmEmail(string userId, string token)
    {
        var count = _context.Pictures.Count();
        var randomIndex = new Random().Next(0, count);
        var randomPicture = _context.Pictures.Skip(randomIndex).FirstOrDefault();
        ViewBag.RandomPicture = randomPicture;

        if (userId == null || token == null)
        {
            return RedirectToAction("Register", "Account");
        }

        var user = await _userManager.FindByIdAsync(userId);
        if (user == null)
        {
            return NotFound($"Не вдалося знайти користувача з ID '{userId}'.");
        }

        var result = await _userManager.ConfirmEmailAsync(user, token);
        if (result.Succeeded)
        {
            return View("ConfirmEmailSuccess"); // або перенаправлення на сторінку
            успішного підтвердження
        }

        return View("Error");
    }

    [HttpPost]
    [Authorize]
    public async Task<IActionResult> Logout()
    {
        await HttpContext.SignOutAsync(IdentityConstants.ApplicationScheme);
        return RedirectToAction("Login", "Account");
    }

    // СКИДАННЯ ПАРОЛЮ
    [HttpGet]
    public async Task<IActionResult> ForgotPassword()
    {
        var count = await _context.Pictures.CountAsync();
        var randomIndex = new Random().Next(0, count);
        var randomPicture = await
        _context.Pictures.Skip(randomIndex).FirstOrDefaultAsync();
        ViewBag.RandomPicture = randomPicture;
        return View();
    }

    // Обробка форми скидання пароля
    [HttpPost]
    public async Task<IActionResult> ForgotPassword(ForgotPasswordViewModel model)
    {
        var count = await _context.Pictures.CountAsync();
        var randomIndex = new Random().Next(0, count);
        var randomPicture = await
        _context.Pictures.Skip(randomIndex).FirstOrDefaultAsync();
        ViewBag.RandomPicture = randomPicture;

        if (!ModelState.IsValid)
        {
            return View(model);
        }
    }

```

```

        Console.WriteLine(model.Email);

        var user = await _userManager.FindByEmailAsync(model.Email);
        if (user == null || !await _userManager.IsEmailConfirmedAsync(user))
        {
            // Не розкривайте, чи існує користувач
            return RedirectToAction("ForgotPasswordConfirmation", "Account");
        }

        // Генерація токена для скидання пароля
        var token = await _userManager.GeneratePasswordResetTokenAsync(user);
        var callbackUrl = Url.Action("ResetPassword", "Account", new { token, email =
user.Email }, protocol: HttpContext.Request.Scheme);

        // Надсилання електронного листа через ваш сервіс
        await _emailServiceMailKit.SendEmailAsync(model.Email, "Password Reset",
            $"You can reset your password by clicking on the link: {callbackUrl}.");

        return RedirectToAction("ForgotPasswordConfirmation", "Account");
    }

    // Підтвердження скидання пароля
    [HttpGet]
    public async Task<IActionResult> ForgotPasswordConfirmation()
    {
        var count = await _context.Pictures.CountAsync();
        var randomIndex = new Random().Next(0, count);
        var randomPicture = await
_context.Pictures.Skip(randomIndex).FirstOrDefaultAsync();
        ViewBag.RandomPicture = randomPicture;

        return View();
    }

    // Вибображення сторінки для скидання пароля
    [HttpGet]
    public async Task<IActionResult> ResetPassword(string token = null, string email =
null)
    {
        var count = await _context.Pictures.CountAsync();
        var randomIndex = new Random().Next(0, count);
        var randomPicture = await
_context.Pictures.Skip(randomIndex).FirstOrDefaultAsync();
        ViewBag.RandomPicture = randomPicture;

        return token == null || email == null ? View("Error") : View(new
ResetPasswordViewModel { Email = email, Token = token });
    }

