ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка торговельної площадки 3D-моделей “Grifon” з використанням JavaScript, Python, HTML, CSS та фреймворків React, Django»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

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

_____ Олексій ЗДОРЕНКО
(підпис)

Виконав: здобувач вищої освіти групи ПД-42

Олексій ЗДОРЕНКО

Керівник: Оксана ЗОЛОТУХІНА
к.т.н., доцент

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здоренку Олексію Олександровичу

1. Тема кваліфікаційної роботи: «Розробка торговельної площадки 3D-моделей “Grifon” з використанням JavaScript, Python, HTML, CSS та фреймворків React, Django.»
керівник кваліфікаційної роботи к.т.н. доц., Оксана ЗОЛОТУХІНА,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи:
   3.1 Офіційна документація JavaScript.
   3.2 Офіційна документація PostgreSQL.
   3.3 Офіційна документація Python.
   3.4 Науково-технічна література.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

4.1 Вступ.
4.2 Аналіз існуючих рішень.
4.3 Розробка веб-сервісу.
4.4 Тестування та аналіз результатів.
4.5 Висновки.

5. Перелік графічного матеріалу: *презентація*
5.1 Титульний слайд.
5.2 Мета, об’єкт та предмет дослідження.
5.3 Задачі дипломної роботи.
5.4 Аналіз аналогів.
5.5 Вимоги до програмного забезпечення.
5.6 Програмні засоби реалізації.
5.7 Архітектура системи
5.8 Діаграма варіантів використання.
5.9 Діаграма User Flow.
5.9 Схема баз даних.
5.10 Екранні форми.
5.11 Апробація результатів дослідження.
5.12 Висновок.

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Огляд та вибір інструментів для створення веб-сервісу.	15.03-21.03.2024	
4	Проектування веб-сервісу.	23.03-30.03.2024	
5	Програмна реалізація веб-сервісу.	01.04-20.04.2024	
6	Тестування розробленого веб-сервісу.	20.04-24.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувач(ка) вищої освіти _____ Олексій ЗДОРЕНКО
(підпис)

Керівник кваліфікаційної роботи _____ Оксана ЗОЛОТУХІНА
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 79 стор., 8 табл., 18 рис., 19 джерел.

Мета роботи – спростити процес купівлі-продажу 3D-моделей за рахунок використання торговельної площадки 3D-моделей “Grifon”

Об’єкт дослідження – процес купівлі-продажу 3D-моделей.

Предмет дослідження – програмне забезпечення для купівлі-продажу 3D-моделей.

Короткий зміст роботи: В даній дипломній роботі було розроблено алгоритм роботи веб-застосунку та програмно реалізовані ключові функціональні можливості, зокрема: Перегляд каталогу 3D моделей, їх пошук, придбання та виставлення на продаж своїх моделей, додавання коментарів до виставлених модей, да відслідковування їх оновлень. Проведено функціональне тестування та тестування безпеки додатку. В роботі використано JavaScript, Python, HTML, CSS та фреймворки React, Django. Сферою використання застосунку є професійні дизайнери та художники, які шукають або пропонують 3D-контент.

КЛЮЧОВІ СЛОВА: ТОРГІВЕЛЬНА ПРОЩАДКА, 3D МОДЕЛЬ, HTML, CSS, WEB-САЙТ, JS, PYTHON.

ЗМІСТ

ВСТУП

3D моделі стали невід'ємною частиною сучасного світу, знаходячи застосування у різноманітних галузях і сферах діяльності. Вони використовуються в архітектурі для створення точних візуалізацій будівель і інтер'єрів, що допомагає архітекторам і дизайнерам демонструвати свої проекти клієнтам та вносити необхідні корективи на ранніх етапах. Крім того, візуалізація в 3D дозволяє більш реалістично представити майбутні об'єкти, що сприяє кращому розумінню їх функціональності та естетичних якостей.

У сфері розваг 3D моделі активно використовуються у створенні комп'ютерних ігор та анімаційних фільмів. Високоякісні моделі персонажів, оточення та реквізиту є ключовими елементами, що забезпечують захоплюючий і реалістичний досвід для гравців і глядачів. Віртуальна реальність (VR) та доповнена реальність (AR) також базуються на 3D моделях, дозволяючи користувачам взаємодіяти з цифровими об'єктами в реальному світі.

У промисловості 3D моделювання використовується для розробки і тестування нових продуктів. Інженери та дизайнери створюють прототипи та перевіряють їх на відповідність технічним вимогам та ергономічним стандартам. Це значно скорочує витрати на розробку і випробування, а також прискорює процес виведення продукту на ринок.

З огляду на широкий спектр застосувань 3D моделей, виникає очевидний попит на створення платформи, де фахівці різних галузей можуть продавати свої 3D моделі. Торговельний майданчик для 3D моделей дозволить дизайнерам, архітекторам, інженерам та іншим професіоналам легко обмінюватися своїми розробками та забезпечувати доступ до високоякісних ресурсів для своїх проектів.

Об’єкт дослідження – процес купівлі-продажу 3D-моделей.

Предмет дослідження – програмне забезпечення для купівлі-продажу 3D-моделей.

Мета роботи – спростити процес купівлі-продажу 3D-моделей за рахунок використання торговельної площадки 3D-моделей “Grifon”.

Основними завданнями цієї роботи є:

1. Провести аналіз ринку торговельної індустрії 3D моделей.
2. Оцінити веб-платформи конкурентів та їх функціонал для визначення сильних та слабких сторін.
3. Визначити вимоги до веб-застосунку з урахуванням його функціональності та зручності використання.
4. Розглянути можливості використання різних технологій та фреймворків для розробки веб-застосунку.
5. Розробити веб-застосунок для реалізації продажу 3D моделей.
6. Виконати тестування зручності та відповідності вимогам користувацького інтерфейсу.

1. АНАЛІЗ ТА ДОСЛІДЖЕННЯ ІСНУЮЧИХ ВЕБ-ЗАСТОСУНКІВ ДЛЯ ПРОДАЖУ 3D МОДЕЛЕЙ.

1.1 Поняття 3D модель

3D модель — це цифрове представлення об'єкта в тривимірному просторі, створене за допомогою спеціалізованого програмного забезпечення. Вона відображає геометричні та візуальні характеристики реальних або уявних об'єктів, дозволяючи користувачам бачити, маніпулювати та взаємодіяти з ними в цифровому середовищі. 3D моделі широко використовуються в різних галузях, включаючи архітектуру, дизайн, анімацію, відеоігри, медицину та виробництво.

3D модель складається з ряду ключових компонентів. По-перше, це вершини (Vertices) — базові точки в просторі, які визначають форму об'єкта. Вершини з'єднуються між собою ребрами (Edges), утворюючи полігони (Polygons), зазвичай трикутники або квадрати. Полігони, у свою чергу, утворюють поверхню об'єкта (PolyMesh), як зображено на рисунку 1.1. Крім геометрії, 3D модель може включати текстури (Textures) — зображення, що накладаються на поверхню моделі для додання деталей, таких як кольори, візерунки або матеріали.

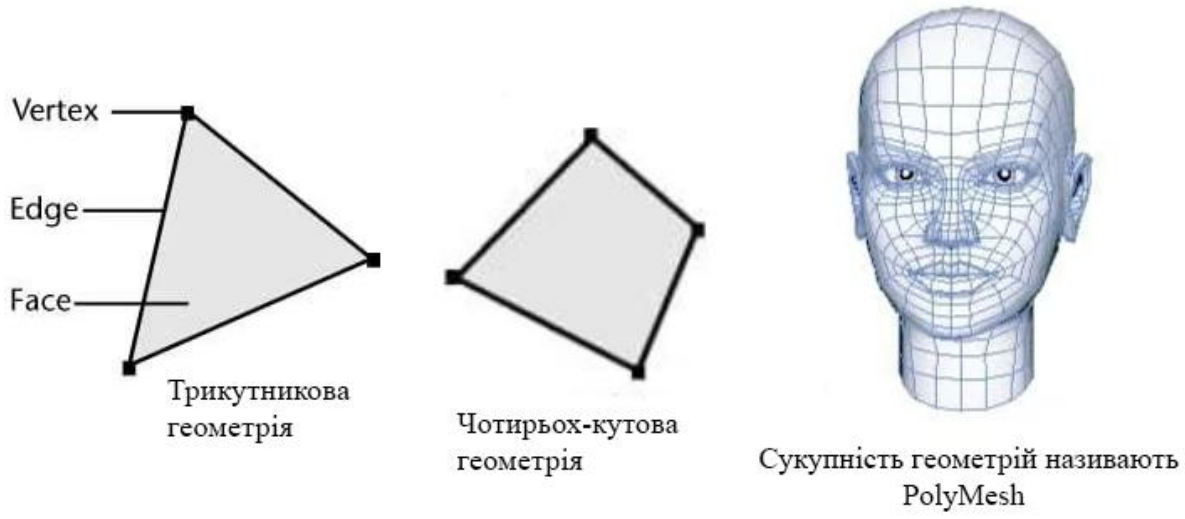


Рис. 1.1 Будова 3D моделі

Для корректного відображення Текстури для 3D моделей створюють UV-розгортку. UV-розгортка (UV mapping) — це процес перетворення тривимірної поверхні об'єкта на двовимірну площину, що дозволяє ефективно накладати текстури. Кожна вершина 3D моделі отримує відповідні координати (U і V) на текстурному зображенні, що дозволяє коректно розміщувати зображення на поверхні об'єкта. UV-розгортка є критично важливим етапом у текстуруванні, оскільки від її якості залежить, наскільки точно текстура відповідатиме геометрії моделі. Погана UV-розгортка може призвести до спотворень, розтягнень або неправильного відображення текстур.

Відображення 3D моделі на екрані комп'ютера здійснюється шляхом проекції її тривимірних координат на двовимірну площину екрану. Це досягається завдяки використанню графічних процесорів (GPU) та спеціальних алгоритмів рендерингу, які обчислюють видимість, освітлення та текстурування об'єкта з певної точки зору (рис. 1.2).

Рис. 1.2 Рендеринг 3D моделі

Рендеринг 3D моделей може здійснюватися за допомогою різних технологій, таких як растеризація (Rasterization) та трасування променів (Ray tracing). Растеризація є швидким і ефективним методом, який використовується в реальному часі в комп'ютерних іграх та інтерактивних застосунках. Трасування променів, навпаки, є більш ресурсомістким, але забезпечує високу якість

зображень шляхом точного обчислення світлових променів і їх взаємодії з поверхнями. Вибір технології рендерингу залежить від конкретних вимог до якості та продуктивності, а також від формату збереження моделі, що визначає, яку інформацію і в якому вигляді буде зберігати модель.

Існує кілька найпоширеніших форматів для збереження 3D моделей, кожен з яких має свої переваги та недоліки. Формат OBJ (Wavefront Object) є одним з найбільш універсальних і підтримується багатьма програмами. Він дозволяє зберігати інформацію про геометрію, текстури та матеріали об'єкта. Формат FBX (Filmbox) широко використовується у виробництві анімації та відеоігор завдяки його здатності зберігати складні сцени, включаючи анімацію, кістки (bones) та інші дані. Формат STL (Stereolithography) є стандартом для 3D друку, оскільки він фокусується на геометрії без текстур і матеріалів. Крім того, формат COLLADA (Collaborative Design Activity) використовується для обміну даними між різними програмами та підтримує широкий спектр інформації, включаючи фізичні властивості об'єктів.

1.2 Актуальність використання 3D моделей.

У сучасному світі 3D моделі стрімко набувають все більшої популярності та стають невід'ємною частиною багатьох сфер людської діяльності. Ці цифрові представлення тривимірних об'єктів відкривають безліч нових можливостей та революціонізують різні аспекти нашого життя. 3D моделі знаходять застосування в широкому спектрі галузей, від промисловості та медицини до розваг та освіти.

- *Застосування у ігровій індустрії.* 3D моделі стали невід'ємною частиною сучасних відеоігор, адже вони роблять їх більш реалістичними, емоційними та захоплюючими. Завдяки 3D моделям розробники створюють персонажів з вражаючою деталізацією, які мають унікальний зовнішній вигляд та особистість. Деталізовані та реалістичні світи, створені за допомогою 3D моделей, занурюють гравців у атмосферу гри, роблячи їх частиною

віртуального світу. Спецефекти, такі як вибухи, вогнені кулі та падіння, додають динаміки та емоційності ігровому процесу. А плавні анімаційні рухи персонажів роблять їх більш живими та реалістичними, що покращує загальне враження від гри.

- *Застосування у кінематографі*. У сучасному кінематографі використання 3D моделей є не лише важливим елементом, але й ключовою складовою для створення вражаючих візуальних ефектів та реалістичних образів. Ця технологія стала необхідністю для кіноіндустрії, що відкрила нові горизонти для режисерів та візуальних ефектів. Одним з основних застосувань 3D моделей у кінематографі є створення віртуальних світів та локацій, які були б неможливі для зйомок у реальному житті. Також 3D моделі широко використовуються для створення та анімації персонажів у кінофільмах, надаючи їм реалістичний вигляд та динаміку. Вплив 3D технологій на розвиток кіноіндустрії виявився значним, розширивши можливості для створення фільмів та створивши нові можливості для розвитку кінематографічної сфери.
- *Застосування в архітектурі та будівництві.* Використання 3D моделей в архітектурі та будівництві має різноманітні переваги. По-перше, вони дозволяють проводити віртуальне проектування, створюючи деталізовані та реалістичні віртуальні моделі будівель і споруд. Це дозволяє архітекторам та інженерам докладно розглядати та аналізувати кожний аспект проекту перед його фізичним втіленням. Далі, 3D моделі допомагають створювати вражаючі візуалізації архітектурних проектів, що полегшує розуміння та оцінку концепції будівлі клієнтами, інвесторами та іншими зацікавленими сторонами. Крім того, за допомогою 3D моделей можна створювати вражаючі віртуальні тури та презентації будівельних об'єктів, що дозволяє оглянути об'єкт з будь-якого кута зору та отримати реалістичне уявлення про його масштаб та архітектурні особливості.

- *Застосування в електроніці та інженерії.* 3D моделі революціонізують сферу електроніки, роблячи процес розробки, виробництва та презентації проектів більш ефективним та наочним. Завдяки 3D моделям інженери можуть створювати віртуальні прототипи електронних пристроїв, економити час та ресурси на фізичному прототипуванні. 3D моделювання також дозволяє моделювати та аналізувати складні системи, передбачаючи їхню поведінку та працездатність. Виробництво електронних компонентів з використанням 3D моделей стає більш оптимізованим, зменшуючи витрати на матеріали та прискорюючи процес. Окрім того, 3D моделі використовуються для візуалізації та презентації проектів, роблячи їх більш зрозумілими та наочними для зацікавлених сторін.

Тому використання 3D моделей в різних сферах, таких як ігрова індустрія, кінематограф, архітектура та будівництво, електроніка та інженерія, відображає надзвичайно широкі можливості цієї технології. У всіх цих галузях 3D моделі стали не тільки важливим інструментом, але й ключовою складовою процесу розробки та виробництва. Їхня роль у створенні реалістичних образів, віртуальних світів, анімації персонажів та спеціальних ефектів визнана як незамінна. 3D моделі дозволяють реалізувати найбільш сміливі творчі ідеї від режисерів та розробників, забезпечуючи неймовірну емоційність та вражаючу реалістичність у всіх аспектах творчого процесу. Таким чином, включення різноманітних сфер використання 3D моделей на розробленій торговельній платформі буде важливим кроком у забезпеченні доступу до цієї потужної технології для широкого кола користувачів та сприятиме подальшому розвитку цих галузей.

1.3 Аналіз та загальна характеристика веб-застосунків для продажу 3D моделей

Веб сервіс для продажу 3D моделей призначений для створення платформи, де дизайнери, художники та розробники можуть продавати свої тривимірні моделі, а покупці можуть легко знаходити та купувати їх для використання у своїх проектах. Далі буде розглянуто основні характеристики веб-застосунку для продажу 3D моделей:

1. Каталог моделей: Веб-застосунок має пропонувати широкий каталог 3D моделей, що охоплює різні категорії, такі як архітектура, транспорт, персонажі, продукти, одяг та багато інших.
2. Цінова політика: Ціни на моделі мають варіюватися від безкоштовних до дуже дорогих, залежно від складності, якості та ексклюзивності моделі. Багато сервісів також пропонують знижки та акції для залучення покупців.
3. Формати файлів: Платформа має підтримувати кілька форматів файлів, таких як OBJ, STL, FBX, 3DS, і т.д. Це дозволяє користувачам вибирати найбільш підходящий формат для їхнього програмного забезпечення та цілей.
4. Рейтинги та відгуки: Платформа повинна надавати можливість залишати відгуки та оцінки моделей, що допоможе користувачам приймати обґрунтовані рішення про покупку.
5. Реєстрація та авторизація користувача: Платформа має мати систему ідентифікації користувача для подальшої можливості продажу власних 3D моделей.
6. Каталогізація та індексація: Платформи повинні мати ефективні алгоритми для індексації та пошуку моделей, забезпечуючи легкий доступ до потрібних ресурсів за категоріями, ключовими словами, авторами тощо.
7. Інструменти для завантаження: Для продавців веб-сервіси пропонують зручні інструменти для завантаження та управління своїми моделями,

включаючи опис, теги, ціну, прев'ю та інші метадані, що полегшує процес продажу.

1.4 Аналіз існуючих веб-сервісів продажу 3D моделей.

Для аналізу було взято 3 відомі торговельні площадки 3D моделей : Turbosquid, Quixel Megascans, та SketchFab.

Turbosquid - це одна з найбільших і найпопулярніших онлайн-платформ для обміну та продажу 3D моделей (Рис. 1.3). Створена у 2000 році, Turbosquid вирізняється своєю величезною базою даних, що налічує мільйони різноманітних 3D моделей.Але конкуренція на платформі може бути значною, оскільки там представлено велику кількість художників і дизайнерів, що може ускладнити продаж моделей для новачків та знизити їхній потенційний прибуток. Щодо якості контролю, навіть з наявністю процесу перевірки якості, деякі моделі з низькою якістю все ще можуть потрапити на платформу, що може призвести до незадоволення покупців. Сприяє цьому не зручний та не деталье представлення продаваємих 3D моделей, що може створити помилкові очікування у покупців щодо її фактичних характеристик та якості.

