МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ

(підпис)

09 червня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 – Комп'ютерних наук,

освітньо- професійної програми «Інформатика»

на тему: «Тематичний інтернет-портал "Галерея патріота"»

здобувача групи ІН-92 Кравченка Артема Андрійовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Артем КРАВЧЕНКО

(підпис)

Керівник
заступник декана з інформатизації,
доцент,
кандидат технічних наук Сергій ПЕТРОВ _____

(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-92 Кравченка Артема Андрійовича

1. Тема роботи: «Тематичний інтернет-портал "Галерея патріота"»
затверджую наказом по СумДУ від *«01» червня 2023 р. № 0475-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 09 червня 2023 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, й формування завдань дослідження. 2) Огляд ресурсів аналогів, пошук оптимальних реалізацій, огляд програмного забезпечення для розробки веб-сайтів. 3) Розробка тематичного інтернет порталу «Галерея Патріота». 4) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проєкту (роботи), із значенням розділів проєкту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, формування завдань дослідження*		
2	*Огляд ресурсів аналогів, пошук оптимальних реалізацій, огляд програмного забезпечення для розробки веб-сайтів*		
3	*Розробка тематичного інтернет порталу «Галерея Патріота»*		
4	*Аналіз отриманих результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 54 стр., 21 рис., 1 додаток, 13 використаних джерел.

Обґрунтування актуальності теми роботи – Тематика кваліфікаційної роботи актуальна у зв'язку з загальними подіями у світі. Постала необхідність у створенні власного національного простору для самовираження українських митців без тотального політичного контролю з боку іноземних компаній, що лоббують власні інтереси, використовуючи цензуру, та водночас допомогти збройним силам України, відкривши додаткову гілку волонтерського руху.

Об'єкт дослідження – платформа, що надає можливість українським художникам представляти свої роботи та взаємодіяти зі спільнотою.

Мета роботи – розробка повноцінного національного тематичного інтернет порталу галереї для митців з можливістю грошових переказів.

Методи дослідження – аналіз конкурентних ресурсів-аналогів.

Результати – розроблено повноцінний тематичний інтернет портал «Галерея патріота» в якому українські митці можуть без остраху висловлювати всі свої думки та просувати патріотичні месседжі, паралельно допомагаючи збройним силам України завдяки системі донатів.

ТЕМАТИЧНИЙ ІНТЕРНЕТ-ПОРТАЛ "ГАЛЕРЕЯ ПАТРІОТА"

ЗМІСТ

ВСТУП

Актуальність. Актуальність кваліфікаційної роботи полягає в необхідності власного, національного простору для розкриття творчих здібностей. Особливо важливим є створення платформи, що дозволить українським художникам представляти свої твори та взаємодіяти з аудиторією без зовнішнього впливу та обмежень, враховуючи сучасну політичну ситуацію. Беручи до уваги той факт, що багато існуючих платформ та соціальних мереж, контролюються іноземними компаніями та інколи підлягають жорсткій модерації, цілеспрямовано пригнічуючи свободу слова для українців, постає необхідність створення національного ресурсу, що забезпечить вільне самовираження та незалежність українських художників.

Мета дослідження. Створення національної платформи для художників, враховуючи методи і підходи вже наявних продуктів аналогів.

Завдання дослідження. Для втілення заданої мети в життя виділено наступні завдання:

- дослідити основні підходи по створенню веб-сайтів;
- дослідити ресурси аналоги та виділити найуспішніші реалізації потреб;
- розробити веб-сайт;

Об’єкт дослідження. Незалежна платформа, що надає можливість українським художникам представляти свої роботи та взаємодіяти зі спільнотою.

Предмет дослідження. Функціонал та можливості веб-сайту "Галерея патріота".

Методи дослідження. Аналіз конкурентів, проведено аналіз існуючих веб-порталів та соціальних мереж, спрямованих на представлення мистецтва, зокрема платформ, що пропонують функціонал схожий з "Галереєю патріота". Це дозволило виявити основні переваги та недоліки існуючих рішень та визначити можливість створення конкурентноспроможного продукту.

Практична значимість. Проект "Галерея патріота" має потенціал для розвитку та створення позитивного впливу на українське мистецтво, сприяючи новим творчим здобуткам художників та зміцненню патріотичних цінностей, водночас допомагаючи країні у скрутний час як додатковий грошовий потік у благодійні фонди через підключену систему донату.

Структура. Кваліфікаційна робота «Тематичний інтернет портал Галерея патріота» складається з наступних розділів: вступ, аналітичний огляд, моделювання та проектування, розробка веб-сайту, висновків, списку використаних джерел та додатків.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Основні підходи до створення веб-сайтів

Загалом підходи до створення веб-сайтів можна охарактеризувати декількома основними етапами, які притаманні для проектів будь якої складності:

1) **Аналіз вимог:** Визначення і аналіз вимог є першим етапом при створенні веб-сайту. Він включає в себе дослідження потреб користувачів, цілей проекту та функціональних вимог до веб-сайту. Результати аналізу вимог формують основу для подальшої розробки та планування;
2) **Макет та дизайн:** Розробка ефективного та зручного інтерфейсу є важливим аспектом при створенні веб-сайту. На цьому етапі визначається зовнішній вигляд та структура сторінок, розміщення елементів, вибір кольорової палітри, шрифтів та інших візуальних елементів, що забезпечують зручну навігацію та привабливість веб-сайту;
3) **Вибір технологій та програмного забезпечення:** Оцінивши вимоги та потреби поставленої задачі можна підібрати комбінацію технологій на програмних засобів, які підходять для оптимального вирішення краще за все;
4) **Розробка, тестування та оптимізація:** Найважливіший етап де програмісти створюють функціональність веб-сайту на основі вимог та дизайну. Після цього веб-сайт піддається тестуванню, що включає перевірку на належну роботу всіх функцій, а також перевірку безпеки та продуктивності.

Проте підходи до самої реалізації можуть бути різні, адже сайти бувають різних типів:

1) **Статичні веб-сайти:** Складаються з набору HTML-сторінок, які зберігаються на сервері і безпосередньо відображаються користувачам. Для створення статичних сайтів використовуються HTML, CSS та

JavaScript. Такий підхід використовують для невеликих проектів без складного функціоналу;

2) **Динамічні веб-сайти:** Сайти побудовані на основі серверних технологій, таких як PHP, Python, Ruby, ASP.NET тощо. Ці технології дозволяють генерувати вміст веб-сторінок на основі запитів користувачів та обробляти дані з баз даних. Динамічні сайти забезпечують більшу гнучкість та можливості, такі як інтерактивність, обробка форм, аутентифікація користувачів;
3) **Контент-менеджерські системи (CMS):** CMS, такі як WordPress, Joomla, Drupal, є популярними рішеннями для швидкого створення та керування веб-сайтами. Вони надають готові шаблони та функціонал для створення різноманітних типів сайтів, включаючи блоги, новинні портали, електронні магазини тощо. CMS дозволяють керувати контентом та візуально налаштовувати вигляд сайту без програмування;
4) **Frontend-фреймворки:** Frontend-фреймворки, такі як React, Angular та Vue.js, дозволяють розробникам будувати складні веб-інтерфейси та взаємодіяти з даними без перезавантаження сторінок. Вони пропонують готові компоненти, роутинг, станові менеджери та інші інструменти для покращення продуктивності та розширюваності;
5) **Backend-фреймворки:** Backend-фреймворки, такі як Django, Ruby on Rails, Express.js, надають зручний спосіб розробки серверної логіки веб-сайту. Вони допомагають управляти маршрутизацією, обробкою запитів, взаємодією з базами даних та реалізацією бізнес-логіки.

Враховуючи власний досвід та знання, а також комплексність платформи, для задоволення всіх необхідних потреб в проекті використано саме комбінацію backend та frontend фреймворків, а також представлені всі основні етапи створення веб-сайту.

1.2 Аналіз продуктів аналогів

Створення власного національного простору в мережі інтернет є актуальним на сьогоднішній день, оскільки світ в цілому наразі занадто покладається на західні розробки, що викликає велику залежність від зовнішньої політики та контролю. Для прикладу можна взяти ту ж всесвітньо відому платіжну систему PayPal, що інтегрується в Україні вже багато років і досі є не повністю інтегрованою. Проте, щоб стати конкурентноспроможним у цьому світі, потрібно або пропонувати щось геть нове та цікаве, або використовувати максимальну комбінацію вже наявних успішних рішень.

Серед наявних широковідомих сайтів галерей для художників є:

1) **Artstation:** Веб-платформа, спеціалізована на представленні творчості художників, дизайнерів, аніматорів та інших творчих професій. Надає можливість створювати власні профілі-портфоліо. Користувачі можуть додавати свої проекти, ілюстрації, моделі, концепції та інші витвори мистецтва. Профілі підтримують розділи для опису та контактної інформації. Має активну спільноту художників, де користувачі можуть обговорювати свої роботи, ділитися порадами та взаємодіяти один з одним. Це стимулює творчий обмін інформацією та можливість отримати зворотний зв'язок від інших учасників. Налаштовано зручні інструменти для пошуку творчих робіт. Користувачі можуть шукати за ключовими словами, категоріями, авторами та іншими фільтрами. Є можливість перегляду зображень в повному розмірі, коментування, виставлення лайків та зберігання улюблених робіт. Також має інтегрований магазин, де художники можуть продавати свої роботи, включаючи принти, моделі, текстури, плагіни та інші цифрові активи. Це надає можливість монетизувати творчість та залучити зацікавлених покупців;
2) **DeviantArt:** Одна з найбільш відомих та найстаріших платформ для художників, заснована ще у 2000-му році. DeviantArt також надає можливості взаємодії та спілкування з іншими художниками через

коментарі та повідомлення. Також має функціонал для створення портфоліо, пошуку та збереженню робіт, та навіть створення персональних «інтернет експозицій», в яких користувачі створюють ніби власний музей з робіт інших художників. Останнім часом також додано функціонал платної підписки, що дозволяє художникам виставляти деякий контент, перегляд якого можливий тільки за гроші.

3) **Behance:** Також одна з провідних платформ для дизайнерів та творчих професій, проте відрізняється більш «солідним» підходом саме до питань працевлаштування митців, надаючи зручні інструменти для роботодавців для пошуку потенційних робітників. Надає можливість створювати та виставляти свої проекти, включаючи графічний дизайн, фотографію, ілюстрацію та багато іншого;
4) **Dribbble:** Спеціалізується на дизайні і пропонує платформу саме для дизайнерів. Дозволяє створювати портфоліо, додавати зображення та інші візуальні матеріали. Dribbble також активно використовується як засіб зв'язку між дизайнерами, де вони можуть ділитися коментарями та взаємодіяти один з одним;
5) **CGSociety:** Спеціалізована платформа для художників комп'ютерної графіки та візуальних ефектів. Пропонує можливості публікації та обговорення художніх робіт, а також надає доступ до навчальних матеріалів, відомостей про вакансії та інших спеціалізованих ресурсів.

Таблиця 1.1 – Порівняння найпопулярніших веб-сайтів галерей

Сайт	**Переваги**	**Недоліки**
1	**2**	**3**
Artstation	Зручні профілі портфоліо, зручні інструменти редагування, інтегрований магазин, активна спільнота художників.	Сильна модерація української свободи слова включно до видалення небажаних робіт. Досить громіздкий, повільна швидкість роботи.

DeviantArt	Можливість платної підписки, інтернет експозиції. Дуже велика спільнота з усього світу.	Настільки слабка модерація, що ресурс погруз в бруді та небажаному контенті.
Behance	Добре налаштовані фільтри релевантності, інструменти для роботодавців.	У більшості орієнтований саме на виробничий дизайн. Не найкращі рішення в реалізації саме перегляду робіт.
Dribbble	Швидкість роботи, також інструменти роботодавця.	Дуже вузький профіль в плані тематики. Не зручне портфоліо.
CGSociety	Добре розвинена спільнота художників комп'ютерної графіки.	Проблеми з доступністю в Україні, занадто вузький профіль.

Спираючись на таблицю 1.1 порівняння найпопулярніших веб-сайтів галерей кожен з розглянутих ресурсів має свої переваги та недоліки. У виборі успішних прикладів реалізацій деякого функціоналу, здавалось би, можна покластись на популярність ресурсу. Наприклад, за офіційними даними, найпопулярніші сайти з представлених це DeviantArt, за офіційним карьерним звітом від 15 вересня 2020-го року від самої компанії – 48 мільйонів користувачів[45], та Behance, за даними Abobe Blog 10.20.2020 – 24 мільйони користувачів[45]. Проте якісь українськи культурні чи національні меседжі простежувались саме на платформах, де щільність українських митців була більша.

Таким сайтом серед проаналізованих був Artstation, тож найближчим до ідеї проекту «Галерея патріота» є саме ця веб-платформа.

Однак в цього порталу є один великий недолік, через те, що компанія велика, її сфери інтересів часто корелюються з ситуацією в світі і під виглядом «мистецтво поза політикою» сайт часто модерується зі схильністю до замовчування українських месседжів, постійно намагаючись налаштовувати фільтри так, щоб роботи наших митців не потрапляли на головну сторінку, видаляють політичні коментарі, а іноді навіть і самі роботи. Це навіть призвело до деякої місцевої революції та кількох бунтів, в результаті чого, до речі,

платформу покинуло більше саме іноземних професійних художників, ніж українських. Тим не менше ресурс непоганий в плані реалізації всіх потреб митця, тому його і взято за приклад, однак деякі розділи, наприклад маркетплейс і інструменти редагування зображень реалізовані не будуть.

1.3 Постановка задачі

Мета проекту – розробити простий, але функціональний національний веб ресурс на якому українськи митці зможуть просувати будь які свої месседжі.

Для реалізації поставленої задачі були пройдені такі етапи:

1) Знайомство з необхідною літературою по розробці веб-сайтів;
2) Огляд ресурсів аналогів;
3) Обрано підхід до реалізації;
4) Обрано середовище розробки, мову реалізації та програмне забезпечення;
5) Розроблено веб-сайт.