    // Обробка скидання пароля
    [HttpPost]
    public async Task<IActionResult> ResetPassword(ResetPasswordViewModel model)
    {
        var count = await _context.Pictures.CountAsync();
        var randomIndex = new Random().Next(0, count);
        var randomPicture = await
_context.Pictures.Skip(randomIndex).FirstOrDefaultAsync();
        ViewBag.RandomPicture = randomPicture;

        if (!ModelState.IsValid)
        {
            return View(model);
        }

        var user = await _userManager.FindByEmailAsync(model.Email);
        if (user == null)
        {

```

```

        return RedirectToAction("ResetPasswordConfirmation");
    }
    var result = await _userManager.ResetPasswordAsync(user, model.Token,
model.NewPassword);
    if (result.Succeeded)
    {
        return RedirectToAction("ResetPasswordConfirmation");
    }

    foreach (var error in result.Errors)
    {
        ModelState.AddModelError(string.Empty, error.Description);
    }
    return View(model);
}
// Підтвердження скидання пароля
[HttpGet]
public async Task<IActionResult> ResetPasswordConfirmation()
{
    var count = await _context.Pictures.CountAsync();
    var randomIndex = new Random().Next(0, count);
    var randomPicture = await
_context.Pictures.Skip(randomIndex).FirstOrDefaultAsync();
    ViewBag.RandomPicture = randomPicture;
    return View();
}
}
}

```

Лістинг MainController.cs

```

using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using System.Security.Claims;
using ArtChateau.Models;
using ArtChateau.Models.Forms;
using Microsoft.EntityFrameworkCore;
using static System.Net.Mime.MediaTypeNames;

namespace ArtChateau.Controllers;

public class MainController : Controller
{
    private readonly ArtDbContext _context;
    private readonly UserManager<User> _userManager;
    private readonly SignInManager<User> _signInManager;
    public MainController(ArtDbContext context, UserManager<User> userManager,
SignInManager<User> signInManager)
    {
        _context = context;
        _userManager = userManager;
        _signInManager = signInManager;
    }
    [HttpGet]
    public async Task<IActionResult> GetUser()
    {
        var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

        if (userId == null)
        {
            return RedirectToAction("Login", "Account");
        }

        var user = await _context.Users
            .FirstOrDefaultAsync(u => u.Id == int.Parse(userId));
    }
}

```

```

        string base64Image = user.Image != null
        ? Convert.ToBase64String(user.Image)
        : null;

        // Повертаємо дані користувача
        return Json(new
        {
            Id = user.Id,
            Name = user.FullName,
            AvatarBase64 = base64Image
        });
    }

    [HttpPost]
    public async Task<IActionResult> ChangePassword([FromBody] ChangePasswordViewModel
model)
    {
        var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
        var user = await _userManager.FindByIdAsync(userId);

        if (user == null)
        {
            return Json(new { success = false, message = "User not found." });
        }

        if (model.NewPassword != model.ConfirmPassword)
        {
            return Json(new { success = false, message = "New password and confirmation do
not match." });
        }

        var result = await _userManager.ChangePasswordAsync(user, model.OldPassword,
model.NewPassword);

        if (!result.Succeeded)
        {
            var errors = result.Errors.Select(e => e.Description).ToList();
            return Json(new { success = false, message = string.Join(", ", errors) });
        }

        await _signInManager.RefreshSignInAsync(user);
        return Json(new { success = true });
    }

    [HttpGet]
    public async Task<IActionResult> Main()
    {
        var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

        if (userId == null)
        {
            return RedirectToAction("Login", "Account");
        }
        ViewBag.UserId = int.Parse(userId);
        ViewBag.UserName = (await _context.Users
            .Where(u => u.Id == int.Parse(userId))
            .FirstOrDefaultAsync())?.FullName;
        var friends = await _context.UserFriends
            .Include(uf => uf.Friend)
            .Where(uf => uf.UserId == int.Parse(userId)).Take(7)
            .ToListAsync();
        ViewBag.Friends = friends;

        var friendIds = await _context.UserFriends
            .Where(uf => uf.UserId == int.Parse(userId))
            .Select(uf => uf.FriendId)
            .ToListAsync();
    }

```