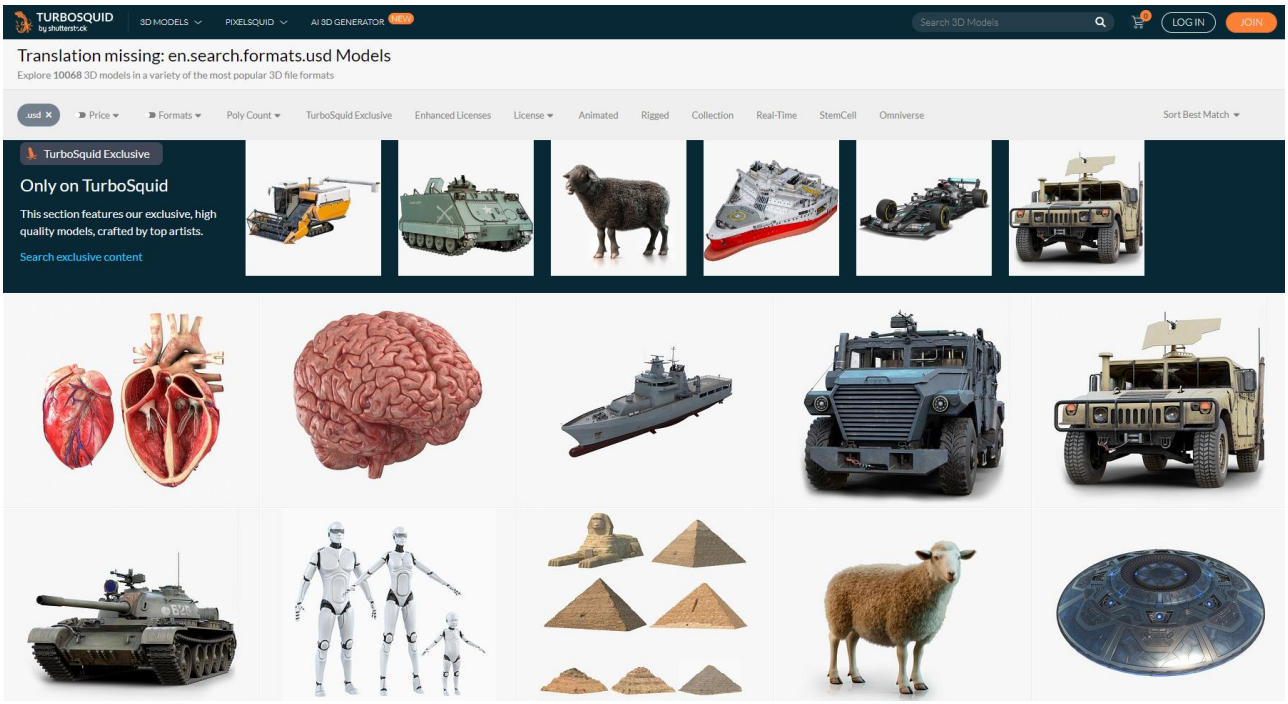


Рис. 1.3 Turbosquid.

Quixel Megascans - торгівельна площадка, заснована компанією Quixel у 2011 році, належить до провідних у сфері 3D-сканування та має значну бібліотеку ресурсів, призначених для використання у фільмах та іграх (Рис. 1.4). Після того, як Epic Games придбала Quixel у 2019 році, весь контент бібліотеки став доступним безкоштовно у рамках рушія Unreal Engine. Однак використання ресурсів у зовнішніх програмах передбачає певні витрати. Користувачі мають можливість обирати роздільну здатність текстури і тип текстури (альбедо, нормальна, зміщення і т. д.). Megascan також пропонує декілька рівнів деталізації (LOD) для кожного 3D-ресурсу. Важливо зауважити, що бібліотека Megascan безкоштовно доступна всередині Unreal Engine, але для використання у зовнішніх програмах доступні різні тарифні плани. Клієнти мають можливість обирати план, щомісяця отримуючи бали за активи. Наприклад, особисті та незалежні плани надають 90 балів на місяць. Повернення коштів пов'язані з підписками та надаються через безпечні методи оплати.

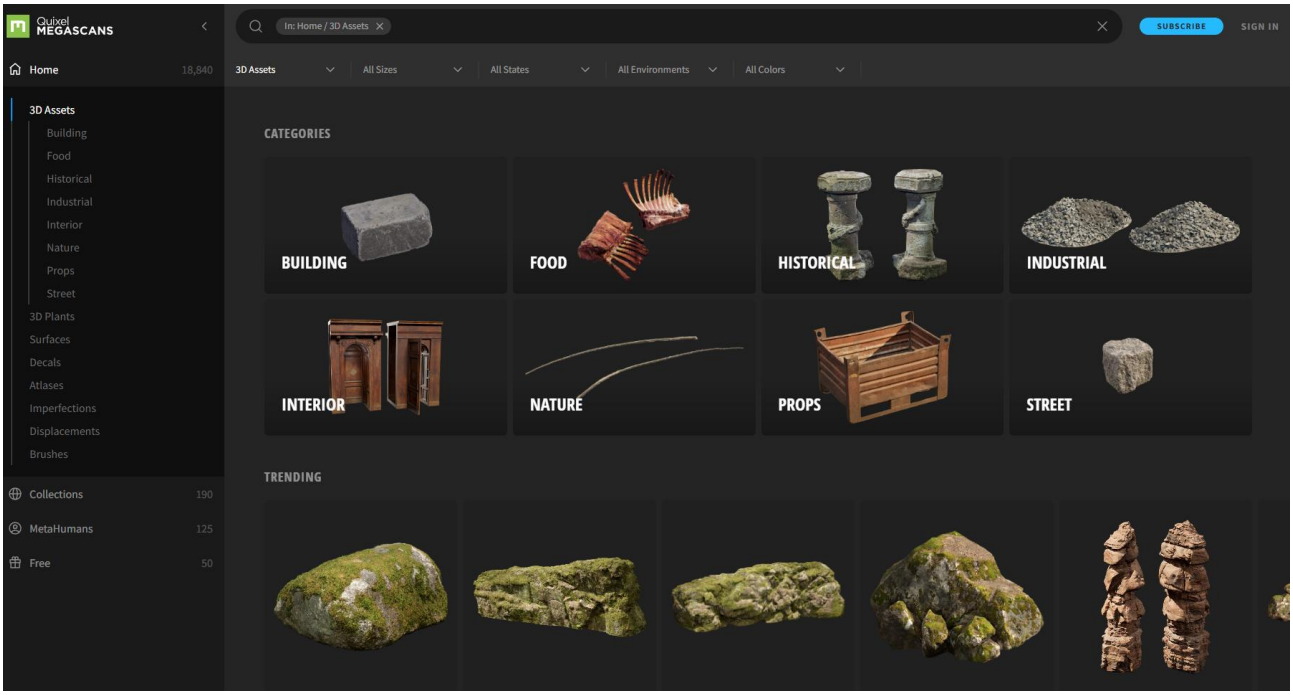


Рис. 1.4 Quixel Megascans.

SketchFab - було засновано в 2011 році як платформу для обміну 3D-моделями, а її ринок було запущено в 2018 році. Його популярність зростала, і наразі має шість мільйонів користувачів.це інноваційна та динамічно розвиваючася онлайн-платформа для обміну та продажу 3D моделей (Рис. 1.5). На відміну від інших конкурентів, SketchFab вирізняється унікальною можливістю, яка забезпечує користувачам прямий та миттєвий доступ до перегляду 3D об'єктів у просторі, просто на веб-сайті. Ця можливість реалізована в реальному часі, що означає, що користувачі можуть взаємодіяти з об'єктами, спостерігати за ними з усіх кутів та оцінювати їхні розміри та деталі безпосередньо на онлайн-платформі. Платформа підтримує віртуальну та розширену реальності, що дозволяє користувачам переглядати 3D моделі у віртуальному середовищі або у своєму реальному оточенні.

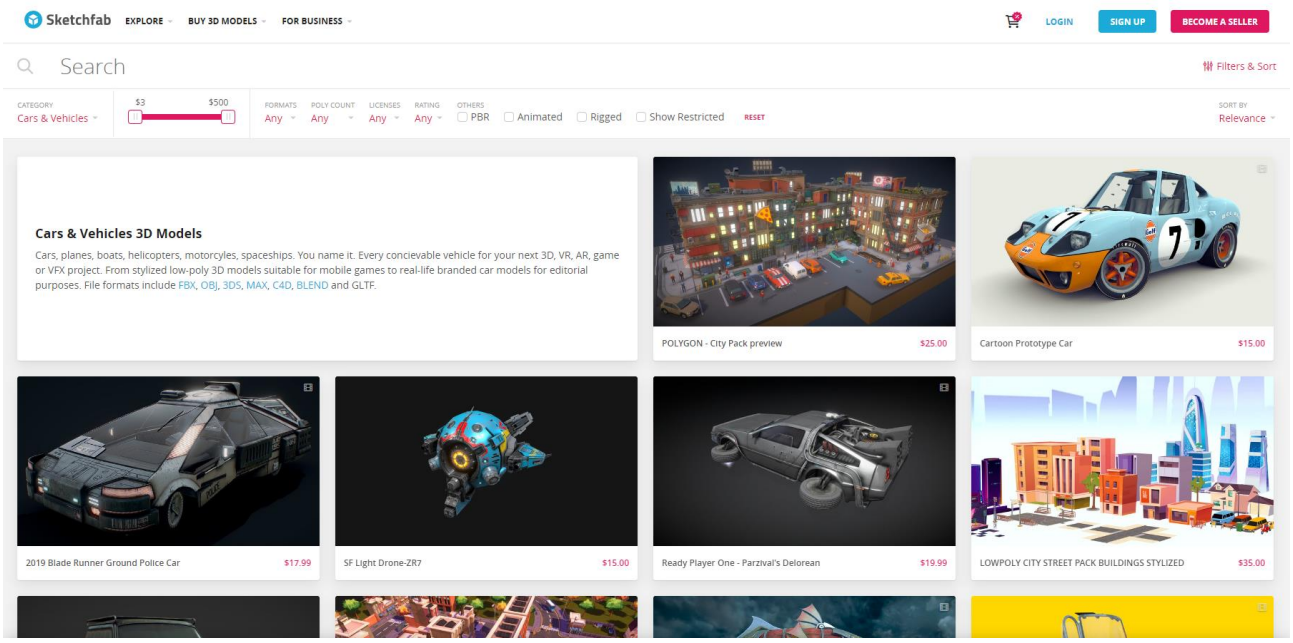


Рис. 1.5 Sketchfab

Виконавши аналіз платформ продажу 3D моделей, було сформульовано таблицю критеріїв, по якій зрівнялись вищеописані сайти:

Таблиця 1.1

Порівняння існуючих аналогів

Критерій порівняння	Turbosquid	Quixel Megascans	SketchFab	GRIFON
Відстежування вподобаних робіт авторів.	-	+	+	+
Підтримка та оновлення вже проданих 3D моделей	-	+	-	+
Можливість написання рецензій та коментарів	-	+	+	+
Не прив'язка до платформ візуалізації 3D моделей	+	-	+	+
Українська локалізація	-	-	-	+

Аналізуючи дані, представлені в таблиці порівняння функціональних можливостей різних платформ для роботи з 3D моделями, можна зробити кілька

важливих висновків щодо веб-застосунку GRIFON, який розробляється в даній дипломній роботі. Таблиця включає такі платформи, як Turbosquid, Quixel Megascans, SketchFab та GRIFON, і охоплює різні критерії порівняння, що мають значення для користувачів та авторів 3D контенту.

Веб-застосунок GRIFON виділяється серед конкурентів своєю широкою функціональністю, яка включає всі важливі критерії порівняння. Зокрема, GRIFON підтримує відстежування вподобаних робіт авторів, що дозволяє користувачам легко знаходити та стежити за оновленнями улюблених моделей. Це є важливою перевагою, яка покращує користувацький досвід та підвищує залученість.

Крім того, GRIFON забезпечує підтримку та оновлення вже проданих 3D моделей. Це означає, що користувачі, які придбали модель, можуть отримувати її оновлення без додаткових витрат, що є значною перевагою для підтримання актуальності контенту. Така функціональність також вигідна для авторів, які можуть постійно вдосконалювати свої роботи та підтримувати довгострокові відносини з клієнтами.

Ще однією важливою перевагою GRIFON є можливість написання рецензій та коментарів. Це дозволяє користувачам ділитися своїми враженнями та відгуками про моделі, що створює додаткову цінність для потенційних покупців та сприяє створенню активної спільноти навколо платформи.

GRIFON також не прив'язаний до конкретних платформ візуалізації 3D моделей, що дає користувачам свободу вибору інструментів для роботи з моделями. Це робить GRIFON універсальним інструментом для різних категорій користувачів, від початківців до професіоналів.

Веб-застосунок пропонує українську локалізацію, що є великою перевагою для україномовних користувачів, які віддають перевагу використанню програмного забезпечення на рідній мові. Це також сприяє популяризації платформи серед місцевих авторів та користувачів, що є важливим фактором для розширення бази користувачів.

У той час як такі платформи, як Turbosquid, Quixel Megascans та SketchFab, не підтримують усі перераховані функції одночасно. Наприклад, Turbosquid не має можливості відстежування вподобаних робіт, підтримки та оновлення проданих моделей, а також написання рецензій та коментарів. Quixel Megascans та SketchFab мають деякі з цих функцій, але не пропонують повного набору можливостей, які є у GRIFON.

Тому для створення комплексної торговельної площадки 3D-моделей, яка буде відповідати потребам як покупців, так і продавців, важливо врахувати всі ці фактори. Платформа повинна пропонувати широкий вибір високоякісних моделей за доступними цінами. Крім того, важливо забезпечити ретельний контроль якості та гнучкі умови для продавців.

2. АНАЛІЗ ТА ОБҐРУНТУВАННЯ ВИБОРУ СЕРЕДОВИЩА ТА ІНСТРУМЕНТІВ РОЗРОБКИ ВЕБ-ЗАСТОСУНКУ

Розробка веб-застосунку починається з чіткого розуміння його архітектури. Веб-застосунок складається з двох основних частин: клієнтської та серверної. Клієнтська частина, або фронтенд, відповідає за інтерфейс користувача та взаємодію з ним. Зазвичай вона написана мовами HTML, CSS або JavaScript. Ці технології дозволяють створювати динамічні сторінки, що реагують на дії користувача, забезпечуючи зручність та ефективність роботи з застосунком.

Серверна частина, або бекенд, займається обробкою запитів, та взаємодією з базами даних. Бекенд забезпечує логіку застосунку, відповідає за аутентифікацію користувачів, зберігання даних та бізнес-логіку. Серверні застосунки можуть бути написані різними мовами програмування, такими як C#, PHP, JavaScript, Java та Python.

Аналіз мов програмування для веб-застосунку є важливим етапом розробки, оскільки від правильного вибору залежить ефективність, швидкість розробки та підтримка застосунку. C# є потужною мовою, яка часто використовується для розробки серверних застосунків на платформі .NET. Вона пропонує розвинену екосистему та багато інструментів для розробників.

- **PHP** є популярною мовою для веб-розробки, особливо для серверних застосунків. Вона легко інтегрується з базами даних і має широкий набір фреймворків, що спрощують розробку. Java є універсальною мовою програмування, яка використовується у великих підприємствах. Вона забезпечує високу продуктивність та безпеку, що робить її популярною для серверних застосунків.
- **Python** є однією з найпопулярніших мов програмування завдяки своїй простоті та великій кількості бібліотек для різних задач, включаючи веб-розробку. Flask та Django є одними з найвідоміших фреймворків для розробки веб-застосунків на Python.

- **Node.js** дозволяє використовувати JavaScript для серверної частини, що забезпечує єдність технологій між клієнтською та серверною частинами. Це спрощує розробку та підтримку застосунку, дозволяючи розробникам використовувати одну мову для всієї системи.
- **C# з .NET** також є потужним інструментом для розробки серверних застосунків. Вона пропонує розвинену екосистему та інтеграцію з іншими сервісами Microsoft. Однак, використання різних мов для клієнтської та серверної частин може ускладнити процес розробки.

В даній дипломній роботі буде використано JavaScript з фреймворком Node.js, та Python з фреймворком Django. Поєднання JavaScript та Python в розробці веб-застосунків дозволяє скористатися перевагами обох мов для створення ефективних, масштабованих і зручних у розробці рішень. JavaScript, з його домінуванням на фронтенді, забезпечує динамічний та інтерактивний інтерфейс користувача, тоді як Python, відомий своєю простотою та багатою екосистемою бібліотек, часто використовується на бекенді для логіки бізнес-процесів та обробки даних.

На фронтенді JavaScript забезпечує динамічну взаємодію з користувачем через такі бібліотеки та фреймворки, як React, Angular чи Vue.js. Ці інструменти дозволяють створювати сучасні односторінкові додатки (SPA) з високою продуктивністю та інтерактивністю. JavaScript також використовується для реалізації асинхронних запитів до сервера через технології, такі як AJAX та Fetch API, що дозволяє динамічно оновлювати вміст сторінки без її повного перезавантаження.

Python, зі свого боку, широко використовується на серверній частині завдяки своїм можливостям обробки даних, машинного навчання та інтеграції з іншими сервісами. Фреймворки, пропонує багато вбудованих функцій, таких як аутентифікація, адміністрування та ORM (об'єктно-реляційне відображення), що значно прискорює процес розробки.

Інтеграція JavaScript і Python здійснюється через API (Application Programming Interface), де серверна частина на Python обробляє запити від клієнта, написані на JavaScript. RESTful API або GraphQL є найпоширенішими підходами для такої інтеграції. На сервері Python обробляє запити, виконує необхідні обчислення або доступ до бази даних і повертає відповіді у форматі JSON, які легко обробляються на фронтенді за допомогою JavaScript.

JavaScript у поєднанні з Node.js і Express.js є популярним вибором для створення серверної частини веб-застосунків завдяки своїй продуктивності, масштабованості та активній спільноті розробників. Node.js, як подієво-орієнтована платформа з неблокуючим введенням-виведенням (I/O), забезпечує можливість обробки великої кількості одночасних запитів з мінімальними затримками. Це робить її особливо придатною для реальних додатків, таких як чат-додатки, стрімінгові сервіси та інтернет-магазини. Використання однієї мови програмування, JavaScript, для обох частин додатка (клієнтської та серверної) спрощує процес розробки і зменшує потребу в навчанні нових технологій.

Express.js є мінімалістичним веб-фреймворком для Node.js, який надає потужний і гнучкий набір інструментів для побудови веб-серверів і API. Завдяки своїй простоті та легкості у використанні, Express.js дозволяє швидко налаштовувати маршрути, обробляти запити і відповіді, а також інтегруватися з різними середовищами виконання (middleware) для розширення функціональності додатка. Це забезпечує швидкий і ефективний розвиток проектів, дозволяючи розробникам зосередитися на бізнес-логіці замість роботи з низькорівневими деталями.

