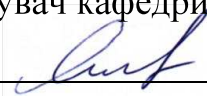


МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова
Навчально-науковий інститут комп'ютерних наук та управління проектами
Кафедра інформаційних управляючих систем та технологій

"Допущений до захисту"

Завідувач кафедри ІУСТ



к.т.н, доц. Михелєв І.Л.

" ____ " _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти "бакалавр"

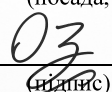
на тему: «Розроблення web-ресурсу для демонстрації художніх робіт»

Виконав: студент групи 4144


_____ Лактіонов М.С.
(підпис)

Керівник роботи:

викладач

(посада, науковий ступінь вчене звання)

_____ Задніпряна О.Б.
(підпис)

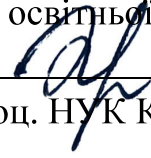
м. Миколаїв – 2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет кораблебудування імені адмірала Макарова
Навчально-науковий інститут комп'ютерних наук та управління проектами
Кафедра інформаційних управляючих систем та технологій
Спеціальність 126 "Інформаційні системи та технології"
Освітня програма "Інформаційні системи та технології"

"ЗАТВЕРДЖУЮ"

Гарант освітньої програми


к.т.н, доц. НУК Книрик Н.Р.

" ____ " ____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
на здобуття ступеня вищої освіти "бакалавр"

Студенту Лактіонову Максиму Сергійовичу

(прізвище, ім'я, по батькові)

1. Тема роботи: Розроблення web-ресурсу для демонстрації художніх робіт

Керівник роботи Задніпряна Ольга Борисівна

Затверджені наказом ректора № 295-уч від "27" березня 2025 року.

2. Термін подання роботи: 20 червня 2025 р.

3. Вихідні дані до проекту (роботи) ДСТУ щодо обробки інформації, літературні джерела, технічна документація на існуючі аналоги систем,

4. Перелік питань, що належать до розробки (найменування розділів) аналіз предметної галузі та постановка задач, розробка проекту інформаційної підсистеми, реалізація проекту підсистеми, охорона праці

5. Перелік презентаційних матеріалів вступ, мета та актуальність роботи, схема запропонованої концепції, діаграма варіантів використання, концептуальна модель, фізична модель, діаграма компонентів, діаграма класів, діаграма переходу станів, діаграма розгортання системи, результати розробки, висновок

6. Консультанти розділів роботи

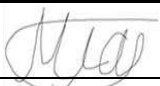
Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Задніпряна О.Б.	03.02.2025	22.02.2025
2	Задніпряна О.Б.	22.02.2025	21.03.2025
3	Задніпряна О.Б.	21.03.2025	25.04.2025
4	Задніпряна О.Б.	25.04.2025	07.06.2025

7. Дата видачі завдання 03 лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів роботи	Примітка
1	Аналіз предметної галузі	13.02.2025	
2	Постановка задачі	15.02.2025	
3	Розробка концепції	22.02.2025	
4	Розробка проекту ІС	21.03.2025	
5	Реалізація проекту	25.04.2025	
6	Розробка питань з охорони праці	07.06.2025	
7	Розробка графічних матеріалів	12.06.2025	
8	Подання роботи на перевірку на плагіат	10.06.2025	
9	Подання роботи рецензенту	15.06.2025	
10	Подання роботи на захист	20.06.2025	

Студент


(підпис)

Лактіонов М.С.

(ПІБ)

Керівник роботи


(підпис)

Задніпряна О.Б.

(ПІБ)

АНОТАЦІЯ

Дипломна робота присвячена розробленню web-ресурсу для демонстрації художніх робіт.

Об'єктом розробки є веб-галерея.

Предметом розробки є розроблення web-ресурсу для демонстрації художніх робіт.

Тема актуальна і може становити інтерес як для широкого кола громадськості, так і для художників і фахівців в області цифрового мистецтва. Поширення художників у сфері розважальної індустрії та мережі інтернет вимагає наявності зручної платформи для презентації своїх робіт та поширення будь-яких типів контенту, які художники можуть надати іншим митцям або звичайним людям, а також для комунікації з потенційними роботодавцями та клієнтами.

Web-ресурс написаний з використанням мови програмування Java, мови розмітки HTML, CSS та СУБД MySQL.

Дипломна робота містить інформацію про сучасне цифрове мистецтво, створення веб-додатків, постановку задачі, проект та розробку програмного забезпечення, результати розробки, демонстрацію роботи web-ресурсу, розділ охорони праці, а також текст програми та інструкцію користувача.

Пояснювальна записка до дипломної роботи виконана на 117 сторінках, містить 22 рисунки, 21 таблицю, 3 додатки та список використаної літератури з 22 найменувань.

Ключові слова: веб-ресурс, веб-галерея, демонстрація художніх робіт, цифрове мистецтво, веб-додаток, художники, візуальне представлення, користувацький інтерфейс, система управління контентом, платформа для митців, презентування цифрового контенту.

ABSTRACT

The thesis is devoted to the development of a web resource for demonstrating works of art.

The object of the development is a web gallery.

The subject of the development is the development of a web resource for demonstrating works of art.

The topic is relevant and may be of interest to both the general public and artists and specialists in the field of digital art. The spread of artists in the entertainment industry and the Internet requires a convenient platform for presenting their works and distributing any types of content that artists can provide to other artists or ordinary people, as well as for communication with potential employers and clients.

The web gallery application is written using the Java programming language, HTML markup languages, CSS and MySQL DBMS.

The thesis contains information about modern digital art, creating web applications, problem statement, software design and development, development results, demonstration of the work of the web gallery, occupational safety section, as well as the program text and user manual.

The explanatory note to the thesis is 117 pages long, contains 22 figures, 21 tables, 3 appendices, and a list of 22 references.

Keywords: web resource, web gallery, art display, digital art, web application, artists, visual representation, user interface, content management system, platform for artists, digital content presentation.

ЗМІСТ

ВСТУП.....	8
РОЗДІЛ 1 Опис предметної області та постановка задачі	10
1.1 Аналіз предметної області	10
1.2 Аналіз існуючих програмних продуктів.....	13
1.3 Розробка концепції на основі аналізу продуктів	17
1.4 Вимоги до додатку web-ресурсу.....	19
1.5 Постановка задачі.....	20
1.6 Вибір засобів розробки.....	21
РОЗДІЛ 2 Проектування програмного забезпечення	25
2.1 Розробка діаграми варіантів використання	25
2.2 Опис основних сутностей.....	26
2.3 Розробка логічної моделі бази даних	30
2.4 Розробка фізичної моделі бази даних	31
2.5 Розробка структури сайту web-ресурсу	36
2.6 Створення дизайну графічного інтерфейсу web-ресурсу	37
2.7 Розробка веб-додатку.....	38
2.8 Тестування функціональності додатку	49
2.9 Розробка діаграми послідовності	51
2.10 Розробка діаграми компонентів та діаграми розгортання	53
РОЗДІЛ 3 Результати розробки програмного забезпечення.....	55
РОЗДІЛ 4 Охорона праці	58
4.1 Вступ.....	58
4.2 Характеристика приміщення	58
4.3 Мікрокліматичні умови	60
4.4 Освітлення.....	61

4.5 Шум та вібрація	61
4.6 Випромінювання	62
4.7 Ергономіка робочого місця	62
4.8 Висновки	63
Висновки	64
Список використаних джерел	65
ДОДАТКИ.....	67
Додаток А. Технічне завдання	67
Додаток Б. Інструкція користувача	71
Додаток В. Текст програми.....	82

ВСТУП

Одним з найбільш поширених способів комунікації в людській культурі завжди були зображення, ілюстрації, візуальні образи. Саме цим здавна займалися і займаються різного роду художники. Однак мистецтво змінюється разом з плином часу, і тепер художники це не ті люди, які проводять місяці в підвалах або на вулиці, намагаючись максимально точно створити репродукцію місцевості або людини, а вкрай активні люди, що створюють найрізноманітніший візуальний контент в терміни які раніше здавалися неймовірними. Цьому посприяли як розвиток цифрового малювання, що скорочує час роботи в рази, так і розвиток інтернету.

Сучасне мистецтво наразі найбільш поширене в мас-культурі, таким як інтернет, відеоігри, фільми, мультфільми, візуальні новели і безліч інших продуктів.

Ілюстрації, постери, концепт-арти, дизайни персонажів, істот і тварин, оточення, предметів, ключові кадри фільмів («сторіборди») – все це створюється саме 2D та 3D художниками. У даній сфері зачастіше не художник шукає роботодавця, а роботодавець художника. Портфоліо художника є його «резюме», тому web-ресурсу є одним з найважливіших та найефективніших способів комунікації художників не тільки з їх аудиторією і один з одним, а з потенційними роботодавцями, які при наявності портфоліо у вигляді галереї можуть відразу оцінити навички та профіль художника, і вирішити, чи підходить він для роботи.

Метою дипломної роботи є розробка web-ресурсу що зможе надати художникам зручну площадку для розміщення та презентації своїх робіт, а також можливість ділитися і іншими типами контенту, які може надати художник: навчальні матеріали, процеси створення робіт, різні файли, новини і т.п.

Для досягнення поставленої мети необхідно розв'язати наступні задачі:

- 1) Проаналізувати дані, описані в предметній області. Визначити таблиці, необхідні для подання предметної області в рамках реляційної моделі даних;
- 2) Виконати аналіз існуючих рішень та розробити концепцію системи;
- 3) Спроекувати базу даних web-ресурсу;
- 4) Розробити структуру і дизайн інтерфейсу клієнтської частини веб-додатку;
- 5) Провести програмну реалізацію серверної частини веб-додатку;
- 6) Провести тестування функціональності веб-додатку.

РОЗДІЛ 1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області

Веб-галерея - платформа, що знаходиться в мережі інтернет, яка призначена для демонстрації і перегляду цифрових або оцифрованих художніх творів користувачів, обміну думками та можливістю ділитися різним контентом. Web-ресурсу дуже поширені в зв'язку з все більш глибоким розвитком та інтеграцією цифрового мистецтва в медіа, яке зараз превалює в усіх сферах розважальної індустрії.

Цифрове мистецтво - напрям в медіамистецтві, заснований на використанні інформаційних (комп'ютерних) технологій, результатом якої є художні твори в цифровій формі. [7]

Веб-сайт - інформаційна одиниця в інтернеті, ресурс з веб-сторінок (документів), які об'єднані спільною темою і пов'язані між один з одним за допомогою посилань. Сайт зареєстрований на одну юридичну або фізичну особу і обов'язково прив'язаний до конкретного домену, який є його адресою. Сайт може складатися як з однієї сторінки, так багатьох. Кожна сторінка ресурсу - це текстовий файл або їх набір, написаний на спеціальній мові розмітки або програмування (HTML, PHP, CSS і ін.). Після завантаження на комп'ютер файли цього формату обробляються в браузері. В кінцевому підсумку користувач бачить завантажену сторінку сайту. [2]

Структурно сайт web-ресурсу складається з невеликої кількості сторінок і не містить багаторівневих меню. Графічний дизайн web-ресурсу може варіюватися в дуже широкому діапазоні - від простого, до вельми складного, ексклюзивного. Головною рисою web-ресурсу є акцент на користувацькому контенті.

Веб-галерея відрізняється стислістю, ємністю і лаконічністю поданої інформації, і мінімалізмом функціональних можливостей, за рахунок чого досягається легкість сприйняття у відвідувачів і клієнтів.

Метою створення web-ресурсу є презентація робіт дизайнерів, художників, фотографів, майстрів різного профілю для того, щоб вони могли набувати популярності серед інших користувачів і як наслідок привертати увагу потенційних роботодавців, клієнтів і замовників.

Загалом веб-галерея дозволяє користувачам завантажувати на сайт свої цифрові зображення, що будуть автоматично поміщені в загальну галерею, де знаходяться зображення усіх користувачів, переглядати зображення та сторінки інших авторів, залишати відмітки «подобається» і коментарі до робіт, а також ділитися різним контентом, новинами і іншим. На багатьох сайтах також існує можливість обміну особистих повідомлень між користувачами.

Розробка веб-сайту складається з таких основних операцій і потребує врахування таких основних компонентів сайту:

- розробка концепції сайту;
- створення зручної навігаційної системи сайту;
- розробка інтерфейсу сайту;
- розробка індивідуального дизайну сайту;
- веб-програмування і верстання;
- створення веб-додатків;
- контент (зміст) сайту.

Веб-додаток — додаток, в якому клієнтом є оглядач Інтернету, а сервером - веб-сервер. Оглядач Інтернету може бути реалізацією так званих тонких клієнтів. Він відображає веб-сторінки і, як правило, входить до складу операційної системи, а його оновлення та супровід виконує постачальник операційної системи. Логіка додатка зосереджена на сервері, а оглядач Інтернету найчастіше відповідає лише за відображення інформації, завантаженої з сервера, і за передачу на сервер даних користувача. Однією з переваг такого підходу є той факт, що клієнти не залежать від конкретної операційної системи користувача, і веб-додатки, таким чином, є крос-платформними сервісами. [6]

Веб-додаток отримує запит від клієнта і виконує обчислення, після цього формує веб-сторінку і відправляє її клієнту через мережу з використанням протоколу HTTP.

Особливості веб-додатків:

- Веб-додаток може бути клієнтом інших служб, наприклад, бази даних або іншого веб-додатку, розташованого на іншому сервері.
- Додаток створюється один раз для довільно вибраної платформи і на ній розгортається. Проте різна реалізація HTML, CSS, DOM і інших специфікацій в браузерях може викликати проблеми при розробці веб-додатків і подальшої підтримки.
- Простота використання: відвідувач сайту може протягом 10-15 секунд зрозуміти, як почати користуватися сайтом.
- Великий час реакції на дії користувача: від моменту натискання на посилання до появи результату часто проходить більше 1 секунди (а буває й більше). Тому людина встигає помітити затримку. Задача розробника - максимально скоротити час реакції додатка.
- Велика кількість користувачів: в великих веб-проектах на сайт приходять десятки користувачів на секунду. Тому варто звертати увагу на оптимізацію швидкості виконання програми.
- Робота з великими масивами даних.
- Мережні, розподілені, паралельні обчислення, системи агентів: сучасні додатки в інтернеті часто спілкуються між собою. Наприклад, сайт "електронний магазин" використовує сайт банку, щоб прийняти оплату за товар. Розважальні сайти показують рекламні блоки з інших сайтів і т.д.

Варто зазначити, що програмне забезпечення – сервіс, а не товар. Однією з головних характеристик сучасних інтернет-додатків є те, що вони поширюються у вигляді сервісу, а не товару. Це, у свою чергу, веде до фундаментальних змін в бізнес-моделях компаній-розробників програм.

Компанія або розробник повинна мати контроль над процесами. Розробка додатків повинна супроводжуватись умінням організувати щоденні операції для підтримки роботи цих додатків. Масивні веб-додатки, які використовуються користувачами щодня, повинні і підтримуватись щодня.

Архітектура веб-додатку [1]:

Веб-додаток складається з клієнтської і серверної частин, тим самим реалізуючи технологію «клієнт-сервер».

Клієнтська частина реалізує користувацький інтерфейс, формує запити до сервера і обробляє відповіді від нього.

Серверна частина отримує запит від клієнта, виконує обчислення, після цього формує веб-сторінку і відправляє її клієнту через мережу з використанням протоколу HTTP.

Саме веб-додаток може виступати в якості клієнта інших служб, наприклад, бази даних або іншого веб-додатку, розташованого на іншому сервері.

1.2 Аналіз існуючих програмних продуктів

Найбільш популярною платформою для даного типу контенту наразі є «Instagram» та «DeviantArt», а для поширення ресурсів існує веб-сайт під назвою «Gumroad».

«Instargram» має наступні функції:

- Завантаження фотографій та відео;
- Публікація «історій» - фото або коротких відео-фрагментів, які існують 24 години;
- Пошук за описом та хештегами;
- Можливість залишати «лайки» та коментарі до зображень.
- Перегляд загальної галереї, фільтри за категоріями та сортування за популярністю;
- Можливість застосування фільтрів до зображень;

- Можливість відправки персональних повідомлень.

Недоліком даної платформи є:

- Складність пошуку через дуже велику кількість контенту;
- Великий обсяг цільової аудиторії: від звичайних людей до майстрів різного профілю;
- Незручні завантаження та перегляд зображень: зображення повинні мати приблизно квадратний формат для коректної роботи, широкоформатні зображення повинні бути обрізані для завантаження на сайт;
- Зображення втрачають якість після загрузки на сервер, та їх не можна завантажити або розглянути у оригінальній роздільній здатності;
- Можливість ввести лише одне посилання на зовнішній ресурс.

Головна сторінка сайту зображена на рис. 1.1:

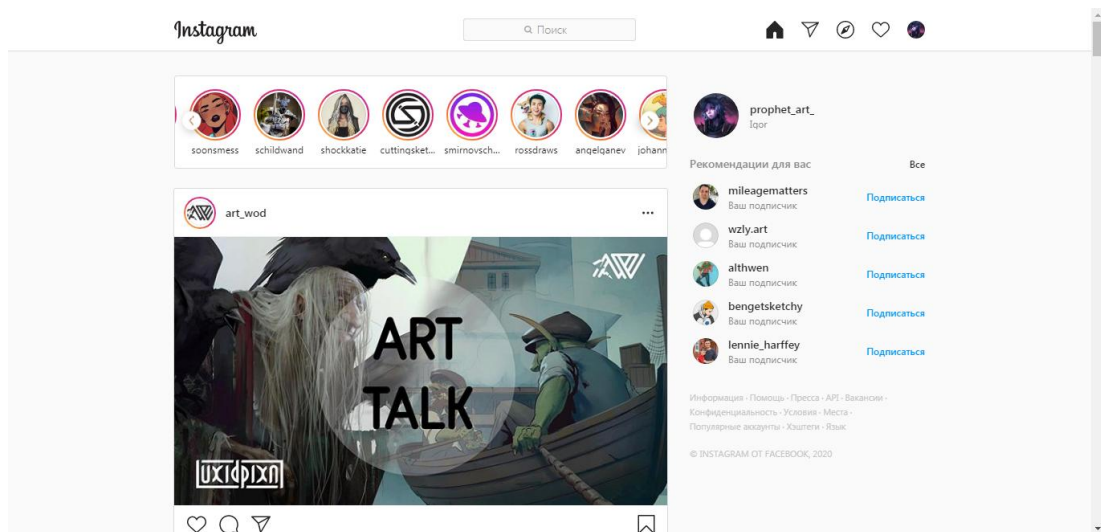


Рисунок 1.1 – Головна сторінка «Instagram»

На рис. 1.2 зображена сторінка профілю користувача.

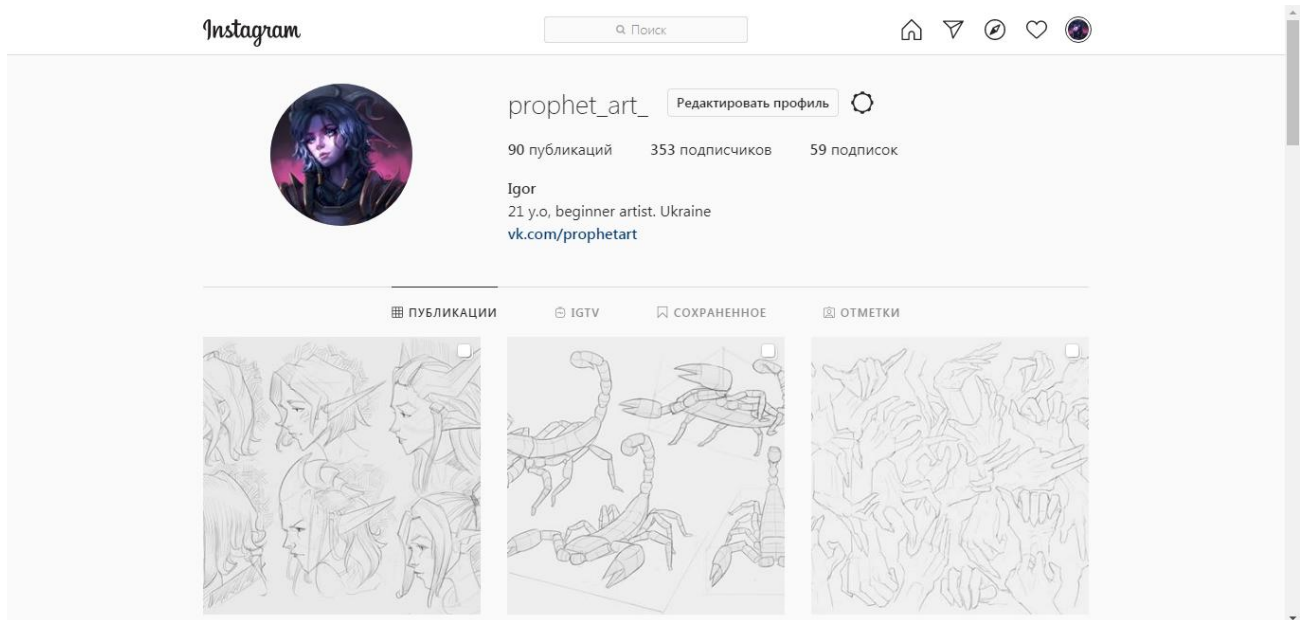


Рисунок 1.2 – Сторінка профілю «Instagram»

«DeviantArt» має більш розширені функції ніж «Instagram» та є більш орієнтованим на митців різного профілю.

Основними функціями даного сайту є:

- Завантаження/перегляд зображень;
- Можливість публікації журналів;
- Можливість створення/приєднання до груп за різними інтересами;
- Перегляд основної галереї, фільтрація за категоріями та сортування за популярністю;
- Пошук за описом і хештегами;
- Можливість створити користувацький дизайн сторінки користувача;
- Можливість залишати «лайки» та коментарі до зображень;
- Можливість додавати зображення інших користувачів до своїх користувацьких колекцій.

Недоліками даної платформи є:

- Незручний перегляд галереї користувача;

- Користувацькі дизайни можуть відволікати та відлякувати потенційних клієнтів та роботодавців із-за перевантаження їх непотрібним контентом;
- Надлишкова кількість інформації та реклами;
- Великий обсяг цільової аудиторії, що веде до великої кількості контенту;
- Незадовільна модерація контенту.

Зовнішній вигляд сайту зображений на рис 1.3.

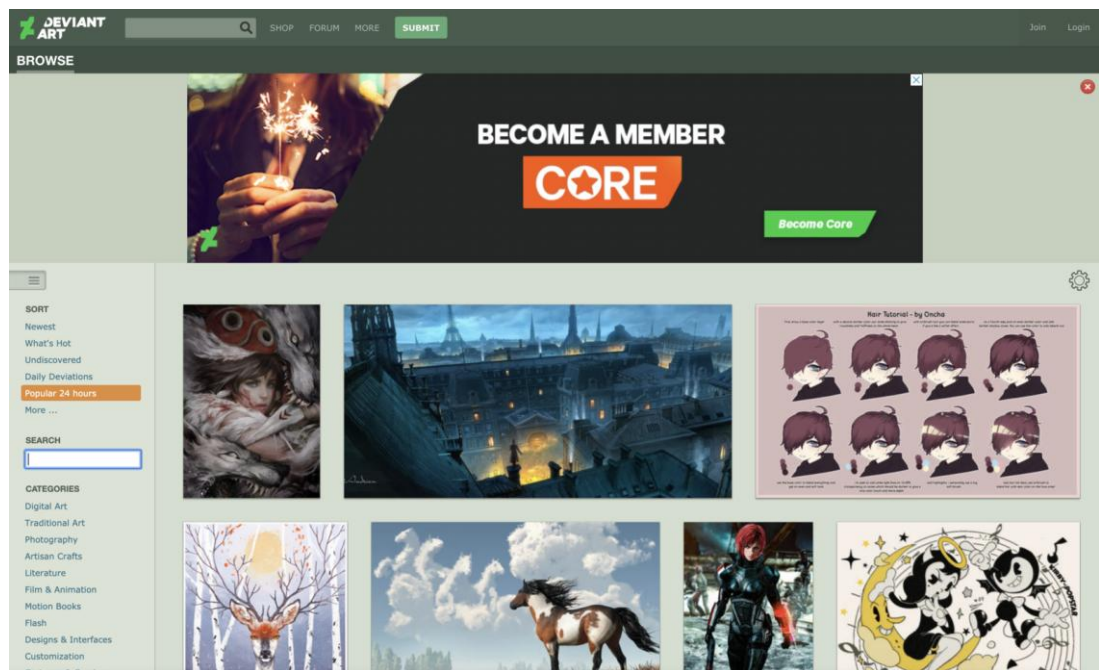


Рисунок 1.3 – Зовнішній вигляд сайту «DeviantArt»

«Gumroad» – сайт, який слугує лише для поширення та покупки продуктів та контенту, який надають різні люди, в тому числі і художники, але це єдина його функція. Зовнішній вигляд сайту зображений на рис 1.4:

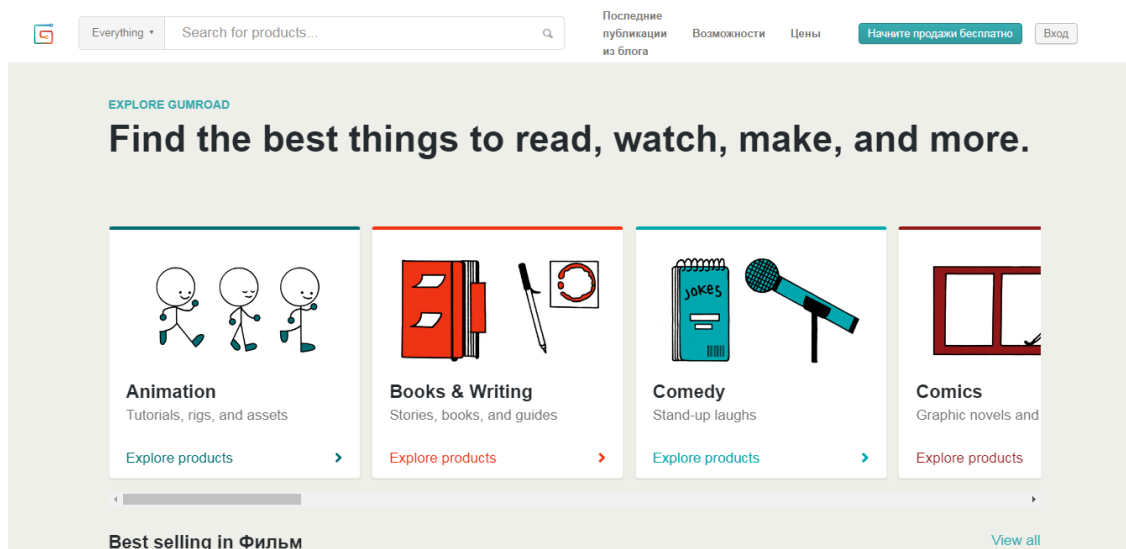


Рисунок 1.4 – Зовнішній вигляд сайту «Gumroad»

1.3 Розробка концепції на основі аналізу продуктів

Аналіз існуючих продуктів показав, що сайти такого типу перевантажені надлишковою інформацією, рекламою та іншими несуттєвими елементами, що відволікають звичайних користувачів та відлякують потенційних роботодавців. Незручний перегляд галереї користувача або загальної галереї також може відштовхнути роботодавців, так як їм необхідно переглядати велику кількість портфоліо за день для пошуку потрібного їм кандидата, і добре виконаний сайт посприє як роботодавцю, так і художнику.