```

var friendPictures = await _context.Pictures
    .Where(p => friendIds.Contains(p.User.Id))
    .OrderByDescending(p => p.CreatedAt)
    .Select(p => new
    {
        p.Id,
        p.Image,
        p.Title,
        p.CreatedAt,
        p.Caption,
        LikeCount = p.Likes.Count(),
        CommentCount = p.Comments.Count(),
        UserImage = p.User.Image,
        UserFullName = p.User.FullName,
        UserId = p.User.Id,
        IsLikedByCurrentUser = p.Likes.Any(l => l.UserId ==
int.Parse(userId)),
        IsSavedByCurrentUser = p.Saves.Any(s => s.UserId ==
int.Parse(userId)),
        Comments = p.Comments.Select(c => new
        {
            c.Id,
            c.UserId,
            c.Text,
            c.CreatedAt,
            UserCommentFullName = c.User.FullName,
            UserCommentId = c.User.Id
        }).ToList()
    })
    .Take(10).ToListAsync();

return View(friendPictures);
}
[HttpPost]
public async Task<IActionResult> GetPosts([FromBody] int count)
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (userId == null)
    {
        return RedirectToAction("Login", "Account");
    }
    ViewBag.UserId = int.Parse(userId);
    ViewBag.UserName = (await _context.Users
        .Where(u => u.Id == int.Parse(userId))
        .FirstOrDefaultAsync())?.FullName;
    var friends = await _context.UserFriends
        .Include(uf => uf.Friend)
        .Where(uf => uf.UserId == int.Parse(userId)).Take(7)
        .ToListAsync();
    ViewBag.Friends = friends;

    var friendIds = await _context.UserFriends
        .Where(uf => uf.UserId == int.Parse(userId))
        .Select(uf => uf.FriendId)
        .ToListAsync();

    var friendPictures = await _context.Pictures
        .Where(p => friendIds.Contains(p.User.Id))
        .OrderByDescending(p => p.CreatedAt)
        .Select(p => new
        {
            p.Id,
            p.Image,
            p.Title,
            p.CreatedAt,

```

```

        p.Caption,
        LikeCount = p.Likes.Count(),
        CommentCount = p.Comments.Count(),
        UserImage = p.User.Image,
        UserFullName = p.User.FullName,
        UserId = p.User.Id,
        IsLikedByCurrentUser = p.Likes.Any(l => l.UserId ==
int.Parse(userId)),
        IsSavedByCurrentUser = p.Saves.Any(s => s.UserId ==
int.Parse(userId)),
        Comments = p.Comments.Select(c => new
        {
            c.Id,
            c.UserId,
            c.Text,
            c.CreatedAt,
            UserCommentFullName = c.User.FullName,
            UserCommentId = c.User.Id
        }).ToList()
    })
    .Skip(count).Take(10).ToListAsync();

    return PartialView("_FriendsPosts", friendPictures);
}
[HttpGet]
public async Task<IActionResult> FriendMain(int id, int pictureId)
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (userId == null)
    {
        return RedirectToAction("Login", "Account");
    }
    ViewBag.UserId = int.Parse(userId);
    ViewBag.UserName = (await _context.Users
        .Where(u => u.Id == int.Parse(userId))
        .FirstOrDefaultAsync())?.FullName;
    var friendPictures = await _context.Pictures
        .Where(p => p.UserId == id && p.IsForGallery == true)
        .OrderByDescending(p => p.CreatedAt)
        .Select(p => new
        {
            p.Id,
            p.Image,
            p.Title,
            p.CreatedAt,
            p.Caption,
            LikeCount = p.Likes.Count(),
            CommentCount = p.Comments.Count(),
            UserImage = p.User.Image,
            UserFullName = p.User.FullName,
            UserId = p.User.Id,
            IsLikedByCurrentUser = p.Likes.Any(l => l.UserId ==
int.Parse(userId)),
            IsSavedByCurrentUser = p.Saves.Any(s => s.UserId ==
int.Parse(userId)),
            Comments = p.Comments.Select(c => new
            {
                c.Id,
                c.UserId,
                c.Text,
                c.CreatedAt,
                UserCommentFullName = c.User.FullName,
                UserCommentId = c.User.Id
            }).ToList()
        })
        .ToListAsync();
}

```