Python є широко використовуваною мовою програмування, відомою своєю простотою і читабельністю коду. Вона активно застосовується в різних галузях, включаючи веб-розробку, науку про дані, машинне навчання та автоматизацію. Django є потужним веб-фреймворком для Python, який забезпечує швидкий розвиток веб-застосунків завдяки своєму "батарейки включено" підходу. Django включає в себе все необхідне для створення повноцінних веб-застосунків: ORM для

роботи з базами даних, шаблони для генерації HTML, форми для обробки введення користувачів, а також інструменти для автентифікації та управління сесіями.

GraphQL є мовою запитів для API, яка дозволяє клієнтам запитувати лише ті дані, які їм потрібні, що зменшує кількість переданих даних і покращує продуктивність. Використання GraphQL з Django дозволяє створювати ефективні і гнучкі API, які легко масштабуються і адаптуються до змін у структурі даних. Django Graphene, бібліотека для інтеграції GraphQL з Django, забезпечує просту і зручну реалізацію GraphQL API, використовуючи потужні можливості Django ORM для управління даними.

Для реалізації зв'язку з користувачем через електронну пошту буде використано Nodemailer. Nodemailer підтримує відправку як простих текстових листів, так і складних HTML-листів з вкладеннями. Це дозволяє створювати естетично привабливі повідомлення, що включають зображення, стилі та інші елементи форматування. Можливість додавання вкладень дозволяє надсилати файли разом з листами, що є важливим для створення системи “Уподобань”, та відправки 3D моделі покупцю після успішної транзакції.

Для розробки серверної частини застосунку буде використовуватись ORM Sequelize. ORM (Object-Relational Mapping) дозволяє розробникам працювати з базами даних за допомогою об'єктів, спрощуючи процес написання SQL-запитів та управління даними. Sequelize є популярним ORM для Node.js, який підтримує різні бази даних, включаючи PostgreSQL, MySQL та SQLite.Вибір ORM Sequelize обґрунтований його потужністю та гнучкістю. Він забезпечує простий і зрозумілий API для роботи з базами даних, підтримує різні типи відносин між таблицями та має велику спільноту, що забезпечує швидке вирішення проблем та доступ до великої кількості ресурсів. Використання Sequelize дозволяє розробникам зосередитися на логіці застосунку, а не на роботі з базами даних, що підвищує продуктивність та ефективність розробки.

Отже, врахувавши всі переваги та недоліки цих мов програмування, було прийнято рішення використовувати JavaScript, зокрема фреймворки Node.js

Express, і Nodemailer, та мову Python з використанням фреймворків Django, та GraphQL, а також ORM Sequelize для написання серверної частини застосунку.

2.1 Аналіз фреймворків для написання серверної частини веб-застосунку

Серверна частина веб-застосунку виконує важливу роль у забезпеченні функціональності веб-додатків. На відміну від клієнтської частини, яка виконується на стороні браузера, серверна частина працює на сервері і забезпечує обробку запитів, управління базами даних, а також логіку бізнес-процесів. Оскільки обрані мови програмування веб-застосунку є JavaScript та Python далі буде наведено список фреймворков які працюють на базі цієї мови:

2.2.1 Sequelize

Sequelize — це потужна ORM (Object-Relational Mapping) бібліотека для Node.js, яка дозволяє розробникам працювати з базами даних за допомогою об'єктно-орієнтованого підходу. Вона підтримує різні реляційні бази даних, включаючи MySQL, PostgreSQL, SQLite і MSSQL, забезпечуючи зручний інтерфейс для взаємодії з ними. Sequelize спрощує процес створення, читання, оновлення та видалення (CRUD) записів у базі даних, що робить її популярною серед розробників веб-застосунків.

Однією з основних особливостей Sequelize є можливість визначення моделей, які представляють таблиці в базі даних. Моделі визначають структуру таблиць, їхні поля, типи даних та взаємозв'язки між іншими моделями. Кожна модель у Sequelize є класом, що розширює основний клас Model, надаючи методи для маніпуляції даними. Це дозволяє розробникам працювати з даними у вигляді об'єктів JavaScript, зберігаючи при цьому всю функціональність реляційної бази даних.

Система взаємозв'язків у Sequelize дозволяє моделювати складні зв'язки між таблицями, включаючи зв'язки один-до-одного (one-to-one), один-до-багатьох (one-

to-many) та багато-до-багатьох (many-to-many). Для цього використовуються спеціальні методи, такі як belongsTo, hasOne, hasMany та belongsToMany. Це дозволяє автоматично створювати відповідні зовнішні ключі та таблиці зв'язків, значно спрощуючи роботу з реляційними даними.

Sequelize має розширену систему плагінів, що дозволяє розширювати його функціональність за допомогою додаткових модулів. Плагіни можуть додавати нові методи до моделей, змінювати поведінку вбудованих методів або додавати нові типи даних. Це робить Sequelize дуже гнучким інструментом, що може бути налаштований.

2.2.2 TypeORM

TypeORM — це ORM (Object-Relational Mapping) бібліотека для TypeScript і JavaScript (Node.js), яка дозволяє розробникам працювати з реляційними базами даних, використовуючи об'єктно-орієнтований підхід. Вона підтримує популярні реляційні бази даних, такі як MySQL, MariaDB, PostgreSQL, SQLite, Microsoft SQL Server, Oracle, а також некласифіковані бази даних, як MongoDB. TypeORM надає потужні інструменти для визначення моделей, побудови складних запитів, керування міграціями та виконання транзакцій.

TypeORM дозволяє розробникам визначати моделі у вигляді класів TypeScript або JavaScript, де кожен клас відповідає таблиці в базі даних. Ці класи можуть включати різні поля з типами даних, індекси, унікальні обмеження, а також взаємозв'язки між таблицями. TypeORM використовує декоратори для маркування властивостей класів як колонок бази даних і для визначення відносин між моделями. Це дозволяє легко читати та розуміти структуру моделі та її зв'язки, зберігаючи при цьому всі переваги типізації TypeScript.

Взаємозв'язки між таблицями у TypeORM можуть бути реалізовані через різні типи відносин: один-до-одного (one-to-one), один-до-багатьох (one-to-many), багато-до-одного (many-to-one) і багато-до-багатьох (many-to-many). Декоратори @OneToOne, @OneToMany, @ManyToOne та @ManyToMany дозволяють

визначити ці відносини безпосередньо в класах моделей. TypeORM автоматично створює необхідні зовнішні ключі та таблиці зв'язків для підтримки цих відносин, що значно спрощує процес моделювання складних структур баз даних.

TypeORM підтримує валідацію даних, що дозволяє визначати різноманітні правила для полів моделей, такі як обов'язковість, унікальність, діапазон значень, формат тощо. Це забезпечує контроль над якістю даних, що зберігаються в базі, і запобігає збереженню некоректних або неповних даних. Використання декораторів @IsEmail, @IsInt, @Length та інших дозволяє легко додавати правила валідації до полів моделей.

Однією з важливих функцій TypeORM є підтримка міграцій бази даних. Міграції дозволяють керувати змінами у структурі бази даних, зокрема створенням, зміненням або видаленням таблиць і колонок, додаванням індексів та обмежень. TypeORM надає інструменти для генерації, виконання та відкату міграцій, що дозволяє відстежувати всі зміни у структурі бази даних та застосовувати їх у різних середовищах розробки, тестування та продуктивності.

TypeORM підтримує транзакції, що забезпечує атомарність, консистентність, ізоляцію та довговічність (ACID) операцій з даними. Транзакції дозволяють виконувати кілька операцій як одну єдину операцію, що гарантує цілісність даних навіть у випадку помилок. TypeORM надає API для створення та керування транзакціями, що дозволяє легко інтегрувати їх у бізнес-логіку застосунку.

2.2.3 Express

Express — це мінімалістичний і гнучкий веб-фреймворк для Node.js, призначений для створення веб-додатків та API. Він надає широкий набір інструментів та функцій, що значно спрощують розробку серверних додатків, роблячи цей процес швидким і ефективним. Express є одним з найпопулярніших веб-фреймворків для Node.js, завдяки своїй простоті, продуктивності та розширюваності.

Однією з ключових особливостей Express є його мінімалістичний дизайн, який дозволяє розробникам додавати лише ті функції, які дійсно потрібні для їхніх додатків. Це досягається за рахунок використання middleware — функцій, які обробляють запити на різних етапах їхнього проходження через сервер. Middleware можуть бути використані для виконання широкого спектру задач, таких як обробка файлів cookie, сесій, запитів на авторизацію, логування, обробка помилок тощо. Завдяки цьому підходу Express залишається легким і швидким, дозволяючи розробникам створювати високопродуктивні серверні додатки.

Express використовує роутинг для визначення кінцевих точок (endpoints) додатку та їхньої логіки обробки. Роутинг дозволяє розробникам задавати різні маршрути для обробки HTTP-запитів, таких як GET, POST, PUT, DELETE тощо. Кожен маршрут може бути пов'язаний з однією або кількома функціями обробки, що виконуються при отриманні відповідного запиту. Це забезпечує високу гнучкість та дозволяє легко керувати складними структурами додатків, зокрема реалізовувати RESTful API.

Express підтримує шаблонизацію, що дозволяє створювати динамічні HTML-сторінки. Він інтегрується з різними движками шаблонів, такими як Pug (раніше відомий як Jade), EJS (Embedded JavaScript) та Handlebars. Це дозволяє розробникам створювати динамічні веб-сторінки, що генеруються на основі даних з бази даних або інших джерел. Підтримка шаблонів також спрощує управління статичними файлами, такими як CSS, JavaScript і зображення, забезпечуючи ефективну організацію та доставку контенту.

Однією з важливих функцій Express є обробка помилок. Express надає вбудовані механізми для обробки помилок, що дозволяє розробникам легко керувати помилками, які виникають у процесі обробки запитів. Спеціальні middleware для обробки помилок можуть бути додані в додаток, що забезпечує централізоване управління помилками та їхню логування. Це дозволяє розробникам ефективно виявляти та виправляти проблеми, що покращує надійність і стабільність додатку.Express підтримує інтеграцію з різними базами

даних, такими як MongoDB, MySQL, PostgreSQL та іншими. Це досягається за допомогою використання сторонніх бібліотек і модулів, таких як Mongoose для MongoDB або Sequelize для реляційних баз даних. Це забезпечує гнучкість у виборі бази даних та спрощує роботу з ними, дозволяючи легко зберігати та витягувати дані з різних джерел.

Express є добре документованим фреймворком, з детальною офіційною документацією, що включає приклади використання, найкращі практики та рекомендації. Це робить його доступним для розробників з різним рівнем досвіду, дозволяючи швидко освоїти основні концепції та розпочати розробку додатків. Офіційна документація, а також численні туторіали та навчальні матеріали, доступні в Інтернеті, допомагають розробникам ефективно використовувати всі можливості фреймворка.

Express має активну спільноту розробників і широкий набір додаткових модулів, що розширюють його функціональність. Спільнота розробників регулярно випускає оновлення та нові версії Express, що включають виправлення помилок, нові функції та покращення продуктивності. Крім того, існує велика кількість сторонніх модулів, доступних через npm (Node Package Manager), які можуть бути використані для додавання нових функцій до додатків на базі Express

2.2.4 Django

Django — це високорівневий веб-фреймворк для мови програмування Python, призначений для швидкої розробки веб-додатків із акцентом на простоту, гнучкість і масштабованість. Він дотримується принципу "не повторюйся" (Don't Repeat Yourself, DRY) та надає широкий спектр вбудованих інструментів і функцій, що допомагають розробникам створювати складні, масштабовані веб-додатки з мінімальними зусиллями. Django створений для того, щоб допомогти розробникам реалізовувати свої проекти швидко і ефективно, забезпечуючи при цьому високу якість та безпеку коду.

Однією з ключових особливостей Django є його архітектура MVC (Model-View-Controller), реалізована через підхід MTV (Model-Template-View). Модель (Model) відповідає за взаємодію з базою даних, забезпечуючи структуру даних та їхню валідацію. Шаблон (Template) відповідає за рендеринг HTML, що дозволяє відокремлювати логіку від представлення. Вигляд (View) обробляє запити, взаємодіє з моделями і передає дані шаблонам для рендерингу. Такий підхід забезпечує чітке розділення відповідальностей і полегшує підтримку та масштабування додатків.

Django поставляється з потужною системою адміністрування, яка генерується автоматично на основі моделей. Ця система забезпечує інтерфейс для керування даними в базі, дозволяючи адміністраторам додавати, редагувати та видаляти записи без написання додаткового коду. Адмін-панель підтримує налаштування прав доступу та дозволяє створювати користувацькі інтерфейси для специфічних потреб проекту. Це значно спрощує управління контентом та даними, зберігаючи час та ресурси розробників.

Django також має вбудовану систему маршрутизації URL, яка дозволяє визначати маршрути для обробки запитів до різних частин додатку. Маршрутизація підтримує динамічні URL, що дозволяє легко створювати SEO-дружні посилання та керувати складними структурами додатків. Крім того, Django забезпечує потужні засоби для обробки форм, включаючи валідацію даних, обробку помилок і автоматичне генерування HTML-форм на основі моделей.

Безпека є однією з головних переваг Django. Він включає вбудовані захисти від найпоширеніших веб-уразливостей, таких як SQL-ін'єкції, міжсайтові скрипти (XSS), міжсайтові підробки запитів (CSRF) та клацання (clickjacking). Django також підтримує захищену аутентифікацію та управління користувачами, що включає систему користувачів і груп, а також механізми для управління паролями та сесіями.

Інтеграція з базами даних у Django здійснюється через ORM (Object-Relational Mapping), що дозволяє розробникам працювати з базами даних,

використовуючи об'єктно-орієнтований підхід. Django підтримує широкий спектр реляційних баз даних, включаючи PostgreSQL, MySQL, SQLite та Oracle. ORM забезпечує автоматичне створення таблиць, індексів та обмежень на основі визначень моделей, що спрощує процес міграцій та оновлень структури бази даних.

Django також підтримує кешування, що дозволяє значно підвищити продуктивність додатків. Він надає різні стратегії кешування, включаючи кешування на рівні шаблонів, представлень та низькорівневе кешування. Підтримка кешування з використанням різних бекендів, таких як Memcached та Redis, дозволяє вибрати оптимальне рішення для конкретних потреб проекту.

Поєднання Django з JavaScript є поширеною практикою, що дозволяє створювати сучасні веб-додатки з багатим клієнтським інтерфейсом. Django надає можливість інтеграції з фронтенд-фреймворками та бібліотеками, такими як React, Angular, Vue.js та іншими. Це дозволяє використовувати Django для розробки бекенду, обробки даних та логіки додатку, тоді як JavaScript забезпечує динамічний і інтерактивний користувацький інтерфейс.

Одним з популярних підходів до інтеграції Django з JavaScript є використання REST API або GraphQL для обміну даними між клієнтом і сервером. Django Rest Framework (DRF) є потужним інструментом для створення RESTful API в Django. Він забезпечує механізми для серіалізації даних, аутентифікації, авторизації та обробки запитів, що дозволяє швидко створювати та масштабувати API. GraphQL також може бути інтегрований з Django за допомогою бібліотек, таких як Graphene-Django, що дозволяє використовувати переваги гнучкого запитування даних та високої продуктивності.

Використання сучасних фронтенд-інструментів, таких як Webpack або Parcel, дозволяє ефективно управляти статичними файлами та пакетами JavaScript у проектах Django. Django надає механізми для інтеграції з цими інструментами, що спрощує процес розробки, збирання та розгортання додатків. Крім того, Django підтримує обробку AJAX-запитів, що дозволяє створювати динамічні інтерфейси без перезавантаження сторінок.

2.2.5 Nodemailer

Nodemailer — це популярний модуль для Node.js, який дозволяє розробникам легко відправляти електронні листи з серверних додатків. Він забезпечує простий і гнучкий API для інтеграції функціоналу надсилання електронної пошти в різноманітні Node.js проекти. Nodemailer підтримує різні транспорти для відправки електронних листів, включаючи SMTP (Simple Mail Transfer Protocol), а також сторонні сервіси, такі як SendGrid, Mailgun та AWS SES (Simple Email Service). Це робить його універсальним інструментом для реалізації поштових функцій у веб-додатках, мобільних додатках та інших сервісах.

Поєднання Nodemailer з JavaScript є природним, оскільки Nodemailer побудований на базі Node.js. JavaScript використовується як основна мова для написання серверних скриптів, які керують процесом відправки електронних листів. Це дозволяє легко інтегрувати Nodemailer у будь-який Node.js проект, використовуючи звичний синтаксис та підходи. Наприклад, відправка листа може бути інтегрована в обробку HTTP-запитів у веб-додатку на базі Express.js, де Nodemailer використовується для відправки підтверджень реєстрації, відновлення паролів, сповіщень про оновлення та інших автоматизованих повідомлень.

Крім основного функціоналу, Nodemailer підтримує розширюваність через використання плагінів і додаткових модулів. Це дозволяє додавати нові можливості та інтегрувати Nodemailer з іншими сервісами та інструментами. Наприклад, можна використовувати плагіни для кодування електронних листів, перевірки спаму, відстеження відкриття листів та клацань по посиланнях. Така гнучкість робить Nodemailer потужним інструментом для реалізації будь-яких поштових потреб у Node.js проектах.

2.2.6 GraphQL

GraphQL — це мова запитів для API та середовище виконання для виконання цих запитів за наявними даними. Розроблений Facebook у 2012 році та випущений

у відкритий доступ у 2015 році, GraphQL призначений для надання клієнтам можливості точно визначати структуру необхідних даних, що дозволяє уникнути проблем надмірної та недостатньої завантаженості (over-fetching та under-fetching). GraphQL дозволяє клієнтам запитувати рівно ті дані, які їм потрібні, що підвищує ефективність роботи API та зменшує обсяг переданих даних.

Основою GraphQL є його декларативний підхід, що дозволяє клієнтам описувати структуру необхідних даних за допомогою запитів (queries). Запити в GraphQL схожі на JSON-подібний синтаксис, де клієнт вказує, які поля йому потрібні від кожного типу об'єкта. Це контрастує з традиційними REST API, де кожен кінцевий пункт (endpoint) повертає фіксований набір даних. У GraphQL клієнти можуть комбінувати запити до різних ресурсів у єдиному запиті, що значно знижує кількість мережевих викликів і підвищує продуктивність додатків.

GraphQL також підтримує мутації (mutations), які використовуються для зміни даних на сервері. Мутації мають подібний синтаксис до запитів, але крім вказівки полів для отримання, вони також дозволяють передавати аргументи для створення, оновлення або видалення даних. Це забезпечує потужний та гнучкий спосіб управління даними на сервері, дозволяючи виконувати складні операції з мінімальними зусиллями.