Ще одним суттєвим недоліком є занадто широка цільова аудиторія, тобто такі сайти призначені як для художників, так і для фотографів, дизайнерів, рукодільних майстрів та інших. І хоча це позитивно впливає на охоплення аудиторії, такий підхід лише перевантажує сайт контентом.

І нарешті для поширення ресурсів використовуються інші платформи, що є незручним як для художників так і користувачів, що їх хочуть їх придбати або завантажити.

Метою дипломної роботи є створення сайту, що об'єднає усі необхідні художникам та користувачам функції у межах одного сайту, а цільовою аудиторією зробити саме художників, що займаються як 2D-графікою, так і 3D:

так, сайт буде містити лише цифрові або оцифровані роботи, прийнятну кількість контенту та мінімум надлишкової інформації або реклами.

Обрана архітектура системи: 3-х рівнева «клієнт-сервер», що зображена на рис. 1.5.

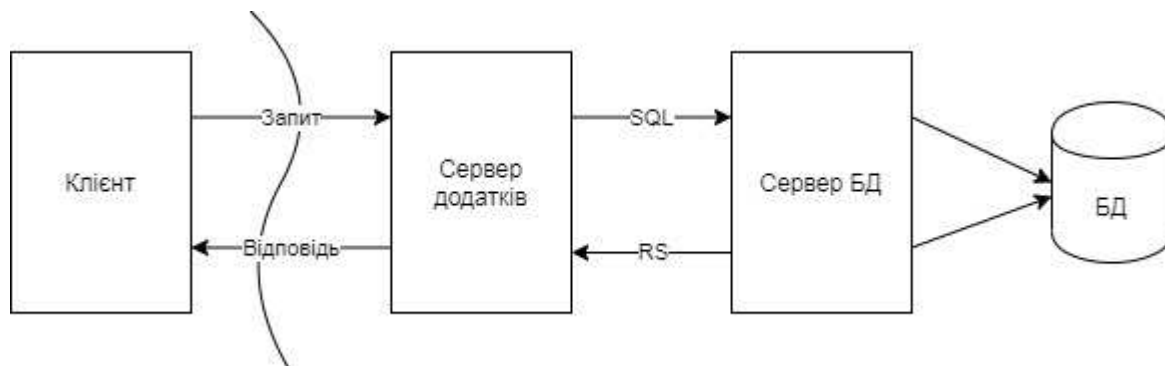


Рисунок 1.5 – Архітектура системи

Представимо графічно існуючу концепцію системи web-ресурсу (рис. 1.6):



Рисунок 1.6 – Існуюча концепція web-ресурсу

Оскільки метою роботи є об'єднання усіх необхідних художникам та користувачам функцій у межах одного сайту, було вирішено додати модулі публікації постів (новин) та завантаження ресурсів (продуктів, навчальних

матеріалів, файлів художніх проектів, та ін.) у систему web-ресурсу. Запропонована концепція web-ресурсу представлена на рис 1.7:

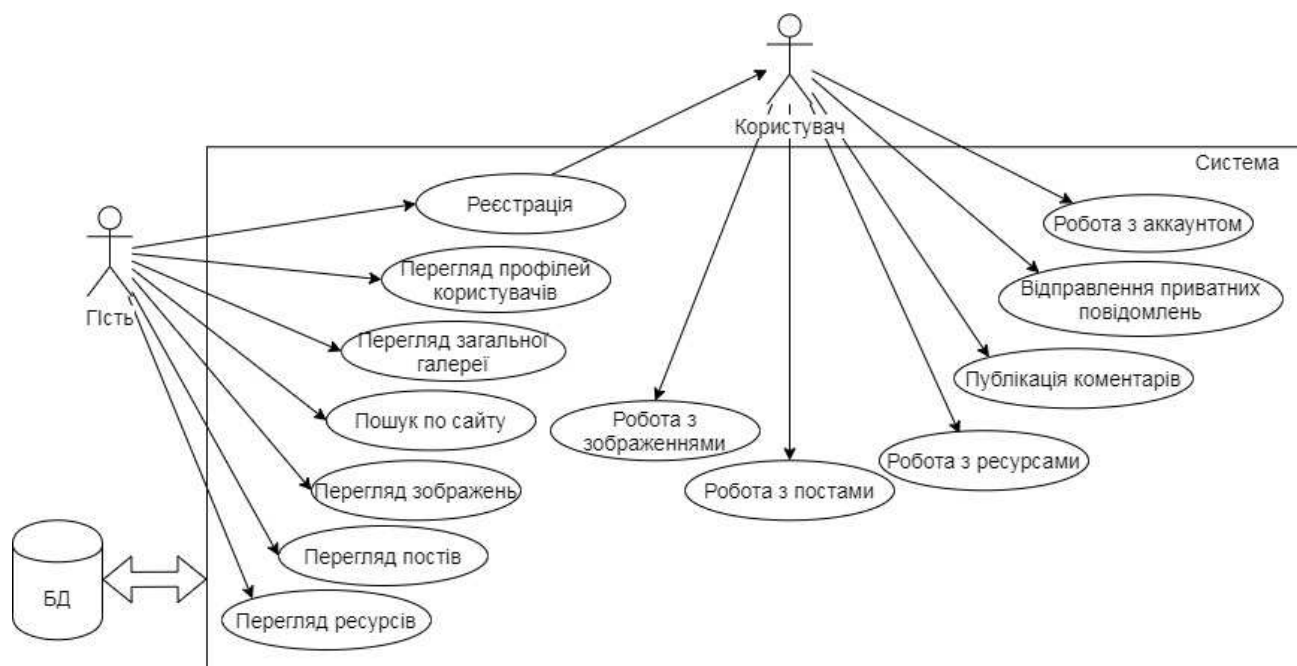


Рисунок 1.7 – Запропонована концепція web-ресурсу

1.4 Вимоги до додатку web-ресурсу

Загальні вимоги до web-ресурсу, що буде розроблятися:

- наявність форми входу/реєстрації.
- наявність горизонтального меню за допомогою якого забезпечується навігація;
- наявність модуля здійснення пошукових запитів;
- можливість завантаження зображень і вказівка до якої категорії вони належать;
- наявність загальної галереї, в якій відображаються зображення всіх користувачів, а також можливість їх фільтрації за категоріями;
- розміщення інформації про користувачів;
- можливість публікації постів (новин) користувачами;
- можливість завантаження ресурсів користувачами та їх розповсюдження;

- можливість залишати коментарі та відмітки «Подобається» до зображень та постів;
- можливість обміну особистими повідомленнями між користувачами.

Структура web-ресурсу повинна передбачати подальший розвиток за допомогою додавання нових розділів, а також можливість їх зміни, впровадження додаткових блоків і модулів.

Основні вимоги до дизайну web-ресурсу:

- графічні елементи сайту повинні бути розроблені з урахуванням специфіки подання інформації на веб-сайтах;
- основні елементи меню та інші елементи навігації повинні бути доступні з усіх сторінок сайту;
- відсутність надлишкової інформації на сторінках сайту;
- дизайн сайту повинен бути сучасним і мінімалістичним, оскільки увага користувача має концентруватися на творчих матеріалах користувачів.

1.5 Постановка задачі

Метою дипломної роботи є розробка додатку web-ресурсу для використання багатьма користувачами, придбання теоретичних знань і практичних умінь з розробки сайтів із застосуванням засобів програмування Java, HTML, CSS, а також з проектування і створення баз даних в середовищі MySQL.

Для реалізації мети дипломної роботи необхідно вирішити наступні завдання:

- 1) Проаналізувати дані, описані в предметній області. Визначити таблиці, необхідні для подання предметної області в рамках реляційної моделі даних;
- 2) Спроекувати логічну та фізичну структуру бази даних web-ресурсу;
- 3) Розробити структуру і дизайн інтерфейсу клієнтської частини веб-додатку;
- 4) Провести програмну реалізацію серверної частини веб-додатку;

5) Провести програмну реалізацію веб-додатку та реалізувати наступні функції:

- Реєстрація та вхід за допомогою логіна/пароля користувача.
- Завантаження, перегляд та видалення зображення.
- Редагування інформації про зображення.
- Перегляд сторінки користувача.
- Редагування інформації на сторінці користувача.
- Видалення сторінки користувача.
- Перегляд сторінок та зображень інших користувачів.
- Публікація, перегляд, редагування та видалення постів користувачів.
- Завантаження, перегляд, редагування та видалення ресурсів користувачами.
- Можливість залишати коментарі та відмітки «Подобається» до постів та зображень.
- Пошук користувачів, зображень, та постів.
- Обмін особистими повідомленнями між користувачами.

6) Провести тестування функціональності веб-додатку.

Результатом виконання поставлених завдань повинен стати додаток web-ресурсу.

1.6 Вибір засобів розробки

Обрана мова програмування – Java з використанням JPA та Hibernate, середовище розробки - IntelliJ IDEA, система управління базами даних – MySQL, веб-сервер – GlassFish. Мова розмітки - HTML та CSS. Середовище для розробки дизайну – Adobe Photoshop.

IntelliJ IDEA — комерційне інтегроване середовище розробки для різних мов програмування (Java, Python, Scala, PHP та ін.) від компанії JetBrains. [11]

Java - строго типізована об'єктно-орієнтована мова програмування. В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи. [12]

Java EE - набір специфікацій і відповідної документації для мови Java, яка описує архітектуру серверної платформи для задач середніх і великих підприємств. [13]

Специфікації деталізовані настільки, щоб забезпечити переносимість програм з однієї реалізації платформи на іншу. Основна мета специфікацій - забезпечити масштабованість додатків і цілісність даних під час роботи системи. Java EE багато в чому орієнтована на використання її через веб, як в інтернеті, так і в локальних мережах.

Hibernate — засіб відображення між об'єктами та реляційними структурами (object-relational mapping, ORM) для платформи Java. Hibernate є вільним програмним забезпеченням, яке поширюється на умовах GNU Lesser General Public License. Hibernate надає легкий для використання каркас (фреймворк) для відображення між об'єктно-орієнтованою моделлю даних і традиційною реляційною базою даних. [14]

GlassFish — сервер застосунків з відкритим сирцевим кодом, який реалізує специфікації Java EE. [15]

HTML (англ. HyperText Markup Language — мова розмітки гіпертексту) — це мова тегів, якою пишуться гіпертекстові документи для мережі Інтернет. Веб-браузери отримують HTML-документи з веб-сервера або з локальної пам'яті і передають документи в мультимедійні веб-сторінки. HTML описує структуру веб-сторінки семантично і спочатку включені сигнали для зовнішнього вигляду документа. [10]

HTML впроваджує засоби для:

- створення структурованого документа шляхом позначення структурного складу тексту: заголовки, абзаци, списки, таблиці, цитати та інше;
- отримання інформації із Всесвітньої мережі через гіперпосилання;

- створення інтерактивних форм;
- включення зображень, звуку, відео, та інших об'єктів до тексту.

CSS — це спеціальна мова стилю сторінок, що використовується для опису їхнього зовнішнього вигляду. Самі сторінки написані мовами розмітки даних. [9]

CSS використовується авторами та відвідувачами веб-сторінок, щоб визначити кольори, шрифти, верстку та інші аспекти вигляду сторінки. Одна з головних переваг — можливість розділити зміст сторінки (або контент, наповнення, зазвичай HTML) від вигляду документу (що описується в CSS).

Таке розділення може покращити сприйняття та доступність контенту, забезпечити більшу гнучкість та контроль за відображенням контенту в різних умовах, зробити контент більш структурованим та простим, прибрати повтори тощо. Один і той самий HTML або XML документ може бути відображений по-різному залежно від використаного CSS. Стилi для відображення сторінки можуть бути:

- Стилi автора (інформація надана автором сторінки):
 - зовнішні таблиці стилів , найчастіше окремий файл або файли .css
 - внутрішні таблиці стилів, включені як частина документу або блоку
 - стилі для окремого елемента
- Стилi користувача
 - локальний .css-файл, вказаний користувачем для використання на сторінках і вказаний в налаштуваннях браузера.
- Стилi переглядача (браузера)
 - стандартний стиль переглядача, наприклад стандартні стилі для елементів, визначені браузером, використовуються коли немає інформації про стиль елемента або вона неповна.

MySQL — вільна система керування реляційними базами даних. Вона використовується, в першу чергу, для створення динамічних веб-сторінок, оскільки має підтримку з боку різноманітних мов програмування. [16]

MySQL вважається гарним рішенням для малих і середніх застосувань. Сирцеві коди сервера компілюються на багатьох платформах.

Можливості сервера MySQL:

- простота у встановленні та використанні;
- підтримується необмежена кількість користувачів, що одночасно працюють із БД;
- кількість рядків у таблицях може досягати 50 млн;
- висока швидкість виконання команд;
- наявність простої і ефективної системи безпеки.

Adobe Photoshop - багатофункціональний графічний редактор, розроблений і поширюваний фірмою Adobe Systems. В основному працює з растровими зображеннями, проте має деякі векторні інструменти. Широко використовується у багатьох різних галузях промисловості, в тому числі веб-дизайну, медицини, кіно, реклами, інженерії та архітектури. [8]

РОЗДІЛ 2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Розробка діаграми варіантів використання

UML - уніфікована мова моделювання - це мова графічного опису для об'єктного моделювання в області розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, який використовує графічні позначення для створення абстрактної моделі системи, що має назву «UML модель». UML був створений для визначення, візуалізації, проектування та документування програмних систем. [17]

Використання UML не обмежується моделюванням програмного забезпечення. Його також використовують для моделювання бізнес-процесів, системного проектування та відображення організаційних структур.

На даний момент додаток web-ресурсу не потребує адміністрації, так як основний контент завантажують самі користувачі. На рис. 2.1 відображена діаграма варіантів використання додатку.

Функціонал додатку розділений на дві частини:

1) Гість - може користуватися навігацією по сайту, пошуком, переглядом постів, зображень та профілів інших користувачів.

2) Користувач – після реєстрації та авторизації відвідувач сайту отримує свою сторінку та має можливість завантажувати, редагувати та видаляти зображення, пости та ресурси, а також відправляти особисті повідомлення до інших користувачів і змінювати інформацію свого профілю.

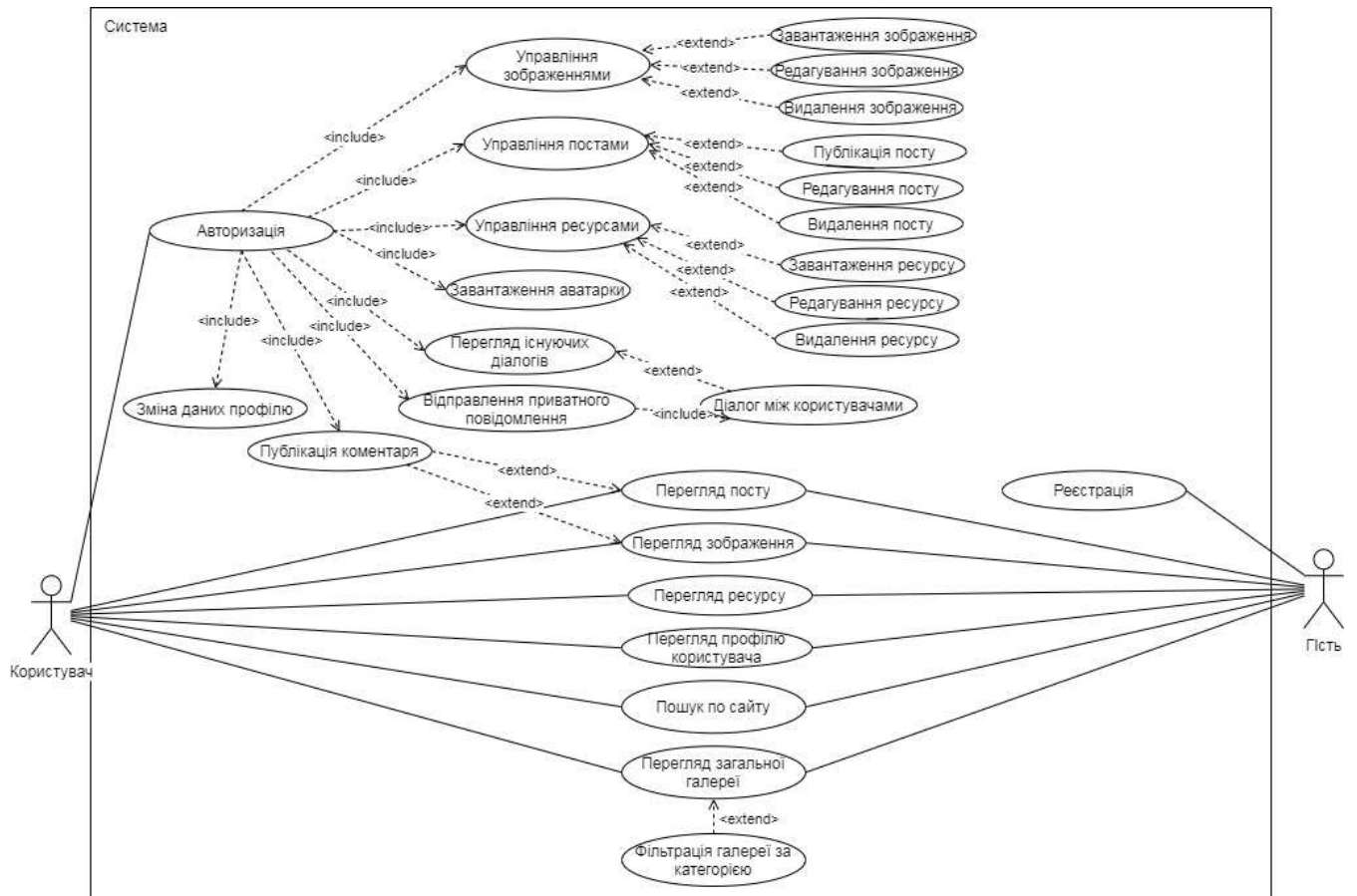


Рисунок 2.1 – Діаграма варіантів використання веб-додатку

2.2 Опис основних сутностей

Після аналізу предметної області основними сутностями бази даних web-ресурсу було виділено:

- Користувач
- Користувацькі посилання
- Зображення
- Категорія зображення
- Коментар зображення
- «Лайк» зображення
- Ресурс
- Категорія ресурсу
- Файли ресурсу

- Пост
- Коментар посту
- «Лайк» посту
- Повідомлення

Опис даних сутностей:

1. Користувач

Дана сутність зберігає інформацію про користувача, яку він вказує при реєстрації на сайті.

Атрибути:

- Логін;
- Пароль;
- Опис «Про себе»;
- Статус;
- Загальна інформація: ім'я, прізвище, e-mail, країна, місто, дата створення аккаунту - кожен атрибут зберігається в окремому полі.
- Посилання на файл зображення «аватару»

2. Користувацькі посилання

Дана сутність містить посилання на зовнішні сайти користувача (такі як Instagram, Vk, та ін.

Атрибути:

- Назва сайту;
- Посилання на сайт;

3. Зображення

Дана сутність містить інформацію безпосередньо про зображення, які можуть завантажувати користувачі на сайт.

Атрибути:

- Назва;
- Опис;
- Посилання на файл;
- Дата публікації;

- Категорія – ідентифікатор категорії с відповідної таблиці;
- Кількість переглядів;

4. Категорія зображення

Дана сутність зберігає основні категорії зображень, які можуть бути завантажені на сайт (напр. «Ілюстрація», «Концепт-арт», «3D-арт» та ін.), та призначена для більш зручного перегляду зображень.

Атрибути:

- Назва;
- Опис категорії;

5. Коментар зображення

Дана сутність зберігає коментарі які залишили користувачі до певних зображень.

Атрибути:

- Ідентифікатор зображення з однойменної таблиці.
- Ідентифікатор користувача з однойменної таблиці.
- Текст коментаря.
- Дата/час публікації.

6. «Лайк» зображення

Дана сутність зберігає відмітки «подобається» які залишили користувачі до певних зображень.

Атрибути:

- Ідентифікатор зображення з однойменної таблиці.
- Ідентифікатор користувача з однойменної таблиці.

7. Ресурс

Дана сутність зберігає інформацію про ресурси, які може завантажувати користувач та ділитися з іншими.

Атрибути:

- Назва.
- Опис.
- Посилання на файл зображення для передогляду ресурсу.

- Дата публікації.
- Ідентифікатор категорії ресурсу з однойменної таблиці.

8. Категорія ресурсу

Дана сутність зберігає основні категорії ресурсів, які можуть бути завантажені на сайт (файли процесів робіт, відеоуроки, цифрові «пензлі» для програм для малювання та ін.), та призначена для більш зручного перегляду ресурсів.

Атрибути:

- Назва;
- Опис категорії;

9. Файли ресурсу

Дана сутність зберігає файли до певного ресурсу.

Атрибути:

- Посилання на файл.
- Тип файлу.
- Розмір.
- Ідентифікатор ресурсу з однойменної таблиці.

10. Пост

Дана сутність зберігає інформацію про пости, які може публікувати користувач з метою розповісти новини, анонси, та ін.

Атрибути:

- Заголовок.
- Текст.
- Посилання на файл зображення (необов'язково).
- Дата публікації.
- Кількість переглядів.

11. Коментар посту

Дана сутність зберігає коментарі які залишили користувачі до певних постів.

Атрибути:

- Ідентифікатор поста з однойменної таблиці.
- Ідентифікатор користувача з однойменної таблиці.
- Текст коментаря.
- Дата/час публікації.

12. «Лайк» посту

Дана сутність зберігає відмітки «подобається» які залишили користувачі до певних постів.

Атрибути:

- Ідентифікатор поста з однойменної таблиці.
- Ідентифікатор користувача з однойменної таблиці.

13. Повідомлення

Дана сутність зберігає повідомлення які відправив користувач іншому користувачу.

Атрибути:

- Текст повідомлення.
- Дата/час відправлення повідомлення.
- Ідентифікатор користувача-відправника з однойменної таблиці.
- Ідентифікатор користувача-отримувача з однойменної таблиці.

2.3 Розробка логічної моделі бази даних

На основі діаграми варіантів використання та опису основних сутностей була створена логічна модель бази даних, що представлена на рис. 2.2:

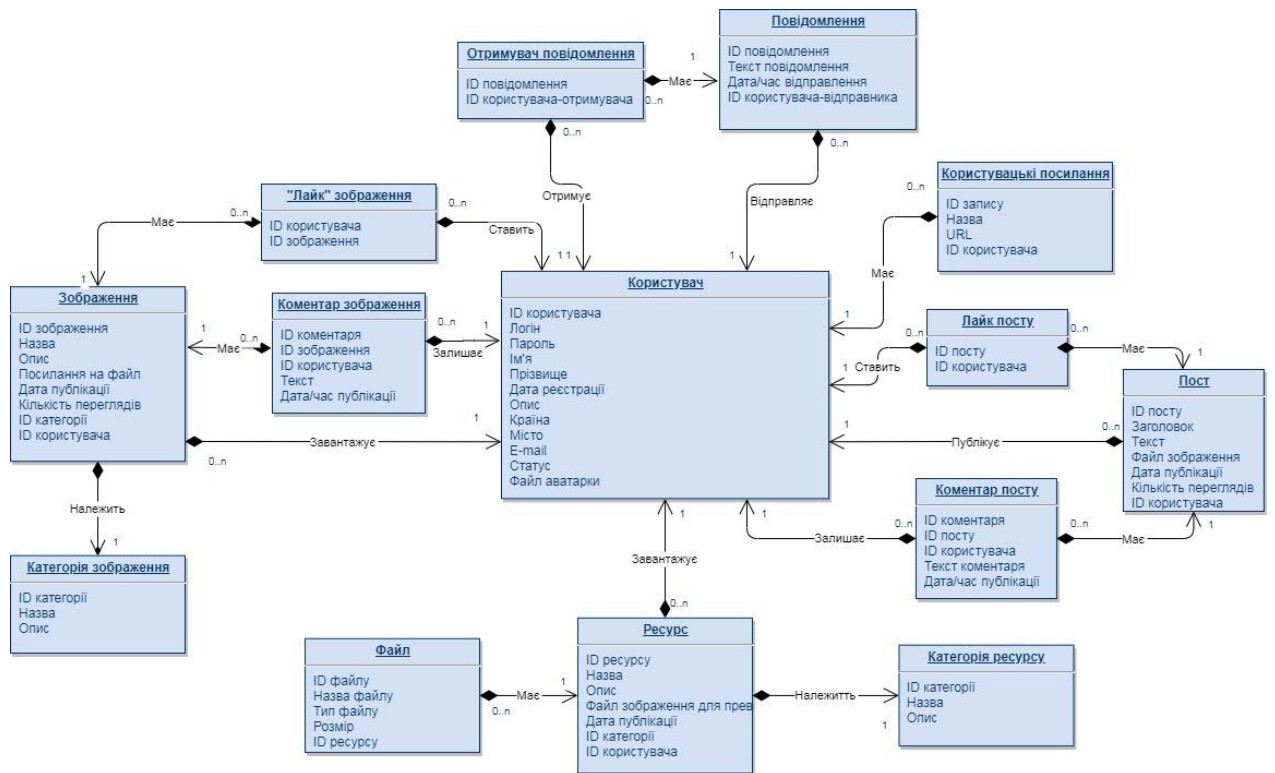


Рисунок 2.2 – Логічна модель бази даних

2.4 Розробка фізичної моделі бази даних

Фізична модель містить всю інформацію, необхідну для реалізації конкретної БД.