```

        ViewBag.HighlightedPictureId = pictureId;
        return View(friendPictures);
    }

    [HttpGet]
    public async Task<IActionResult> Profile()
    {
        var userIdstr = User.FindFirstValue(ClaimTypes.NameIdentifier);

        if (userIdstr == null)
        {
            return RedirectToAction("Login", "Account");
        }
        var userId = int.Parse(userIdstr);

        var totalLikes = await _context.Pictures
            .Where(p => p.UserId == userId)
            .SelectMany(p => p.Likes)
            .CountAsync();

        var friendCount = await _context.UserFriends
            .CountAsync(uf => uf.UserId == userId);

        var userLikes = await _context.Pictures
            .GroupBy(p => p.UserId)
            .Select(g => new
            {
                UserId = g.Key,
                LikesCount = g.Sum(p => p.Likes.Count())
            })
            .OrderByDescending(x => x.LikesCount)
            .ToListAsync();

        var userRank = userLikes
            .Select((x, index) => new { x.UserId, Rank = index + 1 })
            .FirstOrDefault(x => x.UserId == userId);

        ViewBag.TotalLikes = totalLikes;
        ViewBag.FriendCount = friendCount;
        ViewBag.Rank = userRank != null ? userRank.Rank : 0;

        var user = await _context.Users
            .Include(u => u.Pictures.Where(p => p.IsForGallery))
            .FirstOrDefaultAsync(u => u.Id == userId);

        if (user == null)
        {
            return NotFound("User not found");
        }

        return View(user);
    }

    [HttpGet]
    public async Task<IActionResult> FriendProfile(int id)
    {
        var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

        if (userId == null)
        {
            return RedirectToAction("Login", "Account");
        }

        if (int.Parse(userId) == id)

```

```

    {
        return RedirectToAction("Profile", "Main");
    }

    var friend = await _context.Users
        .Include(u => u.Pictures.Where(p => p.IsForGallery))
        .FirstOrDefaultAsync(u => u.Id == id);

    if (friend == null)
    {
        return NotFound("Friend not found");
    }

    var user = await _context.Users
        .Include(u => u.UserFriends)
        .ThenInclude(uf => uf.Friend)
        .FirstOrDefaultAsync(u => u.Id == int.Parse(userId));

    if (user == null)
    {
        return NotFound("User not found");
    }

    ViewBag.User = user;

    var totalLikes = await _context.Pictures
        .Where(p => p.UserId == friend.Id)
        .SelectMany(p => p.Likes)
        .CountAsync();

    var friendCount = await _context.UserFriends
        .CountAsync(uf => uf.UserId == friend.Id);

    var userLikes = await _context.Pictures
        .GroupBy(p => p.UserId)
        .Select(g => new
        {
            UserId = g.Key,
            LikesCount = g.Sum(p => p.Likes.Count())
        })
        .OrderByDescending(x => x.LikesCount)
        .ToListAsync();

    var userRank = userLikes
        .Select((x, index) => new { x.UserId, Rank = index + 1 })
        .FirstOrDefault(x => x.UserId == friend.Id);