Схема (schema) є центральним елементом GraphQL, що визначає типи даних, взаємозв'язки між ними та можливі запити та мутації. Схема описується за допомогою мови визначення схем (SDL, Schema Definition Language), яка дозволяє чітко та однозначно описати структуру даних API. Кожен тип у схемі може мати поля, що визначають доступні дані, та аргументи, які можна використовувати для фільтрації або модифікації результатів. Схема також може включати інтерфейси та юніони (unions), що дозволяє створювати складні та гнучкі структури даних.

GraphQL також включає механізми для обробки помилок та повідомлень. Відповіді GraphQL можуть містити як дані, так і помилки, що дозволяє клієнтам отримувати часткові результати навіть у разі виникнення проблем. Це забезпечує

кращий користувацький досвід, оскільки додатки можуть відображати доступні дані та інформувати користувачів про помилки, що виникли.

Інструменти для розробки та налагодження є важливою складовою екосистеми GraphQL. Одним з найпопулярніших інструментів є GraphiQL, інтерактивна консоль для виконання запитів та мутацій, яка дозволяє розробникам експериментувати з API та отримувати результати в реальному часі. GraphiQL надає автозавершення для запитів на основі схеми, що значно прискорює процес розробки та зменшує кількість помилок. Інші інструменти, такі як Apollo Client Devtools та Relay Devtools, забезпечують розширені можливості для налагодження клієнтів GraphQL у браузері.

2.2 Аналіз фреймворків для написання клієнтської частини веб-застосунку

Клієнтська частина веб-застосунку, або фронтенд, є інтерфейсом, з яким взаємодіє користувач через веб-браузер. Вона відповідає за відображення даних, забезпечення зручної взаємодії та виконання завдань користувача. Основною метою клієнтської частини є надання інтуїтивно зрозумілого і привабливого користувацького інтерфейсу, який забезпечує доступ до функціональності веб-додатку.

Клієнтська частина включає компоненти, що відображають структуру та зміст веб-сторінок, такі як заголовки, абзаци, зображення, списки, таблиці, форми та інші елементи. Ці компоненти розміщуються та стилізуються відповідно до дизайнерських вимог, що забезпечує логічну і привабливу організацію інформації на екрані користувача. Правильне структурування елементів та їхнє візуальне оформлення сприяє легкості сприйняття і використання веб-додатку.

Інтерактивні елементи, такі як кнопки, посилання, випадаючі списки, чекбокси і радіокнопки, забезпечують користувачам можливість взаємодіяти з веб-додатком. Ці елементи реагують на дії користувача, такі як натискання миші,

введення тексту або вибір опцій, що дозволяє виконувати різноманітні завдання, від навігації по сайту до подання форм та здійснення покупок. Важливим аспектом є забезпечення зворотного зв'язку для користувача, наприклад, через зміну кольору кнопок при наведенні, відображення повідомлень про помилки або підтвердження успішних дій.

Кешування є важливою функцією клієнтської частини, яка дозволяє зберігати часто використовувані дані локально у браузері користувача. Це зменшує кількість запитів до сервера, покращує продуктивність і зменшує час завантаження сторінок. Кешування може включати збереження статичних ресурсів, таких як зображення, стилі та скрипти, а також даних з API, що забезпечує швидкий доступ до них без необхідності повторного завантаження.

Забезпечення доступності (accessibility) є критично важливим аспектом клієнтської частини, що гарантує, що веб-додаток буде доступним для всіх користувачів, включаючи людей з обмеженими можливостями. Це включає використання правильних семантичних елементів, забезпечення клавіатурної навігації, надання альтернативного тексту для зображень, а також дотримання стандартів доступності, таких як WCAG (Web Content Accessibility Guidelines). Доступність покращує користувацький досвід і забезпечує, що веб-додаток може використовуватися якомога ширшою аудиторією.

Анімації та переходи відіграють важливу роль у покращенні візуального досвіду користувача. Вони використовуються для привертання уваги до важливих елементів, полегшення навігації та забезпечення плавності взаємодії. Анімації можуть включати прості ефекти, такі як зміна кольору кнопок при наведенні, а також складніші анімації, такі як слайд-шоу, спливаючі вікна або анімовані графіки. Важливо, щоб анімації були плавними і не відволікали користувача від основних завдань.

Інтеграція з API є критично важливою функцією клієнтської частини, що дозволяє отримувати і відправляти дані з сервера. Веб-додатки використовують API для взаємодії з серверною частиною, отримання даних з баз даних, а також

інтеграції з сторонніми сервісами, такими як платіжні системи, сервіси картографії або соціальні мережі. API забезпечують гнучкість і розширюваність веб-додатків, дозволяючи легко додавати нові функції та інтегрувати нові сервіси.

Тому для реалізації вищеописаних пунктів буде проведено аналіз фреймворків для JavaScript:

2.3.1 React

React — це популярна бібліотека JavaScript, розроблена компанією Facebook для створення користувацьких інтерфейсів, особливо односторінкових додатків (Single Page Applications, SPA). Вона дозволяє розробникам будувати складні інтерфейси з компонентів, які є незалежними, багаторазово використовуваними частинами коду. React забезпечує високу продуктивність, завдяки використанню віртуального DOM (Document Object Model), що мінімізує кількість операцій з реальним DOM і таким чином покращує швидкість та ефективність роботи додатку.

Віртуальний DOM — це ще одна важлива технологія, яка робить React таким ефективним. Замість того, щоб напряму взаємодіяти з реальним DOM, React працює з віртуальною копією DOM у пам'яті. Коли відбуваються зміни в стані компонентів, React порівнює нову віртуальну DOM з попередньою версією і визначає мінімальний набір змін, необхідних для оновлення реального DOM. Цей процес, відомий як "реактивне оновлення", значно підвищує продуктивність додатків, зменшуючи кількість дорогих операцій з реальним DOM.

React також підтримує одностороннє потокове управління даними (one-way data flow), що означає, що дані передаються вниз по дереву компонентів через пропси (props), а зміни стану піднімаються вгору через зворотні виклики (callbacks). Це робить дані передбачуваними та легкими для відстеження, що сприяє кращій структурованості коду та полегшує налагодження. Завдяки цьому підходу, розробники можуть створювати складні інтерфейси з великою кількістю взаємодій, зберігаючи при цьому код чистим і зрозумілим.

Для управління станом у великих додатках React часто використовують зовнішні бібліотеки, такі як Redux або MobX. Redux забезпечує централізоване управління станом, використовуючи концепції з функціонального програмування, такі як чисті функції та немутований стан. MobX, навпаки, використовує реактивні принципи, дозволяючи автоматично відстежувати залежності та оновлювати компоненти при зміні стану. Обидві бібліотеки мають свої переваги та недоліки і можуть бути вибрані в залежності від специфіки проекту.

React також добре інтегрується з різними інструментами для побудови та управління проектами. Webpack, наприклад, використовується для збірки модулів, забезпечуючи ефективне управління залежностями та оптимізацію коду. Babel використовується для трансформації сучасного JavaScript коду, включаючи JSX (JavaScript XML), у формат, сумісний з поточними браузерами. Це дозволяє використовувати новітні можливості мови JavaScript, навіть якщо вони ще не підтримуються у всіх браузерах.

Однією з важливих частин екосистеми React є React Router — бібліотека для управління маршрутизацією у додатках. Вона дозволяє створювати динамічні маршрути, забезпечуючи плавну навігацію між різними сторінками або розділами додатку без перезавантаження сторінки. React Router підтримує різні методи навігації, включаючи декларативну маршрутизацію, динамічні маршрути та вкладені маршрути, що робить її потужним інструментом для створення складних SPA.

React також має потужну екосистему інструментів для тестування, що дозволяє розробникам створювати надійні додатки з високим рівнем покриття тестами. Бібліотеки, такі як Jest та React Testing Library, забезпечують інструменти для юніт-тестування компонентів, перевірки взаємодії з користувачем та симуляції різних сценаріїв використання. Це дозволяє впевнитися в коректній роботі додатку та швидко виявляти й виправляти помилки на ранніх стадіях розробки.

2.3.2 MobX

MobX — це бібліотека для управління станом у додатках на базі JavaScript, яка використовує реактивний підхід до обробки змін у стані. Вона дозволяє розробникам створювати додатки з більш чистою архітектурою, де стан та його зміни автоматично відстежуються, забезпечуючи синхронізацію між даними та інтерфейсом користувача без необхідності явного виклику оновлень. MobX побудований на основі концепції "транспарантної реактивності" (transparent reactivity), що означає, що будь-яка зміна стану автоматично приводить до оновлення всіх залежних від нього частин коду.

Основний принцип роботи MobX базується на використанні спостерігачів (observers) та спостережуваних об'єктів (observables). Спостережувані об'єкти можуть бути простими змінними, масивами або структурами даних, які MobX відстежує на предмет змін. Спостерігачі, в свою чергу, автоматично реагують на зміни в спостережуваних об'єктах, оновлюючи відповідні частини інтерфейсу або виконуючи інші дії. Це забезпечує високу продуктивність та синхронізацію стану додатку з його відображенням.

MobX використовує декоратори, щоб позначати спостережувані властивості та обчислювані значення. Декоратор @observable використовується для визначення властивостей, що повинні відстежуватися на предмет змін. Наприклад, при зміні значення спостережуваної властивості, MobX автоматично оновить всі компоненти, що залежні від цієї властивості. Декоратор @computed використовується для обчислення значень, які залежать від спостережуваних властивостей. Ці значення кешуються та автоматично оновлюються при зміні залежностей, забезпечуючи ефективне управління станом.

MobX також підтримує асинхронні дії через використання функцій async і await всередині дій (actions). Декоратор @action позначає функції, які можуть змінювати стан спостережуваних властивостей. Це дозволяє централізовано керувати змінами стану та забезпечує чистоту і передбачуваність коду.

Використання асинхронних дій дозволяє MobX легко інтегруватися з будь-якими асинхронними операціями, такими як запити до API або робота з базами даних.

Однією з ключових переваг MobX є його гнучкість та сумісність з різними фреймворками та бібліотеками. MobX може бути інтегрований з React, Angular, Vue та іншими популярними бібліотеками для створення інтерфейсів. Особливо популярна інтеграція MobX з React, що забезпечує ефективне управління станом у складних додатках. Компоненти React можуть бути перетворені на спостерігачів за допомогою функції observer з бібліотеки mobx-react, що автоматично забезпечує оновлення компонентів при зміні спостережуваних властивостей.

MobX також забезпечує високу продуктивність завдяки використанню принципу "найменших змін". Це означає, що MobX мінімізує кількість оновлень, виконуючи тільки ті операції, які дійсно необхідні для синхронізації стану. Це досягається за рахунок відстеження залежностей між спостережуваними об'єктами та реакціями і виключення зайвих оновлень. Такий підхід дозволяє створювати додатки з високою продуктивністю та низькими затримками.

2.3.3 Vue.js

Vue.js — це прогресивний фреймворк для створення користувацьких інтерфейсів, розроблений для того, щоб бути прийнятним поступово. На відміну від монолітних фреймворків, Vue.js побудований з метою інкрементального впровадження. Його основна бібліотека зосереджена на шарі представлення і може легко інтегруватися з іншими бібліотеками або існуючими проектами. Vue.js також ідеально підходить для створення складних односторінкових додатків, коли використовується спільно з сучасними інструментами та бібліотеками підтримки.

Vue.js використовує шаблони, які дуже схожі на HTML, але з розширеннями для прив'язки даних та директив. Ці шаблони компілюються в функції віртуального DOM, що дозволяє Vue.js відстежувати мінімальні зміни для ефективного оновлення реального DOM. Директиви, такі як v-bind і v-model, дозволяють легко

зв'язувати елементи DOM з даними, що робить шаблони Vue.js потужними та гнучкими.

Компоненти є фундаментальними будівельними блоками у Vue.js. Кожен компонент інкапсулює свою власну логіку, структуру та стиль, що дозволяє створювати багаторазово використовувані та легко підтримувані частини інтерфейсу. Компоненти можуть бути вкладені один в одного, передавати дані через props і піднімати події для зв'язку між собою. Це забезпечує модульність та реюзабельність коду, що є ключовими принципами сучасної розробки інтерфейсів.

Vue.js має потужну систему управління станом за допомогою Vuex, офіційної бібліотеки для управління станом у додатках на базі Vue.js. Vuex дозволяє зберігати стан у централізованому сховищі та управляти ним за допомогою дій та мутацій. Це забезпечує передбачуваність стану та полегшує налагодження додатків. Використання Vuex особливо корисне в великих додатках, де управління станом може стати складним завданням.

Маршрутизація у Vue.js реалізується за допомогою Vue Router, офіційного маршрутизатора для створення односторінкових додатків. Vue Router дозволяє визначати маршрути та компоненти, які повинні бути завантажені при переході за цими маршрутами. Він підтримує динамічні маршрути, вкладені маршрути та захищені маршрути, що робить його потужним інструментом для створення складних додатків з багаторазовою навігацією.

Однією з важливих переваг Vue.js є його екосистема інструментів, яка включає Vue CLI, Vue Devtools та інші корисні інструменти для розробників. Vue CLI надає потужний інтерфейс командного рядка для створення, налаштування та розгортання проектів Vue.js. Він підтримує шаблони проектів, плагіни та налаштування Webpack, що дозволяє легко почати роботу з Vue.js та інтегрувати його з іншими інструментами. Vue Devtools — це розширення для браузерів, яке дозволяє розробникам налагоджувати та аналізувати свої додатки Vue.js в режимі реального часу, забезпечуючи глибокий огляд компонентів, стану та подій.

Vue.js також підтримує серверний рендеринг (Server-Side Rendering, SSR), що дозволяє рендерити сторінки на сервері перед їх відправкою до клієнта. Це забезпечує швидший час завантаження сторінок, покращує SEO та забезпечує кращий користувацький досвід на повільних з'єднаннях. SSR у Vue.js реалізується за допомогою Nuxt.js, фреймворку на базі Vue.js, який спрощує процес налаштування та використання серверного рендерингу.

2.3.4 Redux

Redux — це бібліотека для управління станом додатків, яка часто використовується з бібліотекою React, хоча може бути інтегрована з будь-якою іншою бібліотекою або фреймворком JavaScript. Вона допомагає розробникам керувати станом додатків більш передбачувано та централізовано, використовуючи концепції з функціонального програмування, такі як чисті функції та немутований стан. Redux створено на основі архітектури Flux, запропонованої Facebook, але спрощує та розширює її принципи для полегшення використання та підвищення ефективності.

Redux використовує дії (actions) для опису змін у стані. Дії є простими об'єктами JavaScript, які містять тип дії та, за потреби, додаткові дані. Вони відправляються (dispatched) у сховище для виклику змін у стані. Дії служать єдиним способом ініціювання змін, що робить стан додатку передбачуваним і легко відстежуваним. Це також дозволяє легко інтегрувати інструменти для налагодження, такі як Redux DevTools, які можуть записувати та відтворювати дії для діагностики проблем.

Ред'юсери (reducers) в Redux — це чисті функції, які визначають, як стан додатку змінюється у відповідь на дії. Ред'юсери приймають поточний стан і дію як аргументи та повертають новий стан. Оскільки ред'юсери є чистими функціями, вони не мають побічних ефектів і завжди повертають один і той самий результат для одного і того ж вхідного значення. Це забезпечує передбачуваність змін стану та полегшує тестування. У великих додатках стан може бути розбитий на кілька

підстанів, кожен з яких обробляється своїм ред'юсером, що дозволяє легко масштабувати додаток.

Селектори (selectors) в Redux — це функції, які отримують стан сховища та повертають дані у форматі, зручному для використання в компонентах. Селектори дозволяють ізолювати логіку вибору даних від компонентів, що робить компоненти більш чистими та легко підтримуваними. Вони також можуть бути комбіновані для створення складних вибірок даних з мінімальними витратами на продуктивність.

Побічні ефекти, такі як асинхронні запити до API або робота з локальним сховищем, можуть бути оброблені за допомогою middleware. Middleware у Redux — це функції, які перехоплюють дії перед тим, як вони досягнуть ред'юсерів. Вони можуть виконувати додаткові завдання, такі як логування, виклик асинхронних операцій або модифікація дій. Одними з найпопулярніших middleware для роботи з асинхронними діями є Redux Thunk і Redux Saga. Redux Thunk дозволяє писати дії, які повертають функції замість об'єктів, що дозволяє виконувати асинхронні операції перед відправкою дій. Redux Saga використовує генератори для обробки асинхронних операцій, що дозволяє писати більш читабельний та керований код.

Redux DevTools — це потужний інструмент для налагодження, який дозволяє розробникам відслідковувати всі дії, що відправляються у сховище, та зміни стану, які вони викликають. DevTools підтримують такі функції, як тайм-тревел, що дозволяє перемотувати стан додатку назад і вперед, а також відтворювати послідовності дій для діагностики проблем. Це значно полегшує процес розробки та тестування додатків на базі Redux.

Redux також забезпечує високу масштабованість додатків, завдяки чіткій організації коду та можливості розбиття стану на окремі модулі. Це дозволяє командам розробників працювати незалежно над різними частинами додатку, зберігаючи при цьому узгодженість стану та поведінки. Структура додатків на базі Redux зазвичай включає розділення дій, ред'юсерів та компонентів на окремі файли або модулі, що спрощує підтримку та розвиток коду.

Інтеграція Redux з React забезпечується за допомогою бібліотеки react-redux, яка надає зручні інтерфейси для підключення компонентів React до сховища Redux. Вона забезпечує оптимізоване оновлення компонентів, що зменшує кількість непотрібних рендерів та покращує продуктивність додатку. Використання HOC (Higher-Order Components) або хуків (hooks) з react-redux дозволяє легко підключати компоненти до стану Redux та відправляти дії.

2.3.5 Axios

Axios — це популярна бібліотека JavaScript для виконання HTTP-запитів, яка дозволяє здійснювати запити до серверів з браузера або Node.js. Вона надає потужний і гнучкий API для роботи з асинхронними запитами, підтримуючи всі основні методи HTTP, такі як GET, POST, PUT, DELETE, PATCH та інші. Axios забезпечує простий спосіб взаємодії з RESTful API та обробки даних, що повертаються з серверів, і завдяки своїй універсальності та легкості у використанні здобула широку популярність серед розробників.