На основі логічної моделі бази даних була створена фізична модель бази даних. Дана модель містить безпосередні назви таблиць та атрибутів, які будуть використовуватися в подальшому, типи атрибутів, їх розмір та первинні і зовнішні ключі.

Фізична модель БД для MySQL відображена в таблицях 2.1-2.14

Таблиця 2.1

Таблиця «user»

Поле	Назва атрибуту	Тип	NN	Ключ
id_user	ID користувача	INT	Так	PK
Login	Логін	VARCHAR(45)	Так	
u_password	Пароль	VARCHAR(70)	Так	

Продовження таблиці 2.1

u_name	Ім'я	VARCHAR(50)	Ні	
u_surname	Прізвище	VARCHAR(50)	Ні	
creation_date	Дата реєстрації	DATETIME	Так	
description	Опис	VARCHAR(400)	Ні	
country	Країна	VARCHAR(50)	Ні	
email	Е-mail	VARCHAR(100)	Ні	
city	Місто	VARCHAR(50)	Ні	
user_status	Статус	VARCHAR(100)	Ні	
avatar_filename	Файл аватарки	VARCHAR(500)	Ні	

Таблиця 2.2

Таблиця «image»

Поле	Назва атрибуту	Тип	NN	Ключ
id_image	ID зображення	INT	Так	PK
title	Назва	VARCHAR(100)	Ні	
description	Опис	TEXT	Ні	
filename	Посилання на файл	VARCHAR(500)	Ні	
publish_date	Дата публікації	DATETIME	Ні	
views_counter	Кількість переглядів	INT	Ні	
id_category	ID категорії	INT	Так	FK
user_id_user	ID користувача	INT	Так	FK

Таблиця 2.3

Таблиця «image_likes»

Поле	Назва атрибуту	Тип	NN	Ключ
id_user	ID користувача	INT	Так	FK
id_image	ID зображення	INT	Так	FK

Таблиця 2.4

Таблиця «image_commentary»

Поле	Назва атрибуту	Тип	NN	Ключ
id_entry	ID коментаря	INT	Так	PK
id_image	ID зображення	INT	Так	FK
id_user	ID користувача	INT	Так	FK
comment_text	Текст	TEXT	Ні	
publish_date_time	Дата/час публікації	DATETIME	Так	

Таблиця 2.5

Таблиця «image_category»

Поле	Назва атрибуту	Тип	NN	Ключ
id_category	ID категорії	INT	Так	PK
cat_name	Назва	VARCHAR(40)	Ні	
description	Опис	VARCHAR(200)	Ні	

Таблиця 2.6

Таблиця «post»

Поле	Назва атрибуту	Тип	NN	Ключ
id_user_post	ID посту	INT	Так	PK
post_title	Заголовок	VARCHAR(100)	Так	
post_text	Текст	LONGTEXT	Так	
filename_image	Файл зображення	VARCHAR(500)	Ні	
publish_date	Дата публікації	DATETIME	Так	
views_counter	Кількість переглядів	INT	Так	
user_id_user	ID користувача	INT	Так	FK

Таблиця 2.7

Таблиця «post_likes»

Поле	Назва атрибуту	Тип	NN	Ключ
id_user	ID посту	INT	Так	FK
id_post	ID користувача	INT	Так	FK

Таблиця 2.8

Таблиця «post_commentary»

Поле	Назва атрибуту	Тип	NN	Ключ
id_entry	ID коментаря	INT	Так	PK
id_post	ID посту	INT	Так	FK
id_user	ID користувача	INT	Так	FK
comment_text	Текст	TEXT	Ні	
publish_date_time	Дата/час публікації	DATETIME	Так	

Таблиця 2.9

Таблиця «user_links»

Поле	Назва атрибуту	Тип	NN	Ключ
id_user_links	ID запису	INT	Так	PK
link_name	Назва	VARCHAR(45)	Так	
link_url	URL	TEXT	Так	
id_user	ID користувача	INT	Так	FK

Таблиця 2.10

Таблиця «message»

Поле	Назва атрибуту	Тип	NN	Ключ
id_message	ID повідомлення	INT	Так	PK
text_message	Текст повідомлення	TEXT	Так	
date_time	Дата/час відправлення	DATETIME	Так	
id_user_sender	ID користувача-відправника	INT	Так	FK

Таблиця 2.11

Таблиця «message_recipient»

Поле	Назва атрибуту	Тип	NN	Ключ
id_entry	ID запису	INT	Так	PK
id_message	ID повідомлення	INT	Так	FK
id_user_recipient	ID користувача-отримувача	INT	Так	FK

Таблиця 2.12

Таблиця «resource»

Поле	Назва атрибуту	Тип	NN	Ключ
id_resource	ID ресурсу	INT	Так	PK
title	Назва	VARCHAR(100)	Ні	
description	Опис	TEXT	Ні	
preview_image_filename	Файл зображення для превью	VARCHAR(500)	Ні	
publish_date	Дата публікації	DATETIME	Ні	
id_category	ID категорії	INT	Так	FK
user_id_user	ID користувача	INT	Так	FK

Таблиця 2.13

Таблиця «resource_category»

Поле	Назва атрибуту	Тип	NN	Ключ
id_resource_category	ID категорії	INT	Так	PK
category_name	Назва	VARCHAR(100)	Ні	
category_description	Опис	TEXT	Ні	

Таблиця 2.14

Таблиця «files»

Поле	Назва атрибуту	Тип	NN	Ключ
id_files	ID файлу	INT	Так	PK
filename	Назва файлу	VARCHAR(500)	Так	

type	Тип файлу	VARCHAR(45)	Hi	
size	Розмір	INT	Hi	
id_resource	ID ресурсу	INT	Так	FK

На рисунку 2.3 показано діаграму фізичної моделі бази даних.

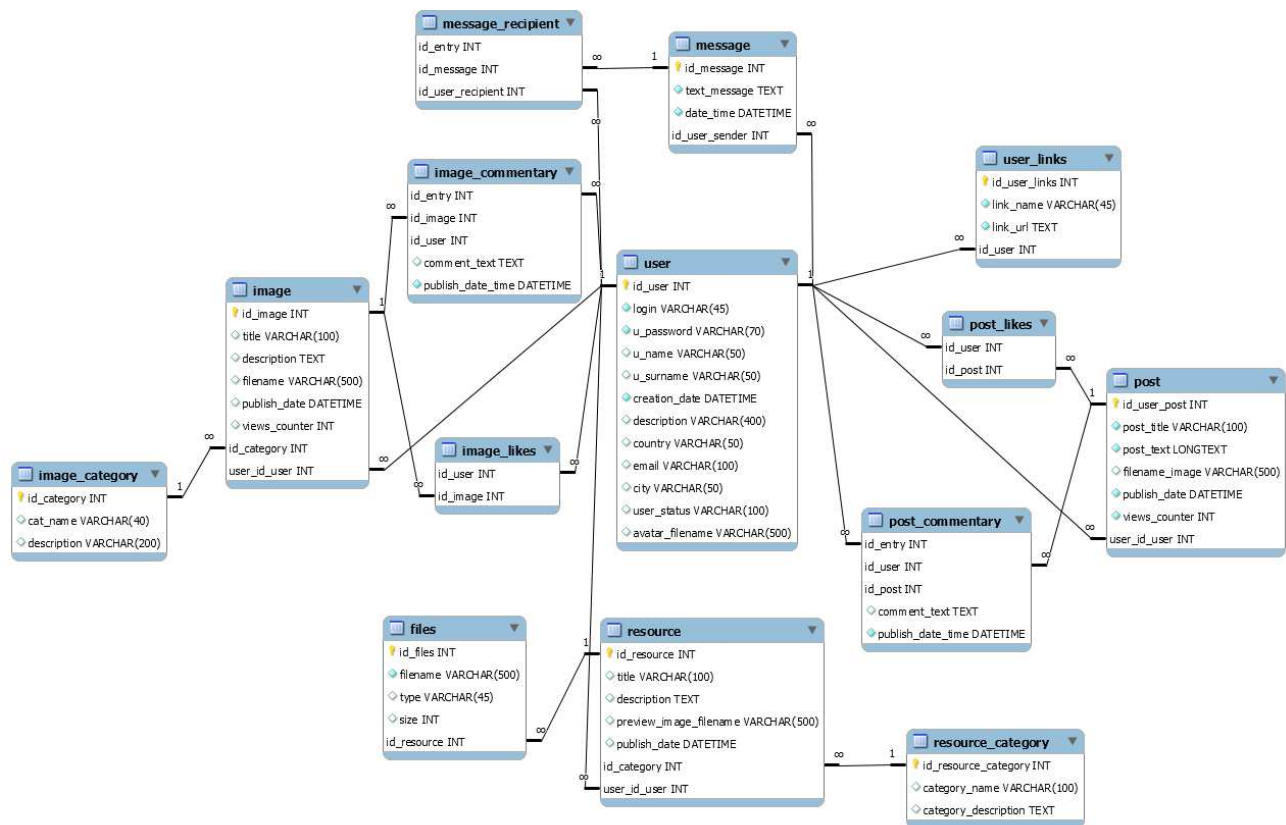


Рисунок 2.3 – Фізична модель бази даних

2.5 Розробка структури сайту web-ресурсу

Використовуючи інформацію попередніх розділів, була сформована структура сайту web-ресурсу, на якій відображені усі необхідні сторінки сайту.

Структура сайту представлена на рис. 2.4:



Рисунок 2.4 – Структура сайту web-ресурсу

2.6 Створення дизайну графічного інтерфейсу web-ресурсу

За допомогою середовища Adobe Photoshop був сформований наступний макет дизайну графічного інтерфейсу для сторінки користувача (рис 2.5). Надалі він буде використаний для верстки сайту за допомогою мови розмітки HTML та CSS.

В процесі створення дизайну були обрані темні кольори, так як вони найбільш вигідно підкреслюють контент, що надається користувачем, не навантажують очі і є досить сучасними.

На всіх сторінках буде відображатися хедер, тобто шапка сайту, яка містить наступні елементи:

- Посилання на головну галерею
- Поле пошуку
- Посилання на сторінку з діалогами користувача
- Посилання на сторінку користувача

Сторінка користувача містить коротку інформацію про користувача, його останній опублікований пост, меню навігації для перегляду публікацій, постів, ресурсів, і повної інформації про користувача відповідно, а також основний блок, який буде відображати контент виділеного пункту меню.

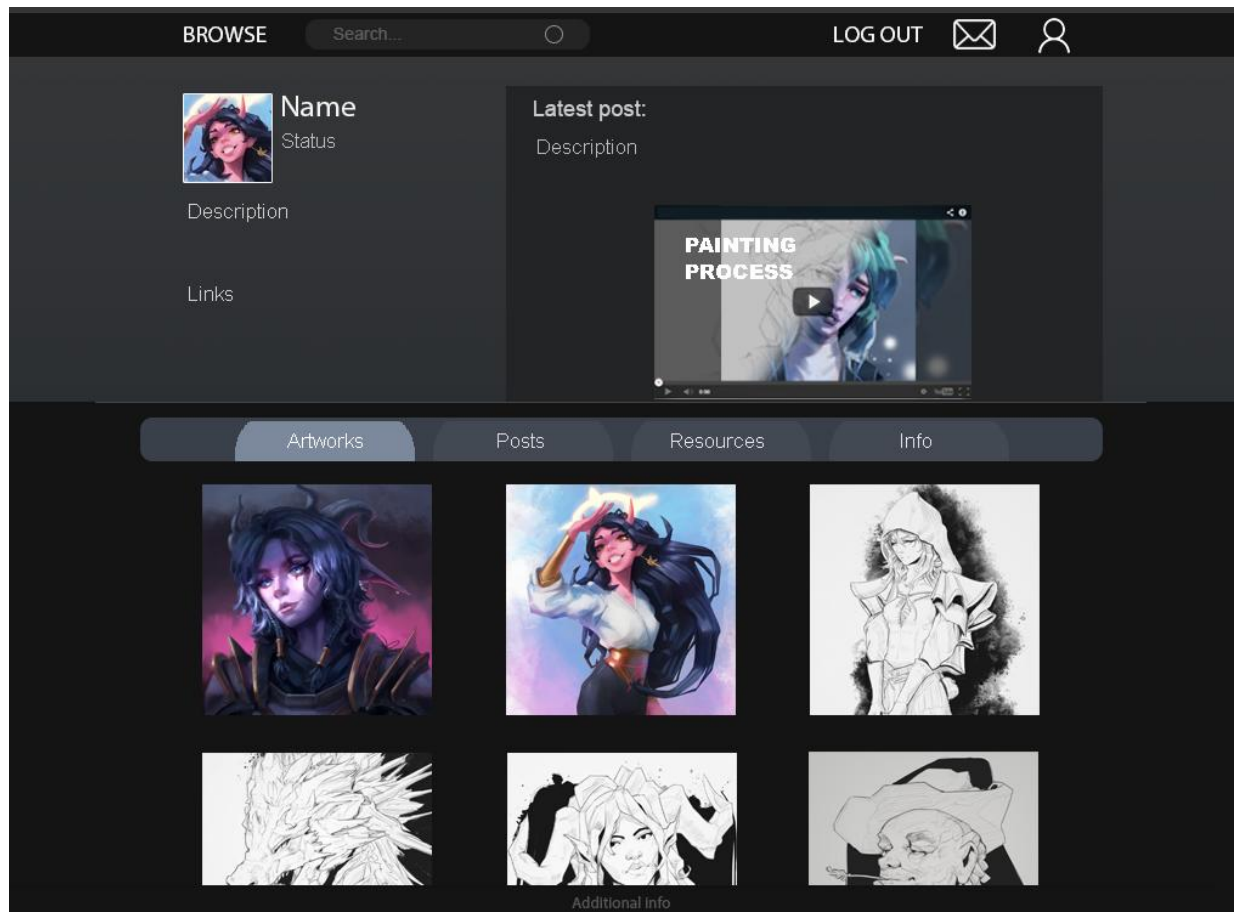


Рисунок 2.5 – Макет дизайну сторінки користувача

2.7 Розробка веб-додатку

Проект веб-додатку складається з клієнтської та серверної частини. Клієнтська частина містить наступні компоненти:

- **jsp-сторінки** - відповідають за інтерфейс сайту та відображуваний контент;
- **style.css** – відповідає за стилі сторінок, їх зовнішній вигляд.

Серверна частина складається з:

- **java-сервлети** - оброблюють інформацію, яка поступає з форм jsp-сторінок, та формують контент цих самих сторінок відповідно.
- **DAO-класи** – відповідають за взаємодію з базою даних; в них реалізовані CRUD-операції: створення, зчитування, видалення та редагування об'єктів бази даних.

- **Entity-класи** – java-класи, сутності, що являють собою відповідні таблиці бази даних.
- **Bean-класи** – сутності що слугують для об'єднання кількох об'єктів в один для зручної передачі даних.

Після створення бази даних за допомогою додатку Database у середовищі IntelliJ IDEA, були сгенеровані Entity-класи за допомогою Java Persistence API, що відповідають таблицям у базі даних. Для безпосередньої роботи з базою даних були створені DAO-класи, в яких за допомогою EntityManager були реалізовані CRUD-операції. У процесі розробки також були додані додаткові функції для коректної обробки запитів. Bean-класи слугують для зберігання та подальшої передачі колекцій відповідних Entity-класів.

На рис. 2.6 показаний фрагмент діаграми класів проекту через занадто велику їх кількість.

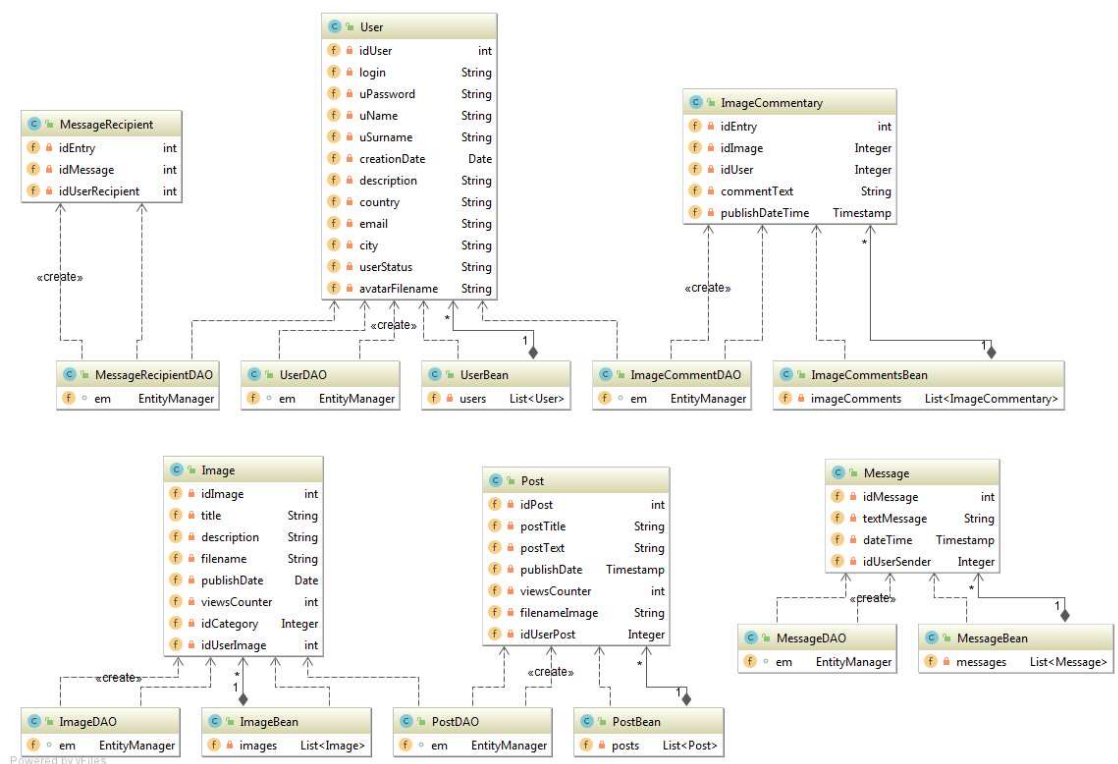


Рисунок 2.6 – Діаграма класів веб-додатку

Jsp-сторінки відповідають за зовнішній вигляд та інтерфейс сайту, використовуючи мову розмітки HTML та Java-код. У файлі style.css описані стилі,

які використовуються для генерації для задання кольорів, шрифтів, стилів, розташування окремих блоків і інших аспектів представлення зовнішнього вигляду jsp-сторінок. Стили були створені за допомогою реалізованого дизайну на рис. 2.5.

На рис. 2.7 відображена web-структура додатку та усі наявні jsp-сторінки. Зображення, що завантажуються на сервер, поділені на відповідні директорії: аватарки користувачів, зображення до постів та зображення до галереї.

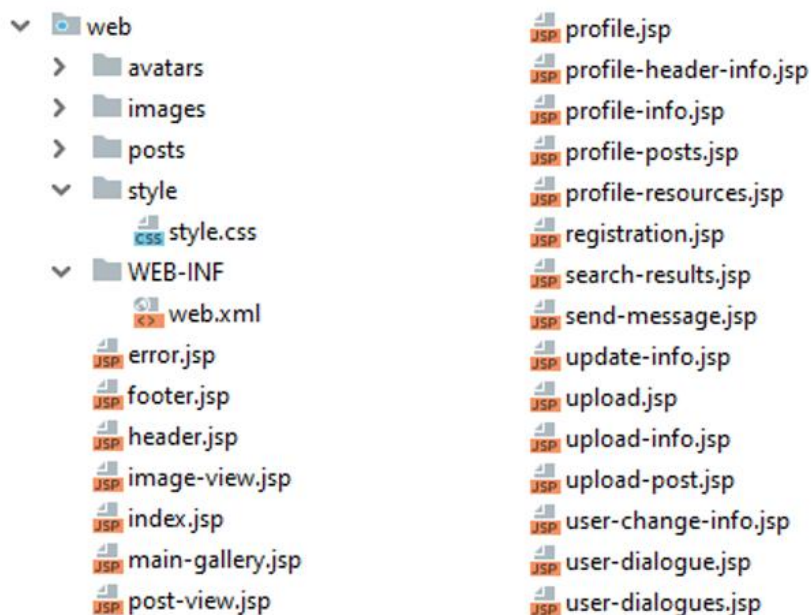


Рисунок 2.7 – Web-структура проекту

У таблиці 2.15 наведені усі створені jsp-сторінки, їх опис, взаємодія з іншими сторінками та сервлетами та наявні поля для вводу.

Таблиця 2.15

Jsp-сторінки

Назва	Опис	Посилання на сторінки/сервлети	Поля вводу даних
header.jsp	Містить хедер сайту, де знаходиться модуль пошуку, є посилання на загальну	MainGalleryServlet; SearchServlet; UserServlet;	search-text

Продовження таблиці 2.15

	галерею, сторінку входу, та якщо користувач вже виконав вхід, посилання на його сторінку, його діалоги та вихід з користувацької сесії.	UserDialoguesServlet; LogoutServlet; index.jsp	
footer.jsp	Містить футер сайту з додатковою інформацією про сайт.		
error.jsp	Містить повідомлення про помилку, яке генерується сервлетом.	index.jsp	
index.jsp	Сторінка входу у користувацький профіль. Містить посилання на сторінку реєстрації.	LogInServlet; registration.jsp	login, password
registration.jsp	Сторінка реєстрації. Містить поля для вводу необхідної інформації про користувача для подальшої його реєстрації на сайті.	RegistrationServlet	login, password, name, surname, country, city, email
main-gallery.jsp	Сторінка загальної галереї; відображає усі наявні на сайті зображення. Містить назви категорій, за якими можна фільтрувати зображення.	MainGalleryServlet; MainGallerySortServlet; ImageViewServlet	

Продовження таблиці 2.15

image-view.jsp	Сторінка перегляду зображення. Містить саме зображення, його опис, інформацію про автора, додаткову інформацію про зображення та секцію з коментарями.	UserServlet; MainGallerySortServlet; ImageCommentAddServlet; PreUpdateImageServlet; DeleteImageServlet; ImageLikeServlet; ImageDislikeServlet; ;	comment -text
post-view.jsp	Сторінка перегляду поста. Містить назву, текст посту, інформацію про автора та, при наявності, зображення, що додано до посту. Також містить секцію з коментарями.	UserServlet; PostCommentAddServlet; PostLikeServlet; PostDislikeServlet	comment -text
profile.jsp	Профіль користувача. Містить коротку інформацію про користувача, що вказана на сторінці «profile-header-info.jsp», та секцію з усіма зображеннями, що завантажив даний користувач.	UserPostsServlet; UserInfoServlet; UserResourcesServlet; ImageViewServlet; UploadTypeServlet;	
profile-header-info.jsp	Коротка інформація про користувача. Є частиною сторінок профіля. Містить аватарку, ім'я, статус, опис, посилання на зовнішні ресурси та відображує останній пост.	UploadTypeServlet; send-message.jsp PostViewServlet;	

Продовження таблиці 2.15

profile-info.jsp	Містить детальну інформацію користувача.	UserServlet; UserPostsServlet; UserResourcesServlet; user-change-info.jsp	
profile-posts.jsp	Містить секцію з усіма опублікованими постами користувача.	UserServlet; UserInfoServlet; UserResourcesServlet; PostViewServlet; upload-post.jsp	
profile-resources.jsp	Містить секцію з усіма завантаженими ресурсами користувача.	UserServlet; UserPostsServlet; UserInfoServlet	
update-info.jsp	Сторінка редагування інформації про зображення.	UpdateImageServlet	title, description, category-select
upload.jsp	Сторінка завантаження файлу зображення на сервер.	UploadServlet	file-select
upload-info.jsp	Сторінка де користувач вводить додаткову інформацію про зображення, що завантажується на сервер.	ImageInfoUploadServlet	title, description, category-select
upload-post.jsp	Сторінка публікації нового посту.	PostUploadServlet; UploadTypeServlet	title, text

Продовження таблиці 2.15

user-change-info.jsp	Сторінка редагування інформації про користувача.	UserChangeInfoServlet	status, description, name, surname, country, city, email
send-message.jsp	Сторінка відправлення приватного повідомлення до вказаного користувача.	SendMessageServlet	text
user-dialogue.jsp	Сторінка діалогу з певним користувачем.	UserServlet; SendMessageServlet	text
user-dialogues.jsp	Сторінка з усіма користувацькими діалогами.	DialogueServlet; UserServlet	
search-results.jsp	Сторінка, що відображує результати пошуку.	PostViewServlet; ImageViewServlet; UserServlet	

Запити, що посилаються з даних jsp-сторінок повинні оброблятися сервлетами. На рис. 2.8 представлені усі сервлети даного проекту.