Цільовою аудиторією проекту будуть художники та люди, що цікавляться мистецтвом в інтернет просторі, отже сайт повинен мати інтуїтивно знайомий інтерфейс з легкою навігацією, до якого вже звикли користувачі мережі, що користуються наявними продуктами аналогами.

Основним функціоналом сайту повинна бути змога створювати власне портфоліо та викладати там роботи, отже як мінімум є необхідність в реалізації функціоналу реєстрації, авторизації та створення сесій, а також підключення взаємодії з файлами картинок.

Додатковий функціонал – зв'язок між поміж спільнотою, в першу чергу коментарі, лайки та лічильник кількості переглядів робіт для фідбеку. Автори коментарів повинні мати змогу редагувати та видаляти свої дописи, так, як і художники редагувати чи видаляти свої пости.

Одна з головних поставлених цілей це грошова допомога армії ЗСУ, тож є також необхідність у функціоналі, що реалізує її. Для цього спершу треба обрати зручну платіжну систему, що буде охоплювати не лише місцевих, але й іноземців,

після чого реалізувати взаємодію з формою через кнопку.

Так як постів може бути велика кількість – важлива реалізація пагінації як на головній сторінці, так і на сторінках портфоліо.

Отже портал повинен складатися з клієнтської на серверної частини, а також бази даних. Серверна частина відповідає за роботу веб-сервера, з'єднання з базою даних, обробку запитів, роутинг та керуванням сесіями користувачів. Клієнтська частина відповідає за зовнішній вигляд, функціонал та інтерактивність.

1.4 Середовище розробки та програмне забезпечення

Visual Studio Code (VS Code) є одним з найпопулярніших та потужних текстових редакторів для розробки програмного забезпечення. Розроблений компанією Microsoft, VS Code став вкрай популярним серед розробників з усього світу.

Має чистий та сучасний інтерфейс, що спрощує роботу з редактором. Головне вікно поділене на різні панелі, такі як файловий провідник, редактор коду, вікна відладки та консоль. Навігація здійснюється шляхом кліків та використання швидких клавіш, що робить роботу з VS Code швидкою та ефективною.

VS Code має потужну систему розширень, яка дозволяє розробникам налаштовувати та розширювати можливості редактора. З великим розмаїттям доступних розширень, розробники можуть встановлювати плагіни для підтримки різних мов програмування, фреймворків, інструментів відладки, систем керування версіями та багато іншого.

Деякі з розширень та плагінів, що було використано у проекті:

1) **ES7+ React/Redux/React-Native snippets:** корисне розширення для розробників, які працюють з технологіями React, Redux та React Native. Воно надає набір скорочень (snippets) - швидких та зручних шаблонів коду, які допомагають прискорити процес розробки та забезпечують

стандартизацію стилю коду;

2) **eCSStractor for VSCode:** розширення, що підтримує масовк виділення стилів, дозволяє виділяти стилі одночасно з кількох елементів або класів, що прискорює процес рефакторингу з багатьох елементах одночасно.

VS Code надає розширені можливості редактора коду, такі як підсвічування синтаксису, автодоповнення, відступи, переміщення по коду, пошук та заміна, розгортання функцій, рефакторинг та багато іншого. Він також підтримує інтеграцію з системами керування версіями, такими як Git. Має вбудовані інструменти для відладки коду, що дозволяють розробникам встановлювати точки зупину, крокувати по коду, переглядати значення змінних та стежити за виконанням програми.

Docker - платформа, що надає можливість виконувати програми в ізольованих контейнерах і дуже добре підходить для створення та управління серверів баз даних.

pgAdmin - безкоштовний інструмент для адміністрування та управління PostgreSQL базами даних. Надає зручний графічний інтерфейс, що дозволяє розробникам та адміністраторам легко взаємодіяти з базами даних та виконувати різноманітні операції, включаючи запити, моніторинг та резервне копіювання.

Insomnia - кросплатформений інструмент для тестування та налагодження HTTP запитів. Надає зручне середовище для взаємодії з веб-серверами та отримання відповідей в реальному часі. Insomnia спрощує процес відправки запитів, перевірки статусу, перегляду та аналізу даних, що повертаються сервером.

1.5 Платіжна система

В якості платіжної системи для проведення грошових переказів однозначно було обрано PayPal за рахунок своєї всесвітньої популярності, легкості міжнародних переказів та тепер більшій доступності в Україні. Також у виборі саме цієї системи зіграло не менш важливу роль необхідність залучення

іноземців до донатів, котрі вже звикли нею користуватись і легкий інструментарій PayPal по автоматичному створенню персональних форм для проведення платежів, котрі можна легко інтегрувати у свій сайт.

За даними самого PayPal його клієнтська база налічує 246 мільйонів користувачів станом на 2018 рік [45].

2 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ

2.1 Розробка макету

Оскільки сайт представляє собою галерею, то в першу чергу він повинен висвітлювати роботи митців вже на головній сторінці, та бути простим у навігації від портфоліо до портфоліо, тому було створено макет на кшталт Artstation.com.

Головна сторінка складається з хедеру та контентної частини, на якій висвітлені роботи та пагінації, рисунок 2.1. Також на головній сторінці присутній основний мотиваційно інформаційний пост на кнопка переходу на форму донату на ЗСУ.

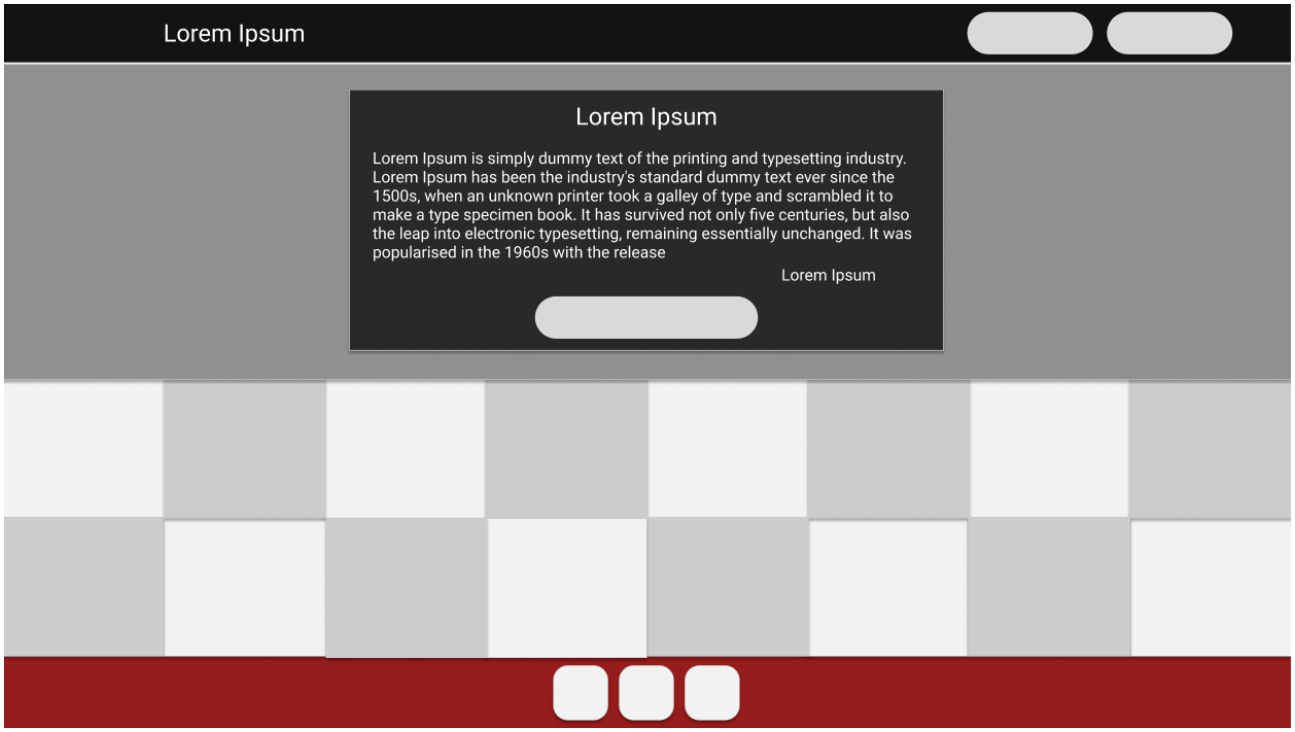

Рисунок 2.1 – Макет головної сторінки сайту

Окрім головної сайт звісно має особисту сторінку користувача яка одночасно являється його портфоліо, та додаткову приватну сторінку особистого кабінету, де він може змінити інформацію про себе. Окрему форму для додавання та редагування поста, а також сторінки логіну та реєстрації. Кожний пост можна відкрити також окремою сторінкою, на якій є можливість залишити коментар,

перейти на профіль автора, поставити вподобайку роботі, або через окрему форму провести грошовий переказ.

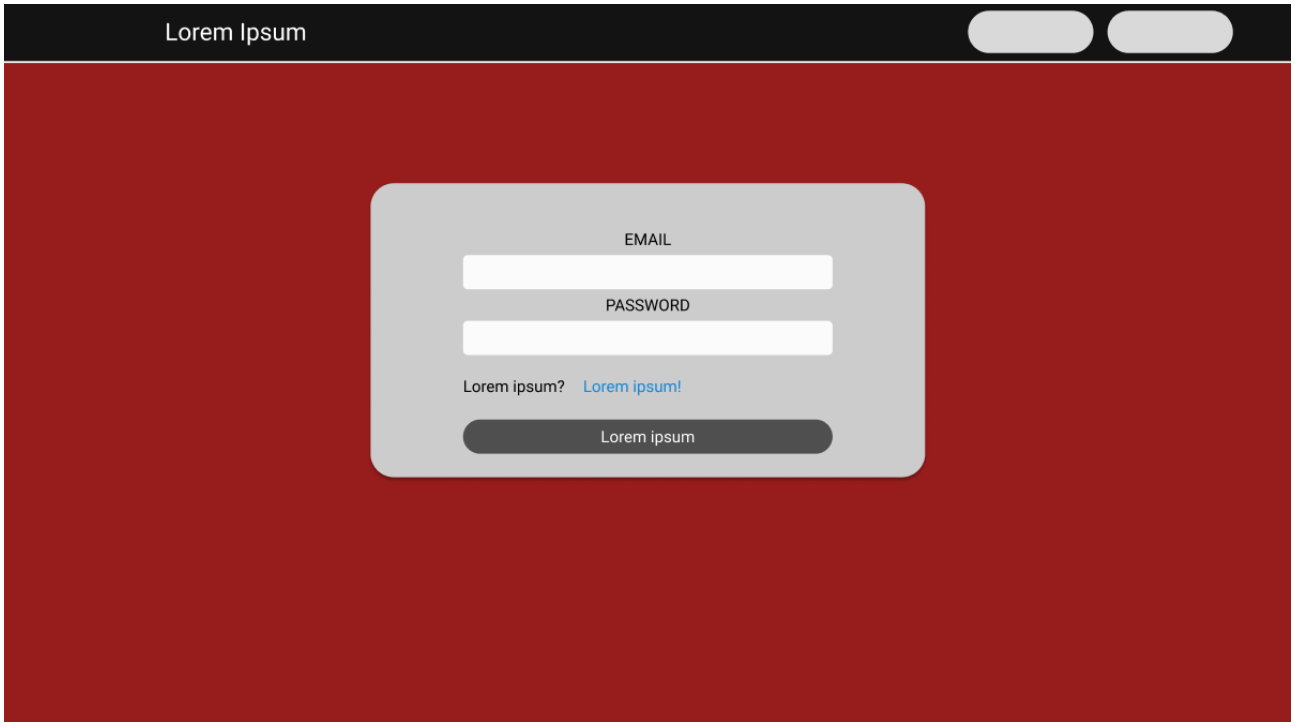

Рисунок 2.2 – Макет сторінки авторизації

Авторизація, рисунок 2.2 , та реєстрація, рисунок 2.3 , користувачів виконується на окремій сторінці, що змінює поля в залежності від того авторизується користувач чи створює новий аккаунт.

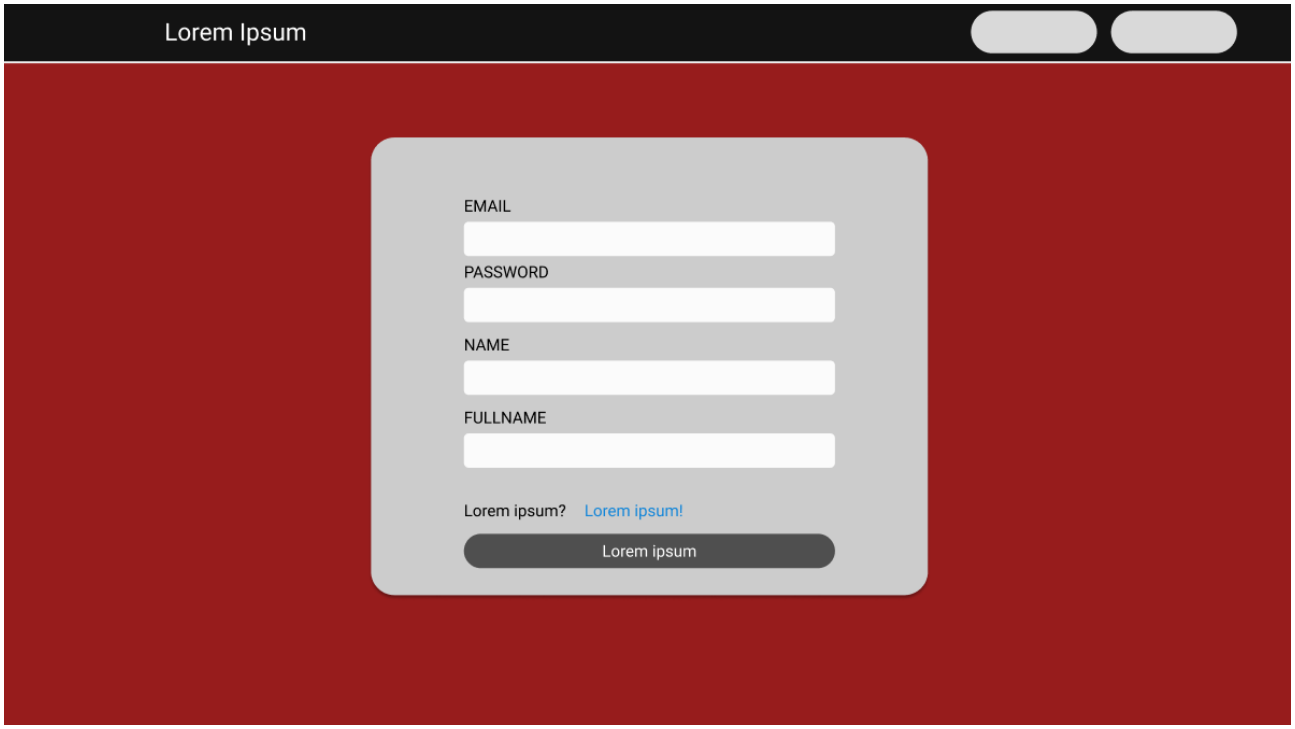

Рисунок 2.3 – Макет сторінки реєстрації

В портфоліо надається загальна інформація про художника, включно з його повним ім’ям, аватаркою та нікнеймом, а також всі його викладені роботи, як це показано на рисунку 2.4 . Якщо сторінку переглядає сам автор, то в нього з’являються також кнопки на додавання нового посту, або ж кнопка переходу до особистих даних. Також присутня пагінація по для списку особистих постів.