Однією з головних переваг Axios є його підтримка промісів (Promises), що дозволяє легко обробляти асинхронні операції. Це значно спрощує управління потоками виконання в коді, дозволяючи розробникам використовувати методи then і catch для обробки успішних відповідей та помилок відповідно. Крім того, Axios повністю підтримує асинхронні функції (async/await), що робить код ще більш зрозумілим та лаконічним. Ця підтримка асинхронності дозволяє ефективно виконувати запити та обробляти відповіді без блокування основного потоку виконання.

Axios надає розширені можливості для налаштування запитів, включаючи налаштування заголовків (headers), параметрів (params), тайм-аутів (timeouts), а також автентифікації (authentication). Це дозволяє розробникам точно контролювати кожен аспект запиту, забезпечуючи високу гнучкість та адаптивність під різні вимоги проекту. Наприклад, можна встановити заголовки

для авторизації, вказати параметри запиту через URL або задати час очікування для запиту, щоб уникнути зависань у випадку повільної відповіді сервера.

Одна з потужних функцій Axios — це можливість створення інтерсепторів (interceptors). Інтерсептори дозволяють перехоплювати та змінювати запити або відповіді перед їх обробкою, що надає можливість виконувати додаткові дії, такі як додавання токенів авторизації, логування або обробка помилок глобально. Інтерсептори можуть бути використані для реалізації повторних спроб запиту у разі помилки або для обробки специфічних кодів відповідей, таких як перенаправлення або оновлення токенів.

Axios підтримує обробку даних у різних форматах, включаючи JSON, XML, і звичайний текст. Він автоматично конвертує дані відповідно до заголовків Content-Type, що полегшує обробку відповідей сервера. Крім того, Axios може працювати з форм-даними (FormData) для завантаження файлів або надсилання даних з форм, що робить його зручним інструментом для інтеграції з веб-формами та обробки файлів на стороні сервера.

У контексті роботи з браузерами, Axios забезпечує підтримку крос-доменних запитів через CORS (Cross-Origin Resource Sharing). Це дозволяє додаткам здійснювати запити до серверів, що знаходяться на інших доменах, зберігаючи при цьому безпеку і конфіденційність даних. Axios автоматично обробляє налаштування CORS, що значно спрощує розробку крос-доменних додатків.

Інтеграція Axios з Node.js робить його потужним інструментом для створення серверних додатків. Axios підтримує всі можливості Node.js для виконання HTTP-запитів, включаючи обробку проксі-серверів, завантаження файлів і роботу з потоками (streams). Це дозволяє використовувати Axios як універсальний інструмент для взаємодії з API як на клієнтській, так і на серверній стороні.

2.3.6 Bootstrap

Bootstrap — це популярний фреймворк для розробки адаптивних веб-інтерфейсів, який спрощує процес створення стильних та функціональних веб-

сайтів та веб-додатків. Розроблений командою Twitter, Bootstrap був вперше випущений у 2011 році і з тих пір став одним із найпоширеніших інструментів у веб-розробці. Основними компонентами Bootstrap є CSS, JavaScript та HTML-шаблони, які забезпечують готові рішення для створення макетів, форм, кнопок, навігаційних елементів та інших інтерфейсних компонентів.

Однією з головних переваг Bootstrap є його система сіток (grid system), яка дозволяє створювати адаптивні макети, що автоматично підлаштовуються під різні розміри екранів. Сіткова система Bootstrap базується на 12-колонному макеті, що забезпечує гнучкість у розташуванні елементів на сторінці. Розробники можуть використовувати класи для створення рядів (rows) та колонок (columns), що робить процес створення складних макетів простим і зрозумілим. Адаптивність досягається завдяки використанню медіа-запитів (media queries), які змінюють розташування елементів залежно від розміру вікна браузера.

Bootstrap включає великий набір готових компонентів, таких як навігаційні панелі (navbars), модальні вікна (modals), випадаючі списки (dropdowns), каруселі (carousels), поповери (popovers) та інші. Ці компоненти легко інтегруються у веб-додаток за допомогою простих HTML-структур і класів CSS, що значно зменшує час розробки та дозволяє швидко створювати сучасні інтерфейси. Кожен компонент має добре задокументовані приклади та інструкції, що допомагають розробникам легко налаштовувати їх під свої потреби.

Фреймворк Bootstrap також включає велику кількість утилітних класів (utility classes), які дозволяють швидко застосовувати стилі до елементів без необхідності написання додаткового CSS. Ці класи охоплюють широкий спектр стилів, таких як відступи (margins), поля (paddings), кольори (colors), шрифти (fonts), вирівнювання (alignment) та інші. Використання утилітних класів значно спрощує процес стилізації елементів та забезпечує консистентність у зовнішньому вигляді додатку.

2.3 Аналіз фреймворків для написання бази данних веб-застосунку

2.3.1 Реляційні та нереляційні бази даних

Реляційні та нереляційні бази даних є двома основними типами систем управління базами даних (СУБД), які використовуються для зберігання та управління даними в різних додатках. Кожен тип бази даних має свої унікальні характеристики, переваги та недоліки, що робить їх відповідними для різних сценаріїв використання.

Реляційні бази даних (РБД) засновані на реляційній моделі, запропонованій Едгаром Коддом у 1970-х роках. Вони зберігають дані у вигляді таблиць (або відносин), де кожен рядок представляє окремий запис, а кожен стовпець — атрибут цього запису. РБД використовують структуру таблиць з чітко визначеними схемами, що забезпечує цілісність даних і дозволяє легко виконувати складні запити за допомогою мови SQL (Structured Query Language). Відомі реляційні СУБД включають MySQL, PostgreSQL, Oracle Database та Microsoft SQL Server.

Основною перевагою реляційних баз даних є їх здатність забезпечувати цілісність даних через використання ключів (primary keys) та зовнішніх ключів (foreign keys), що дозволяє встановлювати зв'язки між таблицями та підтримувати референтну цілісність. Це забезпечує високу надійність і консистентність даних, що є критично важливим для багатьох бізнес-додатків, таких як банківські системи, системи управління замовленнями та CRM-системи.

РБД також мають потужний механізм транзакцій, який забезпечує властивості ACID (атомарність, консистентність, ізоляція, довговічність). Це означає, що всі операції з даними виконуються як єдине ціле, і у випадку збою всі зміни відміняються, що гарантує цілісність даних навіть у разі виникнення помилок або збоїв. Ця особливість робить реляційні бази даних ідеальними для додатків, де важливо забезпечити надійність і точність даних.

Недоліками реляційних баз даних є їх відносна складність у налаштуванні та масштабуванні, особливо для дуже великих наборів даних або додатків з високими

вимогами до продуктивності. Масштабування реляційних баз даних зазвичай здійснюється вертикально (додаванням ресурсів до одного сервера), що може бути обмежено фізичними можливостями апаратного забезпечення. Горизонтальне масштабування (розподіл даних між кількома серверами) є складнішим у реалізації та потребує додаткових механізмів, таких як шардінг або реплікація.

Нереляційні бази даних (NoSQL) включають широкий спектр технологій, що не використовують реляційну модель для зберігання даних. Вони призначені для зберігання великих обсягів даних, часто з різноманітними структурами, що не підходять для традиційних реляційних баз даних. Типи NoSQL баз даних включають документоорієнтовані (MongoDB, CouchDB), ключ-значення (Redis, Riak), графові (Neo4j, JanusGraph) та колоночні (Cassandra, HBase) бази даних.

Основною перевагою нереляційних баз даних є їх здатність до горизонтального масштабування, що дозволяє легко додавати нові сервери до кластеру для збільшення обсягу зберігання та продуктивності. Це робить їх ідеальними для великих веб-додатків, соціальних мереж, аналітичних систем і додатків, що обробляють великі обсяги різнорідних даних у реальному часі.

Однак, нереляційні бази даних мають свої недоліки, серед яких відсутність стандартизованої мови запитів, що може ускладнювати інтеграцію з існуючими системами та інструментами. Крім того, вони зазвичай не забезпечують властивості ACID, що може бути неприйнятним для додатків, де важлива висока надійність і консистентність даних. Багато NoSQL баз даних забезпечують лише "зрештою консистентність" (eventual consistency), що означає, що дані можуть бути тимчасово неконсистентними під час оновлень, але згодом стають консистентними.

Таким чином для даної дипломної роботи буде використана реляційні бази даних надають безліч переваг для додатків, де важлива структурована схема, цілісність даних та надійність транзакцій. Їх потужні можливості для управління даними та підтримка стандартної мови запитів роблять їх незамінними для багатьох

критично важливих веб-додатків. Далі буде проаналізовані наявні реляційні бази даних.

2.3.2 MySQL

MySQL — це потужна система управління базами даних (СУБД), яка використовується для створення, управління та взаємодії з реляційними базами даних. Вона була розроблена в 1995 році шведською компанією MySQL AB, а згодом придбана Sun Microsystems у 2008 році та Oracle Corporation у 2010 році. MySQL є однією з найпопулярніших СУБД у світі, відома своєю продуктивністю, надійністю та легкістю у використанні, що робить її ідеальним вибором для багатьох додатків, від веб-сайтів до складних корпоративних систем.

MySQL надає розробникам можливість створювати і керувати базами даних за допомогою мови SQL (Structured Query Language). Ця мова дозволяє здійснювати різноманітні операції з даними, такі як вставка, оновлення, видалення та вибірка даних, а також виконання складних запитів та агрегатних функцій. Завдяки своїй сумісності зі стандартом SQL, MySQL забезпечує високу гнучкість та універсальність, що дозволяє використовувати її у різних галузях.

MySQL також забезпечує високу надійність та цілісність даних. Вона підтримує транзакції з властивостями ACID (атомарність, консистентність, ізоляція, довговічність), що гарантує коректність та надійність виконання операцій навіть у разі збою системи або інших непередбачених обставин. Це досягається завдяки використанню журналів змін (transaction logs), які дозволяють відновлювати базу даних до консистентного стану у разі збою.

MySQL надає потужні засоби для забезпечення безпеки даних. Вона підтримує різноманітні методи аутентифікації користувачів, включаючи паролі, SSL-сертифікати та плагіни аутентифікації. Крім того, MySQL дозволяє налаштовувати політики контролю доступу на різних рівнях, від рівня бази даних до рівня окремих таблиць і колонок, що забезпечує високий рівень захисту даних від несанкціонованого доступу.

Проте MySQL має і деякі недоліки. Одним з них є обмежена підтримка складних транзакцій і об'єктів, таких як тригери, збережені процедури та користувацькі функції, у порівнянні з іншими реляційними СУБД, такими як PostgreSQL чи Oracle. Це може бути обмеженням для деяких додатків, що вимагають розширених можливостей для обробки бізнес-логіки безпосередньо на рівні бази даних.

Іншим недоліком MySQL є складність у реалізації горизонтального масштабування. Хоча MySQL підтримує реплікацію та шардінг, їх налаштування та управління можуть бути складними та вимагати значних зусиль з боку адміністраторів баз даних. Це може бути проблемою для великих систем, що потребують високої масштабованості та продуктивності.

MySQL також може мати проблеми з продуктивністю при роботі з дуже великими обсягами даних або складними запитами. Висока продуктивність MySQL досягається за рахунок ефективних механізмів індексації та оптимізованих алгоритмів зберігання даних, проте при роботі з великими базами даних або складними запитами може виникати необхідність у додатковій оптимізації або використанні альтернативних СУБД.

2.3.3 PostgreSQL

PostgreSQL — це потужна, об'єктно-реляційна система управління базами даних (ОРСУБД), яка підтримує сучасні стандарти SQL та забезпечує широкий спектр функціональних можливостей для зберігання та обробки даних. Вона була розроблена як частина проєкту POSTGRES в Університеті Каліфорнії, Берклі, під керівництвом Майкла Стоунбрейкера у 1986 році. З часом PostgreSQL перетворився на одну з найпопулярніших та найнадійніших СУБД, широко використовуваних у різних галузях, від малих підприємств до великих корпорацій.

Однією з основних переваг PostgreSQL є його відповідність стандартам SQL і підтримка розширених можливостей SQL, таких як складні запити, підзапити, транзакції, представлення (views), індекси, тригери, збережені процедури та інші.

PostgreSQL підтримує типи даних першого класу, включаючи стандартні числові, текстові, дату і час, а також розширювані типи даних, такі як масиви, JSON, XML та інші. Це забезпечує високу гнучкість і потужність при роботі з різними типами даних.

PostgreSQL також має потужну систему розширюваності. Вона дозволяє користувачам створювати свої власні типи даних, функції, операції та індекси, що значно розширює можливості СУБД. Користувацькі функції можуть бути написані на різних мовах програмування, таких як PL/pgSQL, PL/Tcl, PL/Perl, PL/Python та інших. Це забезпечує високу гнучкість у розробці специфічних рішень для різних бізнес-завдань.

Транзакційна система PostgreSQL відповідає властивостям ACID (атомарність, консистентність, ізоляція, довговічність), що гарантує надійність і коректність виконання операцій з даними. PostgreSQL підтримує багатоверсійну паралельність (MVCC), що дозволяє виконувати одночасні читання і записи без блокувань, забезпечуючи високу продуктивність і масштабованість. Це особливо важливо для додатків з високими вимогами до продуктивності та надійності.

Потужні засоби для забезпечення безпеки даних є ще однією перевагою PostgreSQL. Вона підтримує аутентифікацію користувачів за допомогою паролів, SSL-сертифікатів, Kerberos, LDAP та інших методів. PostgreSQL також забезпечує розширений контроль доступу до даних на різних рівнях, включаючи рівень бази даних, таблиць, рядків і колонок, що дозволяє налаштовувати детальні політики безпеки відповідно до вимог бізнесу.

Незважаючи на свої численні переваги, PostgreSQL має і деякі недоліки. Одним з них є складність налаштування та адміністрування, особливо для користувачів, які не мають досвіду роботи з реляційними базами даних. PostgreSQL надає широкий спектр можливостей для налаштування, що може бути перевагою для досвідчених користувачів, але водночас створює додаткові труднощі для новачків.

Іншим недоліком є відносно висока вимога до апаратних ресурсів, особливо при роботі з великими обсягами даних або виконанні складних запитів. PostgreSQL може вимагати значних обсягів оперативної пам'яті та дискового простору для забезпечення оптимальної продуктивності, що може бути проблемою для деяких організацій.

PostgreSQL також може мати проблеми з продуктивністю при роботі з дуже великими обсягами даних або високими навантаженнями. Хоча PostgreSQL забезпечує високу продуктивність завдяки своїм оптимізованим алгоритмам зберігання даних та індексації, у деяких випадках може виникати необхідність у додатковій оптимізації або використанні спеціалізованих рішень для обробки великих даних.

2.3.2 Oracle

Oracle Database — це потужна система управління базами даних (СУБД), розроблена компанією Oracle Corporation. Вона є однією з найбільш популярних і широко використовуваних СУБД у світі, відома своєю високою продуктивністю, надійністю та масштабованістю. Oracle Database була вперше випущена в 1979 році і з того часу зазнала численних оновлень і поліпшень, що зробило її незамінною для багатьох великих підприємств та організацій, які потребують надійного зберігання та управління даними.

Основною перевагою Oracle Database є її висока продуктивність і здатність обробляти великі обсяги даних. Oracle використовує передові технології для оптимізації роботи з даними, включаючи потужні механізми індексації, стиснення даних та інтелектуальне управління пам'яттю. Це дозволяє забезпечувати швидкий доступ до даних і ефективне виконання складних запитів, що є критично важливим для багатьох бізнес-додатків, таких як банківські системи, системи управління ланцюгами поставок та інші.

Oracle Database також відома своєю надійністю і здатністю забезпечувати високу доступність даних. Вона підтримує різноманітні механізми резервного

копіювання і відновлення, включаючи повні, диференційовані та інкрементні резервні копії. Крім того, Oracle забезпечує можливості для реплікації даних та автоматичного перемикання на резервні сервери у разі збою основного сервера, що гарантує безперервність бізнес-процесів і мінімізує час простою системи.

Oracle Database має потужну систему безпеки, яка забезпечує захист даних від несанкціонованого доступу та забезпечує відповідність вимогам різних регуляторних норм. Вона підтримує різні методи аутентифікації, включаючи паролі, SSL-сертифікати, Kerberos та інші. Крім того, Oracle забезпечує детальний контроль доступу до даних на різних рівнях, включаючи рівень бази даних, таблиць, рядків і колонок, що дозволяє налаштовувати політики безпеки відповідно до вимог бізнесу.

Однак, Oracle Database має і деякі недоліки. Одним з них є висока вартість ліцензій та підтримки, що може бути значним бар'єром для малих та середніх підприємств. Вартість ліцензії залежить від різних факторів, таких як кількість користувачів, обсяг даних та функціональність, що робить її доступною переважно для великих організацій з високими вимогами до управління даними.

Oracle Database також може мати проблеми з продуктивністю при роботі з дуже великими обсягами даних або складними запитами. Незважаючи на потужні можливості для оптимізації роботи з даними, у деяких випадках може виникати необхідність у додатковій оптимізації або використанні спеціалізованих рішень для обробки великих даних. Це може включати налаштування індексів, оптимізацію запитів та використання додаткових модулів для підвищення продуктивності.

Ще одним можливим недоліком є залежність від постачальника (vendor lock-in), що може обмежувати гнучкість організації у виборі технологій та платформ. Оскільки Oracle Database є власницькою технологією, її використання може вимагати спеціалізованих знань і навичок, що може ускладнювати міграцію до інших СУБД або платформ у разі необхідності.

2.4 Висновок

У цьому розділі було проведено детальний аналіз мов програмування та фреймворків, які використовуються для розробки веб-застосунку. Вибір технологій був обґрунтований з точки зору їх функціональних можливостей, продуктивності, легкості використання, а також підтримки спільноти розробників. Основною мовою програмування було обрано JavaScript з використанням Node.js та Python з Django, що дозволяє реалізувати різноманітні сценарії розробки веб-застосунків.

Для розробки серверної частини веб-застосунку було обрано Node.js, який забезпечує високу продуктивність завдяки своїй асинхронній архітектурі. Використання Express.js як основного фреймворку дозволяє створювати швидкі та масштабовані сервери з мінімальними витратами часу на налаштування. Express.js є популярним вибором серед розробників завдяки своїй гнучкості та великій кількості плагінів.

GraphQL був інтегрований для оптимізації запитів до сервера, дозволяючи клієнтам отримувати лише необхідні дані, що знижує навантаження на сервер та покращує продуктивність застосунку. Nodemailer був використаний для реалізації функцій надсилання електронних листів, що є необхідним для забезпечення функціоналу автентифікації та сповіщень користувачів.Для взаємодії з базою даних PostgreSQL було обрано ORM Sequelize, який надає інтуїтивно зрозумілий інтерфейс для роботи з базою даних, забезпечуючи легкість виконання CRUD-операцій та управління міграціями даних.