Продовження таблиці 2.16

LogOutServlet	Виконує вихід з профілю користувача.	index.jsp
RegistrationServlet	Здійснює реєстрацію користувача після вводу необхідних даних та натиску кнопки реєстрації. Якщо дані введені неправильно, то виводиться повідомлення про помилку.	index.jsp error.jsp
MainGallery	Виводить усі зображення, що знаходяться у базі даних.	main-gallery.jsp
MainGallerySort	Фільтрує зображення у головній галереї за заданою категорією.	main-gallery.jsp
ImageViewServlet	Відображує вибране користувачем зображення, його назву, опис, автора, додаткову інформацію та усі присутні коментарі до зображення.	image-view.jsp
ImageLikeServlet	Залишає відмітку «Подобається» до даного зображення.	ImageViewServlet
ImageDislikeServlet	Видаляє відмітку «Подобається» з даного зображення.	ImageViewServlet
ImageCommentAddServlet	Додає коментар до зображення після його вводу та натиску кнопки додавання.	ImageViewServlet
UploadTypeServlet	Визначає тип зображення, що завантажується на сервер.	upload.jsp

Продовження таблиці 2.16

UploadServlet	Завантажує зображення на сервер у відповідну директорію. Якщо файл вибраний неправильно або не вибраний взагалі, то виводиться повідомлення про помилку.	upload-info.jsp upload-post.jsp UserServlet error.jsp
ImageInfoUploadServlet	Додає інформацію до завантаженого зображення після її введення, якщо це зображення завантажується у галерею.	UserServlet
DeleteImageServlet	Видаляє вибране зображення з галереї користувача.	UserServlet
PreUpdateImageServlet	Додатковий сервлет що шукає та відображає необхідні дані для подальшого редагування інформації про зображення.	update-info.jsp
UpdateImageServlet	Оброблює та додає до бази нову введену інформацію про зображення.	UserServlet
UserServlet	Виводить профіль користувача, коротку інформацію про нього, його аватар, останній опублікований пост та усі додані їм зображення в галереї користувача.	profile.jsp
UserPostsServlet	Виводить усі пости, що опублікував користувач.	profile-posts.jsp

Продовження таблиці 2.16

UserInfoServlet	Виводить повну інформацію про користувача.	profile-info.jsp
UserResourcesServlet	Виводить усі ресурси, що додав користувач.	profile-resources.jsp
UserChangeInfoServlet	Виконує редагування інформації про користувача.	UserServlet
UserDialoguesServlet	Виводить усі наявні діалоги, які має користувач з іншими користувачами.	user-dialogues.jsp
SendMessageServlet	Посилає приватне повідомлення до певного користувача після введення тексту на натиску кнопки відправлення.	DialogueServlet
DialogueServlet	Відображує усі повідомлення у діалозі між заданими користувачами.	user-dialogue.jsp
PostViewServlet	Відображує вибраний користувачем пост, його назву, опис, додане зображення до посту (при наявності) та усі присутні коментарі до даного посту.	post-view.jsp
PostUploadServlet	Виконує публікацію посту після введення усіх необхідних даних та натиску кнопки публікації.	UserServlet
PostLikeServlet	Залишає відмітку «Подобається» до даного поста.	PostViewServlet

Продовження таблиці 2.16

PostDislikeServlet	Видаляє відмітку «Подобається» з даного поста.	PostViewServlet
PostCommentAddServlet	Додає коментар до поста після його вводу та натиску кнопки додавання.	PostViewServlet
SearchServlet	Виконує пошук по сайту за користувачами, зображеннями та постами за введеними ключовими словами та відображує результати.	search-results.jsp

Текст програми представлений у Додатку В.

2.8 Тестування функціональності додатку

Для перевірки коректного виконання додатком функцій, були складені тестові набори, показані в таблицях 2.17 - 2.19.

Таблиця 2.17

Тестові набори для перевірки функціональності авторизації

№ Тесту	Подія	Очікуємий результат	Відмітка про
1	Введення некоректних логіна і пароля Login: prophet Password: 123	Авторизація та вхід у профіль користувача	перевірено
2	Введення некоректного логіна і коректного пароля Login: kur Password: 123	Повідомлення про помилку при введенні логіну/паролю	перевірено

Продовження таблиці 2.16

3	Введення коректного логіна і некоректного пароля Login: prophet Password: 456 (Введення некоректної інформації - невірний пароль)	Повідомлення про помилку при введені логіну/паролю	перевірено
---	--	--	------------

Таблиця 2.18

Тестові набори для перевірки функціональності реєстрації

№ Тесту	Подія	Очікуємий результат	Відмітка про виконання
1	Введення унікального логіну Login: prophetX Password: 123 (Введення коректної інформації)	Успішна реєстрація	перевірено
2	Введення вже існуючого логіну Login: prophet Password: 123 (Введення некоректної інформації)	Повідомлення про помилку при реєстрації	перевірено
3	Відсутність введення логіну та паролю Login: Password:	Повідомлення про помилку при реєстрації	перевірено

Таблиця 2.19

Тестові набори для перевірки функціональності завантаження зображення на сервер

№ Тесту	Подія	Очікуємий результат	Відмітка про виконання
1	Вибір коректного файлу	Подальше завантаження зображення на сервер	перевірено
2	Вибір некоректного файлу або відсутність вибору файлу	Повідомлення про помилку завантаження файлу.	перевірено

Представлені в таблицях тести дозволили перевірити програму і переконатися в коректному виконанні функціоналу.

2.9 Розробка діаграми послідовності

Діаграма послідовності — відображає взаємодії об'єктів впорядкованих за часом. Зокрема, такі діаграми відображають задіяні об'єкти та послідовність відправлених повідомлень. [3]

Іншими словами, діаграма послідовностей відображає часові особливості передачі і прийому повідомлень об'єктами.

Діаграми послідовностей можна використовувати для уточнення діаграм прецедентів, більш детального опису логіки сценаріїв використання. Це відмінний засіб документування проекту з точки зору сценаріїв використання. Діаграми послідовностей зазвичай містять об'єкти, які взаємодіють у рамках сценарію, повідомлення, якими вони обмінюються, і які повертаються результати, які пов'язані з повідомленнями.

Нижче наведені дві діаграми послідовності для процесів користувача.

Діаграма послідовності для варіанту використання «Завантаження зображення» представлена на рис. 2.9.

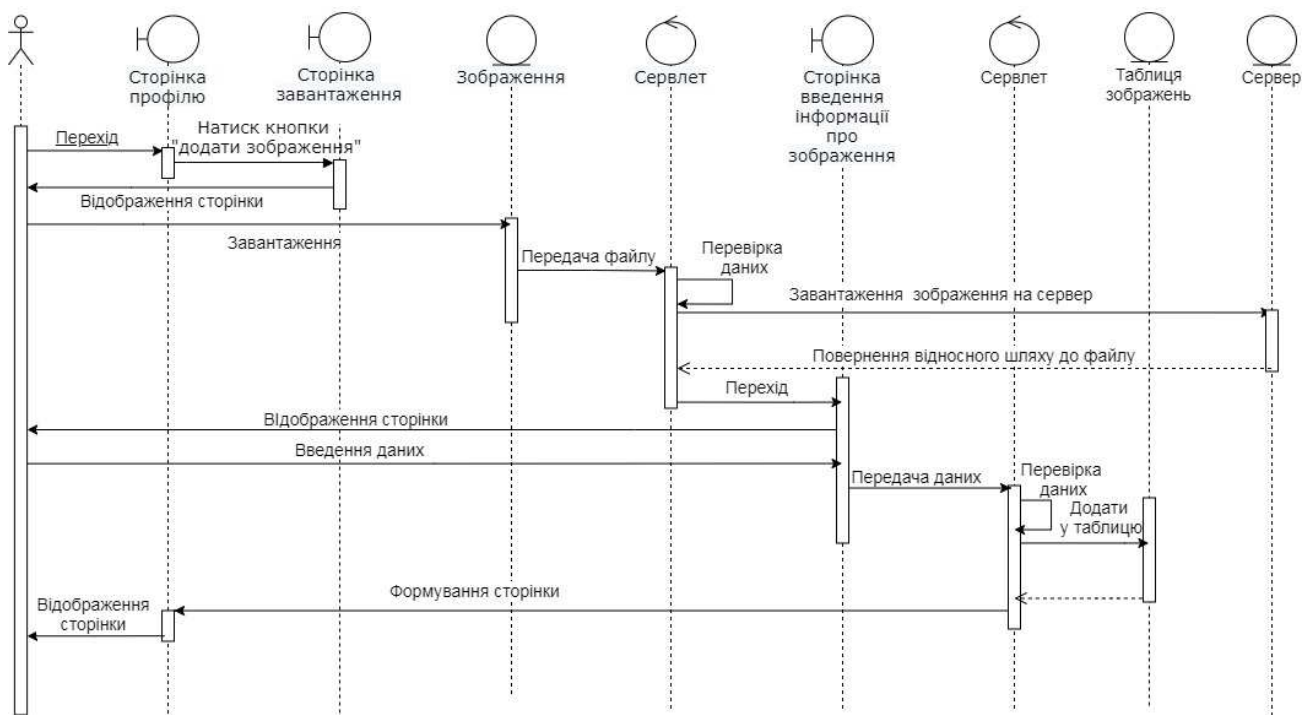


Рисунок 2.9 - Діаграма послідовності для варіанту використання «Завантаження зображення»

Діаграма послідовності для варіанту використання «Публікація коментаря до зображення» представлена на рис. 2.10.

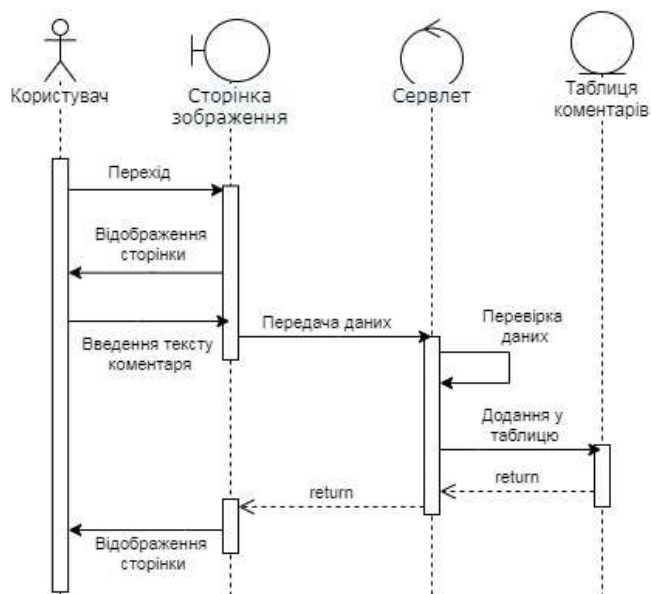


Рисунок 2.10 - Діаграма послідовності для варіанту використання «Публікація коментаря до зображення»

2.10 Розробка діаграми компонентів та діаграми розгортання

Діаграма компонентів — в UML, діаграма, на якій відображені компоненти, залежності та зв'язки між ними.

Діаграма компонентів відображає залежності між компонентами програмного забезпечення, включаючи компоненти вихідних кодів, бінарні компоненти, та компоненти, що можуть виконуватись. Модуль програмного забезпечення може бути представлено як компоненту. Деякі компоненти існують під час компіляції, деякі — під час компонування, а деякі під час роботи програми. [4]

Діаграма компонентів веб-додатку представлена на рис. 2.11:

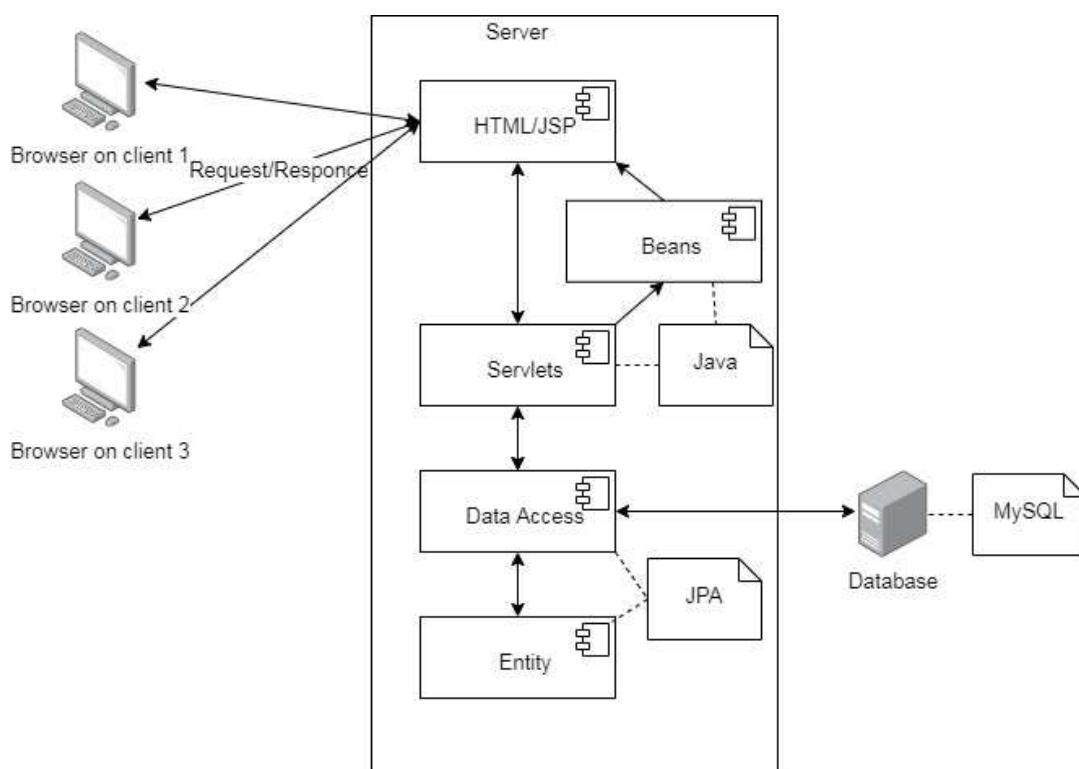


Рисунок 2.11 – Діаграма компонентів веб-додатку

Діаграма розгортання (англ. deployment diagram) — діаграма в UML, на якій відображаються обчислювальні вузли під час роботи програми, компоненти, та об'єкти, що виконуються на цих вузлах. [5]

Діаграма розгортання веб-додатку представлена на рис. 2.12:

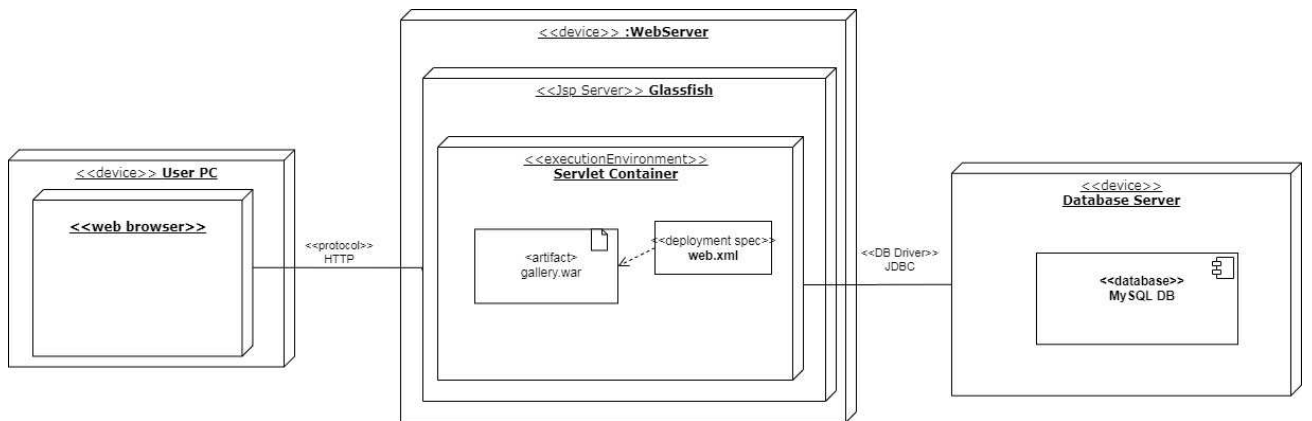


Рисунок 2.12 – Діаграма розгортання веб-додатку

РОЗДІЛ 3 РЕЗУЛЬТАТИ РОЗРОБКИ WEB-РЕСУРСУ

В процесі виконання дипломної роботи була проведена програмна реалізація клієнтської та серверної частини додатку web-ресурсу. Розроблене програмне забезпечення задовольняє вимогам, які поставлені у технічному завданні, що наведене у додатку А.

В процесі роботи були реалізовані наступні функції:

- Реєстрація та вхід за допомогою логіна/пароля користувача.
- Завантаження, перегляд та видалення зображення.
- Редагування інформації про зображення.
- Перегляд сторінки користувача.
- Редагування інформації на сторінці користувача.
- Перегляд сторінок та зображень інших користувачів.
- Публікація та перегляд постів користувачів.
- Можливість залишати коментарі та відмітки «Подобається» до постів та зображень.
- Пошук користувачів, зображень, та постів.
- Обмін особистими повідомленнями між користувачами.

Нереалізовані функції знаходяться в стані розробки.

Під час розробки були використані веб-сервер Glassfish, мова програмування Java, мова розмітки HTML та CSS. Програмний продукт використовує базу даних під управлінням СУБД MySQL. При розробці бази даних було створено 14 таблиць.

Тестування веб-додатку показало, що програмне забезпечення працює стабільно.

Робота з веб-додатком здійснюється з використанням зручного та інтуїтивно зрозумілого інтерфейсу. Розроблена система орієнтована на користувача що вже має навички роботи із графічним інтерфейсом операційної системи та web-додатків.

Розроблений дизайн містить роботи автора даного дипломного проекту для презентації роботи web-ресурсу. На рис. 3.1 та 3.2 зображені сторінка загальної галереї та сторінка користувача відповідно.

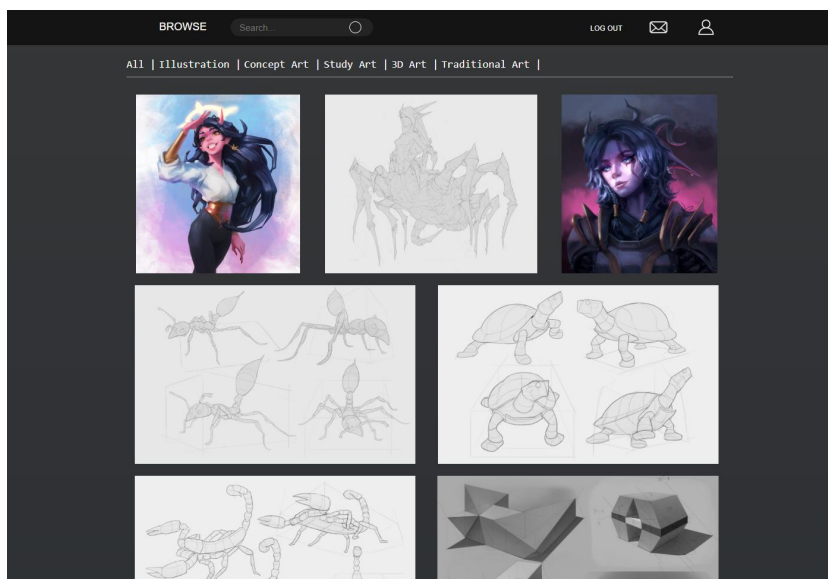


Рисунок 3.1 – Головна сторінка

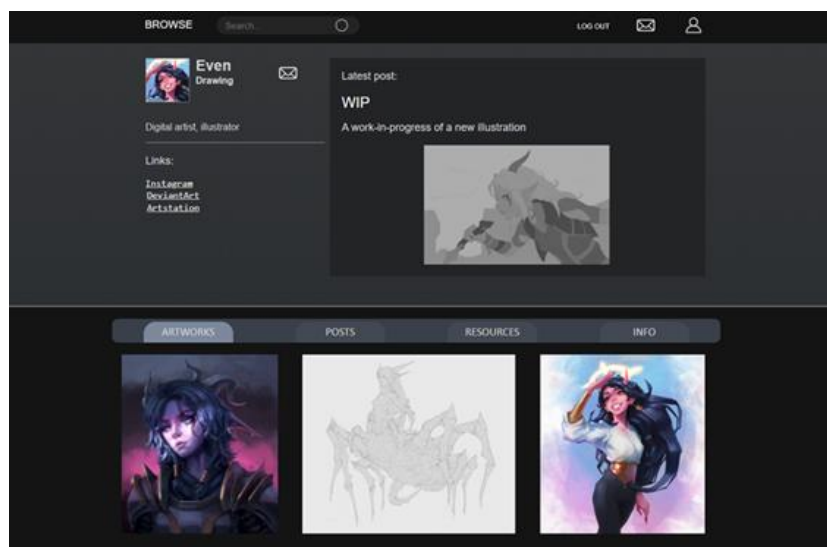


Рисунок 3.2 – Сторінка профілю користувача

Усі наявні сторінки web-ресурсу відображені у додатку Б.

Даний дизайн реалізований за допомогою одного CSS-файлу, що містить каскадні таблиці стилів для формування веб-сторінок і 22-х HTML-файлів з розширенням JSP, що містять JSP-скрипти, JSTL, Java-код та HTML код.

Програмне забезпечення сервера вимагає встановлення наступних програмних продуктів:

- Веб-сервер GlassFish;
- MySQL-сервер;
- MySQL-клієнт;
- JRE/JDK.

Програмно-апаратне забезпечення для клієнта:

- Операційна система, що забезпечує мережеву взаємодію по протоколу TCP/IP;
- Доступ до мережі інтернет;
- Будь-який веб-браузер.

Інструкція користувача для роботи з веб-додатком наведена в Додатку Б.

РОЗДІЛ 4 ОХОРОНА ПРАЦІ

4.1 Вступ

Охорона праці - це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров'я і працездатності людини у процесі трудової діяльності. [18]

Цей Закон визначає основні положення щодо реалізації конституційного права працівників на охорону їх життя і здоров'я у процесі трудової діяльності, на належні, безпечні і здорові умови праці, регулює за участю відповідних органів державної влади відносини між роботодавцем і працівником з питань безпеки, гігієни праці та виробничого середовища і встановлює єдиний порядок організації охорони праці в Україні.

Далі наведено параметри приміщення, яке відповідає чинним правилам безпеки, а також рекомендовану систему вправ для зменшення втомлюваності під час роботи.

4.2 Характеристика приміщення

Набір програмних продуктів, який використовується у дипломній роботі, використовуватиметься в приміщенні, план якого приведено на рис. 4.1.

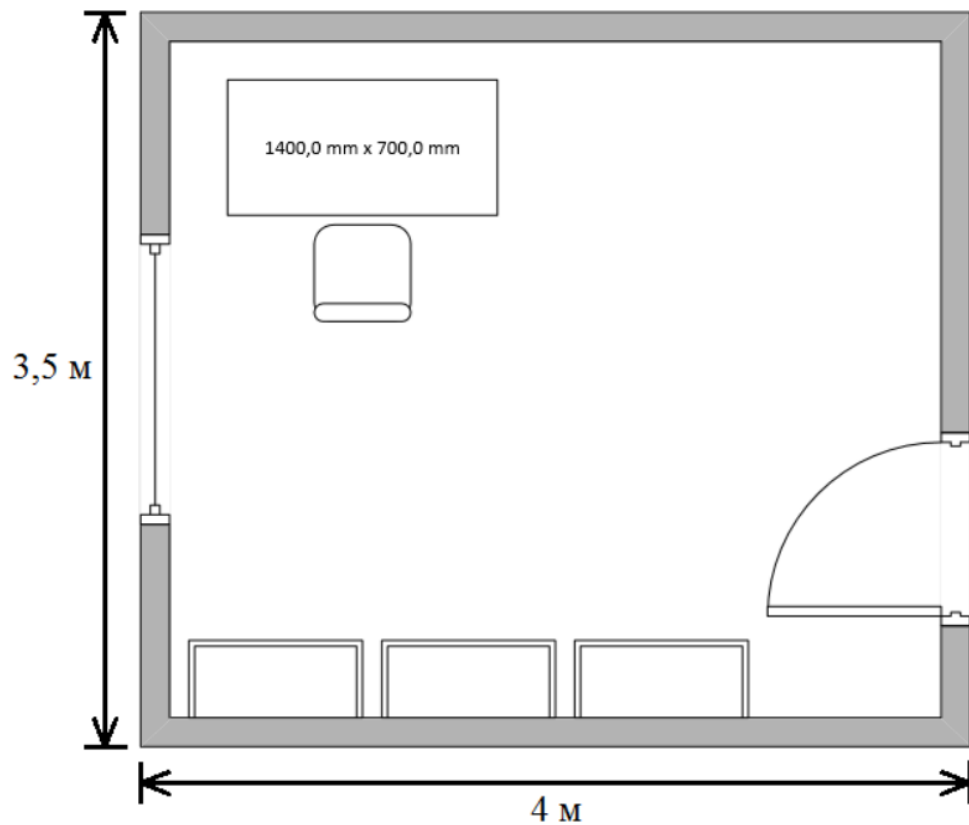


Рисунок 4.1 – План приміщення

Приміщення має одностороннє природне освітлення і загальне штучне освітлення. Стіни і стеля обклеєні світлими шпалерами, підлога вкрита темним ламінатом. У приміщенні відсутні сильні вібрації та шкідливі речовини. Склад повітря в нормі.

У кімнаті знаходиться ПК з 4-ядерним процесором і 23-дюймовим IPS монітором, а також меблі.

Приміщення має довжину 4 м, ширину 3.5 м, висоту стелі 2,7 м. Кількість робочих місць – одне. Приміщення знаходиться на шістнадцятому поверсі 20-поверхової цегляної будівлі. Площа – 14 м², об'єм – 37,8 м³. Виходячи з цього, отримано дані, наведені в табл. 4.1. Нормативні значення згідно з [19].

Таблиця 4.1

Фактичні та нормативні значення параметрів приміщення

Параметр	Норма	Реальні параметри
Площа, S	не менше 6 м ²	14 м ²
Об'єм, V	не менше 15 м ³	37,8 м ³

Отримані показники відповідають чинним нормам і вимогам.