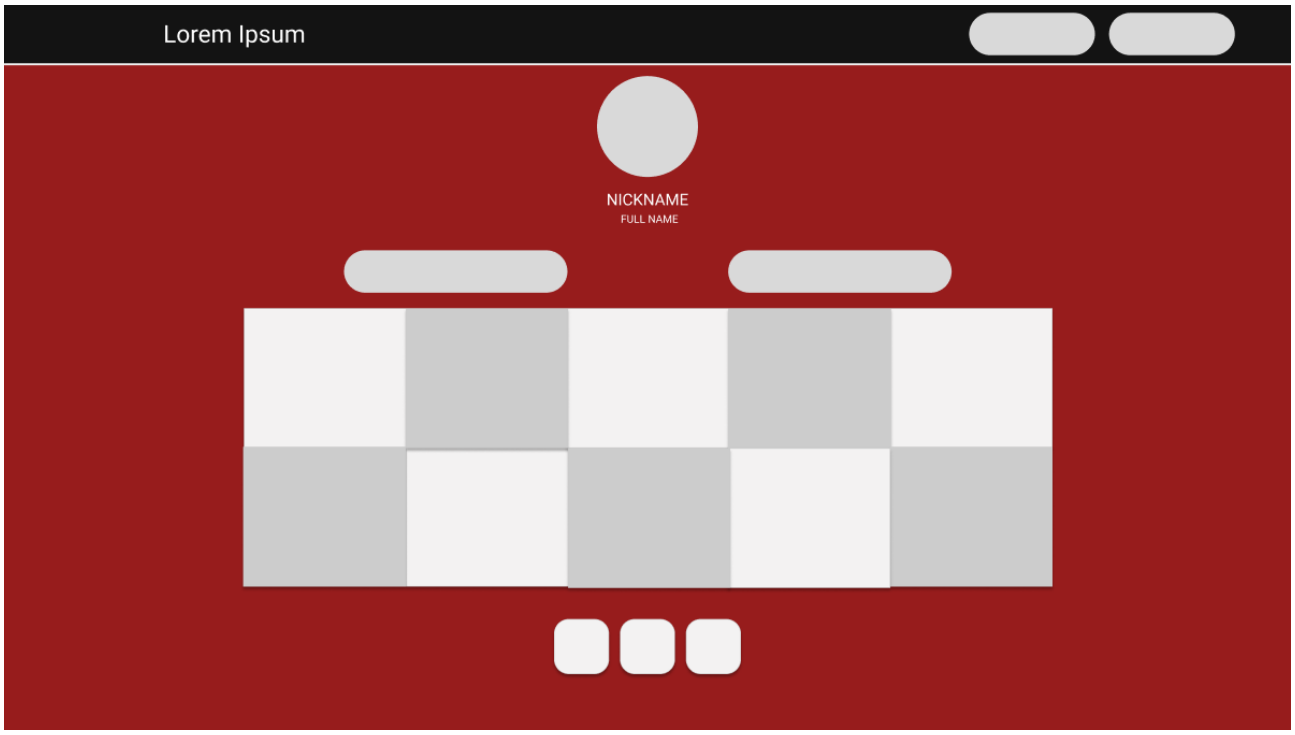

Рисунок 2.4 – Макет портфоліо користувача

Вікно додавання та редагування посту виконано у модальному вигляді, на сторінці особистого профілю, рисунок 2.5 , користувача знаходяться можливості для зміни персональних даних, таких як пароль, імейл та повне ім’я.

Рисунок 2.5 – Макет особистого профілю користувача

Кнопки редагування та видалення посту знаходяться саме на сторінці посту, як це показано на рисунку 2.6 , також там представлена інформація про пост, така як кількість переглядів, лайків, коментарі та кнопка переходу на форму донату на ЗСУ. Для авторизованих користувачів відкрита форма для додавання коментарів, а на наявних коментарях існують кнопки редагування та видалення, які видно лише авторам коментарів.

Весь макет створено за допомоги веб-програми для дизайну і прототипування інтерфейсів Figma.

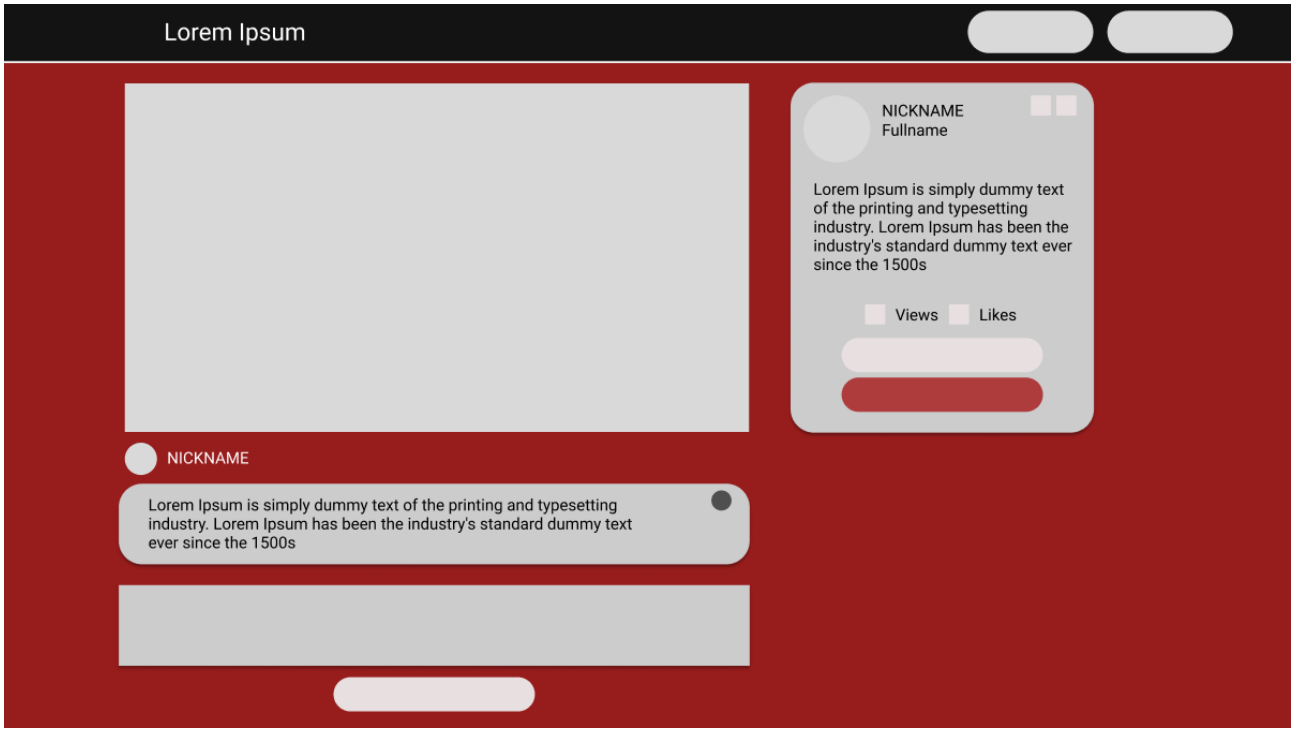

Рисунок 2.6 – Макет сторінки посту

2.2 Проектування моделі бази даних

Базу даних було вирішено зробити реляційну з мінімальною кількістю залежностей. Згідно до діаграми на рисунку 2.7 , основні сутності – User, Post, Comment, Like, Session.

Сутність User має наступні поля: id(PK), name, username, email, password_hash, role, avatarUrl;

Сутність Post має наступні поля: id(PK), title, text, imageUrl, tags, userId, views;

Сутність Comment має наступні поля: id(PK), text, userId, postId;

Сутність Like має наступні поля: id(PK), userId, postId;

Сутність Comment має наступні поля: id(PK), text, userId, postId.

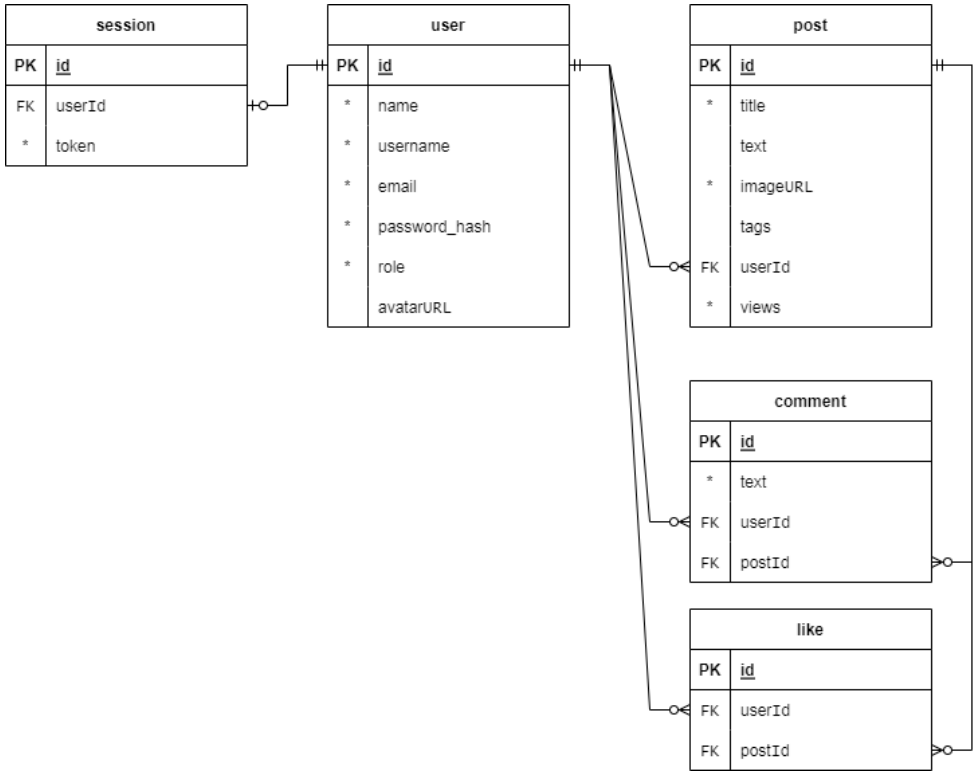

Рисунок 2.7 – Entity relation діаграма бази даних

Сервер бази даних було вирішено розмістити в docker контейнері, завдяки docker-compose файлу з усіма налаштуваннями, та підключити до pgAdmin.

Docker Compose – зручний інструмент для визначення та управління багатоконтейнерними додатками з використанням Docker. Він дозволяє описувати конфігурацію додатку у вигляді файлу YAML, що включає в себе всі залежності, мережі, образи контейнерів, об'ємів, порти, змінні середовища та інші налаштування.

3 РОЗРОБКА ВЕБ-САЙТУ

3.1 Архітектура програмного додатку

Перегляд веб-сайту починається з головної сторінки, на якій висвітлено перелік наявних робіт художників та мотиваційний пост з кнопкою переходу на форму донату. На головній сторінці в залежності від стану авторизації можна перейти або на сторінку авторизації, або на сторінку власного портфоліо. Присутня також кнопка виходу з авторизованого стану.

Також, навіть якщо користувач не зареєстрований, він має змогу перейти з головної сторінки до перегляду конкретного посту, що буде відображатись у неавторизованому стані, тобто відсутня форма додавання коментаря та неможливо поставити лайк.

Сторінка авторизації має подвійний функціонал, одночасно виступає як форма авторизації, так і форма для реєстрації. Між станами можна переключитись окремою кнопкою. Для всіх полів присутня перевірка на валідацію. При створенні аккаунту два поля мусять бути унікальними, це імейл та нікнейм користувача.

Після створення аккаунту чи авторизації користувач має змогу ставити вподобайки, додавати пости через модальне вікно у портфоліо. В цьому вікні доступні поля для введення назви роботи, опису, прикріплення файлу картинки та поле для тегів.

Редагувати пости також через модальне вікно на сторінці посту, де можна редагувати будь-які поля, окрім файлу. Видаляти пост через кнопку також на сторінці посту. Додавати, редагувати та видаляти власні коментарі, змінювати свої персональні дані.

Оскільки одна з головних ідей сайту це підтримка армії України, було вирішено, що кнопка для переходу на форму донату буде доступна незалежно від авторизованого стану.

3.2 Підключення бібліотек

Для реалізації функціональності проекту було обрано певний набір бібліотек, які забезпечують необхідні можливості в області шифрування, мережевого взаємодії, валідації, роботи з базами даних та інші клієнтські бібліотеки.