Клієнтська частина веб-застосунку була реалізована з використанням бібліотеки React, що дозволяє створювати високодинамічні та інтерактивні користувацькі інтерфейси. React забезпечує компонентний підхід до розробки, що сприяє легкому повторному використанню коду та підвищує ефективність розробки.Для управління станом застосунку був використаний MobX, який забезпечує простий та ефективний механізм реактивного програмування. Використання MobX дозволяє зменшити кількість коду, необхідного для

управління станом, та спрощує підтримку додатка.Для забезпечення адаптивного дизайну та швидкого створення стильових рішень було обрано Bootstrap, який надає широкий набір готових компонентів та стилів, що значно прискорює процес розробки інтерфейсів.

Для зберігання даних було обрано реляційну базу даних PostgreSQL, яка забезпечує надійне зберігання даних та підтримує складні запити. PostgreSQL відома своєю стабільністю, високою продуктивністю та відповідністю стандартам SQL, що робить її ідеальним вибором для веб-застосунків з високими вимогами до надійності та безпеки даних.

Загалом, вибір технологій для розробки веб-застосунку був обґрунтований їх відповідністю вимогам проекту та сучасним тенденціям у розробці веб-застосунків. Використання JavaScript з Node.js та Python з Django, а також інтеграція сучасних фреймворків та інструментів, таких як Express.js, GraphQL, Nodemailer, Sequelize, React, MobX та Bootstrap, забезпечує високу продуктивність, гнучкість та масштабованість розробленого веб-застосунку. PostgreSQL виступає надійною базою даних, що гарантує безпеку та стабільність збереження даних. Таким чином, обрана технологічна стек дозволяє ефективно вирішувати завдання, поставлені перед веб-застосунком, та забезпечує його подальший розвиток і підтримку.

3. ОПИС РОЗРОБЛЕНИХ АЛГОРИТМІВ ТА МОДУЛІВ

3.1 Вимоги до розроблюваного веб-застосунку

Розробка веб-застосунку для взаємодії з 3D моделями має на меті створення зручного та функціонального інструменту, який дозволяє користувачам переглядати, шукати, купувати та керувати 3D моделями. Основною метою є забезпечення високої якості користувацького досвіду шляхом інтеграції інтуїтивно зрозумілого інтерфейсу, ефективних функціональних можливостей та безпечного середовища для здійснення транзакцій.

Функціональні вимоги визначають конкретні дії та процеси, які система повинна виконувати для задоволення потреб користувачів. Вони включають в себе:

- Реєстрація нового користувача: можливість створити новий обліковий запис з використанням електронної пошти.
- Вхід в систему: аутентифікація користувачів за допомогою електронної пошти/пароля.
- Перегляд каталогу: користувачі повинні мати можливість переглядати різноманітні 3D моделі, відфільтровані за категоріями, ціною, популярністю тощо.
- Детальний опис моделі: сторінка з детальним описом моделі, включаючи зображення, технічні характеристики, ціни та відгуки.
- Розширений пошук: функція пошуку з використанням ключових слів, тегів та категорій.
- Фільтрація: можливість фільтрувати результати пошуку за різними критеріями, такими як ціна, рейтинг, дата додавання тощо.
- Додавання до кошика: користувачі можуть додавати моделі до кошика для подальшої покупки.

- Додавання до Вподобаних: користувачі можуть додавати вподобані 3D моделі до ‘Вподобані’ для відслідковування оновлень або знижок.
- Оформлення замовлення: процес оформлення замовлення з вибором способу оплати та підтвердження покупки.
- Завантаження моделей: автори можуть завантажувати свої 3D моделі, додавати описи, встановлювати ціни та переглядати статистику завантажень.
- Оцінювання моделей: можливість користувачам залишати оцінки та відгуки про моделі.

Нефункціональні вимоги описують характеристики та якості веб-застосунку, які впливають на його продуктивність, зручність використання, безпеку та масштабованість.

- Час завантаження сторінки: сторінки повинні завантажуватися швидко, навіть при великому навантаженні.
- Захист даних користувачів: використання шифрування для захисту особистої інформації та платіжних даних.
- Оптимізація ресурсів: використання кешування, стиснення зображень та інших методів для оптимізації використання ресурсів.
- Масштабованість бази даних: база даних повинна бути спроектована таким чином, щоб легко розширюватися при зростанні кількості даних.
- Аутентифікація та авторизація: забезпечення безпечного входу та контролю доступу до різних функцій системи.
- Підтримка високої доступності: забезпечення максимально можливого часу безвідмовної роботи сервісу.
- Інтуїтивно зрозумілий інтерфейс: забезпечення зручної навігації та взаємодії з системою для користувачів з різним рівнем підготовки.

3.2 Вимоги до інтерфейсу

Основна концепція розроблювального інтерфейсу для веб-застосунку включає ряд принципів та методів, спрямованих на забезпечення зручності використання, ефективності та привабливості для кінцевого користувача. Ці принципи охоплюють різні аспекти дизайну та розробки, які в сукупності утворюють гармонійний і функціональний продукт.

UX (User Experience) є фундаментом, на якому будується весь інтерфейс. Добре продуманий UX забезпечує інтуїтивність, зручність та привабливість інтерфейсу, що дозволяє користувачам легко орієнтуватися, швидко знаходити потрібну інформацію та виконувати завдання без зайвих зусиль. Це не тільки покращує користувацький досвід, але й підвищує лояльність користувачів, зменшує ймовірність помилок і збільшує загальну продуктивність веб-застосунку. Інтеграція принципів UX у розробку допомагає створити продукт, який відповідає потребам і очікуванням користувачів, забезпечуючи конкурентну перевагу на ринку.

Простота та мінімалізм. Веб-застосунок повинен бути інтуїтивно зрозумілим, з мінімальною кількістю необхідних дій для досягнення користувачем своєї мети. Простота означає уникнення зайвих візуальних елементів, що можуть відволікати або заплутувати користувача. Мінімалізм вимагає, щоб кожен елемент інтерфейсу мав своє конкретне призначення та був візуально ясним і зрозумілим. Це дозволяє користувачам швидко орієнтуватися в застосунку і знаходити необхідну інформацію або функціонал.

Дотримання єдиного стилю. Використання однакових стилів, шрифтів, кольорів та інших візуальних елементів по всьому застосунку забезпечує узгодженість та передбачуваність для користувачів. Стандартні компоненти, такі як кнопки, поля введення та випадаючі списки, повинні використовуватися згідно з загальноприйнятими нормами, щоб користувачі могли легко зрозуміти, як взаємодіяти з застосунком.

Легкість навігації. Структура контенту повинна бути логічною та організованою, з чіткими заголовками та навігаційними меню. Це дозволяє користувачам швидко знайти потрібну інформацію. Включення функції пошуку також є корисним, оскільки вона дозволяє користувачам швидко знаходити необхідні дані або функції без потреби переглядати всі розділи застосунку.

3.3 Опис веб-застосунку

Коли гість переходить на головну сторінку веб-застосунку, він має доступ до основного меню, яке дозволяє переміщуватися по різних розділах сайту. З головної сторінки користувач може перейти до каталогу 3D моделей або використовувати пошукову функцію для швидкого знаходження конкретних моделей. Пошукова функція розміщена в головному меню і дозволяє вводити ключові слова для фільтрації доступних 3D моделей.

Каталог 3D моделей містить повний перелік доступних моделей, які користувач може переглядати. Користувач може переглядати прев’ю моделей і переходити до детального опису кожної моделі, що його зацікавила.

Коли користувач знаходить 3D модель, яка його цікавить, він може перейти на сторінку з детальним описом цієї моделі. На цій сторінці відображається повна інформація про модель, включаючи її назву, опис, автора, дату створення, характеристики та ціна. Також доступні попередній перегляд моделі та відгуки інших користувачів. Авторизовані користувачі можуть залишати коментарі, оцінювати модель та додавати її в "улюблені".

Якщо користувач бажає придбати 3D модель, він може додати її до кошика. При спробі додати модель до кошика, система перевіряє, чи користувач авторизований. Якщо користувач не авторизований, він буде перенаправлений на сторінку авторизації або реєстрації. Після успішної авторизації, вибрана 3D модель додається до кошика.

Користувач може переглядати свій кошик, де відображаються всі додані моделі, загальна вартість та доступні опції для редагування замовлення. Після того як користувач перевірить своє замовлення, він може перейти до сторінки оплати. На сторінці оплати користувач вводить необхідні платіжні дані та підтверджує покупку. Після успішної оплати обрані 3D моделі відправляються на електронну пошту користувача.

Авторизовані користувачі мають доступ до свого профілю, де вони можуть керувати своїми 3D моделями. Користувачі можуть завантажувати нові 3D моделі, надавати їм опис та категорії, а також переглядати коментарі своїх моделей.

Авторизовані користувачі можуть залишати коментарі на сторінках 3D моделей, що сприяє створенню активної спільноти та обміну думками про моделі. Коментарі можуть включати текст, рейтинги та відповіді на інші коментарі. Користувачі також можуть додавати моделі до списку "улюблених". Це дозволяє їм швидко знаходити улюблені моделі, відслідковувати оновлення та нові версії цих моделей.

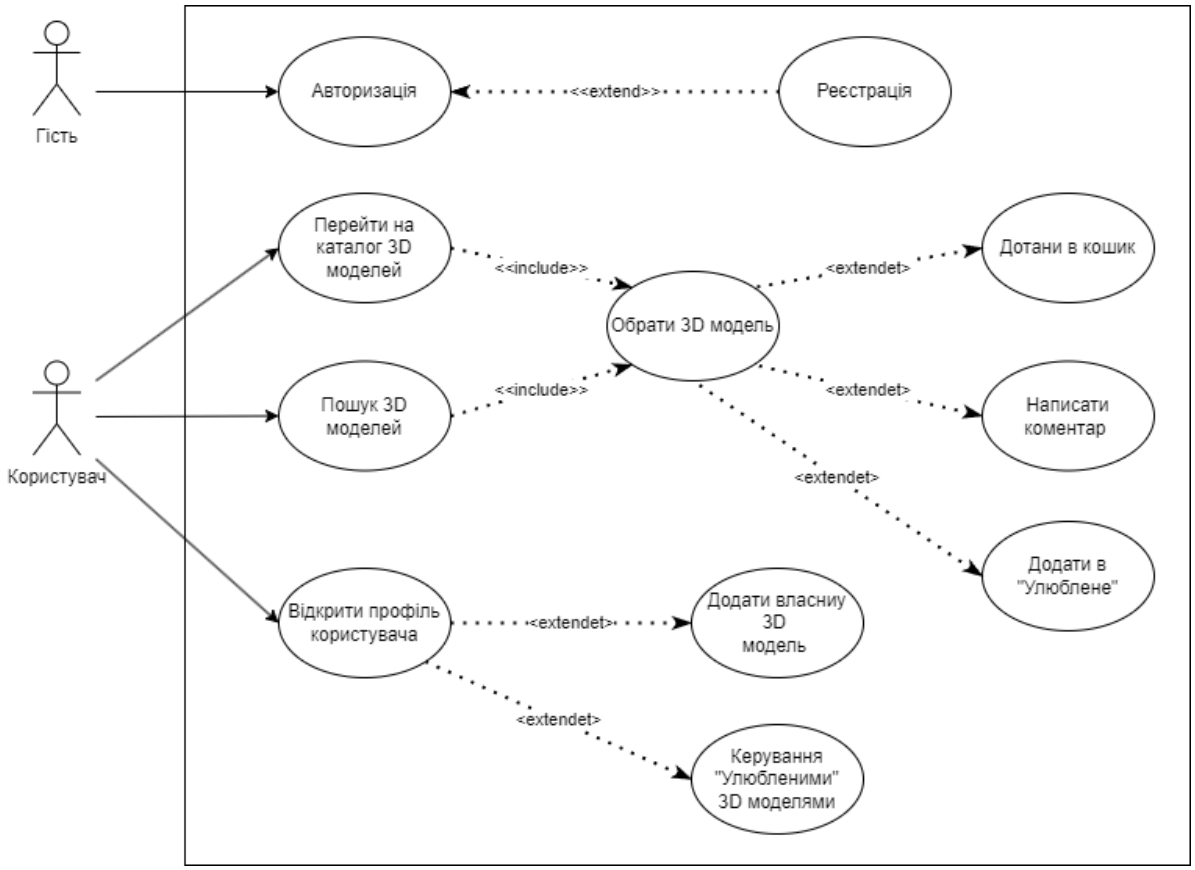


Рис. 3.1 Діаграма варіантів використання

Взаємодія користувача з системою:

1. Перегляд та пошук моделей: Користувач заходить на головну сторінку, переходить до каталогу або використовує пошук для знаходження моделей.
2. Детальна інформація: Користувач переходить на сторінку детального опису моделі, переглядає інформацію та відгуки.
3. Додавання до кошика: Якщо користувач хоче придбати модель, він додає її до кошика, після чого проходить процес авторизації (за необхідності).
4. Оформлення замовлення: Користувач переглядає кошик, переходить до сторінки оплати, вводить платіжні дані та підтверджує покупку.
5. Отримання моделі: Після оплати користувач отримує модель на електронну пошту.

6. Управління профілем: Авторизовані користувачі можуть завантажувати свої моделі, залишати коментарі, додавати моделі до "улюблених" та керувати своїм профілем.

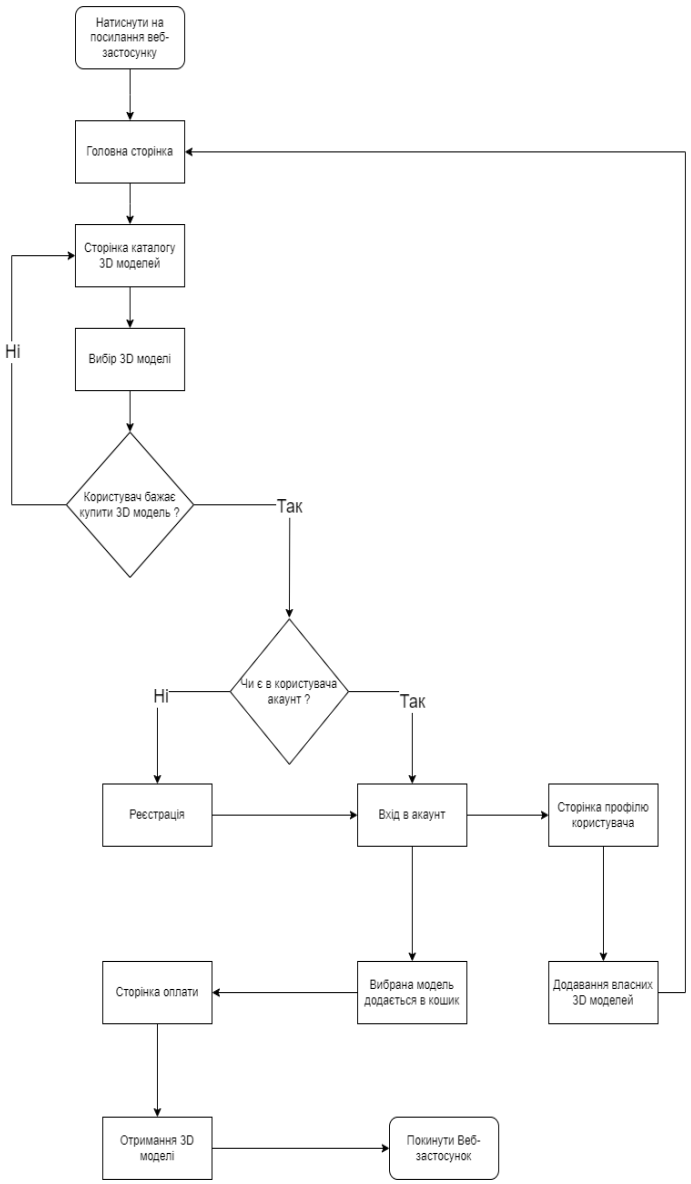


Рис. 3.2 Діаграма User Flow

3.4 Архітектура системи

Архітектура системи "Клієнт-Сервер" є основоположною моделлю для сучасних веб-застосунків, що дозволяє ефективно розподіляти обчислювальні

ресурси та зберігання даних між клієнтськими і серверними компонентами. У рамках дипломної роботи, системна архітектура побудована на використанні Object-Relational Mapping (ORM), шаблонів REST API та формату даних JSON, що забезпечує зручність розробки, масштабованість і продуктивність системи.

Основною складовою клієнтської частини є інтерфейс користувача, який взаємодіє з сервером через веб-браузер. Клієнтська частина відповідає за відображення даних, обробку подій користувача та відправлення запитів до сервера. Вона завантажується і виконується на стороні клієнта, забезпечуючи інтерактивний та динамічний досвід для кінцевих користувачів. Клієнтська частина включає різноманітні компоненти, такі як форми, кнопки, таблиці, графіки та інші елементи, що забезпечують зручність взаємодії та функціональність веб-додатка.

Серверна частина виконує ключові обчислювальні операції, управління бізнес-логікою, обробку запитів та взаємодію з базами даних. Вона реалізована на платформі Node.js, що забезпечує високу продуктивність і неблокуюче введення-виведення, що особливо важливо для обробки одночасних запитів. Використання фреймворку Express.js дозволяє створювати гнучкі та ефективні серверні додатки, забезпечуючи зручне управління маршрутами, обробку запитів та інтеграцію з різними середовищами виконання (middleware).

Основою для взаємодії між клієнтом і сервером є API, що реалізовано за допомогою шаблонів REST (Representational State Transfer). REST API забезпечує стандартизований спосіб доступу до ресурсів системи за допомогою HTTP-методів, таких як GET, POST, PUT і DELETE. Кожен ресурс в системі має унікальний URL, що дозволяє ідентифікувати його і виконувати необхідні операції. REST API визначає чіткі правила для структури URL, методів запитів, статус-кодів відповідей та форматів даних, що забезпечує зручність використання і підтримки системи.

Формат даних JSON (JavaScript Object Notation) використовується для обміну даними між клієнтом і сервером. JSON є легким і читабельним форматом, який дозволяє ефективно серіалізувати та десеріалізувати дані. Клієнт відправляє запити до сервера у форматі JSON, включаючи необхідні параметри і дані, сервер обробляє

ці запити, виконує необхідні операції і повертає відповіді також у форматі JSON. Це забезпечує швидку і надійну передачу даних, зменшуючи навантаження на мережу і підвищуючи продуктивність системи.