4.3 Мікрокліматичні умови

Згідно з [21] цю роботу можна віднести до категорії легка 1а. Джерелами тепла в цьому приміщенні є люди, електроустаткування, освітлювальні прилади в темний час доби і система опалювання взимку. Оператором виділяється до 120 ккал теплової енергії за годину. Оптимальні та фактичні значення параметрів мікроклімату приведені в табл 4.2.

Таблиця 4.2

Значення мікроклімату

Період року	Параметр	Оптимальний	Фактичний
Теплий	Температура	23 – 25 °C	24 °C
	Вологість	40 – 60 %	50 %
	Швидкість повітря	≤ 0.1 м/с	
Холодний	Температура	22 – 24 °C	23 °C
	Вологість	40 – 60 %	55 %
	Швидкість повітря	≤ 0.1 м/с	

Усі показники задовольняють вимогам зазначеним в [10] для робіт категорії легка 1а і є задовільними для здоров'я людини.

4.4 Освітлення

Згідно з [20] ця робота відноситься до V_a розряду зорових робіт.

Передбачається використання природного, штучного і змішаного освітлення.

Природне освітлення здійснюється за допомогою вікна, площа якого складає $S' = 1,8 \cdot 1,5 = 2,7 \text{ м}^2$ і є боковим освітленням.

У світильниках місцевого і загального освітлення використовуються лампи розжарювання потужністю 75 Вт із світловим потоком лампи 940 лм.

Освітлення задовольняє нормам.

4.5 Шум та вібрація

Джерелом шуму в приміщенні є комп'ютер.

Вентилятори (кулери) системного блоку, процесора, відеокарти і блоку живлення є сучасними і мають низький рівень шуму. Згідно з технічною документацією шум, зумовлений кулером в блоці живлення складає 25 дБ, кулером процесора - 30 дБ, загальний, - 34 дБ. Враховуючи незначний рівень шуму від персонального комп'ютера і незначний рівень фонового шуму від іншого устаткування, можна стверджувати, що сумарний рівень шумового забруднення приміщення не перевищує максимально допустимий рівень коригованої звукової потужності і складає не більше 50 дБ.

При роботі з персональним комп'ютером в робочому приміщенні значення характеристик вібрації на робочих місцях не повинна перевищувати допустимих значень.

4.6 Випромінювання

У приміщенні відсутні джерела інфрачервоного, ультрафіолетового і електромагнітного випромінювання, бо монітор ПК вироблений на основі рідкокристалічної матриці, підсвітка якої здійснюється неоновною лампою, що не має сильного електромагнітного випромінювання і сертифіковані в Україні.

Блок живлення є екранованим і не випускає вищезазначених видів випромінювання.

4.7 Ергономіка робочого місця

Обладнання і організація робочого місця працівників з ПЕОМ мають відповідати конструкцій всіх елементів робочого місця.

Висота робочої поверхні складає 700 мм, має простір для ніг 800 мм, довжина столу 1400 мм, ширина – 700 мм. Робочий стілець – обертальне крісло з регулятором висоти.

Робоче місце відповідає вимогам.

Для захисту очей забезпечення комфортного куту огляду 23-дюймового монітора відстань від екрану монітора до користувача складає 600-700 мм, робота за комп'ютером виконується не більше 4 годин в день і проводяться періодичні перерви (через кожних 2 години на 10-15 хвилин). Також періодично виконується комплекс вправ для очей.

Рекомендований комплекс вправ для очей:

- Міцно примружитися на 3-5 секунд. Відкрити очі і не блимати 3-5 секунд. Повторити 7-8 разів.
- Закрити очі і масажувати їх круговими рухами пальців, не натискаючи на очні яблука, протягом 1-2 хвилин.
- Поставити вказівні пальці під кожною бровою і злегка потягнути ними шкіру вгору, щоб піднялися очі. Потримати протягом 1-2 секунд, потім відпустити. Повторити 5 разів.

4.8 Висновки

Аналіз умов праці в розглянутому робочому приміщенні показав, що умови праці з ПЕОМ відповідають вимогам [18, 20, 21, 22] оскільки площа та об'єм не є меншими за нормативні значення, а рівні шуму, вібрації і загазованості не перевищують нормативних обмежень.

Для підтримання параметрів мікроклімату в приміщенні встановлено радіатор центральної водяної системи опалення, що складається з 11 секцій.

Ергономіка робочого місця і режим зорової роботи задовольняють вимогам і сприяють зниженню втоми.

ВИСНОВКИ

Результатом виконання дипломної роботи є створений додаток web-ресурсу.

Був проведений аналіз предметної області, аналіз існуючих рішень та розробка концепції web-ресурсу, спроектована база даних, структура, дизайн та інтерфейс клієнтської частини веб-додатку, і на основі цього була розроблена і протестована серверна частина веб-додатку та основні функції web-ресурсу.

Додаток розроблений з використанням мови програмування Java, мови розмітки HTML та CSS, та бази даних під управлінням СУБД MySQL. Проект був розроблений у середовищі IntelliJ IDEA.

Подальшим розвитком проекту планується розширення функціоналу, зокрема реалізація модулю завантаження ресурсів, їх поширення і продажу, впровадження інтерфейсу адміністратора web-ресурсу, розміщення додатку на хостингу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Веб-застосунок [Електронний ресурс] // Вікіпедія: онлайн енциклопедія. - Режим доступу: https://en.wikipedia.org/wiki/Web_application, вільний.
2. Веб-сайт [Електронний ресурс] // Вікіпедія. – Режим доступу: <https://uk.wikipedia.org/wiki/Веб-сайт>, вільний.
3. Діаграма послідовності [Електронний ресурс] // Andrew's code. – Режим доступу: <http://flash.retejo.info/cxefpagxo/uml/diagrama-poslidovnosti>, вільний.
4. Діаграма компонентів [Електронний ресурс] // Вікіпедія: онлайн енциклопедія. - Режим доступу: https://en.wikipedia.org/wiki/Component_diagram, вільний.
5. Діаграма розгортання [Електронний ресурс] // Вікіпедія: онлайн енциклопедія. - Режим доступу: https://en.wikipedia.org/wiki/Deployment_diagram, вільний.
6. Особливості web-додатків [Електронний ресурс] – Режим доступу: <http://sites.znu.edu.ua/webprog/lect/1191.ukr.html>, вільний.
7. Цифрове мистецтво [Електронний ресурс] // Вікіпедія: онлайн енциклопедія. - Режим доступу: https://en.wikipedia.org/wiki/Digital_art, вільний.
8. Adobe Photoshop [Електронний ресурс] // Вікіпедія: онлайн енциклопедія. - Режим доступу: https://en.wikipedia.org/wiki/Adobe_Photoshop, вільний.
9. CSS [Електронний ресурс] // Вікіпедія: онлайн енциклопедія. - Режим доступу: <https://ua.wikipedia.org/wiki/CSS>, вільний.
10. HTML [Електронний ресурс] // Вікіпедія: онлайн енциклопедія. - Режим доступу: <https://ua.wikipedia.org/wiki/HTML>, вільний.
11. IntelliJ IDEA [Електронний ресурс] // Вікіпедія: онлайн енциклопедія. - Режим доступу: https://en.wikipedia.org/wiki/IntelliJ_IDEA, вільний.
12. Java [Електронний ресурс] // Вікіпедія: онлайн енциклопедія. - Режим доступу: <https://en.wikipedia.org/wiki/Java>, вільний.

13. Jakarta EE [Електронний ресурс] // Вікіпедія: онлайн енциклопедія. - Режим доступу: https://ua.wikipedia.org/wiki/Jakarta_EE, вільний.
14. Hibernate [Електронний ресурс] // Вікіпедія: онлайн енциклопедія. - Режим доступу: <https://uk.wikipedia.org/wiki/Hibernate>, вільний.
15. GlassFish [Електронний ресурс] // Вікіпедія: онлайн енциклопедія. - Режим доступу: <https://uk.wikipedia.org/wiki/GlassFish>, вільний.
16. MySQL [Електронний ресурс] // Вікіпедія: онлайн енциклопедія. - Режим доступу: <https://ru.wikipedia.org/wiki/MySQL>, вільний.
17. UML [Електронний ресурс] // Вікіпедія: онлайн енциклопедія. - Режим доступу: https://uk.wikipedia.org/wiki/Unified_Modeling_Language, вільний.
18. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин: ДСанПіН 3.3.2.007-98. Втратили чинність згідно з розпорядженням КМУ від 08.04.2025 № 317-р. – К.: Постанова Головного державного санітарного лікаря України, 1998. – № 7.
19. Про охорону праці: Закон України від 14.10.1999 р. № 2694-ХІІ [Електронний ресурс]. – Режим доступу: https://urist.com.ua/act/pro_okhoronu_pratsi
20. Визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою: ДСТУ Б В.1.1-36:2016. – Чинний з 01.01.2017. – К.: Мінрегіонбуд України, 2016.
21. Світлотехніка. Освітлення робочих місць в приміщеннях: ДСТУ EN 12464-1:2016. – К.: Державне підприємство «УкрНДНЦ», 2016.
22. Санітарні норми мікроклімату виробничих приміщень: ДСН 3.3.6.042-99. – К., 2000. – 16 с.
23. Ергономіка теплового середовища: ДСТУ ISO 7730:2005. – К.: Держспоживстандарт України, 2005.

ДОДАТКИ

Додаток А. Технічне завдання

Вступ

Назва програмного забезпечення: «Web-галерея», далі – програмний продукт

Основна область застосування цієї програми – мережа інтернет.

1 Підстави для розробки

Документ: завдання на дипломну роботу «Розроблення web-ресурсу для демонстрації художніх робіт».

2 Призначення розробки

Функціональне та експлуатаційне призначення програми: презентація робіт художниками та надання їм можливості розповсюджувати будь-який контент, які вони можуть надати, а також використання сайту як інструменту для зручної комунікації з потенційними роботодавцями та клієнтами.

3 Вимоги до програмного продукту

3.1 Вимоги до функціональних характеристик

Програма повинна надавати користувачу можливість виконання перерахованих нижче функцій:

- Реєстрація та вхід за допомогою логіна/пароля користувача.
- Завантаження, перегляд та видалення зображення.
- Редагування інформації про зображення.
- Перегляд сторінки користувача.
- Редагування інформації на сторінці користувача.
- Видалення сторінки користувача.
- Перегляд сторінок та зображень інших користувачів.
- Публікація, перегляд, редагування та видалення постів користувачів.

- Завантаження, перегляд, редагування та видалення ресурсів користувачами.
- Можливість залишати коментарі та відмітки «Подобається» до постів та зображень.
- Пошук користувачів, зображень, та постів.
- Обмін особистими повідомленнями між користувачами.

3.2 Вимоги до надійності

Надійне функціонування програми повинне бути забезпечене виконанням сукупності організаційно-технічних заходів, перелік яких наведений нижче:

- 1) організацією безперебійного живлення технічних засобів;
- 2) використанням ліцензійного програмного забезпечення;
- 3) регулярним виконанням вимог ДСТУ 51188-98. Захист інформації.

Випробування програмних засобів на наявність комп'ютерних вірусів.

Відмови програми можливі внаслідок некоректних дій оператора (користувача) при взаємодії з ПЗ. Програма повинна виконувати перевірку введених даних перед посиланням до серверу. У випадку, коли введені дані не відповідають вимогам, програма повинна вивести повідомлення про помилку, та запропонувати відредагувати вхідні дані.

3.3 Вимоги до експлуатації

Кінцевий користувач програми повинен мати практичні навички роботи із графічним інтерфейсом користувача операційної системи та web-додатків.

3.4 Вимоги до складу та параметрів технічних засобів

Для використання програмного забезпечення, необхідні наступні технічні засоби:

- Операційна система, що забезпечує мережеву взаємодію по протоколу TCP/IP;
- Доступ до мережі інтернет;
- Веб-браузер.

3.5 Вимоги до інформаційної та програмної сумісності

Для користування програмою користувачу необхідно лише мати браузер для перегляду веб-сторінок.

3.6 Вимоги до маркування та пакування

Програма розташована на сервері тому необхідності у фізичній копії немає.

3.7 Вимоги до транспортування та зберігання

Див. п. 3.6.

4 Вимоги до програмної документації

Склад програмної документації повинен містити:

- 1) технічне завдання;
- 2) текст програми;
- 3) інструкцію користувача.

5 Стадії та етапи розробки

Стадії та етапи розробки наведено в таблиці А.1.

Таблиця А.1

Основні етапи робіт за стадіями, терміни їх розробки та види документів.

Стадія створення програмного проекту	Етапи робіт на стадії	Вид документу	Початок виконання	Кінець виконання
Перед проектне дослідження	Аналіз існуючих програмних продуктів даного спрямування; розробка пропонування про доцільність створення програмного продукту	Технічний звіт		
Технічні задачі	Розробка ТЗ, яке включає основні вимоги до програмного продукту	Технічне завдання		
Технічний проект	Вибір раціонального варіанту програмного продукту, задовольняючого вимогам технічного завдання	Технічний проект		
Ескізний проект	Прийняття принципових рішень по структурі програмного продукту, усім видам забезпечення, попередній розрахунок економічної ефективності та	Ескізний проект		

	технічних показників програмного продукту			
Робочій проект	Формування програмного продукту в цілому	Робочій проект		
Виготовлення, налагодження та іспит	Виготовлення та налагодження компонентів програмного продукту.	Акт передачі програмного продукту		

6 Порядок прийому та контролю

Підготовлена дипломна робота представляється студентом науковому керівнику.

Рішенням кафедри або навчального підрозділу може бути введений обов'язковий публічний захист дипломної роботи.

Порядок і критерії оцінювання курсової роботи розробляються професорсько-викладацьким складом кафедри, обговорюються на засіданні кафедри і затверджуються розпорядженням керівника підрозділу.

Додаток Б. Інструкція користувача

Б.1 Головна сторінка

Головна сторінка web-ресурсу зображена на рис. Б.1. На головній сторінці відображаються всі роботи, завантажені користувачами в порядку «від нових до старих». Навігація по сайту здійснюється за допомогою хедера.

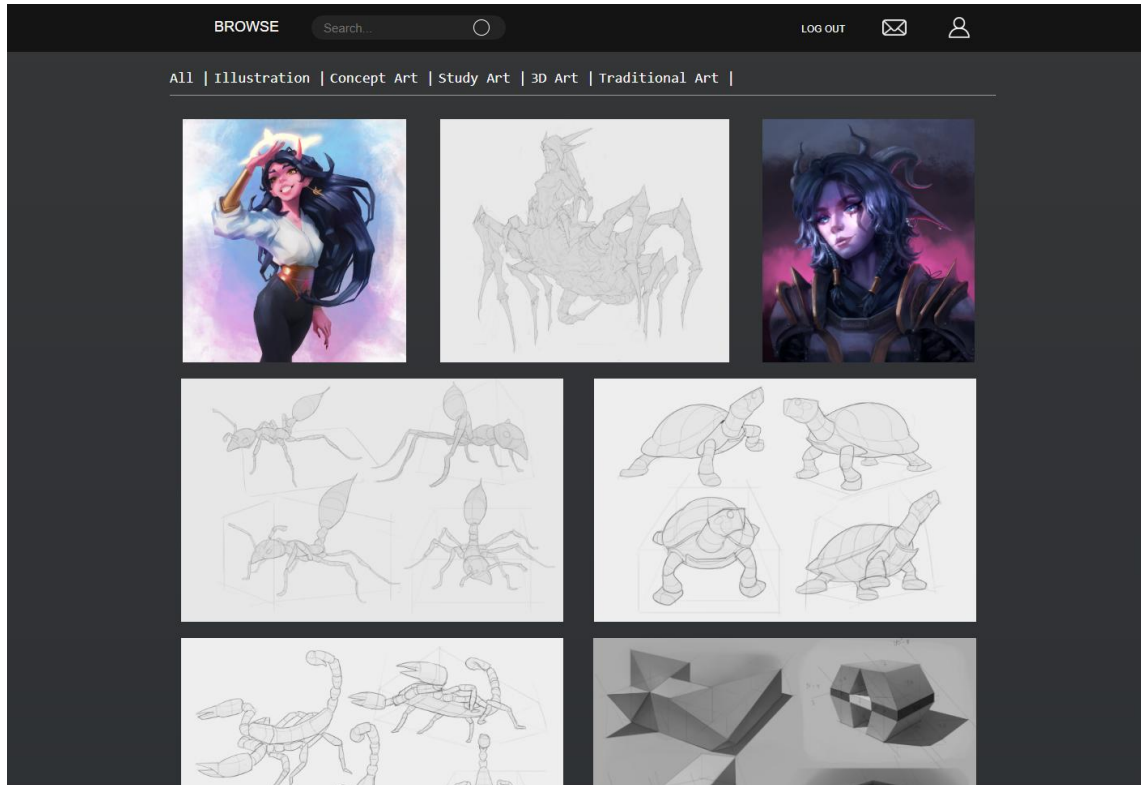


Рисунок Б.1 – Головна сторінка

Зображення у загальній галереї можуть бути відсортовані за категоріями, до яких належать ці зображення. На рис. Б.2 показані зображення з категорії «Study Art» (учбові малюнки).

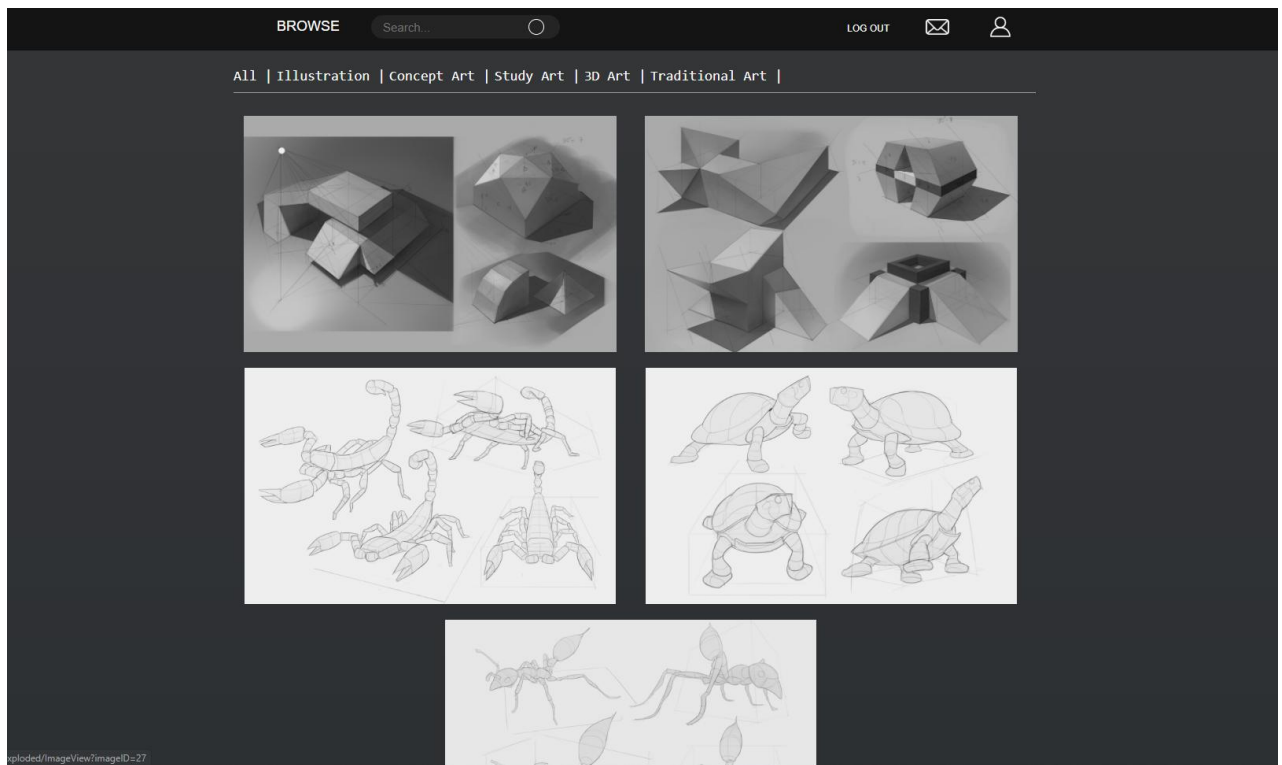


Рисунок Б.2 – Категорія «Study Art»

Б.2 Сторінка входу

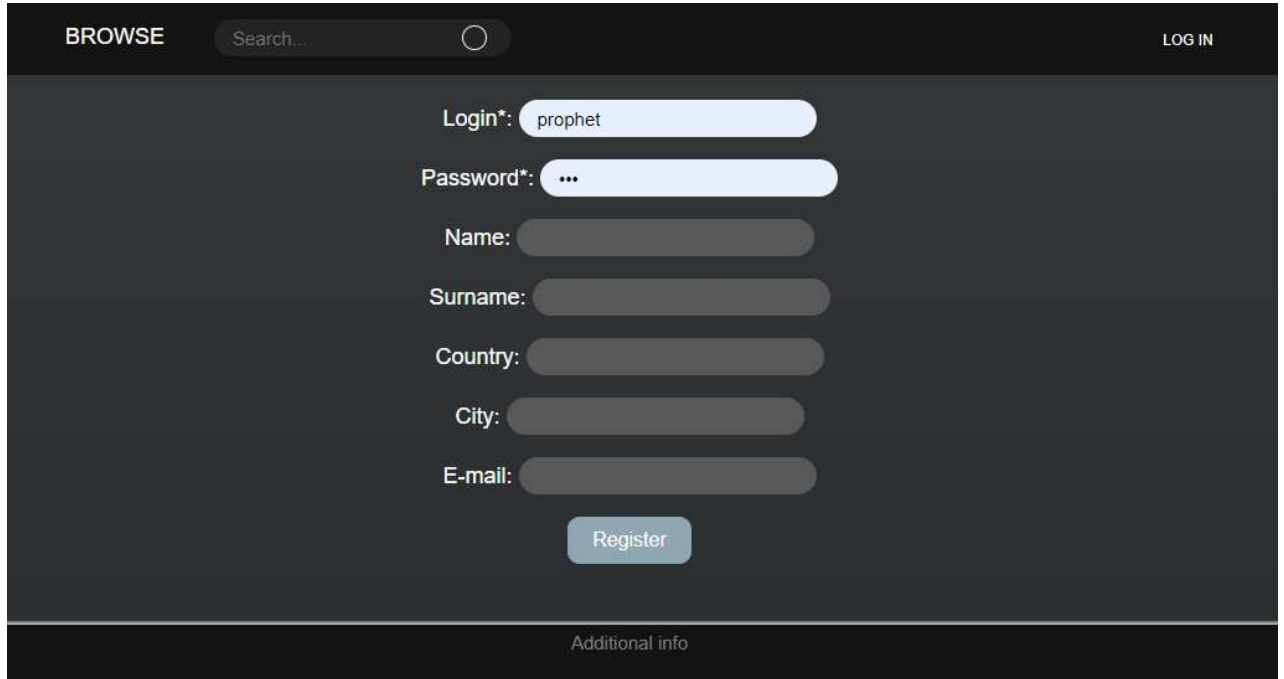
Сторінка входу відображена на рис. Б.3.

Для входу в аккаунт необхідно ввести свої логін і пароль. Якщо користувач не зареєстрований на сайті, необхідно перейти за посиланням «Register».

Рисунок Б.3 – Сторінка входу

Б.3 Сторінка реєстрації

Сторінка реєстрації користувача відображена на рис. Б.4. Для реєстрації необхідно ввести наступні дані: логін (повинен бути унікальним) і пароль (обов'язково), ім'я, прізвище, країну, місто і електронну пошту (за бажанням).



The screenshot shows a registration form on a dark background. The form includes the following elements:

- Top navigation: 'BROWSE' button, a search bar with 'Search...' text, and a 'LOG IN' button.
- Form fields:
 - 'Login*': text input with 'prophet' entered.
 - 'Password*': text input with three dots indicating a password.
 - 'Name:': text input.
 - 'Surname:': text input.
 - 'Country:': text input.
 - 'City:': text input.
 - 'E-mail:': text input.
- Buttons: A 'Register' button below the form fields.
- Footer: A link labeled 'Additional info' at the bottom center.

Рисунок Б.4 – Сторінка реєстрації

Б.4 Сторінка профілю

Сторінка профілю відображена на рис. Б.5. На ній відображається коротка інформація про користувача, його останній доданий пост, і усі його завантажені роботи. Для того, щоб завантажити роботу або аватар користувача, треба, перебуваючи на своїй сторінці, натиснути кнопку «+» у відповідному місці.

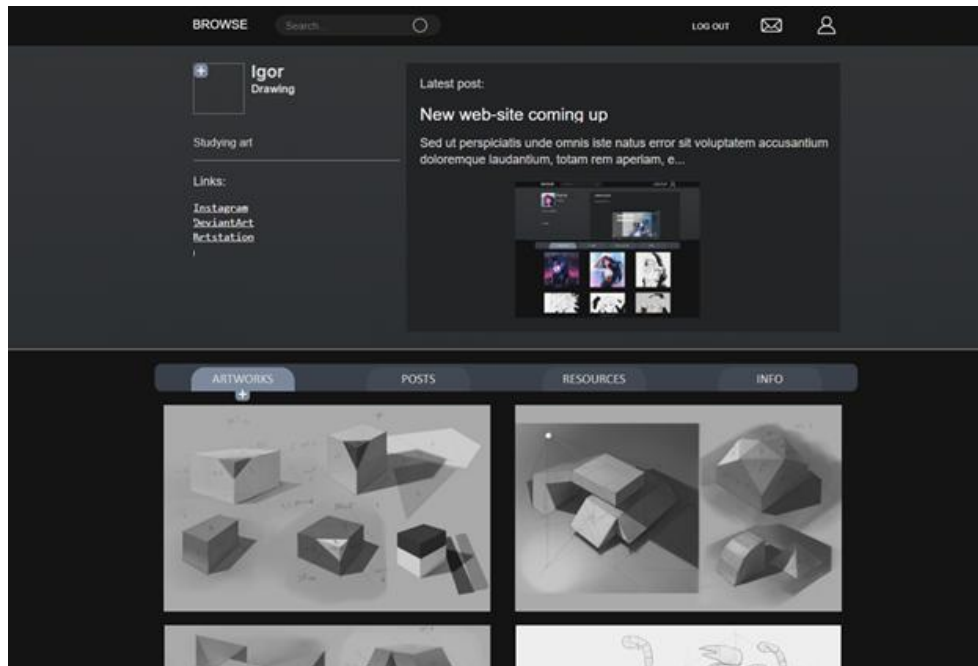


Рисунок Б.5 – Сторінка профілю користувача

Для того щоб переглянути пости користувача необхідно перейти у вкладку «Posts» (рис Б.6).