    ViewBag.TotalLikes = totalLikes;
    ViewBag.FriendCount = friendCount;
    ViewBag.Rank = userRank != null ? userRank.Rank : 0;

    return View(friend);
}

[HttpGet]
public async Task<IActionResult> Setting()
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (userId == null)
    {
        return RedirectToAction("Login", "Account");
    }

    var user = await _context.Users.FindAsync(int.Parse(userId));

    if (user == null)
    {

```

```

        return NotFound("User not found");
    }

    return View(user);
}

[HttpGet]
public async Task<IActionResult> SearchUsers(string query)
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (userId == null)
    {
        return RedirectToAction("Login", "Account");
    }

    var users = await _context.Users
        .Where(u => u.FullName.Contains(query) && u.Id != int.Parse(userId))
        .Select(u => new
        {
            u.Id,
            u.FullName,
            u.Image
        })
        .ToListAsync();

    return Json(users);
}

[HttpPost]
public async Task<IActionResult> AddComment(IFormCollection form)
{
    var userIdstr = User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (userIdstr == null)
    {
        return RedirectToAction("Login", "Account");
    }

    var userId = int.Parse(userIdstr);
    var pictureId = Convert.ToInt32(form["PictureId"]);
    var text = form["Text"].ToString();

    if (string.IsNullOrEmpty(text))
    {
        return BadRequest(new { success = false, message = "Invalid comment data." });
    }

    var newComment = new Comment
    {
        PictureId = pictureId,
        UserId = userId,
        Text = text,
        CreatedAt = DateTime.Now
    };

    await _context.Comments.AddAsync(newComment);
    await _context.SaveChangesAsync();

    var user = await _context.Users.FindAsync(newComment.UserId);
    var commentCount = await _context.Comments.CountAsync(c => c.PictureId ==
pictureId);

    return Ok(new { success = true, username = user.FullName, commentCount, commentId =
newComment.Id });
}

```

```

[HttpPost]
public async Task<IActionResult> Like([FromBody] int id)
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (userId == null)
    {
        return RedirectToAction("Login", "Account");
    }
    var picture = await _context.Pictures.Include(p => p.Likes).FirstOrDefaultAsync(p
=> p.Id == id);

    if (picture == null)
    {
        return NotFound(new { success = false, message = "Picture not found." });
    }

    var existingLike = picture.Likes.FirstOrDefault(l => l.UserId ==
int.Parse(userId));

    if (existingLike != null)
    {
        _context.Likes.Remove(existingLike);
    }
    else
    {
        var newLike = new Like { UserId = int.Parse(userId), PictureId = id };
        await _context.Likes.AddAsync(newLike);
    }

    await _context.SaveChangesAsync();
    var likeCount = picture.Likes.Count();

    return Ok(new { success = true, likeCount });
}

public async Task<IActionResult> Save([FromBody] int id)
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (userId == null)
    {
        return RedirectToAction("Login", "Account");
    }
    var picture = await _context.Pictures.Include(p => p.Saves).FirstOrDefaultAsync(p
=> p.Id == id);

    if (picture == null)
    {
        return NotFound(new { success = false, message = "Picture not found." });
    }

    var existingSave = picture.Saves.FirstOrDefault(l => l.UserId ==
int.Parse(userId));

    if (existingSave != null)
    {
        _context.Saves.Remove(existingSave);
    }
    else
    {
        var newSave = new Save { UserId = int.Parse(userId), PictureId = id };
        await _context.Saves.AddAsync(newSave);
    }

    await _context.SaveChangesAsync();
}

```

```

        return Ok(new { success = true });
    }

    [HttpGet]
    public async Task<IActionResult> AddPicture()
    {
        var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

        if (userId == null)
        {
            return RedirectToAction("Login", "Account");
        }

        var user = await _context.Users
            .Include(u => u.Pictures)
            .FirstOrDefaultAsync(u => u.Id == int.Parse(userId));

        if (user == null)
        {
            return NotFound("User not found");
        }

        return View(user);
    }

    [HttpPost]
    public async Task<IActionResult> ToggleFollow([FromBody] int id)
    {
        var currentUserId = User.FindFirstValue(ClaimTypes.NameIdentifier);

        if (currentUserId == null)
        {
            return RedirectToAction("Login", "Account");
        }