У рамках дослідження для реалізації функціональності проекту було використано наступні бібліотеки:

1) **Dotenv:** Бібліотека dotenv використовується для завантаження змінних середовища з файлу .env у процес додатка. Дозволяє налаштовувати конфігураційні параметри, такі як ключі API, налаштування бази даних і інші, безпосередньо з файлу .env, що спрощує управління конфігурацією проекту;
2) **Express:** Використовується для реалізації серверної частини проекту. Надає зручні функції для маршрутизації, обробки запитів, обробки помилок, налаштування сервера та іншого;
3) **Cors:** Використовується для керування політикою Same-origin і Cross-origin Resource Sharing (CORS) для серверного додатка. Дозволяє налаштовувати доступ до ресурсів сервера з різних джерел і забезпечує безпеку і захист від несанкціонованого доступу;
4) **Bcrypt:** Використовується для хешування паролів користувачів. Вона забезпечує безпечне зберігання паролів у базі даних шляхом створення солі і обчислення хеш-значення паролю з використанням алгоритму bcrypt;
5) **Express-fileupload:** Використовується для завантаження файлів на сервер. Вона надає простий спосіб обробки завантаження файлів з форми на веб-сторінці, що дозволяє користувачам взаємодіяти з сервером шляхом завантаження файлів;
6) **Express-validator:** Бібліотека для перевірки та валідації даних, що надходять до сервера. Вона надає набір функцій для перевірки різних типів даних, включаючи рядки, числа, електронну пошту і багато іншого.

Express-validator забезпечує надійну перевірку даних перед їх обробкою сервером;

7) **Fs:** Використовується для роботи з файловою системою на сервері. Вона надає методи для створення, читання, запису та видалення файлів і каталогів. Fs дозволяє взаємодіяти з файловою системою, що дозволяє зберігати, отримувати та обробляти файли на сервері;
8) **Jsonwebtoken:** Використовується для створення і перевірки JSON Web Tokens (JWT). JWT використовується для аутентифікації та авторизації користувачів. Jsonwebtoken дозволяє генерувати токени, які містять інформацію про користувача і можуть бути використані для підтвердження його ідентичності;
9) **Pg і Pg-hstore:** Використовуються для зв'язку з базою даних PostgreSQL. Вони надають інтерфейс для виконання запитів до бази даних, створення, модифікації та видалення даних. Pg-hstore дозволяє зберігати дані у форматі hstore, який дозволяє зберігати ключ-значення парами;
10) **Sequelize:** Використовується для роботи з об'єктно-реляційним відображенням (ORM) для бази даних. Вона надає зручний спосіб взаємодії з базою даних, дозволяючи виконувати запити, створювати таблиці, встановлювати зв'язки між таблицями та багато іншого;
11) **Uuid:** Для генерації унікальних ідентифікаторів. Вона надає методи для створення UUID, які можуть бути використані для ідентифікації різних об'єктів в системі;
12) **React:** основна бібліотека React, яка дозволяє розробляти інтерактивні користувацькі інтерфейси;
13) **React router dom:** Бібліотека для маршрутизації в React-додатка;
14) **Axios:** Бібліотека для виконання HTTP-запитів з клієнтської частини;
15) **Bootstrap:** CSS-фреймворк, який надає набір стилів і компонентів для швидкої розробки користувацького інтерфейсу;

16) **React Bootstrap:** набір компонентів Bootstrap, адаптованих для використання з React;
17) **Jwt decode:** Бібліотека для розшифрування JWT-токенів;
18) **Mobx:** для керування станом даних в програмах з використанням шаблону стор, дії, представлення;
19) **Mobx react lite:** Бібліотека, що надає підтримку MobX для React-компонентів;
20) **React Icons:** Набір іконок для використання в React-додатках.

Для підключення бібліотек до проекту використовується система керування пакетами Node.js – npm.

3.3 Підключення бази даних

База даних PostgreSQL та інтерфейс адміністрування PgAdmin ініціалізовано за допомогою Docker Compose. Використовується файл конфігурації docker-compose.yml [A1], який містить налаштування для контейнерів db та pgadmin.

У сервісі db використовується офіційний образ PostgreSQL. Встановлення та налаштування бази даних здійснюється за допомогою параметрів environment. Вказується логін та пароль для користувача бази даних, а також назва бази даних, яку необхідно створити.

Сервіс pgadmin використовує образ dpage/pgadmin4, який надає інтерфейс адміністрування для PostgreSQL. За допомогою параметрів environment встановлюється електронна адреса та пароль адміністратора PgAdmin. Порт 80 в контейнері мапується на порт 5050 на локальній машині для доступу до інтерфейсу PgAdmin через веб-браузер.

Для підняття бази даних та PgAdmin потрібно виконати команду docker-compose up у директорії, де знаходиться файл docker-compose.yml. Після успішного запуску контейнерів, база даних буде доступна на порту 5432, а PgAdmin - на порту 5050 як це показано на рисунку 3.1 .

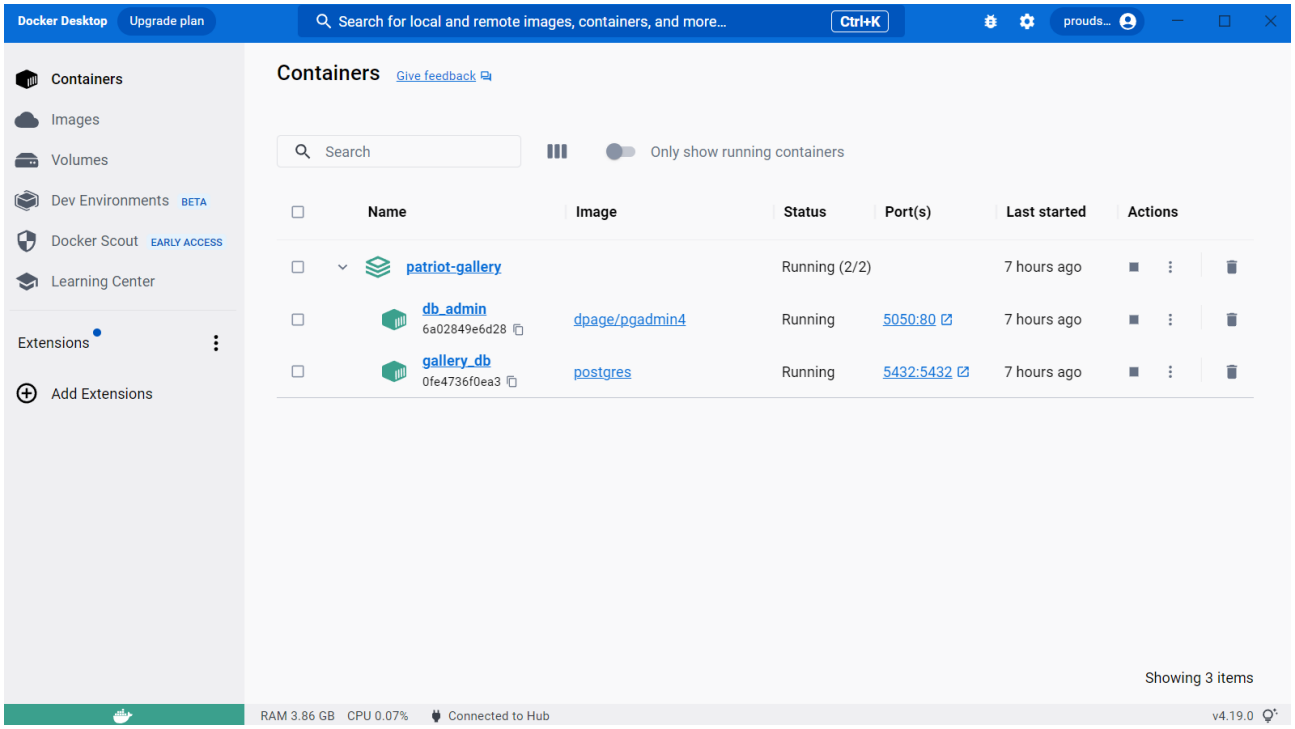

Рисунок 3.1 – Контейнери бази даних та PgAdmin

Можна використовувати зазначені логін та пароль для входу в PgAdmin та налаштування з'єднання з базою даних, рисунок 3.2 .

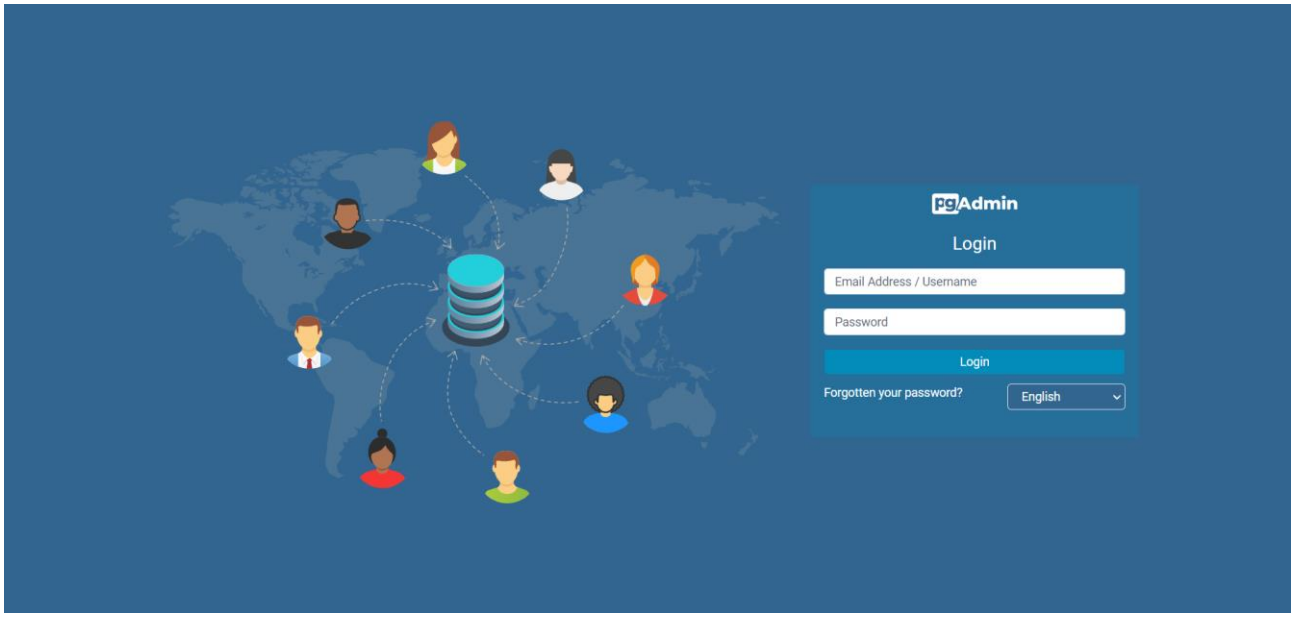

Рисунок 3.2 – Сторінка входу в PgAdmin

Для підключення бази даних до PgAdmin необхідно зробити декілька кроків в терміналі проекту. По-перше командою «docker container ls» отримати список контейнерів, обрати контейнер postgres та ввести команду для нього

«docker inspect [ім’я контейнеру]». Після цього внизу знайти його айпі адресу, та перейшовши в PgAdmin на сторінку реєстрації серверу ввести її у відповідне поле, а також заповнити всі інші необхідні поля, відповідно до даних файлу docker-compose. Після цього база даних буде доступна для перегляду, як це показано на рисунку 3.3 .

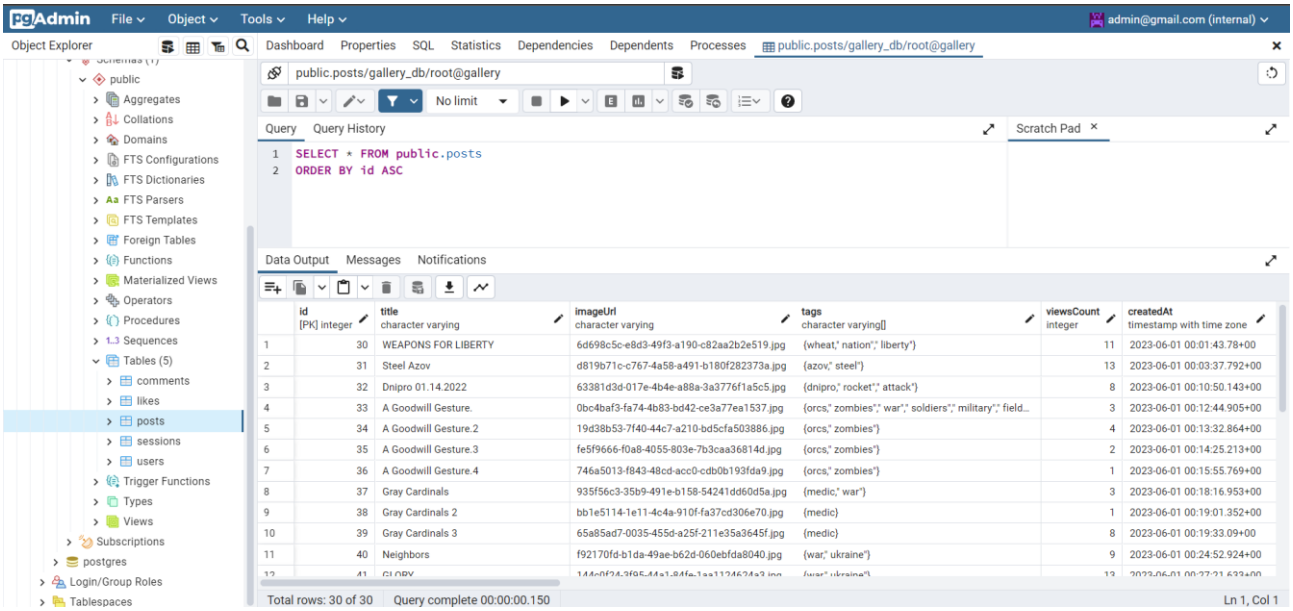

Рисунок 3.3 – База даних в PgAdmin

Цей підхід дає можливість швидко та зручно підняти середовище бази даних для розробки та тестування дипломного проекту, спрощуючи процес установки та налаштування окремої бази даних та інструментів адміністрування.