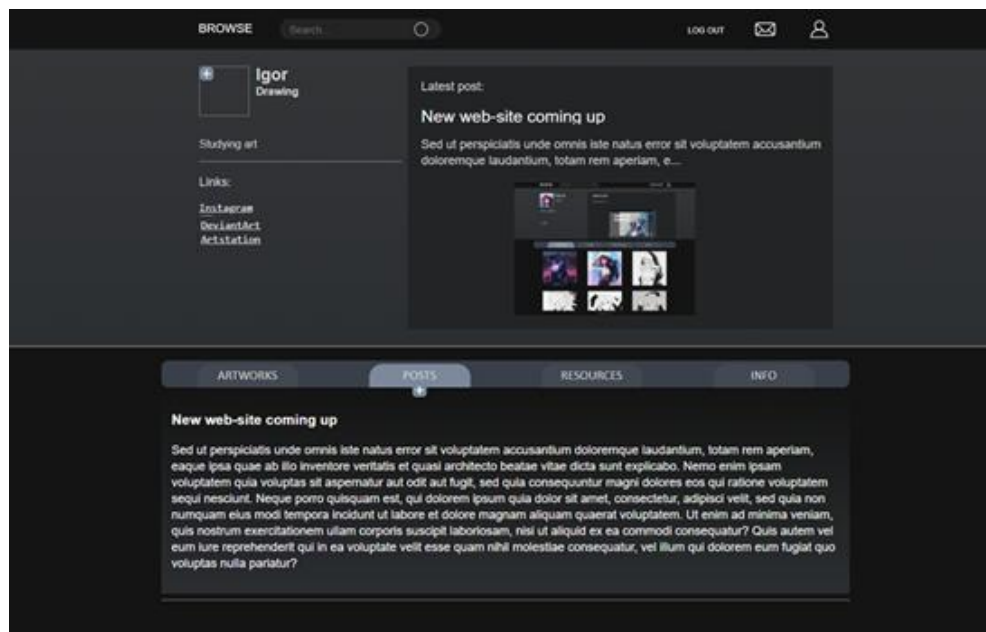


Рисунок Б.6 – Пости користувача

Для того щоб переглянути повну інформацію про користувача необхідно перейти у вкладку «Info».

Для того, щоб відредагувати інформацію про себе, необхідно натиснути на кнопку «Update» На даній сторінці користувач має змогу вказати свій статус та опис, що буде відображено на сторінці користувача.

На рис. Б.7. відображена сторінка іншого користувача. Є можливість переглянути його роботи, пости, ресурси або повну інформацію про користувача. Також можна відправити особисте повідомлення даному користувачу.

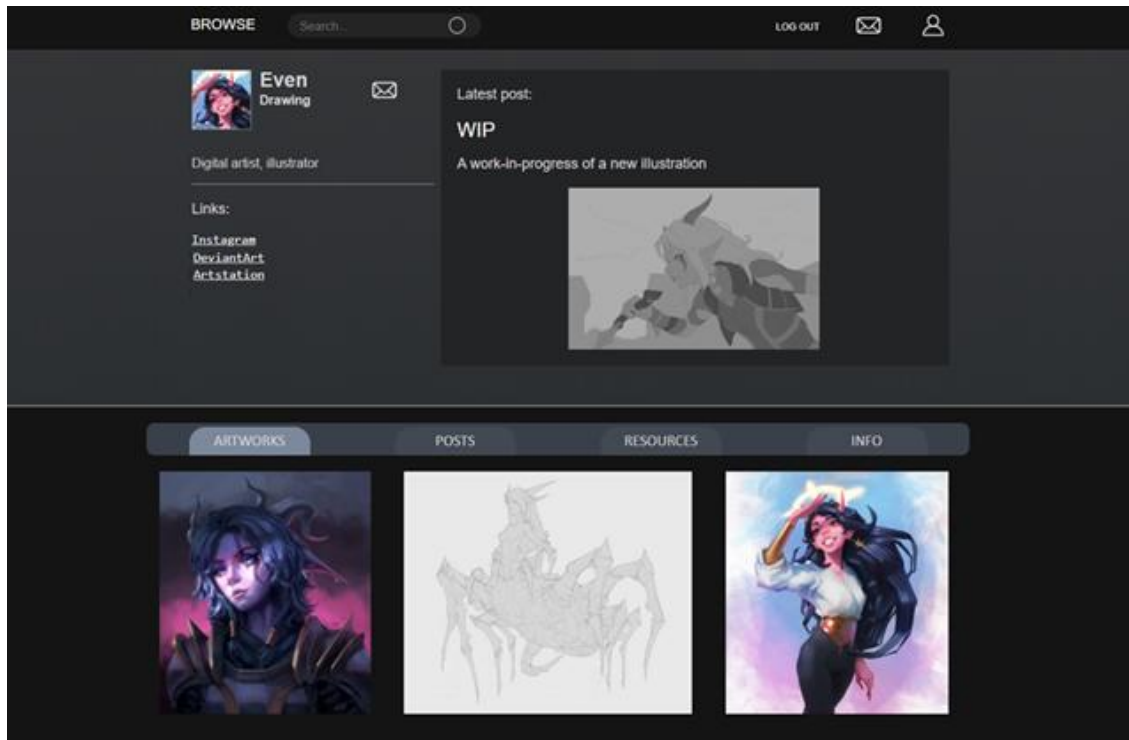


Рисунок Б.7 – Сторінка профілю іншого користувача

Б.5 Сторінка перегляду зображення

Сторінка перегляду зображення представлена на рис Б.8. На даній сторінці знаходиться вибрана робота користувача у найбільш оптимальному розмірі для перегляду, її найменування, опис, дата публікації та кількість переглядів, ім'я та аватар автора, а також найменування категорії до якої належить зображення, що також є гіперпосиланням до цієї категорії у галереї користувача.

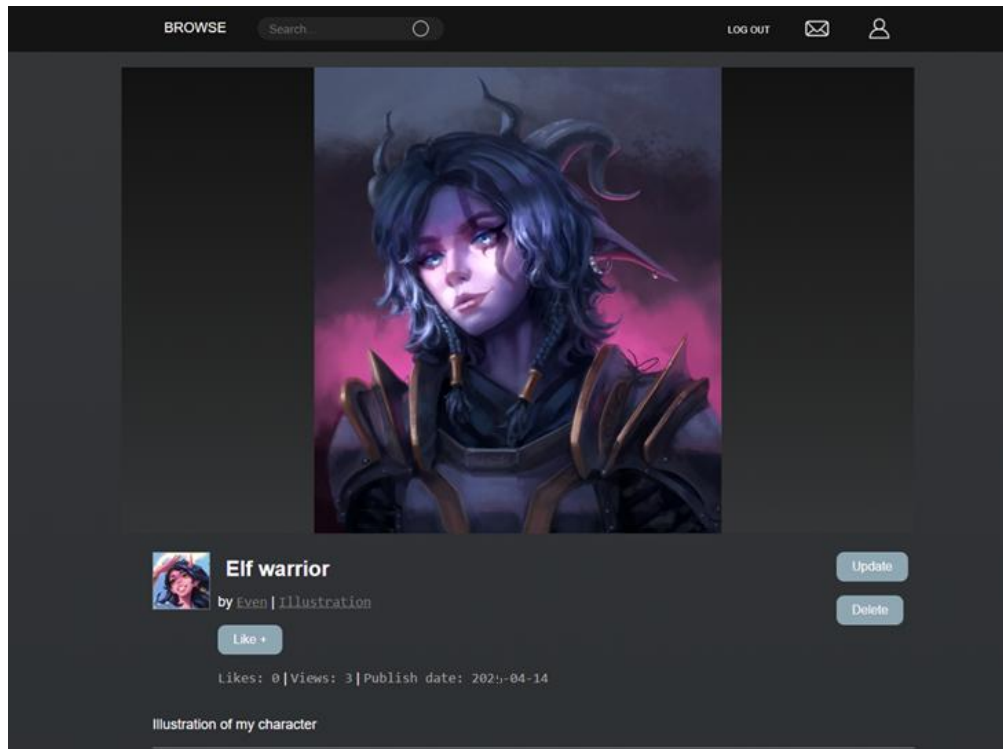


Рисунок Б.8 – Сторінка перегляду зображення

До зображення будь-який користувач може залишити відмітку «Подобається», натиснув кнопку «Like +» або зняти її відповідно, та також додати коментар.

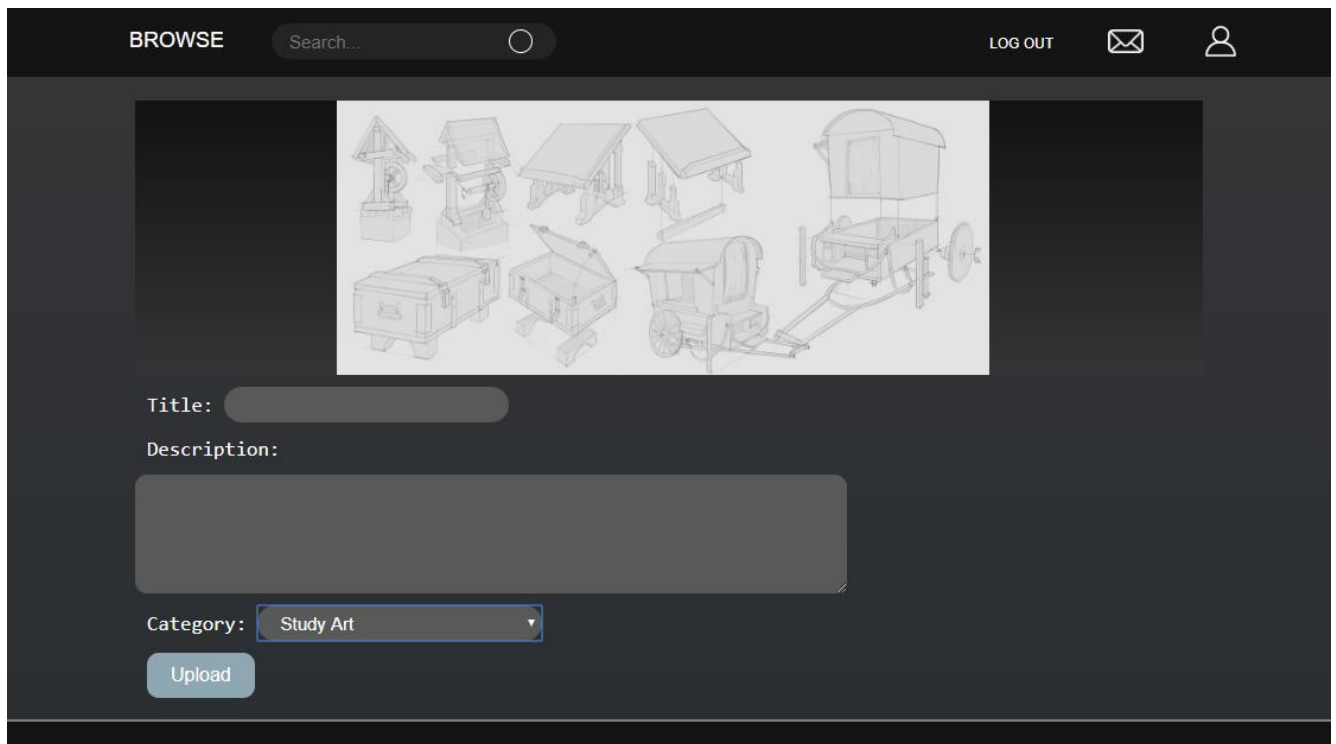
Якщо це зображення користувача, який увійшов у свій аккаунт, він має змогу оновити інформацію про дане зображення або видалити його за допомогою кнопок «Update» або «Delete» у правій частині вікна відповідно. Сторінка редагування зображення відображена на рис Б.9. На даній сторінці можна відредагувати заголовок, опис та категорію зображення.

The screenshot shows a web application interface with a dark theme. At the top, there is a navigation bar with the word "BROWSE" on the left, a search bar with the placeholder text "Search..." in the center, and "LOG OUT" on the right. To the right of "LOG OUT" are two icons: a bell for notifications and a user profile icon. Below the navigation bar is a large rectangular area containing a central image of a character with blue hair and horns. To the left of this image is a dark vertical bar, and to the right is another dark vertical bar. Below the central image, there is a form for editing information. It includes a "Title:" label followed by a text input field containing "Elf warrior". Below this is a larger text area with the placeholder text "Illustration of my character". Underneath the text area is a "Description:" label. Below the description label is a "Category:" label followed by a dropdown menu currently showing "Illustration". At the bottom left of the form is a blue "Upload" button. At the bottom center of the entire interface is a link labeled "Additional info".

Рисунок Б.9 – Сторінка редагування інформації про зображення

Б.6 Сторінка завантаження зображення на сервер

Сторінка завантаження зображення на сервер відображена на рис. Б.10. На даній сторінці, після вибору необхідного зображення з файлової системи, необхідно ввести заголовок та опис зображення, а також вказати, до якої категорії воно відноситься. Після натиску на кнопку «Upload» зображення з'явиться на сторінці користувача та буде додано до загальної галереї.



The screenshot shows a web interface for uploading an image. At the top, there is a navigation bar with the word "BROWSE" on the left, a search bar with the placeholder text "Search..." in the center, and "LOG OUT" on the right. To the right of "LOG OUT" are two icons: an envelope and a user profile. Below the navigation bar is a large rectangular area for the image upload. Inside this area, there is a preview of the image being uploaded, which shows several 3D wireframe models of mechanical devices. Below the image preview, there are input fields for "Title:" and "Description:". The "Description:" field is a larger text area. Below these fields is a "Category:" label followed by a dropdown menu that currently shows "Study Art". At the bottom of the form is an "Upload" button.

Рисунок Б.10 – Сторінка завантаження зображення

Б.7 Сторінка перегляду поста

На сторінці перегляду поста знаходиться вибраний пост користувача, його найменування, основний текст, дата публікації, кількість переглядів та «лайків», ім'я та аватар автора, та зображення до поста.

До постів, як і до зображень, будь-який користувач може ставити «лайки» та додавати коментарі.

Б.8 Сторінка додавання посту

Для того щоб додати новий пост, необхідно натиснути на кнопку «+» на вкладці «Posts» при перегляді сторінки користувача (рис. Б.6). До поста можна додати зображення, але це не є обов'язковим. Останній доданий пост буде відображено на сторінці користувача. Сторінка додавання посту відображена на рис. Б.11.

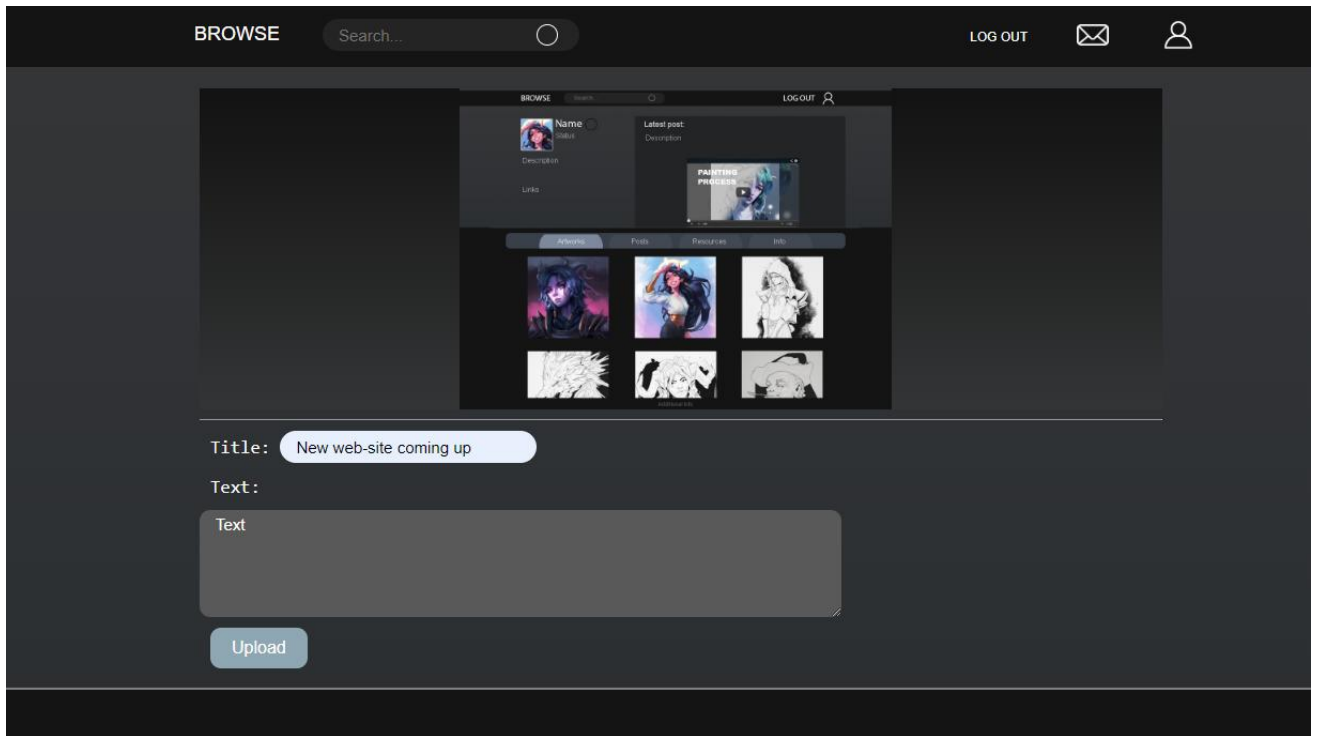


Рисунок Б.11 – Сторінка додавання посту

Б.9 Сторінки особистих повідомлень

Будь який користувач може відправити іншому користувачу особисте повідомлення. Для того, щоб це зробити, треба, переглядаючи сторінку необхідного користувача, натиснути на кнопку з іконкою листа поряд з ім'ям користувача. Сторінка для написання повідомлення зображена на рис. Б.12:

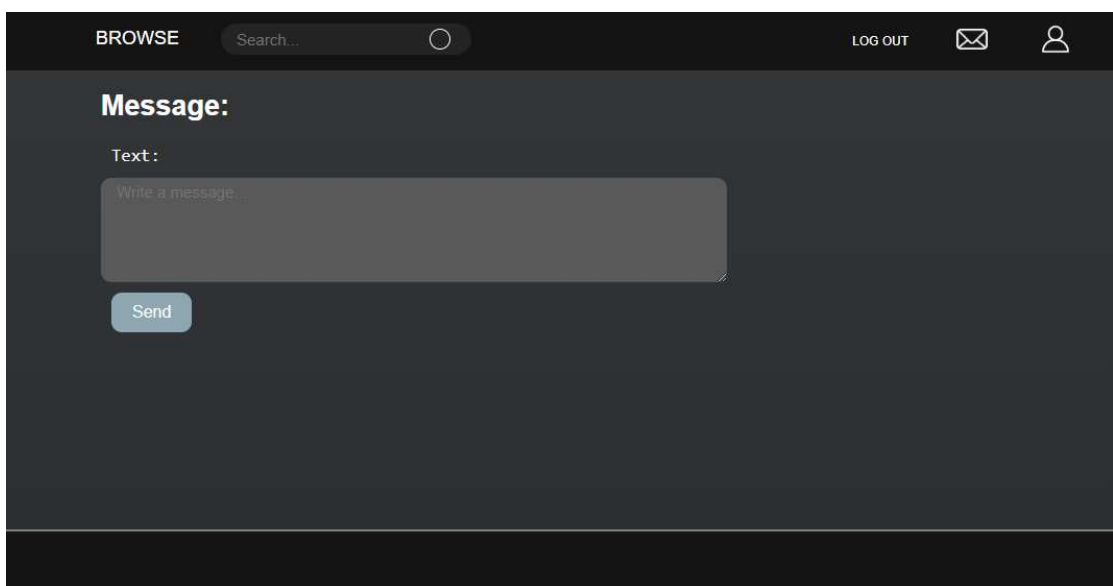


Рисунок Б.12 – Сторінка для відправлення особистого повідомлення

Після написання даного повідомлення буде відкритий діалог з даним користувачем, де будуть відображені усі особисті повідомлення між користувачами.

Для того щоб переглянути усі існуючі діалоги с користувачами, необхідно натиснути на кнопку з іконкою листа у хедері.

Б.10 Сторінка пошуку

Для того, щоб скористатись пошуком на сайті, необхідно ввести ключові слова у поле пошуку у хедері. Пошук ведеться по користувачам (Ім'я/Фамілія), постам (Назва/Текст) та зображенням (Назва/Опис).

Результати пошуку по ключовому слову «Light» зображені на рис. Б.13. Як можна бачити, були знайдені усі зображення що містять у найменуванні слово «Light».

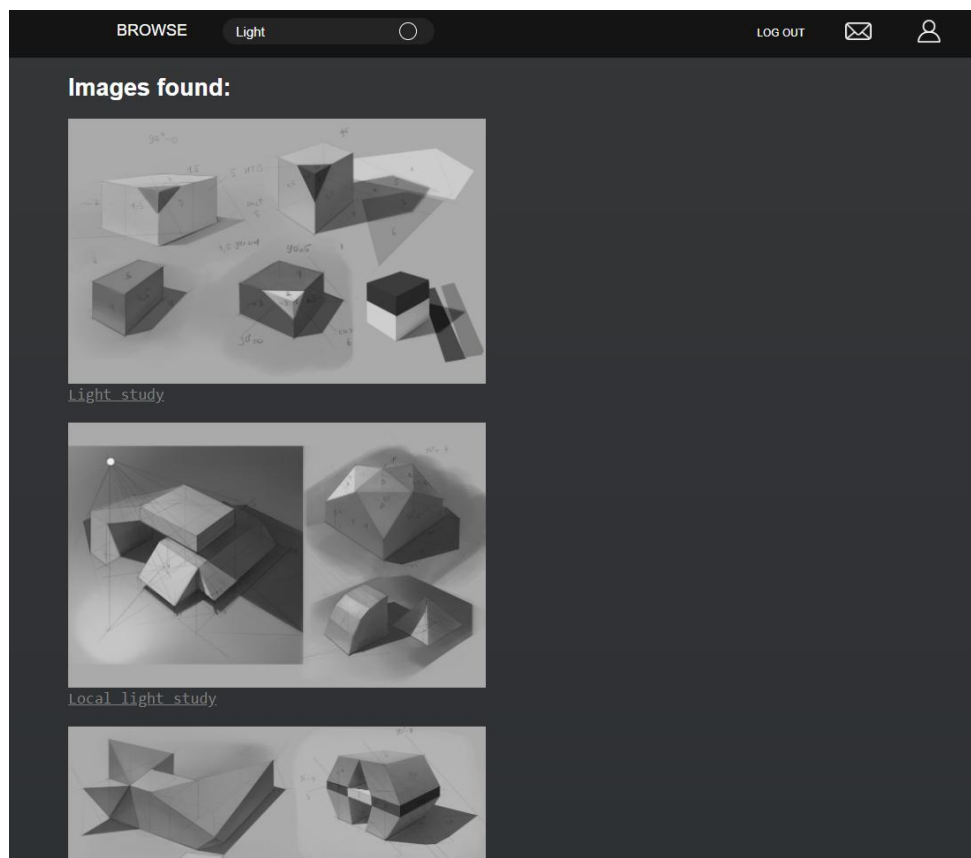


Рисунок Б.13 – Результати пошуку по ключовому слову «Light»

Результати пошуку по ключовому слову «Web» зображені на рис. Б.14. Як можна бачити, був знайдений пост що містить у найменуванні слово «Web».