        var currentUser = await _context.Users
            .Include(u => u.UserFriends)
            .FirstOrDefaultAsync(u => u.Id == int.Parse(currentUserId));

        if (currentUser == null)
        {
            Console.WriteLine("Current user not found.");
            return NotFound("User not found");
        }

        var friend = await _context.Users.FirstOrDefaultAsync(u => u.Id == id);
        if (friend == null)
        {
            Console.WriteLine($"Friend with ID {id} not found.");
            return NotFound("Friend not found");
        }

        var existingFriendship = currentUser.UserFriends
            .FirstOrDefault(uf => uf.FriendId == id);

        if (existingFriendship != null)
        {
            _context.UserFriends.Remove(existingFriendship);
        }
        else
        {
            await _context.UserFriends.AddAsync(new UserFriend
            {
                UserId = currentUser.Id,
                FriendId = id
            });
        }
    }

```

```

        await _context.SaveChangesAsync();
        return Ok();
    }

    [HttpPost]
    public async Task<IActionResult> Setting(User model, IFormFile? Image)
    {
        var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

        if (userId == null)
        {
            return RedirectToAction("Login", "Account");
        }

        var user = await _userManager.FindByIdAsync(userId);
        if (user == null)
        {
            return NotFound("User not found");
        }

        // Перевірка унікальності email
        var existingUser = await _userManager.FindByEmailAsync(model.Email);
        if (existingUser != null && existingUser.Id != int.Parse(userId))
        {
            //ModelState.AddModelError(nameof(model.Email), "Зазначений користувач уже існує!");
            ModelState.AddModelError("Email", "Цей email вже використовується іншим користувачем.");
            return View(model);
        }

        // Оновлення полів профілю
        user.FullName = model.FullName;
        user.Email = model.Email; // _userManager автоматично нормалізує Email
        user.Paypal = model.Paypal;
        user.PhoneNumber = model.PhoneNumber;
        user.Bio = model.Bio;
        user.Location = model.Location;
        user.Website = model.Website;
        user.Instagram = model.Instagram;
        user.Youtube = model.Youtube;

        if (Image != null && Image.Length > 0)
        {
            using (var memoryStream = new MemoryStream())
            {
                await Image.CopyToAsync(memoryStream);
                user.Image = memoryStream.ToArray();
            }
        }

        var result = await _userManager.UpdateAsync(user);
        if (!result.Succeeded)
        {
            // Обробка помилок, якщо оновлення не вдалося
            foreach (var error in result.Errors)
            {
                ModelState.AddModelError("", error.Description);
            }
            return View(model);
        }

        return Redirect("/Main/Profile");
    }

    [HttpPost]
    public async Task<IActionResult> AddPicture(IFormFile? Image, string Caption)
    {

```

```

var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
if (userId == null)
{
    return RedirectToAction("Login", "Account");
}

var user = await _context.Users.FindAsync(int.Parse(userId));
if (user == null)
{
    return NotFound("User not found");
}

var newPicture = new Picture
{
    Caption = Caption,
    IsForGallery = true,
    User = user
};

if (Image != null && Image.Length > 0)
{
    using (var memoryStream = new MemoryStream())
    {
        await Image.CopyToAsync(memoryStream);
        newPicture.Image = memoryStream.ToArray();
    }
}
else
{
    return BadRequest("No image file provided.");
}

_context.Pictures.Add(newPicture);
await _context.SaveChangesAsync();

return Ok();
}
[HttpPost]
public async Task<IActionResult> DeletePicture([FromBody] int id)
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (userId == null)
    {
        return RedirectToAction("Login", "Account");
    }
    var picture = await _context.Pictures.FindAsync(id);
    if (picture == null)
    {
        return NotFound();
    }

    _context.Pictures.Remove(picture);
    await _context.SaveChangesAsync();

    return Ok();
}
[HttpPost]
public async Task<IActionResult> SalePicture([FromBody] int id)
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

    if (userId == null)
    {
        return RedirectToAction("Login", "Account");
    }
    var picture = await _context.Pictures.FindAsync(id);

```