Для підключення бази даних до дипломного проекту використовуються наступні файли конфігурації та скрипти: .env, db_connect.js та index.js.

Файл .env містить основні змінні порту(PORT), назву бази даних(DB_NAME), ім’я головного користувача(DB_USER), пароль до бази даних(DB_PASSWORD), назву хоста(DB_HOST), порт бази даних(DB_PORT) та секретний ключ для jsonwebtoken(JWT_SECRET).

У файлі db_connect.js[A2] наведено налаштування для з'єднання з базою даних. Використовується об'єкт Sequelize з пакету sequelize для створення

з'єднання з PostgreSQL базою даних. Параметри з'єднання отримуються з оточення та передаються як аргументи при створенні об'єкту Sequelize.

Файл index.js[A3] відповідає за запуск серверної частини додатка. У цьому файлі після підключення необхідних модулів та налаштування маршрутів, виконується підключення до бази даних і запуск сервера. Об'єкт sequelize, створений у файлі db_connect.js, використовується для перевірки з'єднання та синхронізації моделей з базою даних.

У функції start спочатку виконується перевірка з'єднання з базою даних за допомогою методу authenticate об'єкта sequelize. Після успішного з'єднання викликається метод sync, який синхронізує моделі з базою даних. Після цього сервер починає слухати на вказаному порту, а на консоль виводиться повідомлення про успішний запуск сервера.

Таким чином, підключення бази даних до дипломного проекту здійснюється за допомогою конфігураційних файлів .env та db_connect.js. Параметри з'єднання отримуються з .env файлу, а об'єкт sequelize використовується для взаємодії з базою даних у файлі index.js.

3.4 Серверна частина

Серверна частина веб-ресурсу виконується на платформі Node.js та базується на фреймворку Express.js. Вона складається з набору роутерів, контролерів, middleware і валідаційних функцій, які забезпечують обробку запитів і виконання функціоналу ресурсу.

Основна робота файлу index.js включає наступні кроки:

- Підключення необхідних модулів, зокрема пакету express для створення сервера та пакету Router для маршрутизації;
- Створення об'єкту router за допомогою Router(). Цей об'єкт використовуватиметься для визначення шляхів та відповідних обробників;

- Підключення модулів маршрутизації за допомогою методу use(path, router). Цей метод встановлює шляхи до різних модулів на основі вказаних роутерів. Наприклад, router.use('/user', userRouter) встановлює шлях /user для userRouter, який містить маршрути, пов'язані з користувачами;
- Використання пакету express-fileupload для обробки завантаження файлів та вказання що папка 'static' є статичною папкою, яка буде доступна клієнтам. Файли, розташовані у цій папці, можуть бути запитані і звернуті через сервер;
- Отримання змінних середовища та запуск серверу на порту.

Файл routes/index.js[А4] визначає головний роутер, який використовується для групування і маршрутизації запитів до різних частин ресурсу. В нашому випадку, ми маємо роутери для користувачів, постів, коментарів і лайків.

Кожен з роутерів, наприклад, userRouter.js, postRouter.js, commentRouter.js і likeRouter.js, має свої власні маршрути і пов'язані з ними контролери. Роутери використовуються для обробки специфічних запитів, пов'язаних з користувачами, постами, коментарями і лайками.

Контролери UserController.js, PostController.js, CommentController.js і LikeController.js, містять функції, які виконують потрібні операції для кожного типу запиту. Наприклад, створення користувача, отримання всіх постів, оновлення коментаря тощо. Контролери взаємодіють з відповідними моделями даних для збереження і отримання інформації.

У проекті використовуються моделі для збереження та отримання даних. Моделі взаємодіють з базою даних і надають інтерфейс для виконання операцій з даними.

В кваліфікаційній роботі використовуються наступні моделі:

1) **User:** Модель користувача відповідає за збереження інформації про користувачів системи. Вона містить поля, такі як електронна пошта, пароль, ім'я та інші характеристики користувача;

2) **Post:** Модель поста відповідає за збереження інформації про пости на веб-ресурсі. Ця модель містить поля, такі як заголовок, текст, теги і інші атрибути, що відображають вміст поста;
3) **Comment:** Модель коментаря відповідає за збереження коментарів, пов'язаних з постами або іншими елементами системи. Вона містить поля, такі як текст коментаря, ідентифікатор поста, ідентифікатор автора тощо.;
4) **Like:** Модель лайка представляє інформацію про лайки, які користувачі ставлять постам або іншим елементам системи. Вона зберігає ідентифікатори користувачів та постів, які були лайкнуті.

Кожна модель має відповідний файл, який визначає її структуру, включаючи поля, типи даних та зв'язки з іншими моделями або таблицями бази даних. Вони використовуються контролерами для отримання або збереження даних, пов'язаних з конкретною моделлю.

Деякі роутери і контролери використовують middleware для виконання певних перевірок перед виконанням функцій. checkAuth.js[A5] перевіряє наявність токена авторизації у заголовку запиту і перевіряє його достовірність. Якщо токен валідний, то дані користувача розшифровуються і додаються до об'єкту запиту (req.user), що дозволяє ідентифікувати користувача під час обробки запиту.

Для валідації даних використовуються функції з файлу Validation.js[A6]. Наприклад, registerValid перевіряє формат пошти, довжину пароля, довжину ім'я і нікнейма тощо. Ці функції використовуються у роутерах разом з validationHandler, який перевіряє наявність помилок після валідації і повертає їх у відповідь у випадку необхідності.

Крім того, файл ErrorHandler.js визначає middleware для обробки помилок. Якщо сталася помилка під час обробки запиту або валідації, відповідний екземпляр ServerError створюється з кодом помилки і повідомленням, і повертається у відповідь. Якщо невідома помилка, то повертається загальне повідомлення про непередбачувану помилку.

Таким чином, серверна частина веб-ресурсу забезпечує маршрутизацію запитів, виконання функціоналу ресурсу, перевірку авторизації, валідацію даних та обробку помилок. Цей архітектурний підхід дозволяє організувати логіку сервера і підтримувати чітку структуру проекту.

В проекті використовується пакет jsonwebtoken для роботи з JWT (JSON Web Token). JWT є механізмом аутентифікації, який дозволяє створювати токени для автентифікації користувачів. У файлі checkAuth.js[A5] відбувається перевірка автентифікації користувача за допомогою JWT. Основна робота з JWT відбувається наступним чином:

- Запит включає заголовок Authorization, який містить JWT. Токен передається у форматі "Bearer <token>";
- В файлі checkAuth.js рядок const token = req.headers.authorization.split(' ')[1]; отримує токен з заголовка запиту;
- Токен перевіряється за допомогою методу jwt.verify(token, process.env.JWT_SECRET). process.env.JWT_SECRET представляє секретний ключ, який використовується для підпису та перевірки токенів. Якщо токен не валідний або його перевірка не вдається, генерується помилка. Якщо перевірка токена успішна, розкодовані дані зберігаються у полі req.user, що дозволяє ідентифікувати користувача для подальшої обробки запиту.

Робота з токеном JWT також використовується в окремій функції контроллеру користувачів UserController, а саме у функції tokenGen[A7], що відповідає за генерацію токену відповідно до наданих даних користувача. Термін дії токену встановлено на один день, пароль шифрується та верифікується за допомоги бібліотеки bcrypt.

Механізм авторизації на реєстрації у проекті виконано на основі створення сесій користувача, що зберігаються в окремій таблиці бази даних. За контроль

сесій відповідає окрема функція sessionManager[A8], що перевіряє наявність існуючих сесій, при повторній авторизації видаляє стару сесію і створює нову.

3.5 Клієнтська частина

У цьому розділі описано клієнтську частину та функціональність веб-додатку на основі React-бібліотеки.

Ініціалізація додатку та встановлення кореневого елемента, в який буде рендеритися додаток відбувається у файлі index.js[A9]. Він імпортує основні модулі, стори, створює контекст додатку та передає його в провайдер контексту. Додаток рендериться за допомогою компонента <App />, головного компоненту, що відповідає за взаємодію з контекстом, перевірку статусу авторизації користувача та відображення відповідних компонентів залежно від стану. За допомогою хука useEffect, відбувається перевірка статусу авторизації завдяки HTTP-запиту до сервера. Додаток відображає головне меню <NavBar />, компонент <AppRouter />[A10], та підвал, <Footer />.

Файл .env містить змінну середовища REACT_APP_API_URL, яка визначає URL-адресу серверної частини додатку. Використовується для встановлення з'єднання з сервером при виконанні HTTP-запитів.

Маршрутизація:

Всі основні шляхи винесено в окремий файл consts[A11] в папці utils, для полегшення зміни шляхів у разі необхідності. Наприклад, LOGIN_ROUTE містить шлях до сторінки входу, REGISTER_ROUTE шлях до сторінки реєстрації тощо. Використання констант також допомагає уникнути повторення рядків з шляхами.

За маршрутизацію в додатку відповідає компонент <AppRouter />. Залежно від статусу авторизації користувача, компонент <AppRouter /> відображає відповідні компоненти для авторизованих користувачів (authRoutes) або публічні компоненти (publicRoutes). У випадку, якщо шлях не знайдено, відбувається перенаправлення на головну сторінку.

Логіку відображення компонентів відповідно до шляху винесено в окремий файл routes, він містить імпорт компонентів сторінок, які використовуються в додатку, а також константи, що визначають шляхи сторінок.

- Auth сторінка відповідає за авторизацію користувача і містить форму входу та реєстрації;
- Main сторінка представляє головну сторінку додатку, яка відображає загальний вміст, список постів та мотиваційний пост з кнопкою донату, залежно від контексту;
- Portfolio сторінка відображає сторінку портфоліо, із списком постів художника та деякою інформацією про нього, а також кнопками додавання посту та переходу на сторінку профілю, якщо переглядач є автором портфоліо;
- Post сторінка відображає окремий пост із вмістом, коментарями, лайками, кнопкою донату, інформацією про автора та можливістю перегляданння зображення в повному розмірі через модальне вікно. Також тут реалізовано кнопки видалення та редагування посту які видно лише автору;
- Profile відображає сторінку користувача з особистою інформацією, аватаркою та можливістю зміни персональних даних.

Керування станами та стори:

У додатку, для керування станом, використовується підхід, заснований на MobX. Цей підхід базується на створенні окремих "сторів", які містять дані та функціонал для керування цими даними. Кожен стор відповідає за певну частину стану додатка та забезпечує його доступність для компонентів, що його використовують.

Один з головних принципів MobX – це концепція реактивності. За допомогою MobX, зміни в сторі автоматично відслідковуються та спричиняють

оновлення компонентів, які використовують ці дані. Це дозволяє забезпечити автоматичну актуалізацію відображення даних у відповідь на зміни стану.

Компоненти, що потребують доступу до сторів, використовують useContext для отримання доступу до контексту, який містить ці стори. Вони можуть використовувати методи та властивості сторів для отримання та оновлення даних, а також підписуватися на зміни стану, щоб автоматично оновлювати своє відображення.

Створено наступні стори:

1) **UserStore:** Відповідає за управління даними користувача. Він містить функціонал, пов'язаний з аутентифікацією, таким як встановлення статусу авторизації користувача та збереження даних користувача;
2) **PostStore:** Відповідає за управління даними постів. Містить також функціонал, пов'язаний з отриманням автора посту, та встановлення налаштувань пагінації, включаючи ліміт відображення постів на сторінці;
3) **CommentStore:** Відповідає за управління даними коментарів та їх авторів;
4) **LikesStore:** Відповідає за управління даними лайків.

АПІ:

Взаємодію з сервером за допомогою HTTP запитів(АПІ) реалізовано на базі бібліотеки Axios та логічно розділено на окремі файли, кожен з яких відповідає за свій сегмент.

У файлі http/index.js[A12] створені два екземпляри axios - $host та $authHost. $host використовується для звичайних запитів, а $authHost - для запитів, які потребують авторизації. Також встановлено перехоплювач запитів authInterceptor, який додає авторизаційний заголовок до запиту, використовуючи токен, який зберігається у локальному сховищі.

У файлі userAPI.js[A13] передбачено методи для взаємодії з API, що стосуються користувачів. registration виконує запит на реєстрацію користувача, login на авторизацію, updateUser на оновлення інформації про користувача, check для перевірки авторизованості, logout для виходу з системи та знищення сесії

користувача, а також getUser для отримання повної інформації про користувача, окремий запит для зміни персональних даних.

Файл postAPI.js містить методи для роботи з постами, createPost створює новий пост, fetchPosts отримує список постів з пагінацією незалежно від автора, fetchPostsBy отримує список постів з пагінацією від одного автора, fetchOnePost отримує один конкретний пост автора, fetchAuthor отримує інформацію про автора, виключаючи деякі дані, updatePost оновлює існуючий пост, а deletePost видаляє пост.

У likesAPI.js надано методи для взаємодії з лайками, createLike створює новий лайк або видаляє старий, якщо лайк вже існує, а fetchLikesBy отримує список лайків по одному конкретному посту.

Файл commentAPI.js містить методи для роботи з коментарями, createComment створює новий коментар, fetchCommentsBy отримує список коментарів для певного посту, getAuthor отримує інформацію про автора коментару, updateComment оновлює існуючий коментар з редагованими даними, а removeComment видаляє коментар.

Усі ці файли використовують $host та $authHost для здійснення HTTP запитів до сервера з використанням відповідних шляхів та параметрів. Інформація про токен також зберігається у локальному сховищі за допомогою localStorage.

Пости:

За відображення постів відповідає окремий компонент Posts[А14], в якому реалізовано одразу і отримання постів з контексту і фільтрація їх по автору, якщо місце їх відображення це портфоліо художника. Потім дані передаються на окремий компонент Post[А15], який вже відповідає за відображення картки посту.

Коментарі:

За відображення коментарів відповідає компонент Comments[А16], який отримує коментарі для конкретного посту та відображає напис «коментарів немає» у випадку, якщо пост не має ще жодного. Потім дані про коментар

передаються на компонент CommentCard, що відповідає за відображення вмісту коментаря, його автора та відображення меню редагування, котре видно лише автору коментаря.