Управління базами даних здійснюється за допомогою ORM (Object-Relational Mapping), що забезпечує зручний інтерфейс для взаємодії з реляційними базами даних. ORM дозволяє працювати з базами даних, використовуючи об'єктно-орієнтований підхід, що спрощує процес створення, читання, оновлення та видалення даних. У даному проекті використовується ORM Sequelize, який забезпечує абстракцію роботи з базою даних PostgreSQL. Sequelize дозволяє легко визначати моделі даних, встановлювати зв'язки між ними, а також автоматично генерувати SQL-запити для взаємодії з базою даних.

База даних PostgreSQL є ключовим компонентом системи, що забезпечує надійне зберігання та управління даними. PostgreSQL підтримує складні запити, транзакції та багаторівневу цілісність даних, що гарантує надійність і консистентність даних. Використання PostgreSQL у поєднанні з ORM Sequelize дозволяє ефективно взаємодіяти з даними, забезпечуючи високу продуктивність та гнучкість системи.

Архітектура системи також включає використання аутентифікації та авторизації для забезпечення безпеки доступу до ресурсів. Це досягається за допомогою токенів доступу, таких як JWT (JSON Web Token), які забезпечують безпечну передачу інформації про користувача між клієнтом і сервером. Серверна частина обробляє процеси аутентифікації та авторизації, перевіряючи облікові дані користувачів і надаючи доступ до захищених ресурсів відповідно до прав доступу.

Клієнтська частина взаємодіє з сервером через REST API, відправляючи запити і отримуючи відповіді у форматі JSON. Це дозволяє клієнту динамічно оновлювати інтерфейс відповідно до отриманих даних, забезпечуючи інтерактивний і зручний досвід користувача. Клієнтська частина може виконувати операції, такі як отримання списку ресурсів, створення нових записів, оновлення

існуючих даних та видалення записів, використовуючи відповідні методи REST API.

Забезпечення продуктивності і масштабованості системи досягається за рахунок оптимізації взаємодії між клієнтом і сервером, використання кешування, асинхронної обробки запитів і оптимізації бази даних. Це дозволяє системі ефективно обробляти великі обсяги даних і підтримувати високу швидкість відгуку навіть при значних навантаженнях.

Загальна архітектура системи "Клієнт-Сервер" з використанням ORM, шаблонів REST API та JSON забезпечує високу продуктивність, надійність, безпеку та зручність у використанні. Вона дозволяє ефективно розподіляти обчислювальні ресурси, забезпечувати гнучку взаємодію між клієнтськими і серверними компонентами та підтримувати масштабованість і підтримуваність системи.

3.5 Архітектура серверної частини

Архітектура розробленої клієнтської та серверної частини веб-застосунку базується на використанні шаблонів REST API для забезпечення ефективної взаємодії між клієнтом і сервером. Нижче наведено детальний опис основних компонентів обох частин згідно з наданою схемою нище (Рис. 3.3).

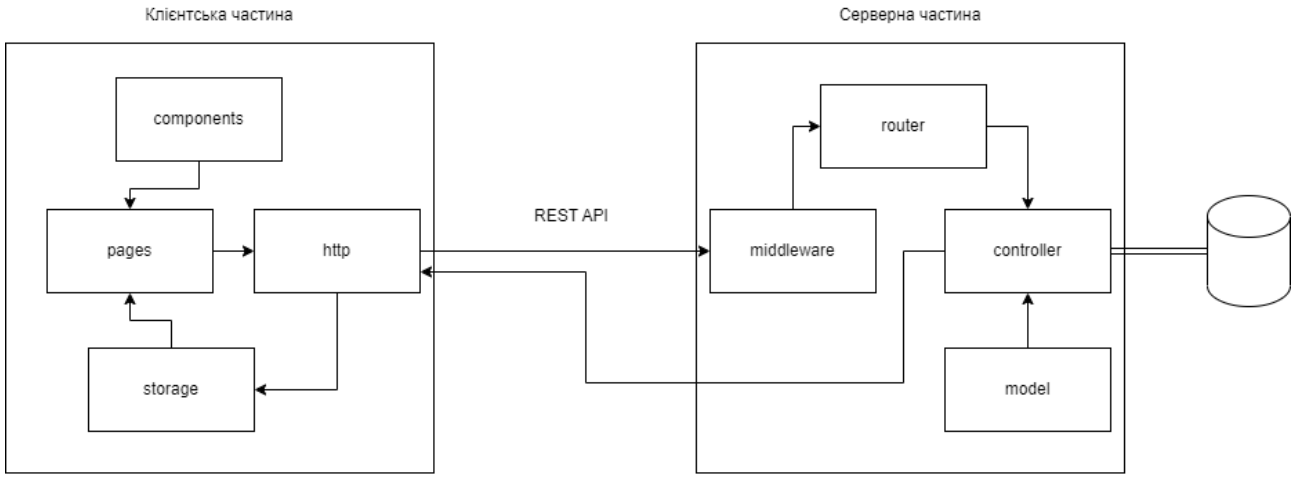


Рис. 3.3 Архітектура застосунку

Компонент "components" відповідає за реалізацію багаторазово використовуваних елементів інтерфейсу користувача, таких як кнопки, форми, модальні вікна та інші UI-елементи. Ці компоненти є базовими будівельними блоками, які використовуються для побудови сторінок (pages).

Компонент "pages" відповідає за складання повних сторінок веб-застосунку з окремих компонентів. Кожна сторінка включає набір компонентів і відповідає за відображення певного контенту та функціональності. Наприклад, сторінка може містити форму входу, список продуктів або інформаційну панель користувача.

Компонент "http" відповідає за здійснення HTTP-запитів до серверної частини. Він забезпечує взаємодію з REST API сервера, відправляючи запити на отримання, створення, оновлення та видалення даних. Компонент http обробляє відповіді сервера і передає отримані дані до відповідних сторінок і компонентів.

Компонент "storage" відповідає за зберігання даних на клієнтській стороні. Це може бути реалізовано через локальне сховище (localStorage), сесійне сховище (sessionStorage) або інші методи зберігання. Storage зберігає стан додатку, наприклад, інформацію про аутентифікацію користувача або тимчасові дані, що використовуються на сторінках.

Компонент "router" відповідає за маршрутизацію HTTP-запитів до відповідних контролерів. Коли сервер отримує запит від клієнта, маршрутизатор визначає, який контролер має обробити цей запит на основі URL та HTTP методу (GET, POST, PUT, DELETE тощо). Маршрутизатор діє як посередник, направляючи запити до відповідних обробників.

Компонент "middleware" є проміжним шаром, який обробляє запити до того, як вони досягнуть контролера. Middleware функції можуть виконувати різні завдання, такі як аутентифікація користувачів, логування запитів, обробка помилок, парсинг JSON, налаштування CORS (Cross-Origin Resource Sharing) тощо. Вони допомагають розширювати функціональність застосунку та забезпечують додатковий рівень контролю і безпеки.

Компонент "controller" є основним обробником логіки запитів. Контролери отримують запити від маршрутизатора, виконують бізнес-логіку і взаємодіють з моделями для доступу до даних. Після обробки запиту контролер формує відповідь і повертає її клієнту. Контролери забезпечують розподіл відповідальностей, ізолюючи логіку обробки запитів від інших компонентів системи.

Компонент "model" відповідає за взаємодію з базою даних. Моделі визначають структуру даних і методи для доступу до них. Вони використовують ORM (Object-Relational Mapping), у цьому випадку Sequelize, для взаємодії з реляційною базою даних PostgreSQL. Моделі забезпечують абстракцію роботи з даними, дозволяючи виконувати операції створення, читання, оновлення та видалення (CRUD) без необхідності написання SQL-запитів.

База даних є центральним сховищем даних для веб-застосунку. У даній архітектурі використовується PostgreSQL, яка забезпечує надійне зберігання даних, підтримку транзакцій і складних запитів. PostgreSQL інтегрується з ORM Sequelize, що дозволяє моделям взаємодіяти з базою даних через об'єктно-орієнтований інтерфейс.

Серверна частина взаємодіє з клієнтською частиною через REST API, який забезпечує стандартизований спосіб обміну даними. Клієнтська частина відправляє HTTP-запити до REST API, які маршрутизуються, обробляються middleware, і передаються до відповідних контролерів. Контролери виконують необхідні операції, взаємодіють з моделями для доступу до даних і формують відповіді, які повертаються клієнту у форматі JSON.

Взаємодія компонентів:

1. Клієнтська частина відправляє HTTP-запит до серверної частини через REST API.
2. Router отримує запит і направляє його до відповідного контролера.
3. Middleware обробляє запит перед його передачею до контролера (наприклад, перевіряє аутентифікацію).

4. Controller виконує бізнес-логіку і взаємодіє з моделлю для виконання операцій з базою даних.
5. Model використовує ORM Sequelize для взаємодії з базою даних PostgreSQL.
6. Результати операцій повертаються через контролер і middleware до клієнта у форматі JSON.

3.6 Архітектура бази даних

Розроблена архітектура бази даних для веб-застосунку представлена через ERD-діаграмою (Entity-Relationship Diagram), яка відображає сутності (таблиці), їх атрибути (поля) та взаємозв'язки (зв'язки) між ними. Ця структура забезпечує організацію даних, що необхідні для функціонування веб-застосунку, з акцентом на забезпечення цілісності даних, продуктивності та масштабованості. Нижче наведено детальний опис кожної сутності та зв'язків між ними:

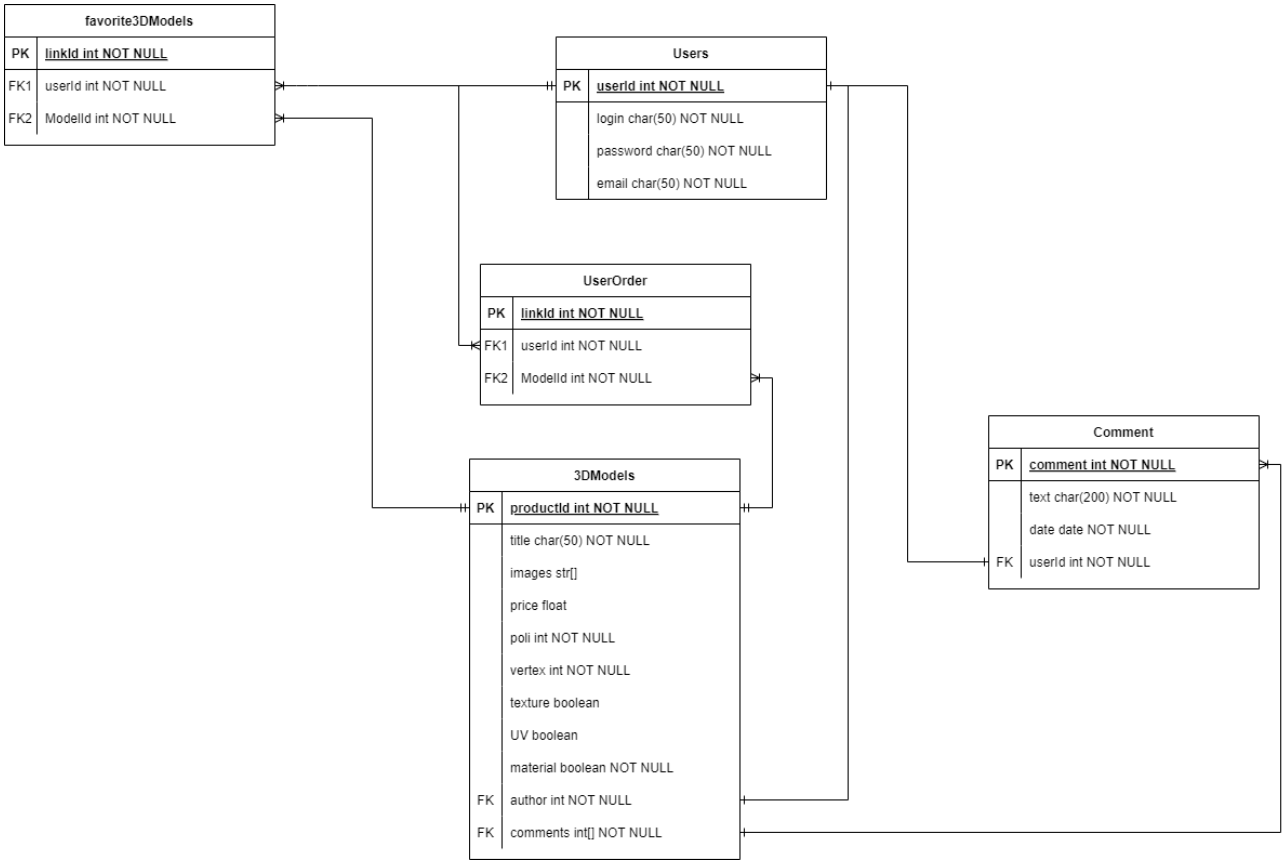


Рис. 3.4 Структура баз данних

3DModels сутність зберігає інформацію про 3D-моделі. modelID є первинним ключем, а userID – зовнішнім ключем, який пов'язує модель з користувачем, що її завантажив. Інші атрибути зберігають деталі про модель та час створення.

Таблиця 3.1

Поля таблиці 3DModels

Атрибут	**Тип даних**	**Опис**
productid	Integer(PK)	Унікальний ідентифікатор продукта
title	Text	Назва 3D моделі
images	String[]	Зображення 3D моделі
Price	Float	Ціна 3D моделі
poly	Integer	Кількість полігонів 3D моделі
vertex	Integer	Кількість вершин 3D моделі
texture	Boolean	Наявність текстури
uv	Boolean	Наявність UV-розгортки
material	Boolean	Наявність Матеріала
author	Integer(FK)	Зовнішній ключ з таблиці Users

Users сутність зберігає інформацію про користувачів системи. userID є первинним ключем, що забезпечує унікальність кожного користувача. Інші атрибути зберігають особисті дані користувача.

Таблиця 3.2

Поля таблиці Users

Атрибут	**Тип даних**	**Опис**
userid	Integer(PK)	Унікальний ідентифікатор користувача
login	Char	Ім'я користувача
userid	Integer(PK)	Унікальний ідентифікатор користувача
login	Char	Ім'я користувача

UserOrder сутність зберігає інформацію про замовлення користувачів. orderID є первинним ключем, userID і modelID – зовнішні ключі, що пов'язують замовлення з користувачем та моделлю відповідно. Інші атрибути зберігають деталі про замовлення.

Таблиця 3.3

Поля таблиці UserOrder

Атрибут	**Тип даних**	**Опис**
linkid	Integer(PK)	Унікальний ідентифікатор замовлення
userid	Integer(FK)	Зовнішній ключ з таблиці Users
modelid	Integer(FK)	Зовнішній ключ з таблиці 3DModels

Comment сутність зберігає інформацію про коментарі користувачів до 3D-моделей. commentID є первинним ключем, а userID і modelID – зовнішніми ключами, що пов'язують коментар з користувачем та 3D-моделлю відповідно. Інші атрибути зберігають текст коментаря та час його створення і оновлення.

Таблиця 3.4

Поля таблиці Comment

Атрибут	**Тип даних**	**Опис**
comment	Integer(PK)	Унікальний ідентифікатор коментарю
text	Char	Текст коментарю
date	date	Дата створення коментарю
userid	Integer(FK)	Зовнішній ключ з таблиці Users

favorite3dmodels сутність реалізує зв'язок багато-до-багатьох між користувачами та 3D-моделями. linkID є первинним ключем, userID та modelID є зовнішніми ключами, що пов'язують користувачів з улюбленими моделями.

Таблиця 3.5

Поля таблиці Favorite3DModels

Атрибут	**Тип даних**	**Опис**
linkid	Integer(PK)	Унікальний ідентифікатор запису
userid	Integer(FK)	Зовнішній ключ з таблиці Users
modelid	Integer(FK)	Зовнішній ключ з таб. 3DModels

4. ТЕСТУВАННЯ РОЗРОБЛЕНОГО WEB-СЕРВІСУ

4.1 Функціонал веб-застосунку

Інтерфейс розробленого веб-застосунку "GRIFON" структуровано таким чином, щоб забезпечити зручність користування для всіх категорій користувачів, незалежно від їхнього досвіду з подібними системами.

Рис. 4.1 Головна сторінка веб-застосунку

Головна сторінка є центральним елементом веб-застосунку. Вона має кілька ключових компонентів, які розташовані таким чином, щоб забезпечити легкий доступ до основних функцій і інформації.

“Логін/Реєстрація”: Ця кнопка дозволяє користувачам зареєструвати новий акаунт або увійти у вже існуючий. Реєстрація та автентифікація є необхідними для отримання повного доступу до всіх функцій веб-застосунку.

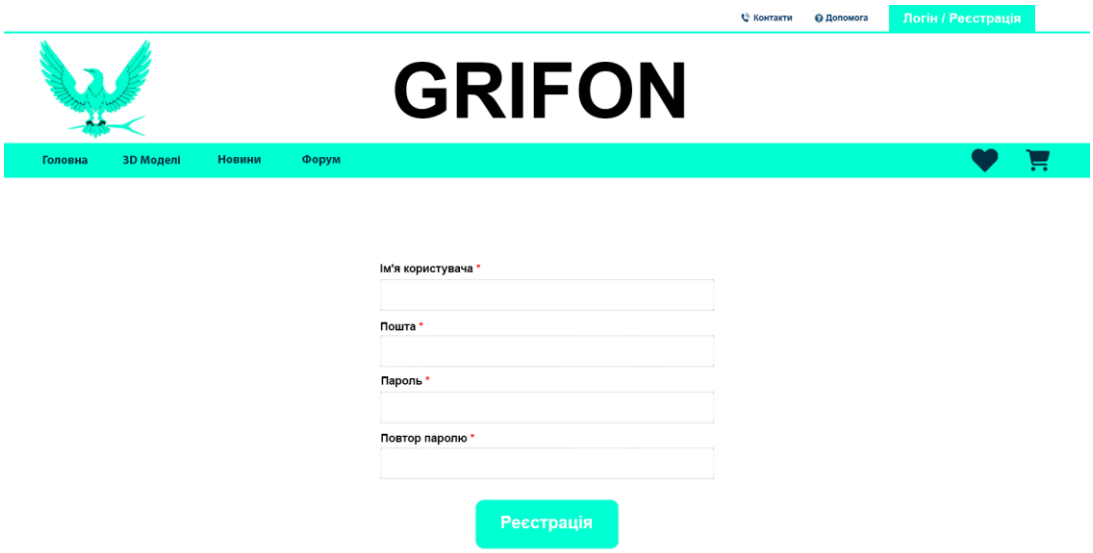


Рис. 4.2 Сторінка реєстрації

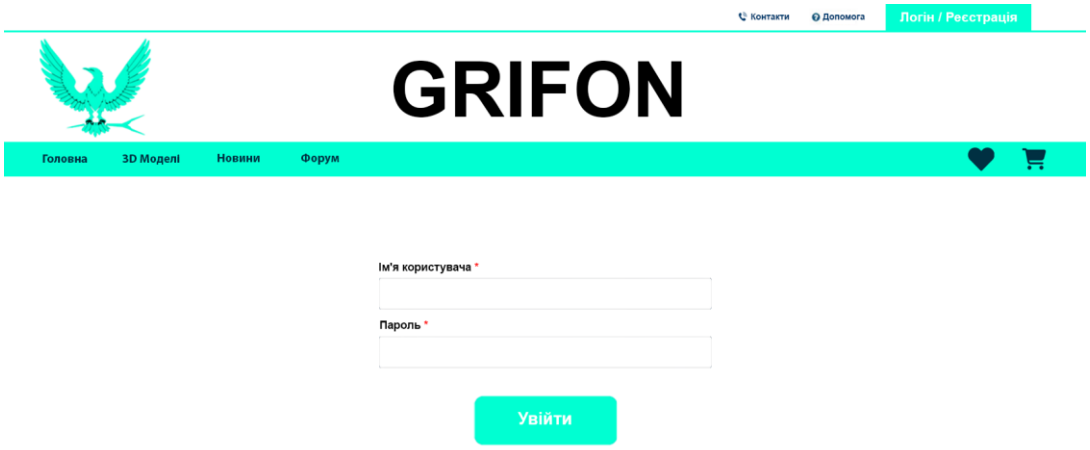


Рис. 4.3 Сторінка авторизації

“Контакти”: В цій секції користувач може знайти контактні дані розробників або адміністраторів веб-застосунку, а також посилання на сторінки GRIFON у соціальних мережах.