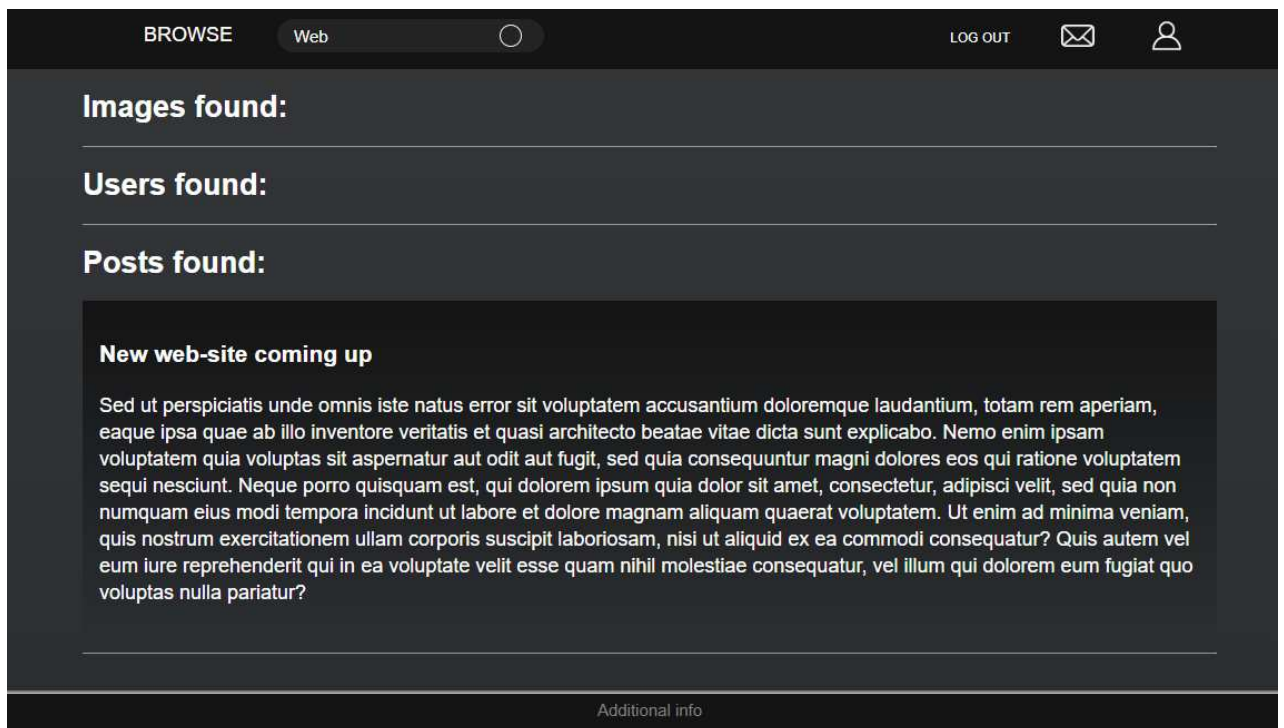


Рисунок Б.14 – Результати пошуку по ключовому слову «Web»

Додаток В. Текст програми

HEADER.JSP

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<div class="header-top">
  <a class="home-link" href="MainGallery">Browse</a>
  <form action="SearchSite" method="post">
    <div class="search">
      <input type="text" placeholder="Search..." name="searchText"
class="round" />
      <input type="submit" class="corner" value="" />
    </div>
  </form>
  <ul class="user-nav clearfix">
    <c:if test="${userID!=0}">
      <li class="profile-icon"><a href="profile"></a></li>
      <li class="note-bell-icon"><a href="UserDialogues"></a></li>
      <li>
        <a href="LogOut">Log Out</a>
      </li>
    </c:if>
    <c:if test="${userID==0}">
      <li>
        <a href="index.jsp">Log In</a>
      </li>
    </c:if>
  </ul>
</div>
```

FOOTER.JSP

```
<div class="footer">
  <div class="content">
    <p>Additional info</p>
  </div>
</div>
```

INDEX.JSP

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<c:set var="userID" value="0" scope="session"/>
<html>
  <head>
    <title>Gallery</title>
    <link rel="stylesheet" href="style/style.css">
  </head>
  <body>
    <div class="header-top-bg">
      <div class="page-header">
        <div class="container">
          <jsp:include page="header.jsp"/>
        </div>
      </div>
      <div class="container">
        <div class="login">
          <h2>Sign In:</h2>
```



```

        <form action="LogIn" method="post">
            <p><input type="text" name="login" placeholder="Login..."
class="enter-field"></p>
            <p><input type="password" name="password"
placeholder="Password..." class="enter-field"></p>
            <input type="submit" class="submit" value="Submit">
        </form>
        <p class="register"><a href="registration.jsp">Register</a></p>
    </div>
</div>
</div>
<jsp:include page="footer.jsp"/>
</body>
</html>

```

REGISTRATION.JSP

```

<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>Registration</title>
    <link rel="stylesheet" href="style/style.css">
</head>
<body>
    <div class="header-top-bg">
        <div class="page-header">
            <div class="container">
                <jsp:include page="header.jsp"/>
            </div>
        </div>
        <div class="container">
            <div class="login">
                <form method="post" action="Registration">
                    <p>Login*: <input type="text" class="enter-field"
name="login"></p>
                    <p>Password*: <input type="password" class="enter-field"
name="password"></p>
                    <p>Name: <input type="text" class="enter-field" name="name"></p>
                    <p>Surname: <input type="text" class="enter-field"
name="surname"></p>
                    <p>Country: <input type="text" class="enter-field"
name="country"></p>
                    <p>City: <input type="text" class="enter-field" name="city"></p>
                    <p>E-mail: <input type="text" class="enter-field"
name="email"></p>
                    <button type="submit" class="submit">Register</button>
                </form>
            </div>
        </div>
    </div>
    <jsp:include page="footer.jsp"/>
</body>
</html>

```

MAIN-GALLERY.JSP

```

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>

```

```

        <title>Title</title>
        <link rel="stylesheet" href="style/style.css">
</head>
<body>
<jsp:useBean id="imageBean" scope="request" class="beans.ImageBean"/>
<jsp:useBean id="imageCategoryBean" scope="request"
class="beans.ImageCategoryBean"/>
<div class="header-top-bg">
    <div class="page-header">
        <div class="container">
            <jsp:include page="header.jsp" />
        </div>
    </div>
    <div class="container">
        <div class="main-categories">
            <li><a href="MainGallery">All | </a></li>
            <c:forEach items="${imageCategoryBean.categories}" var="category">
                <li><a
href="MainGallerySort?cat=${category.idCategory}">${category.catName} | </a> </li>
            </c:forEach>
        </div>
        <hr/>
        <div class="artworks-section">
            <c:forEach items="${imageBean.images}" var="image">
                <a class="artwork-block"
href="ImageView?imageID=${image.idImage}"></a>
            </c:forEach>
        </div>
    </div>
</div>
<jsp:include page="footer.jsp"/>
</body>
</html>

```

IMAGE-VIEW.JSP

```

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jstl/sql" %>
<jsp:useBean id="image" scope="request" type="entities.Image"/>
<jsp:useBean id="author" scope="request" type="entities.User"/>
<jsp:useBean id="category" scope="request" type="entities.ImageCategory"/>
<jsp:useBean id="imageCommentsBean" scope="request"
type="beans.ImageCommentsBean"/>
<jsp:useBean id="userCommentsBean" scope="request" type="beans.UserBean"/>
<html>
<head>
    <title>${image.title}</title>
    <link rel="stylesheet" href="style/style.css">
</head>
<body>
<div class="header-top-bg">
    <div class="page-header">
        <div class="container">
            <jsp:include page="header.jsp" />
        </div>
    </div>
    <div class="container">
        <div class="image-view">

```

```

        <div class="artwork-view"></div>
    </div>
    <br/>
    <div class="image-view-info">
        <div class="user-info-picture"><a
href="profile?profileID=${author.idUser}"></a></div>
        <div class="image-view-text">
            <h2>${image.title}</h2>
            <p>by <a class="under-title"
href="profile?profileID=${author.idUser}"> ${author.uName}</a> | <a class="under-
title"
href="MainGallerySort?cat=${category.idCategory}">${category.catName}</a></p>
            <c:if test="${userID != 0}">
                <c:choose>
                    <c:when test="${like == false}">
                        <form action="ImageLike" method="post">
                            <input type="hidden" name="imageID"
value="${image.idImage}">
                            <button class="submit" value="Like +">Like +</button>
                        </form>
                    </c:when>
                    <c:otherwise>
                        <form action="ImageDislike" method="post">
                            <input type="hidden" name="imageID"
value="${image.idImage}">
                            <button class="submit" value="Liked!">Dislike</button>
                        </form>
                    </c:otherwise>
                </c:choose>
            </c:if>
            <p><em class="add-info">Likes: ${imageCountLikes}</em> | <em
class="add-info">Views: ${image.viewsCounter}</em> | <em class="add-
info">Publish date: ${image.publishDate}</em></p>

        </div>
        <c:if test="${userID==author.idUser}">
            <div class="image-view-control">
                <form method="post" action="PreUpdateImage">
                    <button name="imageID" class="submit" value="${image.idImage}"
type="submit">Update</button>
                </form>
                <form method="post" action="DeleteImage">
                    <button name="imageID" class="submit" value="${image.idImage}"
type="submit">Delete</button>
                </form>
            </div>
        </c:if>
        <p>${image.description}</p>
        <hr/>
        <p class="under-title">Comments:</p>
        <c:if test="${userID!=0}">
            <form action="ImageCommentAdd" method="post">
                <textarea class="enter-field-textarea" placeholder="Write a
comment..." name="commentText"></textarea>
                <input type="hidden" name="imageID" value="${image.idImage}">
                <p><input type="submit" class="submit" value="Submit"></p>
            </form>

```

```

        </c:if>
        <hr/>
        <c:forEach items="${imageCommentsBean.imageComments}" var="comment">
            <c:forEach items="${userCommentsBean.users}" var="user">
                <c:if test="${user.idUser == comment.idUser}">
                    <div class="comment">
                        <div class="user-info-picture">
                            <a href="profile?profileID=${user.idUser}"></a>
                        </div>
                        <div class="image-view-text">
                            <a class="under-title"
href="profile?profileID=${user.idUser}"> ${user.userName}</a>
                            <p>${comment.commentText}</p>
                            <p class="under-title" id="comment-
date">${comment.publishDateTime.toGMTString()}</p>
                        </div>
                    </div>
                </div>
            <hr/>
        </c:if>
    </c:forEach>
</c:forEach>

</div>
</div>
</div>
<jsp:include page="footer.jsp" />
</body>
</html>

```

POST-VIEW.JSP

```

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jstl/sql" %>
<jsp:useBean id="post" scope="request" type="entities.Post"/>
<jsp:useBean id="author" scope="request" type="entities.User"/>
<!--<jsp:useBean id="postCommentsBean" scope="request"
type="beans.PostCommentsBean"/>--%>
<!--<jsp:useBean id="userCommentsBean" scope="request" type="beans.UserBean"/>--%>
<html>
<head>
    <title>${post.postTitle}</title>
    <link rel="stylesheet" href="style/style.css">
</head>
<body>
<div class="header-top-bg">
    <div class="page-header">
        <div class="container">
            <jsp:include page="header.jsp" />
        </div>
    </div>
    <div class="container">
        <br/>
        <div class="image-view-info post-block">
            <div class="user-info-picture"><a
href="profile?profileID=${author.idUser}"></a></div>
            <div class="image-view-text">
                <h2>${post.postTitle}</h2>

```

```

        <p>by <a class="under-title"
href="profile?profileID=${author.idUser}"> ${author.uName}</a> </p>
        <c:if test="${userID != 0}">
            <c:choose>
                <c:when test="${like == false}">
                    <form action="PostLike" method="post">
                        <input type="hidden" name="postID"
value="${post.idPost}">
                        <button class="submit" value="Like +">Like
+</button>
                    </form>
                </c:when>
                <c:otherwise>
                    <form action="PostDislike" method="post">
                        <input type="hidden" name="postID"
value="${post.idPost}">
                        <button class="submit"
value="Liked!">Dislike</button>
                    </form>
                </c:otherwise>
            </c:choose>
        </c:if>
        <p><em class="add-info">Likes: ${postCountLikes}</em> | <em
class="add-info">Views: ${post.viewsCounter}</em> | <em class="add-info">Publish
date: ${post.publishDate.toGMTString()}</em></p>

    </div>
    <!--<c:if test="${userID==author.idUser}">--%>
        <!--<div class="image-view-control">--%>
            <!--<form method="post" action="PreUpdateImage">--%>
                <!--<button name="imageID" class="submit"
value="${image.idImage}" type="submit">Update</button>--%>
            <!--</form>--%>
            <!--<form method="post" action="DeleteImage">--%>
                <!--<button name="imageID" class="submit"
value="${image.idImage}" type="submit">Delete</button>--%>
            <!--</form>--%>
        <!--</div>--%>
    <!--</c:if>--%>
    <p>${post.postText}</p>
    <div class="post-image-view">
        
    </div>
    <hr/>
    <p class="under-title">Comments:</p>
    <c:if test="${userID!=0}">
        <form action="PostCommentAdd" method="post">
            <textarea class="enter-field-textarea" placeholder="Write a
comment..." name="commentText"></textarea>
            <input type="hidden" name="postID" value="${post.idPost}">
            <p><input type="submit" class="submit" value="Submit"></p>
        </form>
    </c:if>
    <hr/>
    <c:forEach items="${postCommentsBean.postComments}" var="comment">
        <c:forEach items="${userCommentsBean.users}" var="user">
            <c:if test="${user.idUser == comment.idUser}">
                <div class="comment">

```

```

        <div class="user-info-picture">
            <a href="profile?profileID=${user.idUser}"></a>
        </div>
        <div class="image-view-text">
            <a class="under-title"
href="profile?profileID=${user.idUser}"> ${user.userName}</a>
            <p>${comment.commentText}</p>
            <p class="under-title" id="comment-
date">${comment.publishDateTime.toGMTString()}</p>
        </div>
    </div>
</div>
<hr/>
</c:if>
</c:forEach>
</c:forEach>

</div>
</div>
</div>
<jsp:include page="footer.jsp" />
</body>
</html>

```

PROFILE.JSP

```

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<jsp:useBean id="imageBean" scope="request" class="beans.ImageBean"/>
<jsp:useBean id="user" scope="request" type="entities.User"/>
<html>
<head>
    <title>USERNAME -- GalleryName</title>
    <link rel="stylesheet" href="style/style.css">
</head>
<body>
<div class="header-top-bg">
    <div class="page-header">
        <div class="container">
            <jsp:include page="header.jsp" />
            <jsp:include page="profile-header-info.jsp" />
        </div>
    </div>
</div>

<div class="main-content">
    <div class="container">
        <ul class="main-navigation">
            <li class="active">
                <a href="#">Artworks</a>
                <c:if test="${userID==user.idUser}">
                    <form action="UploadType">
                        <button class="upload-btn">+</button>
                        <input type="hidden" name="type" value="images">
                    </form>
                </c:if>
            </li>
            <li>
                <a href="profile-posts?profileID=${user.idUser}">Posts</a>
            </li>
            <li>

```

```

        <a href="profile-
resources?profileID=${user.idUser}">Resources</a>
    </li>
    <li>
        <a href="profile-info?profileID=${user.idUser}">Info</a>
    </li>
</ul>
<div class="artworks-section">
    <c:forEach items="${imageBean.images}" var="image">
        <a class="artwork-block"
href="ImageView?imageID=${image.idImage}"></a>
    </c:forEach>
</div>
</div>
</div>
<jsp:include page="footer.jsp"/>
</body>
</html>

```

PROFILE-HEADER-INFO.JSP

```

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<jsp:useBean id="user" scope="request" type="entities.User"/>
<jsp:useBean id="postLatest" scope="session" type="entities.Post"/>
<div class="profile-info clearfix">
    <div class="user-info">
        <div class="user-info-picture">
        <c:if test="${user.avatarFilename.equals('')} && userID==user.idUser}">
            <form action="UploadType">
                <button class="upload-btn">+</button>
                <input type="hidden" name="type" value="avatars">
            </form>
        </c:if>
    </div>
    <div class="user-brief">
        <c:if test="${userID!=user.idUser && userID!=0}">
            <div class="note-bell-icon"><a href="send-
message.jsp?idUserRecipient=${user.idUser}"></a></div>
        </c:if>
        <h2>${user.uName}</h2>
        <h3>${user.userStatus}</h3>
    </div>
    <div class="description">
        <p>${user.description}</p>
    </div>
    <hr/>
    <p>Links:</p>
    <ul class="links">
        <li><a href="#">Instagram</a></li>
        <li><a href="#">VK</a></li>
        <li><a href="#">DeviantArt</a></li>
        <li><a href="#">Artstation</a></li>
    </ul>
</div>
<div class="latest-post">
    <p>Latest post:</p>
    <a href="PostView?postID=${postLatest.idPost}">${postLatest.postTitle}</a>

```

```

        <p>${text_preview}</p>
        <div class="latest-post-content">
            
        </div>
    </div>
</div>

```

PROFILE-INFO.JSP

```

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page contentType="text/html; charset=UTF-8" %>
<jsp:useBean id="user" scope="request" type="entities.User"/>
<html>
<head>
    <title>USERNAME -- GalleryName</title>
    <link rel="stylesheet" href="style/style.css">
</head>
<body>
<div class="header-top-bg">
    <div class="page-header">
        <div class="container">
            <jsp:include page="header.jsp" />
            <jsp:include page="profile-header-info.jsp" />
        </div>
    </div>
</div>
<div class="main-content">
    <div class="container">
        <ul class="main-navigation">
            <li>
                <a href="profile?profileID=${user.idUser}">Artworks</a>
            </li>
            <li>
                <a href="profile-posts?profileID=${user.idUser}">Posts</a>
            </li>
            <li>
                <a href="profile-resources?profileID=${user.idUser}">Resources</a>
            </li>
            <li class="active">
                <a href="#">Info</a>
            </li>
        </ul>
        <div class="info-section">
            <p>Full name: ${user.userName} ${user.uSurname}</p>
            <p>Description: ${user.description}</p>
            <p>Registration date: ${user.creationDate}</p>
            <p>Country: ${user.country}</p>
            <p>City: ${user.city}</p>
            <p>E-Mail: ${user.email}</p>
        </div>
        <c:if test="${userID == user.idUser}">
            <form action="user-change-info.jsp?profileID=${user.idUser}"
method="post">
                <input type="submit" class="submit" value="Update">
            </form>
        </c:if>
    </div>
</div>
</div>

```



```

<jsp:include page="footer.jsp"/>
</body>
</html>

```

PROFILE-POSTS.JSP

```

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page contentType="text/html; charset=UTF-8" %>
<jsp:useBean id="user" scope="request" type="entities.User"/>
<jsp:useBean id="imageBean" scope="request" class="beans.ImageBean"/>
<jsp:useBean id="postBean" scope="request" type="beans.PostBean"/>
<html>
<head>
    <title>USERNAME -- GalleryName</title>
    <link rel="stylesheet" href="style/style.css">
</head>
<body>
<div class="header-top-bg">
    <div class="page-header">
        <div class="container">
            <jsp:include page="header.jsp" />
            <jsp:include page="profile-header-info.jsp" />
        </div>
    </div>
</div>
<div class="main-content">
    <div class="container">
        <ul class="main-navigation">
            <li>
                <a href="profile?profileID=${user.idUser}">Artworks</a>
            </li>
            <li class="active">
                <a href="#">Posts</a>
                <c:if test="${userID==user.idUser}">
                    <form action="upload-post.jsp">
                        <button class="upload-btn">+</button>
                    </form>
                </c:if>
            </li>
            <li>
                <a href="profile-resources?profileID=${user.idUser}">Resources</a>
            </li>
            <li>
                <a href="profile-info?profileID=${user.idUser}">Info</a>
            </li>
        </ul>
        <div class="posts-section">
            <c:forEach items="${postBean.posts}" var="post">
                <div class="post-block-preview">
                    <a
href="PostView?postID=${post.idPost}"><h3>${post.postTitle}</h3></a>
                    <p>${post.postText}</p>
                </div>
                <hr/>
            </c:forEach>
        </div>
    </div>
</div>
<jsp:include page="footer.jsp"/>
</body>

```

</html>

UPLOAD.JSP

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>File Upload</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <link rel="stylesheet" href="style/style.css">
</head>
<body>
<div class="header-top-bg">
    <div class="page-header">
        <div class="container">
            <jsp:include page="header.jsp" />
        </div>
    </div>
<div class="container">
    <div class="upload-page">
        <p>Select the file:</p>
        <form method="post" action="Upload" enctype="multipart/form-data" >
            <input type="file" name="file"/>
            <br/> <br/>
            <input type="submit" class="submit" name="upload"/>
        </form>
    </div>
</div>
</div>
<jsp:include page="footer.jsp" />
</body>
</html>
```

UPLOAD-INFO.JSP

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
    <link rel="stylesheet" href="style/style.css">
</head>
<body>
<div class="header-top-bg">
    <div class="page-header">
        <div class="container">
            <jsp:include page="header.jsp" />
        </div>
    </div>
<jsp:useBean id="imageCategoryBean" scope="request"
class="beans.ImageCategoryBean"/>
<div class="container">
    <div class="upload-page">
        <form action="ImageInfoUpload" method="post">
            <div class="image-view">
                <div class="artwork-block"></div>
            </div>
            <p>Title: <input name="title" class="enter-field" type="text"></p>
```

```

        <p>Description:</p> <textarea name="description" class="enter-
field-textarea" rows="4" cols="40"></textarea>
        <p>Category: <select class="enter-field" name="category">
            <c:forEach items="${imageCategoryBean.categories}"
var="category">
                <option
value="${category.idCategory}">${category.catName}</option>
            </c:forEach>
        </select></p>
        <input type="hidden" name="filename" value="${filename}">
        <p><input type="submit" class="submit" value="Upload"></p>
    </form>
</div>
</div>
</div>
</body>
</html>

```

USER-DIALOGUE.JSP

```

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<jsp:useBean id="user" scope="session" type="entities.User"/>
<jsp:useBean id="secondUser" scope="request" type="entities.User"/>
<jsp:useBean id="messagesBean" scope="request" type="beans.MessageBean"/>
<html>
<head>
    <title>Search Results</title>
    <link rel="stylesheet" href="style/style.css">
</head>
<body>
<div class="header-top-bg">
    <div class="page-header">
        <div class="container">
            <jsp:include page="header.jsp" />
        </div>
    </div>
    <div class="container">
        <div class="send-msg">
            <form action="SendMessage" method="post">
                <textarea name="text" placeholder="Write a message..."
class="enter-field-textarea" rows="6" cols="100"></textarea>
                <input type="hidden" name="idUserRecipient"
value="${secondUser.idUser}"><input type="submit" class="submit" value="Send">
            </form>
        </div>
        <div class="dialogue">
            <c:forEach items="${messagesBean.messages}" var="message">
                <c:if test="${user.idUser == message.idUserSender}">
                    <div class="user-msg">
                        <div class="msg-content">
                            <div class="user-info-picture">
                                <a href="profile?profileID=${user.idUser}"></a>
                            </div>
                            <div class="image-view-text">
                                <a class="under-title"
href="profile?profileID=${user.idUser}"> ${user.userName}</a>
                                <p>${message.textMessage}</p>

```

```

        <p class="under-
title">${message.dateTime.toGMTString()}</p>
        </div>
        </div>
    </div>
</c:if>
<c:if test="${secondUser.idUser == message.idUserSender}">
    <div class="comment">
        <div class="user-info-picture">
            <a
href="profile?profileID=${secondUser.idUser}"></a>
        </div>
        <div class="image-view-text">
            <a class="under-title"
href="profile?profileID=${secondUser.idUser}"> ${secondUser.uName}</a>
            <p>${message.textMessage}</p>
            <p class="under-
title">${message.dateTime.toGMTString()}</p>
        </div>
    </div>
</c:if>
</c:forEach>
</div>
</div>
</div>
<jsp:include page="footer.jsp"/>
</body>
</html>

```

USER-DIALOGUES.JSP

```

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>Search Results</title>
    <link rel="stylesheet" href="style/style.css">
</head>
<body>
<jsp:useBean id="userBean" scope="request" class="beans.UserBean"/>
<div class="header-top-bg">
    <div class="page-header">
        <div class="container">
            <jsp:include page="header.jsp" />
        </div>
    </div>
    <div class="container">
        <h2>Current dialogues:</h2>
        <div class="userSearch">
            <c:forEach items="${userBean.users}" var="user">
                <div class="user-info-picture ">
                    <a href="profile?profileID=${user.idUser}"></a>
                </div>
                <div class="image-view-text">
                    <a href="profile?profileID=${user.idUser}"> ${user.uName}
${user.uSurname}</a>
                    <p class="under-title">${user.city} ${user.country}</p>

```

```

        </div>
        <form action="Dialogue" method="post">
            <input type="hidden" value="{user.idUser}"
name="idUserRecipient">
            <input type="submit" class="submit" value="Open Dialogue">
        </form>
        <hr/>
    </c:forEach>
</div>
</div>
</div>
<jsp:include page="footer.jsp"/>
</body>
</html>

```

Image.java

```

package entities;

import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import java.sql.Date;
import java.util.Objects;

@Entity
public class Image {
    private int idImage;
    private String title;
    private String description;
    private String filename;
    private Date publishDate;
    private int viewsCounter;
    private Integer idCategory;
    private int idUserImage;

    public Image() {
    }

    public Image(String title, String description, String filename, Date
publishDate, Integer idCategory, int idUserImage) {
        this.title = title;
        this.description = description;
        this.filename = filename;
        this.publishDate = publishDate;
        this.idCategory = idCategory;
        this.idUserImage = idUserImage;
    }

    @Id
    @Column(name = "id_image")
    public int getIdImage() {
        return idImage;
    }

    public void setIdImage(int idImage) {

```

```

        this.idImage = idImage;
    }

    @Basic
    @Column(name = "title")
    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    @Basic
    @Column(name = "description")
    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    @Basic
    @Column(name = "filename")
    public String getFilename() {
        return filename;
    }

    public void setFilename(String filename) {
        this.filename = filename;
    }

    @Basic
    @Column(name = "publish_date")
    public Date getPublishDate() {
        return publishDate;
    }

    public void setPublishDate(Date publishDate) {
        this.publishDate = publishDate;
    }

    @Basic
    @Column(name = "views_counter")
    public int getViewsCounter() {
        return viewsCounter;
    }

    public void setViewsCounter(int viewsCounter) {
        this.viewsCounter = viewsCounter;
    }

    @Basic
    @Column(name = "id_category")
    public Integer getIdCategory() {
        return idCategory;
    }

```

```

    public void setIdCategory(Integer idCategory) {
        this.idCategory = idCategory;
    }

    @Basic
    @Column(name = "id_user_image")
    public int getIdUserImage() {
        return idUserImage;
    }

    public void setIdUserImage(int idUserImage) {
        this.idUserImage = idUserImage;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Image image = (Image) o;
        return idImage == image.idImage &&
            viewsCounter == image.viewsCounter &&
            idUserImage == image.idUserImage &&
            Objects.equals(title, image.title) &&
            Objects.equals(description, image.description) &&
            Objects.equals(filename, image.filename) &&
            Objects.equals(publishDate, image.publishDate) &&
            Objects.equals(idCategory, image.idCategory);
    }

    @Override
    public int hashCode() {
        return Objects.hash(idImage, title, description, filename, publishDate,
            viewsCounter, idCategory, idUserImage);
    }
}