Форма додавання коментаря виконана в окремому компоненті CommentAdd, який відображається лише якщо користувача авторизовано.

Кнопка донату та форма PayPal:

Винесена в окремий компонент. Основна функціональність компонента полягає в тому, що він надає посилання на зовнішній ресурс PayPal для здійснення пожертвувань. Посилання відкривається у новій вкладці браузера завдяки атрибутам target="_blank". При натисканні на кнопку, користувач буде перенаправлений на сторінку PayPal з передвстановленим ідентифікатором кнопки пожертвування.

Сама форма пожертвувань виконана за допомоги безкоштовного інструментарію PayPal по створенню кнопок для пожертвувань. Як зазначено на рисунку 3.4 форма має додатковий функціонал зі включення комісії в суму пожертвування та можливість «підписки», щоб робити автоматичні перекази щомісяця, або кожен рік.

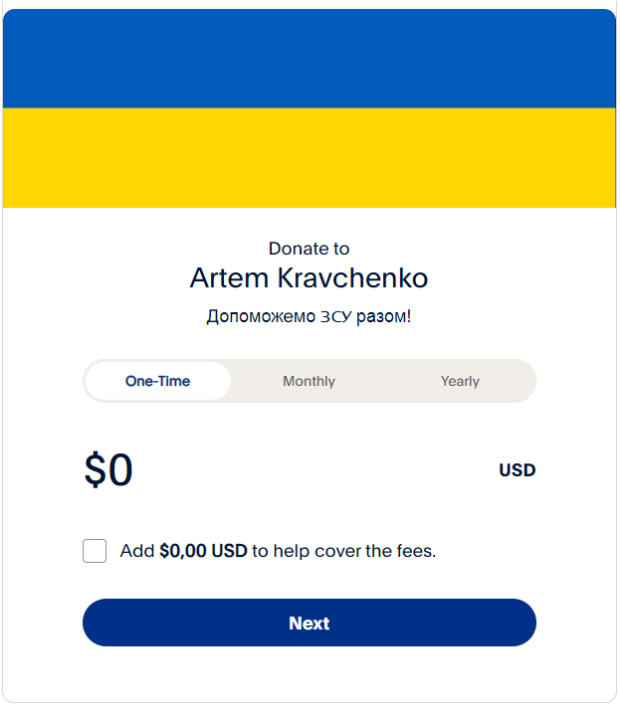

Рисунок 3.4 – Форма пожертвувань

В проекті також реалізовано два модальних вікна для створення та редагування посту на основі бібліотеки Bootstrap для більш швидкої розробки, всі інші стилі CSS винесено в окремі файли відповідно до розташування компонентів.

3.6 Використання програмного додатку

Будь який користувач має змогу переглянути головну сторінку з усією наявною інформацією на ній, одразу провести донат по кнопці в мотиваційному пості, а також переходити до перегляду портфоліо зареєстрованих користувачів з головної сторінки, чи до авторизації по кнопці в хедері, рисунок 3.5 .

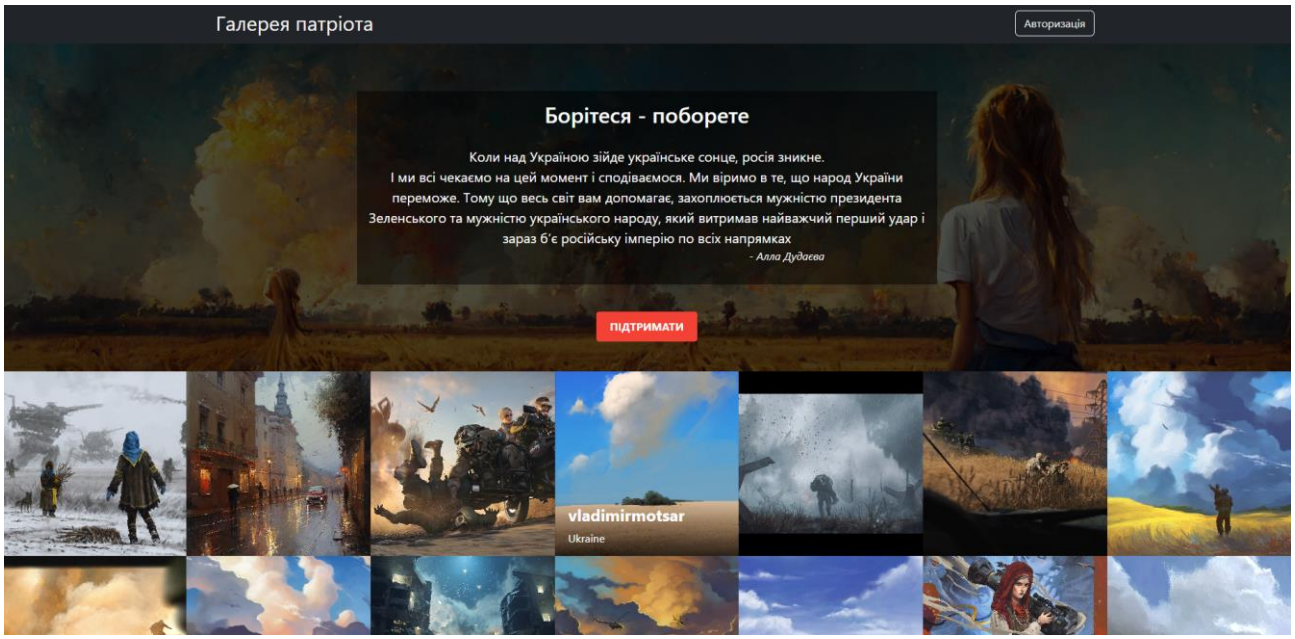

Рисунок 3.5 – Головна сторінка сайту

При наведені на пости з’являється інформація про автора та його ім’я.

Авторизація або реєстрація відбувається на окремій сторінці, рисунок 3.6 , вміст якої можна переключати в залежності від того, що вам треба. Всі поля валідуються і у випадку помилки виникає вікно оповіщення.

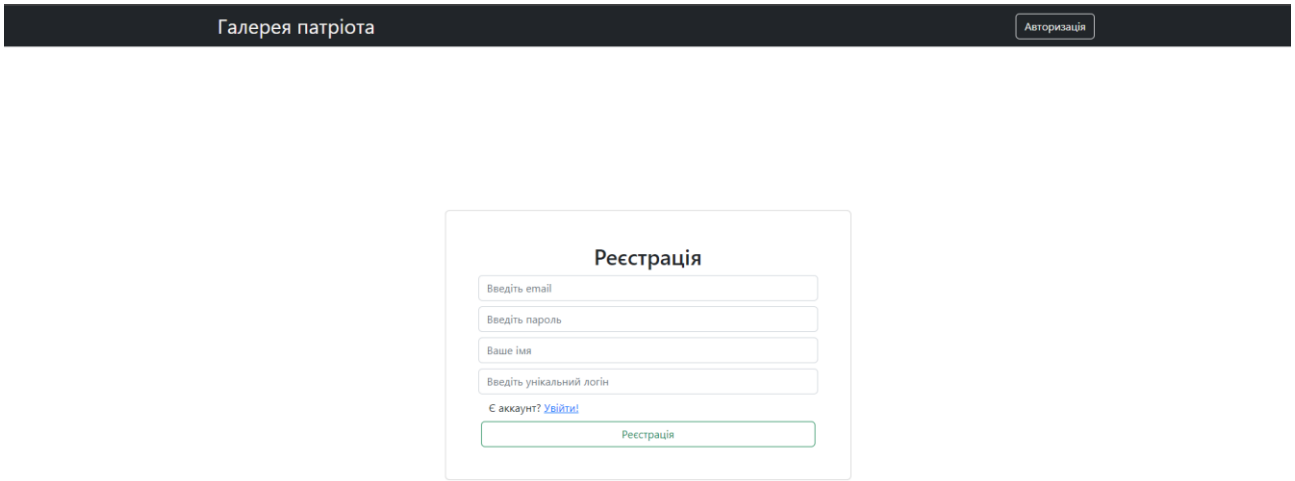

Рисунок 3.6 – Сторінка реєстрації

Рисунок 3.7 показує як перемикається форма, якщо натиснути кнопку «Увійти». За таким же принципом форма переключається назад, якщо натиснути кнопку «Зареєструватись».

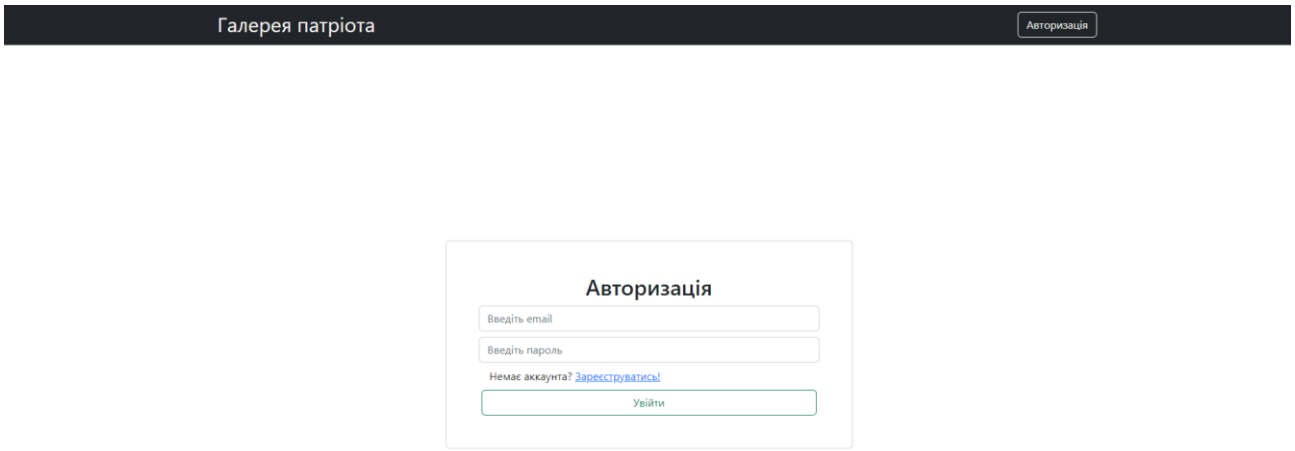

Рисунок 3.7 – Сторінка авторизації

Зареєструвавшись чи авторизувавшись користувачу надається низка можливостей. Перш за все це можливість додавання особистих робіт через портфоліо, яке можна побачити на рисунку 3.8 , додавання виконується через модальне вікно, рисунок 3.9 , поля якого також проходять валідацію, або редагування вже існуючих через сторінку самого посту, рисунок 3.10 , також у модальному вікні, рисунок 3.11 .