```

        if (picture == null)
        {
            return NotFound();
        }

        picture.IsForSale = !picture.IsForSale;

        _context.Pictures.Update(picture);
        await _context.SaveChangesAsync();

        return Ok();
    }
    [HttpGet]
    public async Task<IActionResult> LikedProfilePictures()
    {
        var currentUserId = User.FindFirstValue(ClaimTypes.NameIdentifier);

        if (currentUserId == null)
        {
            return RedirectToAction("Login", "Account");
        }

        int userId = int.Parse(currentUserId);

        var likedPictures = await _context.Pictures
            .Where(p => p.Likes.Any(l => l.UserId == userId) && p.UserId != userId &&
p.IsForGallery == true)
            .Include(p => p.User)
            .OrderByDescending(p => p.Likes.FirstOrDefault(l => l.UserId ==
userId).Id)
            .ToListAsync();

        // Якщо немає лайкнутих картин
        if (!likedPictures.Any())
        {
            return View("LikedProfilePictures", new List<Picture>());
        }

        return View("LikedProfilePictures", likedPictures);
    }

    [HttpGet]
    public async Task<IActionResult> LikedShopPictures()
    {
        var currentUserId = User.FindFirstValue(ClaimTypes.NameIdentifier);

        if (currentUserId == null)
        {
            return RedirectToAction("Login", "Account");
        }

        int userId = int.Parse(currentUserId);

        var likedPictures = await _context.Pictures
            .Where(p => p.Likes.Any(l => l.UserId == userId) && p.IsForSale == true)
            .Include(p => p.User)
            .OrderByDescending(p => p.Likes.FirstOrDefault(l => l.UserId ==
userId).Id)
            .ToListAsync();

        // Якщо немає лайкнутих картин
        if (!likedPictures.Any())
        {
            return View("LikedShopPictures", new List<Picture>());
        }
    }

```

```

        return View("LikedShopPictures", likedPictures);
    }

    [HttpGet]
    public async Task<IActionResult> SavedProfilePictures()
    {
        var currentUserId = User.FindFirstValue(ClaimTypes.NameIdentifier);

        if (currentUserId == null)
        {
            return RedirectToAction("Login", "Account");
        }

        int userId = int.Parse(currentUserId);

        var savedPictures = await _context.Pictures
            .Where(p => p.Saves.Any(s => s.UserId == userId) && p.IsForGallery == true)
            .Include(p => p.User)
            .OrderByDescending(p => p.Saves.FirstOrDefault(l => l.UserId ==
userId).Id)
            .ToListAsync();

        // Якщо немає лайкнутих картин
        if (!savedPictures.Any())
        {
            return View("SavedProfilePictures", new List<Picture>());
        }

        return View("SavedProfilePictures", savedPictures);
    }

    [HttpPost]
    public async Task<IActionResult> DeleteComment([FromBody] int id)
    {
        var currentUserId = User.FindFirstValue(ClaimTypes.NameIdentifier);

        if (currentUserId == null)
        {
            return RedirectToAction("Login", "Account");
        }

        var comment = await _context.Comments.FirstOrDefaultAsync(c => c.Id == id);

        if (comment == null)
        {
            return NotFound("Comment not found");
        }

        _context.Comments.Remove(comment);
        await _context.SaveChangesAsync();

        return Ok();
    }

    [HttpPost]
    public async Task<IActionResult> GetUsers([FromBody] int pictureId)
    {
        Console.WriteLine(pictureId);

        var users = await _context.Likes
            .Where(like => like.PictureId == pictureId)
            .Select(like => new {
                Id = like.User.Id, // Додаємо Id користувача
                FullName = like.User.FullName
            })
            .ToListAsync(); // Асинхронний виклик

        return Json(users);
    }
}

```