```

User.java

```

package entities;

import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import java.sql.Date;
import java.util.Objects;

@Entity
public class User {
    private int idUser;
    private String login;
    private String uPassword;
    private String uName;
    private String uSurname;
    private Date creationDate;
    private String description;
    private String country;
    private String email;
    private String city;
}

```

```

    private String userStatus;
    private String avatarFilename;

    public User() {
    }

    public User(String login, String uPassword, String uName, String uSurname,
Date creationDate, String description, String country, String email, String city,
String avatarFilename) {
        this.login = login;
        this.uPassword = uPassword;
        this.uName = uName;
        this.uSurname = uSurname;
        this.creationDate = creationDate;
        this.description = description;
        this.country = country;
        this.email = email;
        this.city = city;
        this.avatarFilename = avatarFilename;
    }

    public User(String login, String uPassword, String uName) {
        this.login = login;
        this.uPassword = uPassword;
        this.uName = uName;
    }

    @Id
    @Column(name = "id_user")
    public int getIdUser() {
        return idUser;
    }

    public void setIdUser(int idUser) {
        this.idUser = idUser;
    }

    @Basic
    @Column(name = "login")
    public String getLogin() {
        return login;
    }

    public void setLogin(String login) {
        this.login = login;
    }

    @Basic
    @Column(name = "u_password")
    public String getuPassword() {
        return uPassword;
    }

    public void setuPassword(String uPassword) {
        this.uPassword = uPassword;
    }

    @Basic

```



```

@Column(name = "u_name")
public String getuName() {
    return uName;
}

public void setuName(String uName) {
    this.uName = uName;
}

@Basic
@Column(name = "u_surname")
public String getuSurname() {
    return uSurname;
}

public void setuSurname(String uSurname) {
    this.uSurname = uSurname;
}

@Basic
@Column(name = "creation_date")
public Date getCreationDate() {
    return creationDate;
}

public void setCreationDate(Date creationDate) {
    this.creationDate = creationDate;
}

@Basic
@Column(name = "description")
public String getDescription() {
    return description;
}

public void setDescription(String description) {
    this.description = description;
}

@Basic
@Column(name = "country")
public String getCountry() {
    return country;
}

public void setCountry(String country) {
    this.country = country;
}

@Basic
@Column(name = "email")
public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

```

```

@Basic
@Column(name = "city")
public String getCity() {
    return city;
}

public void setCity(String city) {
    this.city = city;
}

@Basic
@Column(name = "user_status")
public String getUserStatus() {
    return userStatus;
}

public void setUserStatus(String userStatus) {
    this.userStatus = userStatus;
}

@Basic
@Column(name = "avatar_filename")
public String getAvatarFilename() {
    return avatarFilename;
}

public void setAvatarFilename(String avatarFilename) {
    this.avatarFilename = avatarFilename;
}

@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    User user = (User) o;
    return idUser == user.idUser &&
        Objects.equals(login, user.login) &&
        Objects.equals(uPassword, user.uPassword) &&
        Objects.equals(uName, user.uName) &&
        Objects.equals(uSurname, user.uSurname) &&
        Objects.equals(creationDate, user.creationDate) &&
        Objects.equals(description, user.description) &&
        Objects.equals(country, user.country) &&
        Objects.equals(email, user.email) &&
        Objects.equals(city, user.city) &&
        Objects.equals(userStatus, user.userStatus) &&
        Objects.equals(avatarFilename, user.avatarFilename);
}

@Override
public int hashCode() {
    return Objects.hash(idUser, login, uPassword, uName, uSurname,
creationDate, description, country, email, city, userStatus, avatarFilename);
}
}

```

UserDAO.java

package dao;

```

import entities.ImageCommentary;
import entities.User;

import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import java.sql.Date;
import java.util.List;

@Stateless
public class UserDAO {
    @PersistenceContext
    EntityManager em;

    public List<User> findAll() {
        return em.createQuery("select u from User u", User.class).getResultList();
    }

    public User findById(int userId) {
        return em.find(User.class, userId);
    }

    public User findByLogin(String userLogin, String password) {
        List userEntities = em.createQuery("SELECT u FROM User u WHERE u.login = :login AND " +
            " u.uPassword = :password").setParameter("login",
userLogin).setParameter("password", password).getResultList();
        return (User) userEntities.get(0);
    }

    public List<User> findByNameOrSurname(String text) {
        return em.createQuery("SELECT u FROM User u WHERE u.uSurname like :text
OR " +
            " u.uName like :text", User.class).setParameter("text", "%" + text
+ "%").getResultList();
    }

    public User addUser(String login, String uPassword, String uName) {
        User user = new User(login, uPassword, uName);
        em.persist(user);
        return user;
    }

    public void removeUser(int userId) {
        User user = em.find(User.class, userId);
        em.remove(user);
    }

    public User updateUser(User user) {
        return em.merge(user);
    }

    public void addAvatar(User user, String filename) {
        user.setAvatarFilename(filename);
        em.merge(user);
    }
}

```

```

        public User addUser(String login, String password, String name, String
surname, Date date, String description, String country, String city, String email,
String avatar_filename) {
            User user = new User(login, password, name, surname, date, description,
country, city, email, avatar_filename);
            em.persist(user);
            return user;
        }
    }
}

```

ImageDAO.java

```

package dao;

import entities.Image;

import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;
import java.sql.Date;
import java.util.List;

@Stateless
public class ImageDAO {

    @PersistenceContext
    EntityManager em;

    public List<Image> findAll() {
        return em.createQuery("select i from Image i order by i.idImage DESC",
Image.class).getResultList();
    }

    public Image findById(int imageId) {
        return em.find(Image.class, imageId);
    }

    public void removeImage(int imageId) {
        Image image = em.find(Image.class, imageId);
        em.remove(image);
    }

    public Image updateImage(Image image) {
        return em.merge(image);
    }

    public void createNewImage(Image image) {
        em.persist(image);
    }

    public Image createNewImage(String title, String description, String filename,
Date publish, int categoryID, int idUser) {
        Image image = new Image(title, description, filename, publish, categoryID,
idUser);
        em.persist(image);
        return image;
    }
}

```

```

    public List<Image> findUserImages(int idUser) {
        return em.createQuery("select i from Image i where i.idUserImage = :idUser", Image.class).setParameter("idUser", idUser).getResultList();
    }

    public List<Image> findbyCategory(int idCategory) {
        return em.createQuery("select i from Image i where i.idCategory = :idCategory", Image.class).setParameter("idCategory", idCategory).getResultList();
    }

    public void plusView(int idImage){
        em.createQuery("UPDATE Image i set i.viewsCounter = i.viewsCounter + 1 where i.idImage = :idImage", Image.class).setParameter("idImage", idImage).executeUpdate();
        em.merge(findById(idImage));
    }

    public List<Image> findByTitleOrDescription(String text) {
        return em.createQuery("SELECT i FROM Image i WHERE i.title like :text OR"
+
            " i.description like :text", Image.class).setParameter("text", "%"
+ text + "%").getResultList();
    }
}

```

MessageDAO.java

```

package dao;

import entities.Message;

import javax.ejb.EJB;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;
import java.sql.Date;
import java.sql.Timestamp;
import java.util.List;

@Stateless
public class MessageDAO {

    @PersistenceContext
    EntityManager em;

    public List<Message> findAll() {
        return em.createQuery("select i from Message i", Message.class).getResultList();
    }

    public Message findById(int messageId) {
        return em.find(Message.class, messageId);
    }

    public Message findByDateTextSender(String text, Timestamp timestamp, int idSender) {

```

```

        return em.createQuery("select m from Message m where m.textMessage = :text
and m.dateTime = :datetime and m.idUserSender = :idSender",
Message.class).setParameter("text", text).setParameter("datetime", timestamp).
        setParameter("idSender", idSender).getResultList().get(0);
    }

    public void removeMessage(int messageId) {
        Message message = em.find(Message.class, messageId);
        em.remove(message);
    }

    public Message updateMessage(Message message) {
        return em.merge(message);
    }

    public void createNewMessage(Message message) {
        em.persist(message);
    }

    public Message createNewMessage(String text, Timestamp timestamp, int
idSender) {
        Message message = new Message(text, timestamp, idSender);
        em.persist(message);
        em.flush();
        em.merge(message);
        return message;
    }

    public List<Message> findUserMessages(int idUserSender, int idUserRecipient) {
        return em.createNativeQuery("select message.* from message INNER JOIN
message_recipient on message.id_message = message_recipient.id_message where
id_user_recipient = " + idUserRecipient + " and id_user_sender = " + idUserSender
+ " or id_user_recipient = " + idUserSender + " and id_user_sender = " +
idUserRecipient + " ORDER BY date_time desc ";", Message.class).getResultList();
    }
}

```

MessageRecipientDAO.java

```

package dao;

import entities.MessageRecipient;
import entities.User;

import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;
import java.sql.Date;
import java.util.List;
import java.util.stream.Collectors;

@Stateless
public class MessageRecipientDAO {

    @PersistenceContext
    EntityManager em;

    public List<MessageRecipient> findAll() {

```

```

        return em.createQuery("select i from MessageRecipient i",
MessageRecipient.class).getResultList();
    }

    public MessageRecipient findById(int messageRecipientId) {
        return em.find(MessageRecipient.class, messageRecipientId);
    }

    public void removeMessageRecipient(int messageRecipientId) {
        MessageRecipient messageRecipient = em.find(MessageRecipient.class,
messageRecipientId);
        em.remove(messageRecipient);
    }

    public MessageRecipient updateMessageRecipient(MessageRecipient
messageRecipient) {
        return em.merge(messageRecipient);
    }

    public void createNewMessageRecipient(MessageRecipient messageRecipient) {
        em.persist(messageRecipient);
    }

    public MessageRecipient createNewMessageRecipient(int idMessage, int
idRecipient) {
        MessageRecipient messageRecipient = new MessageRecipient(idMessage,
idRecipient);
        em.persist(messageRecipient);
        return messageRecipient;
    }

    public List<MessageRecipient> findUserMessageRecipients(int idUser) {
        return em.createQuery("select i from MessageRecipient i where i =
:idUser", MessageRecipient.class).setParameter("idUser", idUser).getResultList();
    }

    public List<User> findAllUsersDialogues(int idUser){
        List users = em.createNativeQuery("select User.* from user where id_user
in (select id_user_recipient from message_recipient INNER JOIN message m on
message_recipient.id_message = m.id_message where id_user_sender = " + idUser +
");", User.class).getResultList();
        users.addAll(em.createNativeQuery("select User.* from user where id_user
in (select id_user_sender from message INNER JOIN message_recipient on
message.id_message = message_recipient.id_message where id_user_recipient = " +
idUser + ");", User.class).getResultList());
        return (List) users.stream().distinct().collect(Collectors.toList());
    }
}

```

PostDAO.java

```

package dao;

import entities.Image;
import entities.Post;

import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import java.sql.Timestamp;

```

```

import java.util.List;

@Stateless
public class PostDAO {

    @PersistenceContext
    EntityManager em;

    public List<Post> findAll() {
        return em.createQuery("select i from Post i", Post.class).getResultList();
    }

    public Post findById(int postId) {
        return em.find(Post.class, postId);
    }

    public void removePost(int postId) {
        Image image = em.find(Image.class, postId);
        em.remove(image);
    }

    public Post updatePost(Post post) {
        return em.merge(post);
    }

    public void createNewPost(Post post) {
        em.persist(post);
    }

    public Post createNewPost(String title, String text, String filename,
Timestamp publish, int idUser) {
        Post post = new Post(title, text, publish, filename, idUser);
        em.persist(post);
        return post;
    }

    public List<Post> findUserPosts(int idUser) {
        return em.createQuery("select i from Post i where i.idUserPost = :idUser",
Post.class).setParameter("idUser", idUser).getResultList();
    }

    public void plusView(int idPost){
        em.createQuery("UPDATE Post i set i.viewsCounter = i.viewsCounter + 1
where i.idPost = :idPost", Image.class).setParameter("idPost",
idPost).executeUpdate();
        em.merge(findById(idPost));
    }

    public List<Post> findByTitleOrText(String text) {
        return em.createQuery("SELECT i FROM Post i WHERE i.postTitle like :text
OR" +
            " i.postText like :text", Post.class).setParameter("text", "%" +
text + "%").getResultList();
    }

    public Post findLatestPost(int idUser) {
        return em.createQuery("select i from Post i where i.idUserPost =
:idUser ORDER BY i.publishDate DESC", Post.class).setParameter("idUser",
idUser).getResultList().get(0);
    }
}

```



```
    }  
}
```

UserBean.java

```
package beans;  
  
import entities.Image;  
import entities.User;  
  
import java.util.List;  
  
public class UserBean {  
    public UserBean() {  
    }  
  
    public UserBean(List<User> users) {  
        this.users = users;  
    }  
  
    private List<User> users;  
  
    public List<User> getUsers() {  
        return users;  
    }  
  
    public void setUsers(List<User> users) {  
        this.users = users;  
    }  
}
```

ImageBean.java

```
package beans;  
  
import entities.Image;  
  
import java.util.List;  
  
public class ImageBean {  
    public ImageBean() {  
    }  
  
    public ImageBean(List<Image> images) {  
        this.images = images;  
    }  
  
    private List<Image> images;  
  
    public List<Image> getImages() {  
        return images;  
    }  
  
    public void setImages(List<Image> images) {  
        this.images = images;  
    }  
}
```

PostBean.java

```

package beans;

import entities.Post;

import java.util.List;

public class PostBean {
    public PostBean() {
    }

    public PostBean(List<Post> posts) {
        this.posts = posts;
    }

    private List<Post> posts;

    public List<Post> getPosts() {
        return posts;
    }

    public void setPosts(List<Post> posts) {
        this.posts = posts;
    }
}

```

UserServlet

```

package servlets;

import beans.ImageBean;
import beans.PostBean;
import dao.ImageDAO;
import dao.PostDAO;
import dao.UserDAO;
import entities.Post;
import entities.User;

import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet(name = "UserServlet", urlPatterns = {"/profile"})
public class UserServlet extends HttpServlet {

    @EJB
    UserDAO userDAO;
    @EJB
    ImageDAO imageDAO;
    @EJB
    PostDAO postDAO;

    protected void processRequest(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        User currentUser;
    }
}

```

```

        try {
            currentUser =
userDAO.findById(Integer.parseInt(request.getParameter("profileID")));
        }
        catch (Exception e){
            currentUser =
userDAO.findById(Integer.parseInt(String.valueOf(request.getSession().getAttribute
("userID"))));
        }
        ImageBean imageBean = new
ImageBean(imageDAO.findUserImages(currentUser.getIdUser()));
        request.setAttribute("imageBean", imageBean);
        request.getSession().setAttribute("postLatest", new Post());
        request.getSession().setAttribute("text_preview", "");
        try {
            postDAO.findLatestPost(currentUser.getIdUser());
            request.getSession().setAttribute("postLatest",
postDAO.findLatestPost(currentUser.getIdUser()));
            String textPrev =
postDAO.findLatestPost(currentUser.getIdUser()).getPostText();
            if (textPrev.length() > 120) {
                textPrev = textPrev.substring(0, 120) + "...";
            }
            request.getSession().setAttribute("text_preview", textPrev);
        }
        catch (Exception ignored) {
        }
        request.setAttribute("user", currentUser);

getServletContext().getRequestDispatcher("/profile.jsp").forward(request,
response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        processRequest(request, response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        processRequest(request, response);
    }
}

```

UserDialoguesServlet

```

package servlets;

import beans.ImageBean;
import beans.PostBean;
import beans.UserBean;
import dao.*;
import entities.Message;
import entities.User;

import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;

```

```

import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.IOException;
import java.sql.Timestamp;

@WebServlet(name = "UserDialoguesServlet", urlPatterns = {"/UserDialogues"})
public class UserDialoguesServlet extends HttpServlet {

    @EJB
    UserDAO userDAO;
    @EJB
    MessageDAO messageDAO;
    @EJB
    MessageRecipientDAO messageRecipientDAO;

    protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException {
        UserBean userBean = new
UserBean(messageRecipientDAO.findAllUsersDialogues((Integer)
request.getSession().getAttribute("userID")));
        request.setAttribute("userBean", userBean);
        request.getRequestDispatcher("/user-dialogues.jsp").forward(request,
response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        processRequest(request, response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        processRequest(request, response);
    }
}

```

UploadServlet

```

package servlets;

import beans.ImageCategoryBean;
import dao.ImageCategoryDAO;
import dao.UserDAO;
import entities.User;

import javax.ejb.EJB;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.annotation.MultipartConfig;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
import java.io.*;
import java.util.logging.Level;

import static com.sun.xml.internal.ws.spi.db.BindingContextFactory.LOGGER;

@WebServlet(name = "UploadServlet", urlPatterns = {"/Upload"})

```

```

@MultipartConfig
public class UploadServlet extends HttpServlet {

    @EJB
    UserDAO userDAO;

    @EJB
    ImageCategoryDAO imageCategoryDAO;

    protected void processRequest(HttpServletRequest request,
                                   HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");

        HttpSession session = request.getSession();
        String type = (String) session.getAttribute("type");

        User user = userDAO.findById((int) session.getAttribute("userID"));
        String userDir = user.getLogin();

        final String path = "C:\\Users\\Zuko\\IdeaProjects\\Gallery1\\web\\" +
type + "\\\" + userDir;
        final Part filePart = request.getPart("file");
        final String fileName = getFileName(filePart);

        File dir = new File(path);
        if (!dir.exists())
            dir.mkdir();

        OutputStream out = null;
        InputStream filecontent = null;
        final PrintWriter writer = response.getWriter();

        try {
            out = new FileOutputStream(new File(path + File.separator
                + fileName));
            filecontent = filePart.getInputStream();

            int read = 0;
            final byte[] bytes = new byte[1024];

            while ((read = filecontent.read(bytes)) != -1) {
                out.write(bytes, 0, read);
            }
            if (type.equals("images")) {
                request.setAttribute("filename", userDir + File.separator +
fileName);
                ImageCategoryBean imageCategoryBean = new
ImageCategoryBean(imageCategoryDAO.findAll());
                request.setAttribute("imageCategoryBean", imageCategoryBean);
                request.getRequestDispatcher("/upload-info.jsp").forward(request,
response);
            }
            else if (type.equals("avatars")){
                userDAO.addAvatar(user, userDir + File.separator + fileName);
                request.getRequestDispatcher("/profile").forward(request,
response);
            }
        }
    }
}

```

```

        else if (type.equals("posts")){
            request.setAttribute("filename", userDir + File.separator +
fileName);
            request.setAttribute("postImageAdded", 1);
            request.getRequestDispatcher("/upload-post.jsp").forward(request,
response);
        }
        else request.getRequestDispatcher("/profile").forward(request,
response);
    } catch (FileNotFoundException fne) {
        request.setAttribute("message", "You did not specify a file to
upload.");
        request.getRequestDispatcher("/error.jsp").forward(request, response);
    } finally {
        if (out != null) {
            out.close();
        }
        if (filecontent != null) {
            filecontent.close();
        }
        if (writer != null) {
            writer.close();
        }
    }
}

private String getFileName(final Part part) {
    final String partHeader = part.getHeader("content-disposition");
    LOGGER.log(Level.INFO, "Part Header = {0}", partHeader);
    for (String content : part.getHeader("content-disposition").split(";")) {
        if (content.trim().startsWith("filename")) {
            return content.substring(
                content.indexOf('=') + 1).trim().replace("\"", "");
        }
    }
    return null;
}

protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    processRequest(request, response);
}

protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    processRequest(request, response);
}
}

```

ImageInfoUploadServlet

```

package servlets;

import dao.ImageDAO;
import entities.Image;

import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;

```

```

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.IOException;
import java.sql.Date;

@WebServlet(name = "ImageInfoUploadServlet", urlPatterns = {"/ImageInfoUpload"})
public class ImageInfoUploadServlet extends HttpServlet {

    @EJB
    ImageDAO imageDAO;

    protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException {
        HttpSession session = request.getSession();
        imageDAO.createNewImage(request.getParameter("title"),
            request.getParameter("description"),
            request.getParameter("filename"), new
Date(System.currentTimeMillis()),
            Integer.parseInt(request.getParameter("category")), (Integer)
session.getAttribute("userID"));

        getServletContext().getRequestDispatcher("/profile").forward(request,
response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        processRequest(request, response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        processRequest(request, response);
    }
}

```

RegistrationServlet

```

package servlets;

import dao.UserDAO;
import entities.User;

import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.IOException;
import java.sql.Date;

@WebServlet(name = "RegistrationServlet", urlPatterns = {"/Registration"})
public class RegistrationServlet extends HttpServlet {

```

```

@EJB
UserDAO userDAO;

    protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException {
    try {
        userDAO.addUser(
            request.getParameter("login"),
            request.getParameter("password"),
            request.getParameter("name"),
            request.getParameter("surname"),
            new Date(System.currentTimeMillis()),
            request.getParameter("description"),
            request.getParameter("country"),
            request.getParameter("city"),
            request.getParameter("email"),
            ""
        );

        getServletContext().getRequestDispatcher("/index.jsp").forward(request, response);
    }
    catch (Exception e){
        request.setAttribute("message", "The login already exist or you have
not entered login and/or password");
        request.getRequestDispatcher("/error.jsp").forward(request, response);
    }
}

    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        processRequest(request, response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        processRequest(request, response);
    }
}

```

PostViewServlet

```

package servlets;

import beans.ImageCommentsBean;
import beans.PostCommentsBean;
import beans.UserBean;
import dao.*;
import entities.*;

import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.ArrayList;

```



```

import java.util.Collections;
import java.util.List;
import java.util.Vector;

@WebServlet(name = "PostViewServlet", urlPatterns = {"/PostView"})
public class PostViewServlet extends HttpServlet {

    @EJB
    PostDAO postDAO;
    @EJB
    UserDAO userDAO;
    @EJB
    PostLikeDAO postLikeDAO;
    @EJB
    PostCommentDAO postCommentDAO;

    protected void processRequest(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        Post post =
postDAO.findById(Integer.parseInt(request.getParameter("postID")));
        User author =
userDAO.findById(postDAO.findById(post.getIdPost()).getIdUserPost());
        postDAO.plusView(post.getIdPost());
        int sessionUser = (int) request.getSession().getAttribute("userID");
        if (sessionUser != 0)
            request.setAttribute("like",
postLikeDAO.checkPostLike(post.getIdPost(), sessionUser));
        request.setAttribute("userCommentsBean", new
UserBean(postCommentDAO.findUsersComments(post.getIdPost())));
        request.setAttribute("postCountLikes",
postLikeDAO.findLikesCount(post.getIdPost()));
        request.setAttribute("author", author);
        request.setAttribute("post", post);
        request.setAttribute("postCommentsBean", new
PostCommentsBean(postCommentDAO.findPostComments(post.getIdPost())));
        request.getRequestDispatcher("/post-view.jsp").forward(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        processRequest(request, response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        processRequest(request, response);
    }
}

```

MainGallery

```

package servlets;

import beans.ImageBean;
import beans.ImageCategoryBean;
import dao.ImageCategoryDAO;
import dao.ImageDAO;

import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;

```

```

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet(name = "MainGalleryServlet", urlPatterns = {"/MainGallery"})
public class MainGalleryServlet extends HttpServlet {

    @EJB
    ImageCategoryDAO imageCategoryDAO;
    @EJB
    ImageDAO imageDAO;

    protected void processRequest(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException {
        ImageBean imageBean = new ImageBean(imageDAO.findAll());
        request.setAttribute("imageBean", imageBean);
        ImageCategoryBean imageCategoryBean = new
ImageCategoryBean(imageCategoryDAO.findAll());
        request.setAttribute("imageCategoryBean", imageCategoryBean);
        getServletContext().getRequestDispatcher("/main-
gallery.jsp").forward(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        processRequest(request, response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        processRequest(request, response);
    }
}

```

ImageViewServlet

```

package servlets;

import beans.ImageCommentsBean;
import beans.UserBean;
import dao.*;
import entities.Image;
import entities.ImageCategory;
import entities.ImageCommentary;
import entities.User;

import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Vector;

@WebServlet(name = "ImageViewServlet", urlPatterns = {"/ImageView"})

```

```

public class ImageViewServlet extends HttpServlet {

    @EJB
    ImageDAO imageDAO;
    @EJB
    UserDAO userDAO;
    @EJB
    ImageCategoryDAO imageCategoryDAO;
    @EJB
    ImageCommentDAO imageCommentDAO;
    @EJB
    ImageLikeDAO imageLikeDAO;

    protected void processRequest(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        Image image =
imageDAO.findById(Integer.parseInt(request.getParameter("imageID")));
        User author =
userDAO.findById(imageDAO.findById(image.getIdImage()).getIdUserImage());
        ImageCategory category = imageCategoryDAO.findById(image.getIdCategory());
        ImageCommentsBean imageCommentsBean = new
ImageCommentsBean(imageCommentDAO.findImageComments(image.getIdImage()));
        imageDAO.plusView(image.getIdImage());
        int sessionUser = (int) request.getSession().getAttribute("userID");
        if (sessionUser != 0)
            request.setAttribute("like",
imageLikeDAO.checkImageLike(image.getIdImage(), sessionUser));
        request.setAttribute("userCommentsBean", new
UserBean(imageCommentDAO.findUsersComments(image.getIdImage())));
        request.setAttribute("imageCountLikes",
imageLikeDAO.findLikesCount(image.getIdImage()));
        request.setAttribute("image", image);
        request.setAttribute("author", author);
        request.setAttribute("category", category);
        request.setAttribute("imageCommentsBean", imageCommentsBean);
        request.getRequestDispatcher("/image-view.jsp").forward(request,
response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        processRequest(request, response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        processRequest(request, response);
    }
}

```