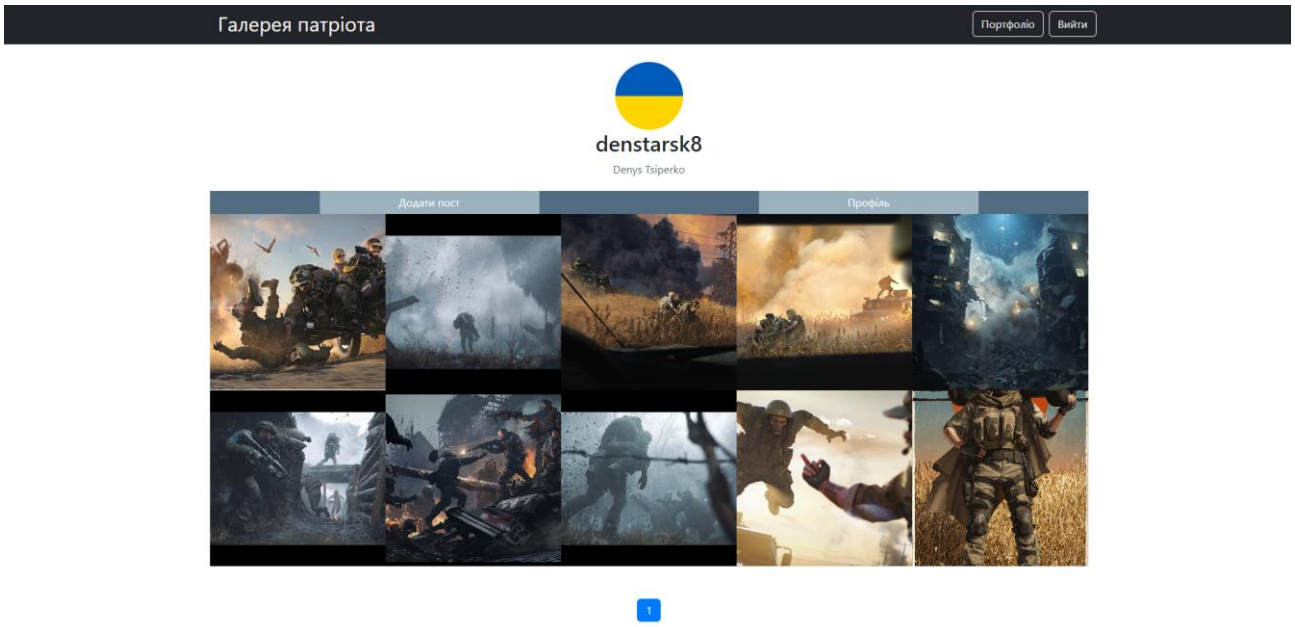

Рисунок 3.8 – Сторінка портфоліо

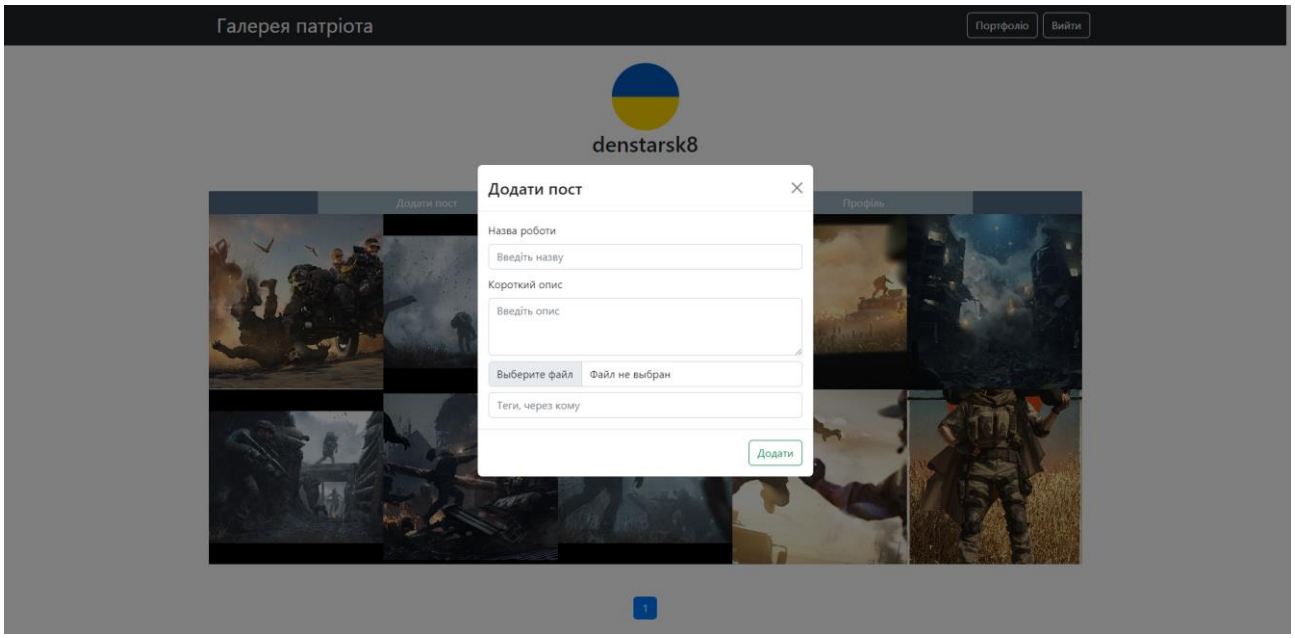

Рисунок 3.9 – Модальне вікно додавання посту

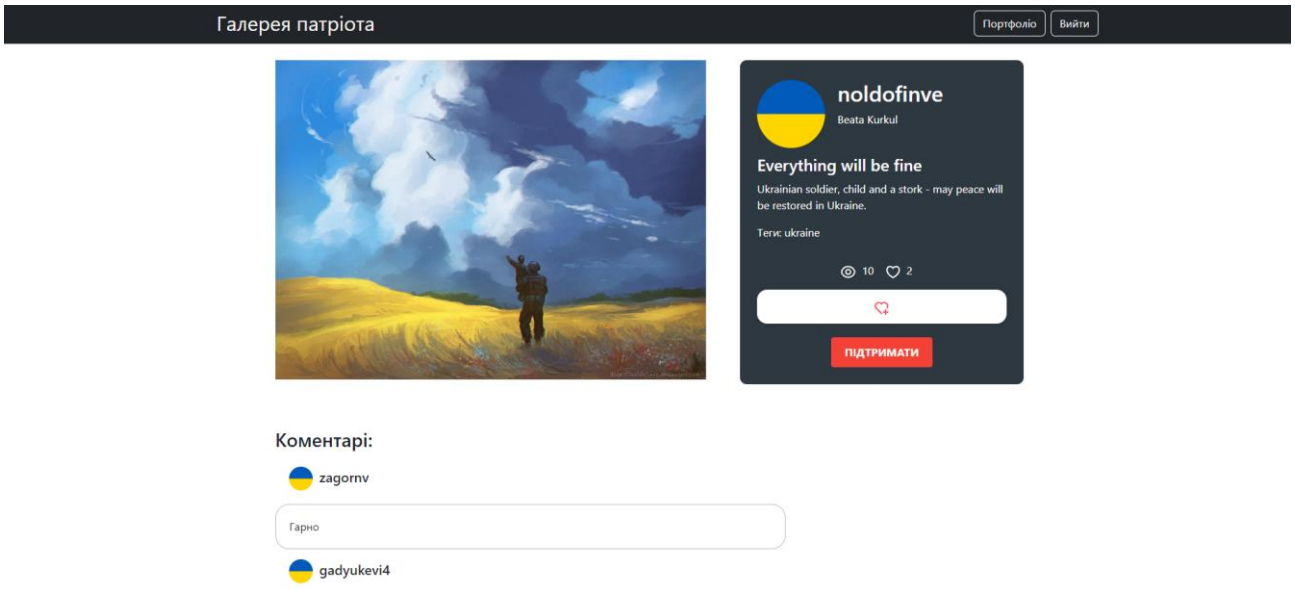

Рисунок 3.10 – Сторінка посту

Також автору на сторінці поста становиться доступною кнопка видалення посту, меню підтвердження видалення також модальне, рисунок 3.12 .

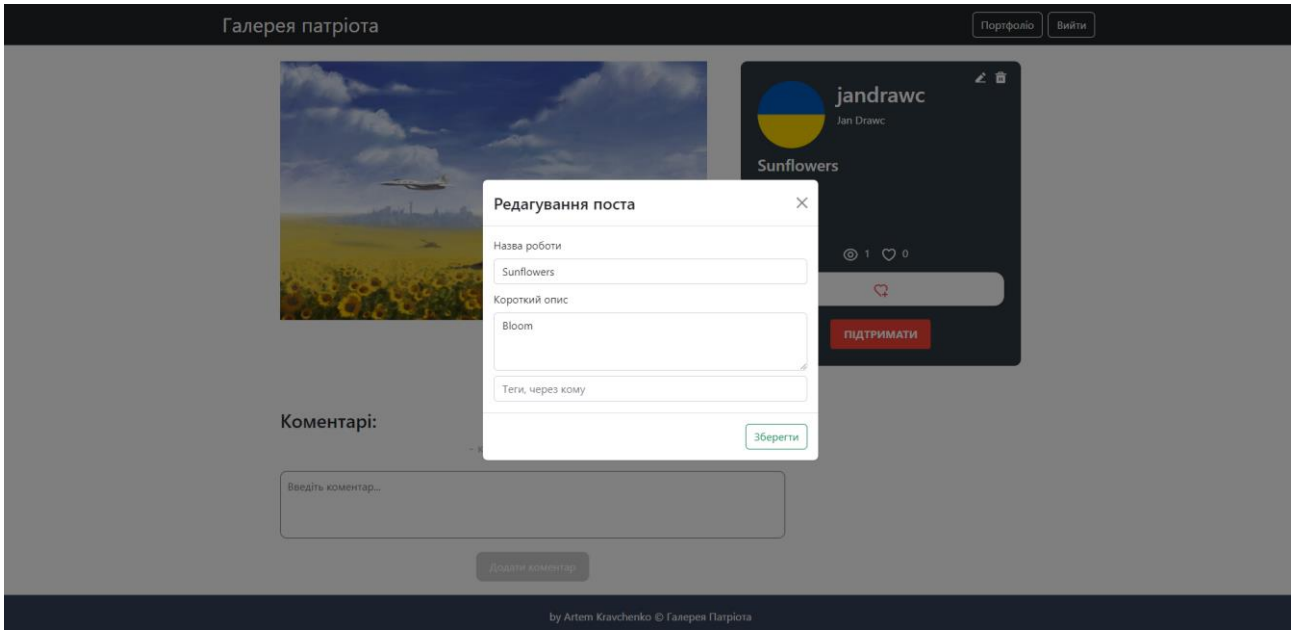

Рисунок 3.11 – Модальне вікно редагування посту

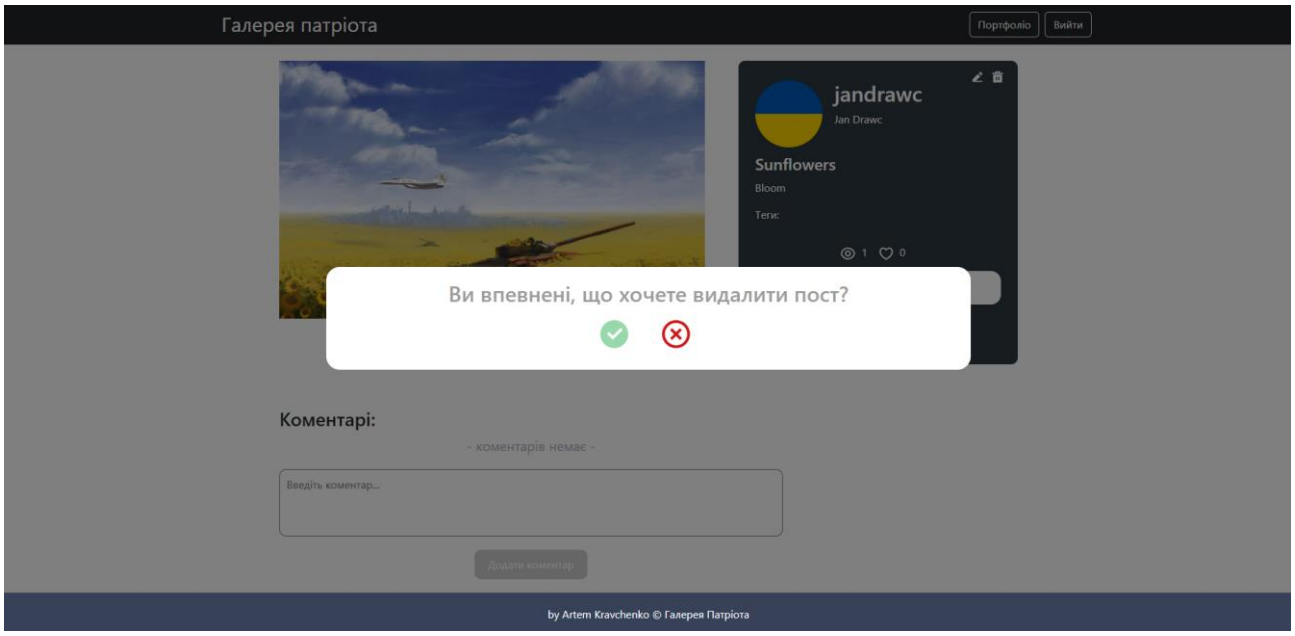

Рисунок 3.12 – Модальне вікно підтвердження видалення

Через сторінку портфоліо художник може перейти до сторінки з особистими даними та редагувати їх, як показано на рисунку 3.13 .

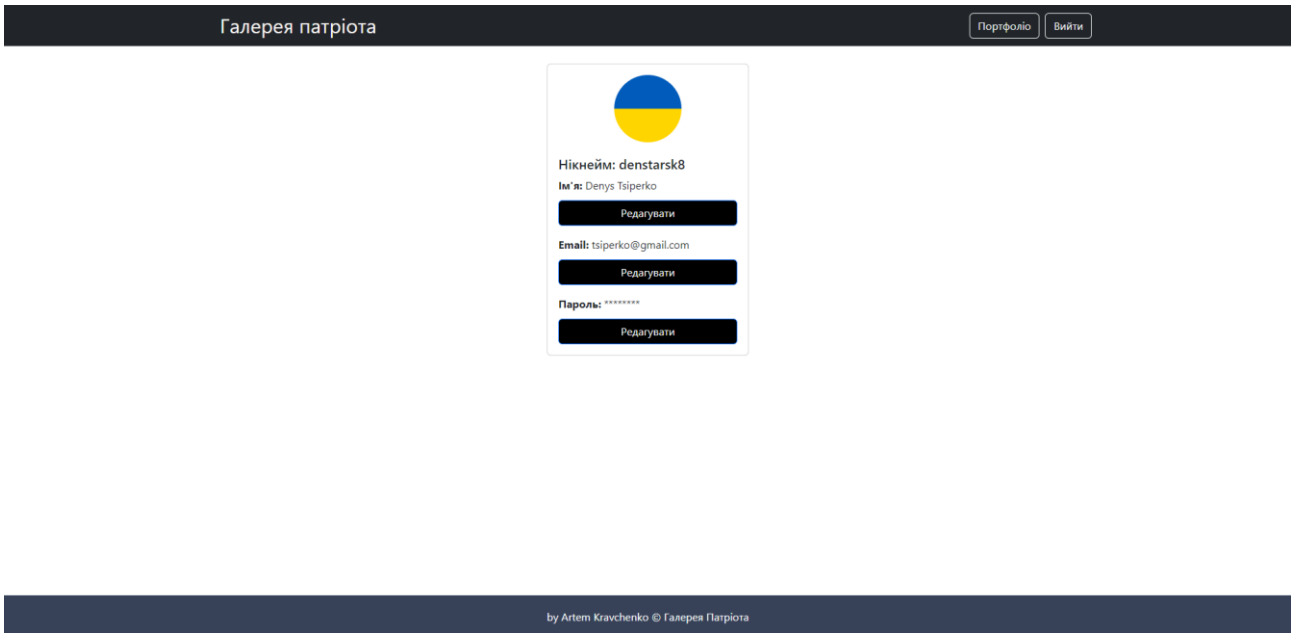

Рисунок 3.13 – Сторінка профілю користувача

Також авторизованим користувачам надається можливість ставити вподобайки, коментувати та керувати власними коментарями, редагувати чи

видаляти їх через випадаюче меню, приклад випадаючого меню показано на рисунку 3.14 .

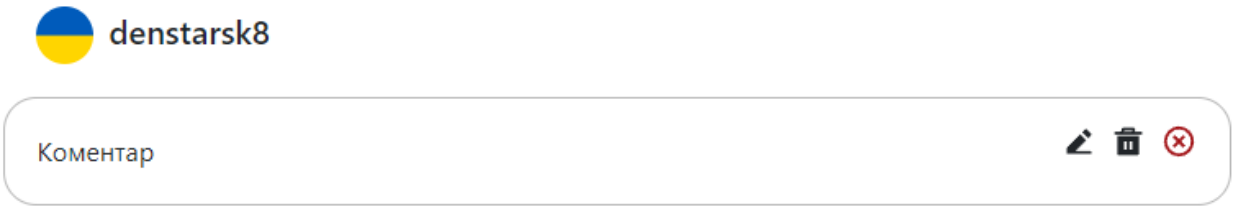

Рисунок 3.14 – Меню редагування коментаря

І нарешті при натисканні на кнопку донату користувача буде перенаправлено на форму грошового переказу в новій вкладці, яку вже було представлено на рисунку 3.4 .

ВИСНОВКИ

Мету роботи по створенню національного вільного простору для українських митців досягнуто в результаті створення повноцінного веб-порталу «Галерея патріота».

Реалізовано основний функціонал сайту, такий як авторизація, реєстрація, додавання постів на ресурс, форма донату, оброблення помилок, запитів, валідацію. Додатковий функціонал з коментування, виставлення вподобайок, лічильник переглядів, можливості редагування та видалення як постів так і коментарів, керування особистими даними користувача.

В результаті виконання кваліфікаційної роботи було виконано наступні завдання:

1) Здобуто необхідні знання по створенню веб-сайтів;
2) Досліджено основні підходи по створенню веб-сайтів;
3) Проаналізовано та досліджено ресурси аналоги, виділено найуспішніші реалізації власних потреб;
4) Розроблено веб-сайт.

В майбутньому можливе розширення функціоналу сайту в сторону більшої кількості інструментів по редагуванню малюнка та розбиття донату на дві частини, щоб авторам робіт теж йшов невеликий відсоток за пророблену роботи. Це буде стимулювати українських митців активно приймати участь у житті сайту, що в свою чергу буде розвивати національне мистецтво. Нажаль для реалізації такого функціоналу треба домовлятись з банком та вирішувати деякі питання стосовно податків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. PayPal Global Scale: presentation. URL: https://www.paypalobjects.com/digitalassets/c/website/marketing/global/shared/global/media-resources/documents/Q4_2018_PayPal_Global_Scale.pdf (date of access: 01.05.2023).

2. Careers at DeviantArt: website. URL: https://web.archive.org/web/20201027000916/https://www.deviantart.com/team/journal/Careers-at-DeviantArt-855261915 (date of access: 15.04.2023).

3. Allen W. What's new in the creative community. Adobe Blog. URL: https://blog.adobe.com/en/publish/2020/10/20/whats-new-in-the-creative-community (date of access: 12.05.2023).

4. Е. Порселло, А. Бенкс . React: сучасні шаблони для розробки додатків 2-е видання: навч. посіб. Україна: Бестселери O'reilly, 2022. 320с.

5. E. Brown. Web Development with Node and Express, 2nd Edition: навч. посіб. USA: O'reilly Media, 2018. 326 с.

6. A. Banks. Learning React: Functional Web Development with React and Redux: навч. посіб. USA: O'reilly Media, 2017. 320 с.

7. I. Miell. A. H. Sayers. Docker in Practice: навч. посіб. UK: Manning, 2016. 372 с.

8. J. Robbins. Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics, Fourth Edition: довідник. Canada: O'Reilly Media, 2012. 624 с.

9. A. Cooper. R. Reimann. D. Cronin. C. Noessel. About Face: The Essentials of Interaction Design, Fourth Edition: довідник. USA: Wiley, 2014. 720 с.

10. C. J. Date. Introduction to Database Systems, An, Eight Edition: довідник. USA: Wesley Longman Publishing Co., 2003. 1040 с.

11. D. Flanagan. The Definitive Guide: Activate Your Web Pages (Definitive Guides), 6th Edition: довідник. USA: O'Reilly Media, 2011. 1096 с.

12. Node.js tutorial in Visual Studio Code: website. URL: https://code.visualstudio.com/docs/nodejs/nodejs-tutorial (date of access: 13.04.2023).

13. PostgreSQL Tutorial: website. URL: https://www.postgresqltutorial.com/ (date of access: 12.04.2023).

ДОДАТОК А
ЛІСТИНГ ПРОГРАМНОГО КОДУ

А1 файл docker-compose

```
version: "3.8"
services:
  db:
    container_name: gallery_db
    image: postgres
    volumes:
      - ./db:/db
    restart: always
    environment:
      POSTGRES_USER: root
      POSTGRES_PASSWORD: 123456
      POSTGRES_DB: gallery_db
    ports:
      - "5432:5432"
  pgadmin:
    container_name: db_admin
    image: dpage/pgadmin4
    restart: always
    environment:
      PGADMIN_DEFAULT_EMAIL: admin@gmail.com
      PGADMIN_DEFAULT_PASSWORD: secret
    ports:
      - "5050:80"
```

А2 db_connect

```
const { Sequelize } = require('sequelize');

module.exports = new Sequelize(
    process.env.DB_NAME,
    process.env.DB_USER,
    process.env.DB_PASSWORD,
    {
        dialect: 'postgres',
        host: process.env.DB_HOST,
        port: process.env.DB_PORT
    }
)
```

А3 server/index.js

```
require('dotenv').config()
const express = require('express');
const cors = require('cors');
const fileUpload = require('express-fileupload');
const path = require('path');

const sequelize = require('./db_connect');
const models = require('./models/index.js');
const router = require('./routes/index.js');
```

```
const errorHandler = require('./middleware/ErrorHandler.js')

const PORT = process.env.PORT || 4500;

const app = express();
app.use(cors());
app.use(express.json());
app.use(fileUpload({}));
app.use(express.static(path.resolve(__dirname, 'static')));
app.use('/api', router);
app.use(errorHandler);


const start = async () => {
    try {
        await sequelize.authenticate();
        await sequelize.sync();
        app.listen(PORT, () => console.log(`Server started at port
${PORT}`));
    } catch (err) {
        console.log(err);
    }
}

start();
```

A4 server/routes/index.js

```
const Router = require('express');
const router = new Router();

const userRouter = require('./userRouter.js');
const postRouter = require('./postRouter.js');
const commentRouter = require('./commentRouter.js');
const likeRouter = require('./likeRouter.js');

router.use('/user', userRouter);
router.use('/post', postRouter);
router.use('/comment', commentRouter);
router.use('/likes', likeRouter);

module.exports = router;
```

A5 server/middleware/checkAuth.js

```
const jwt = require('jsonwebtoken');

module.exports = function (req, res, next) {
    if (req.method === 'OPTIONS') {
        next();
    }

    try {
        const token = req.headers.authorization.split(' ')[1];
        if (!token) {
```

```
            return res.json({
                message: 'Не авторизовано'
            });
        }

        const decoded = jwt.verify(token, process.env.JWT_SECRET);
        req.user = decoded;
        next();
    } catch (err) {
        res.json('Не авторизовано');
    }
};
```

A6 server/middleware/Validation.js

```
const { body } = require('express-validator');

const registerValid = [
    body('email', 'Неправильний формат пошти!').isEmail(),
    body('password', 'Пароль має бути не менше 5
символів!').isLength({ min: 5 }),
    body('name', 'Занадто коротке ім\'я!').optional().isLength({
min: 2 }),
    body('userName', 'Занадто короткий нікнейм!').isLength({ min:
2 }),
];

const postValid = [
    body('title', 'Занадто короткий заголовок!').isLength({ min: 2
}).isString(),
    body('text', 'Занадто короткий опис!').optional().isLength({
min: 2 }).isString(),
    body('tags', 'Неправильний формат
тегів!').optional().isString(),
];

module.exports = {
    registerValid,
    postValid
}
```

A7 tokenGen

```
const tokenGen = (id, email, role) => {
    return jwt.sign(
        { id, email, role },
        process.env.JWT_SECRET,
        { expiresIn: '1d' }
    )
}
```

A8 sessionManager

```
const sessionManager = async (id, token) => {
    const session = await SessionModel.findOne({
        where: { userId: id }
    },);
```

```
    await session.destroy();

    return await SessionModel.create({ userId: id, token: token
});
}
```

A9 client/index.js

```
import React, { createContext } from 'react';
import { createRoot } from 'react-dom/client';
import App from './App';
import UserStore from './store/UserStore';
import PostStore from "./store/PostStore";
import CommentStore from './store/CommentStore';
import LikesStore from './store/LikesStore';
export const Context = createContext(null);
createRoot(document.getElementById('root')).render(
    <Context.Provider value={{
        user: new UserStore(),
        posts: new PostStore(),
        comments: new CommentStore(),
        likes: new LikesStore(),
    }}>
        <App />
    </Context.Provider>
);
```

A10 AppRouter.js

```
import React, { useContext } from 'react';
import { Routes, Route, Navigate } from 'react-router-dom';
import { authRoutes, publicRoutes } from '../routes';
import { MAIN_ROUTE } from '../utils/consts';
import { Context } from '..';

const AppRouter = () => {
    const { user } = useContext(Context);

    return (
        <Routes>
            {user._isAuth && authRoutes.map(({ path, Component })
=> (
                <Route key={path} path={path} element={<Component
/>} />
            ))}

            {publicRoutes.map(({ path, Component }) => (
                <Route key={path} path={path} element={<Component
/>} />
            ))}
            <Route path="*" element={<Navigate to={MAIN_ROUTE} />}
/>
        </Routes>
    );
};
```

```
export default AppRouter;
```

A11 utils/consts.js

```
export const LOGIN_ROUTE = '/login';
export const REGISTER_ROUTE = '/register';
export const MAIN_ROUTE = '/';
export const POST_ROUTE = '/post/posts';
export const PROFILE_ROUTE = '/profile';
export const PORTFOLIO_ROUTE = '/portfolio';
```

A12 http/index.js

```
import axios from "axios";

const $host = axios.create({
    baseURL: process.env.REACT_APP_API_URL
});

const $authHost = axios.create({
    baseURL: process.env.REACT_APP_API_URL
});

const authInterceptor = config => {
    config.headers.authorization = `Bearer
${localStorage.getItem('token')}`
    return config
}

$authHost.interceptors.request.use(authInterceptor)

export {
    $host,
    $authHost
}
```

A13 http/userAPI.js

```
import { $authHost, $host } from "./index";
import jwt_decode from "jwt-decode";

export const registration = async (email, password, name,
userName) => {
    const { data } = await $host.post('api/user/register', {
email, password, name, userName, role: 'USER' });
    localStorage.setItem('token', data.token);
    return jwt_decode(data.token);
}

export const updateUser = async (id, newUser) => {
    const { data } = await $authHost.patch('api/user//portfolio/'
+ id + '/profile', newUser);
    return data;
}
```

```
export const login = async (email, password) => {
    const { data } = await $host.post('api/user/login', { email,
password });
    localStorage.setItem('token', data.token);
    return jwt_decode(data.token);
}

export const check = async () => {
    const { data } = await $authHost.get('api/user/auth',);
    localStorage.setItem('token', data.token);
    return jwt_decode(data.token);
}

export const logout = async (token) => {
    await $authHost.post('api/user/logout', { token });
    localStorage.removeItem('token');
}

export const getUser = async () => {
    const { data } = await $authHost.get('api/user/user', );
    return data;
}
```

A14 Posts.js

```
import React, { useContext } from "react";
import { Context } from "../..";
import Post from "./Post";
import { observer } from "mobx-react-lite";
import { useParams } from "react-router-dom";

const Posts = observer(() => {
  const { posts } = useContext(Context);
  const { id } = useParams();

  const filteredPosts = id
    ? posts.posts.filter((post) => post.userId === Number(id))
    : posts.posts;

  return (
    <div
      style={{
        display: "grid",
        gridTemplateColumns: `repeat(auto-fit, minmax(250px,
${posts.posts.length > 4 ? "1fr" : "25%"
          }))`,
      }}
    >
      {filteredPosts.map((post) => (
        <Post key={post.id} post={post}/>
      ))}
    </div>
  );
```

```
});

export default Posts;
```

A15 Post.js

```
import React, { useEffect, useState } from "react";
import { observer } from "mobx-react-lite";
import { useNavigate } from "react-router-dom";
import "./css/OnePost.css";
import { POST_ROUTE } from "../../utils/consts";
import { fetchAuthor } from "../../http/postAPI";

const Post = observer(({ post }) => {
  const navigate = useNavigate();
  const [author, setAuthors] = useState({});

  const handleClick = () => {
    navigate(POST_ROUTE + "/" + post.id);
  };

  const id = post.id;
  useEffect(() => {
    fetchAuthor(id).then((data2) => setAuthors(data2));
  }, [id]);

  return (
    <div className="one-post-container" onClick={handleClick}>
      <div className="one-post-image-container">
        <img
          className="one-post-image"
          src={process.env.REACT_APP_API_URL + post.imageUrl}
          alt="Post"
        />
        <div className="one-post-details">
          <div className="one-post-details-content">
            <h1>{author.userName}</h1>
            <div>{post.title}</div>
          </div>
        </div>
      </div>
    </div>
  );
});

export default Post;
```

A16 Comments.js

```
import React, { useContext, useEffect } from "react";
import { observer } from "mobx-react-lite";
import CommentCard from "./CommentCard";
import CommentAdd from "./CommentAdd";
import { Context } from "../..";
import { fetchCommentsBy } from "../../http/commentAPI";
```

```
const Comments = observer(({ postId }) => {
    const { comments } = useContext(Context);
    const { user } = useContext(Context);

    useEffect(() => {
        fetchCommentsBy(postId).then(data => {
            comments.setComments(data);
        });
    }, [comments, postId]);

    return (
        <div style={{maxWidth: '750px'}}>
            <h3 style={{ marginTop: '2rem' }}>Коментарі:</h3>
            {comments.comments.length > 0 ? (
                comments.comments.map(comment => (
                    <CommentCard
                        key={comment.id}
                        comment={comment}
                    />
                ))
            ) : (
                <div style={{ color: '#adb5bd', fontSize: '20px',
textAlign: 'center',  minWidth: '400px' }}>
                    - коментарів немає -
                </div>
            )}
            {user.isAuth ? (
                <CommentAdd postId={postId} />) : (
                <div style={{ color: '#adb5bd', fontSize: '18px',
textAlign: 'center', minWidth: '400px' }}>
                    Авторизуйтеся, щоб коментувати
                </div>
            )}
        </div>
    );
});

export default Comments;
```