“Допомога”: Ця кнопка веде до розділу з відповідями на часто задавані питання (FAQ).

Під логотипом розміщена навігаційна панель, яка містить чотири основні посилання:

“Головна”: Посилання на головну сторінку, яка служить точкою відправлення для користувача і містить узагальнену інформацію та доступ до ключових функцій.

“3D Моделі”: Ця вкладка дозволяє як зареєстрованим, так і не зареєстрованим користувачам переглядати бібліотеку 3D моделей.

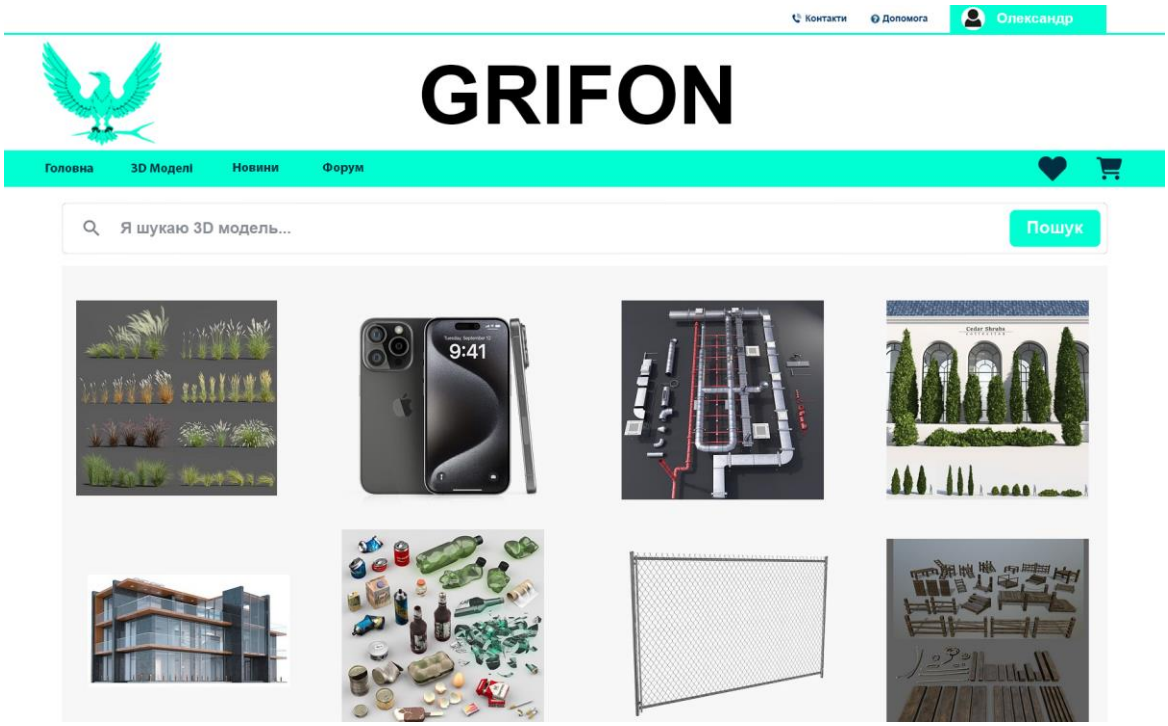


Рис. 4.4 Сторінка “3D Моделі”

“Новини”: У цьому розділі містяться актуальні новини, пов’язані з веб-застосунком. Тут користувачі можуть дізнатися про нові оновлення, функції, події та інші важливі анонси.

Рис. 4.5 Сторінка “Новини”

“Форум”: Форум є місцем, де користувачі можуть взаємодіяти один з одним, обговорювати різні теми, ділитися інформацією та ставити запитання.

Центральне місце на головній сторінці займає банер з пошуком. Пошукова система дозволяє користувачам швидко знаходити 3D моделі, які їх цікавлять. Використання регулярних виразів (REGEXP) у пошуку дозволяє користувачам вводити часткові назви моделей, що спрощує процес пошуку і робить його більш гнучким.

Після успішної реєстрації або входу в систему, кнопка “Логін/Реєстрація“ змінюється на кнопку з ім'ям користувача, що веде до профілю акаунту.

В своєму профілі користувач може додавати свої 3D моделі на веб застосунок в вкладці “Мої моделі”, переглядати вподобані 3D моделі в владці “Вподобані”, або вийти з свого акаунту через вкладку “Мій порофіль”.

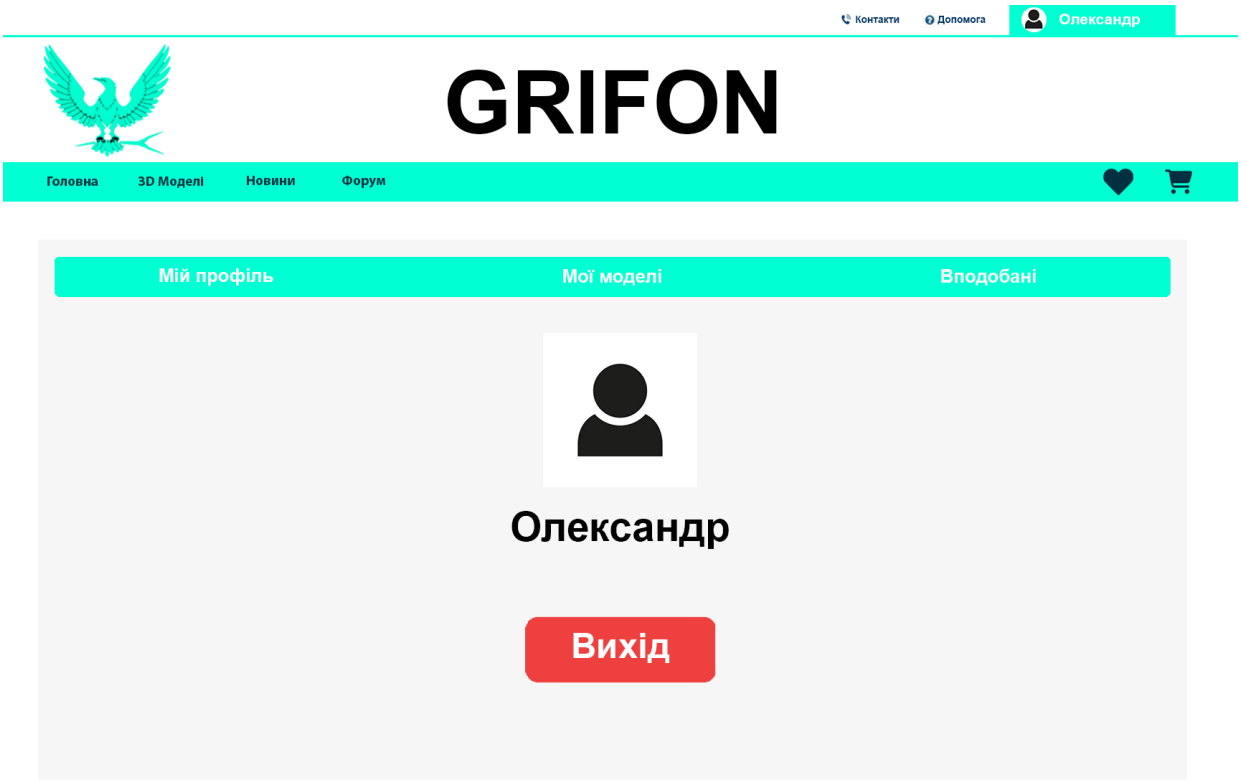


Рис. 4.6 Сторінка профілю користувача

Процес додавання 3D моделі на веб-застосунок включає заповнення форми з кількома полями, які описують модель та її характеристики.

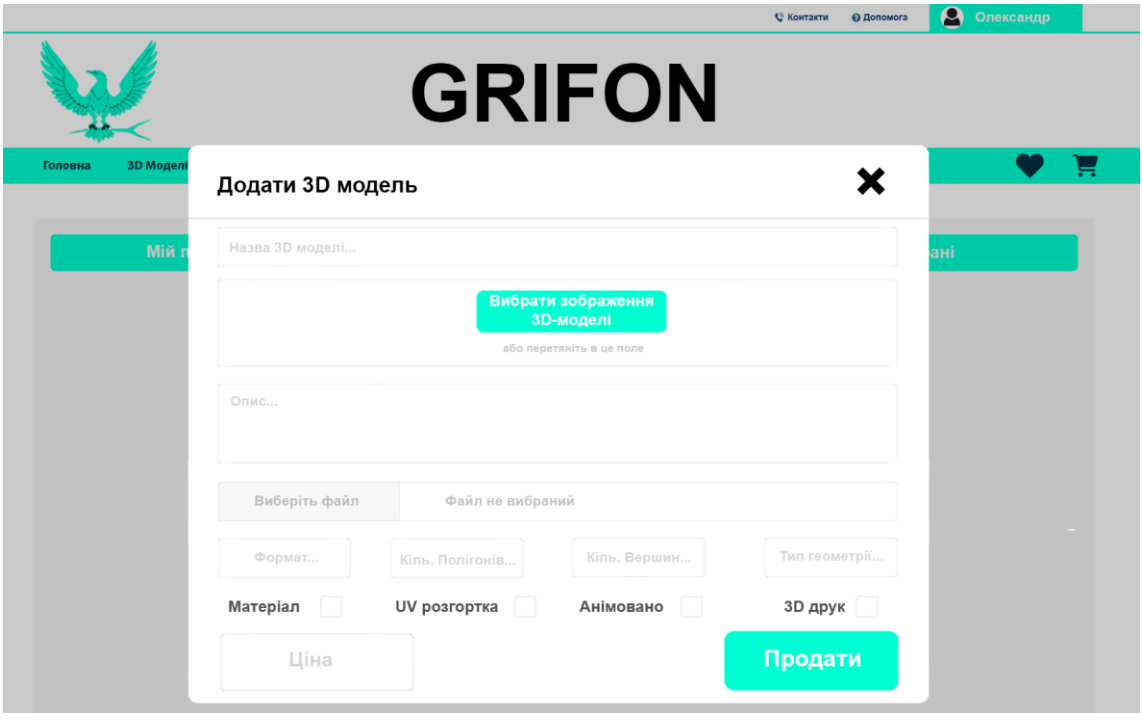


Рис. 4.6 Сторінка додавання 3D моделі на веб-застосунок

Нижче наведено короткий опис кроків, які користувач виконує для додавання нової 3D моделі:

1. Назва 3D моделі. Користувач вводить назву 3D моделі, яка буде використовуватись для ідентифікації та пошуку моделі на сайті.
2. Зображення 3D моделі. Користувач завантажує зображення 3D моделі, які будуть використовуватись для відображення в каталозі веб-застосунку.
3. Опис 3D моделі. Користувач надає детальний опис 3D моделі, який включає інформацію про модель, її призначення та особливості.
4. Файл з 3D моделлю. Користувач завантажує файл з 3D моделлю на сервер. Формат файлу може бути одним із підтримуваних (наприклад, .obj, .stl, .fbx).
5. Формат 3D моделі. Користувач вибирає формат файлу 3D моделі зі списку підтримуваних форматів.
6. Кількість полігонів. Користувач вводить кількість полігонів, з яких складається 3D модель, що є важливим показником складності моделі.
7. Кількість вершин. Користувач вводить кількість вершин, що також є важливим технічним показником моделі.
8. Тип геометрії. Користувач вказує тип геометрії моделі, наприклад, трикутники або н-кутники.
9. Наявність матеріалу. Користувач вказує, чи має модель матеріали, які визначають її зовнішній вигляд (наприклад, текстури, кольори).
10. Наявність UV-розгортки. Користувач вказує, чи має модель UV-розгортку, яка необхідна для правильного нанесення текстур на модель.
11. Наявність анімації. Користувач вказує, чи має модель анімацію, яка може включати рухи або зміни форми моделі з часом.
12. Готовність до 3D друку. Користувач вказує, чи готова модель до 3D друку, що означає, що модель відповідає вимогам для друку на 3D принтері.
13. Ціна. Користувач встановлює ціну моделі, яка буде відображатися на сайті і яку інші користувачі зможуть сплатити для завантаження моделі.

Після заповнення всіх полів та завантаження файлу, користувач натискає кнопку "Додати модель", після чого дані передаються на сервер для збереження у базі даних. Модель стає доступною для перегляду, пошуку та придбання іншими користувачами веб-застосунку.

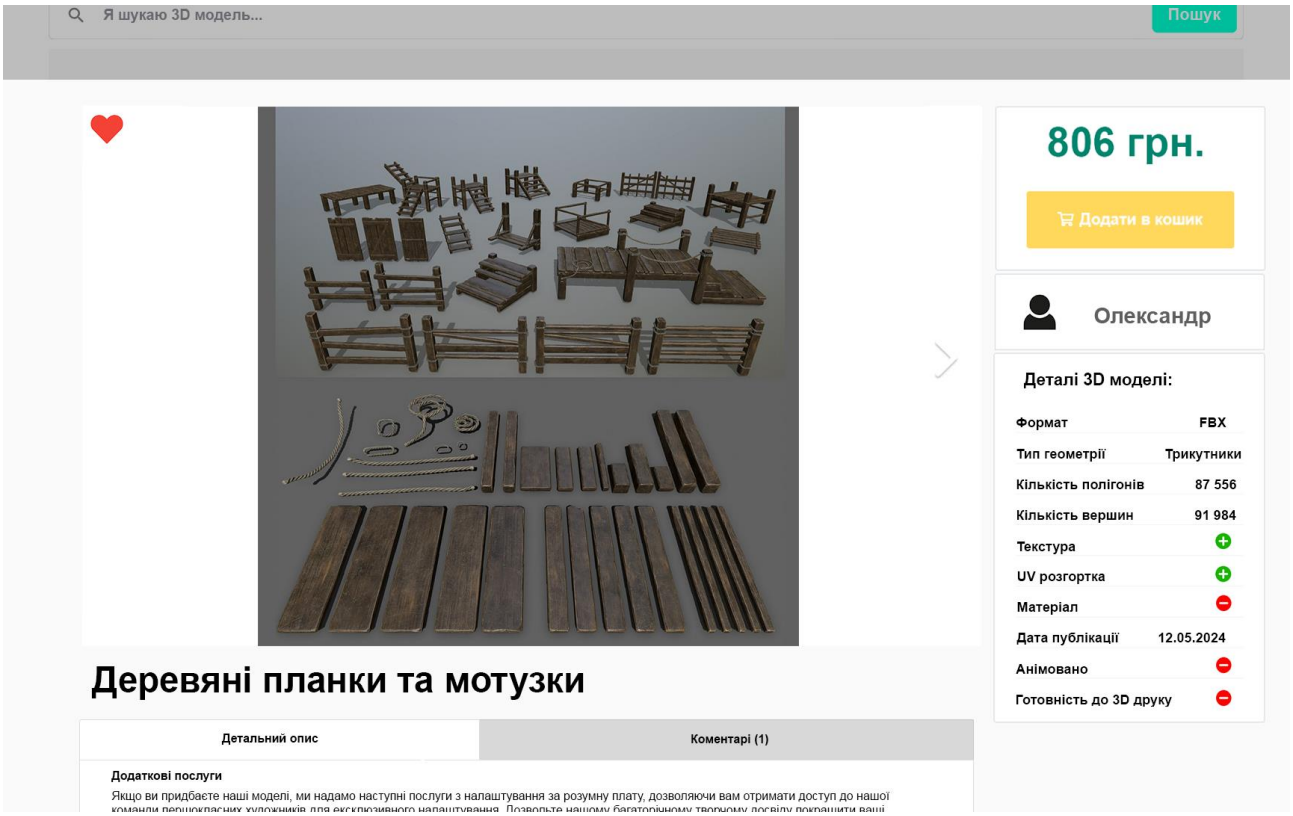


Рис. 4.7 Сторінка прикладу виставленої 3D моделі

Користувач також може додати вибрану 3D модель в ‘Вподобані’, натиснувши на іконку серця. Вподобана модель надає користувачу можливість відслідковувати оновлення 3D моделі від автора, або відслідковувати акції та знижки, які діють на цю модель.

При бажанні купити 3D модель, користувач натискає “Додати в кошик” що дозволяє користувачу підготуватися до покупки. Ця система забезпечує інтуїтивно зрозумілий і зручний спосіб збереження моделей для подальшого придбання та управління ними перед остаточною оплатою. Нижче наведено детальний опис процесу додавання 3D моделей до кошика, його компоненти та взаємодію між ними.

Після додавання вибраної 3D моделі до кошика користувач має можливість продовжити процес покупки, який включає оплату моделі та її отримання через електронну пошту. Цей процес складається з кількох етапів, кожен з яких забезпечує безперебійну та безпечну взаємодію користувача з системою.

На сторінці кошика користувач має можливість переглянути всі додані 3D моделі, включаючи їх назви, кількість, індивідуальну та загальну вартість. Користувач також може редагувати вміст кошика, змінюючи кількість моделей або видаляючи непотрібні елементи. Система автоматично оновлює загальну суму замовлення відповідно до внесених змін.

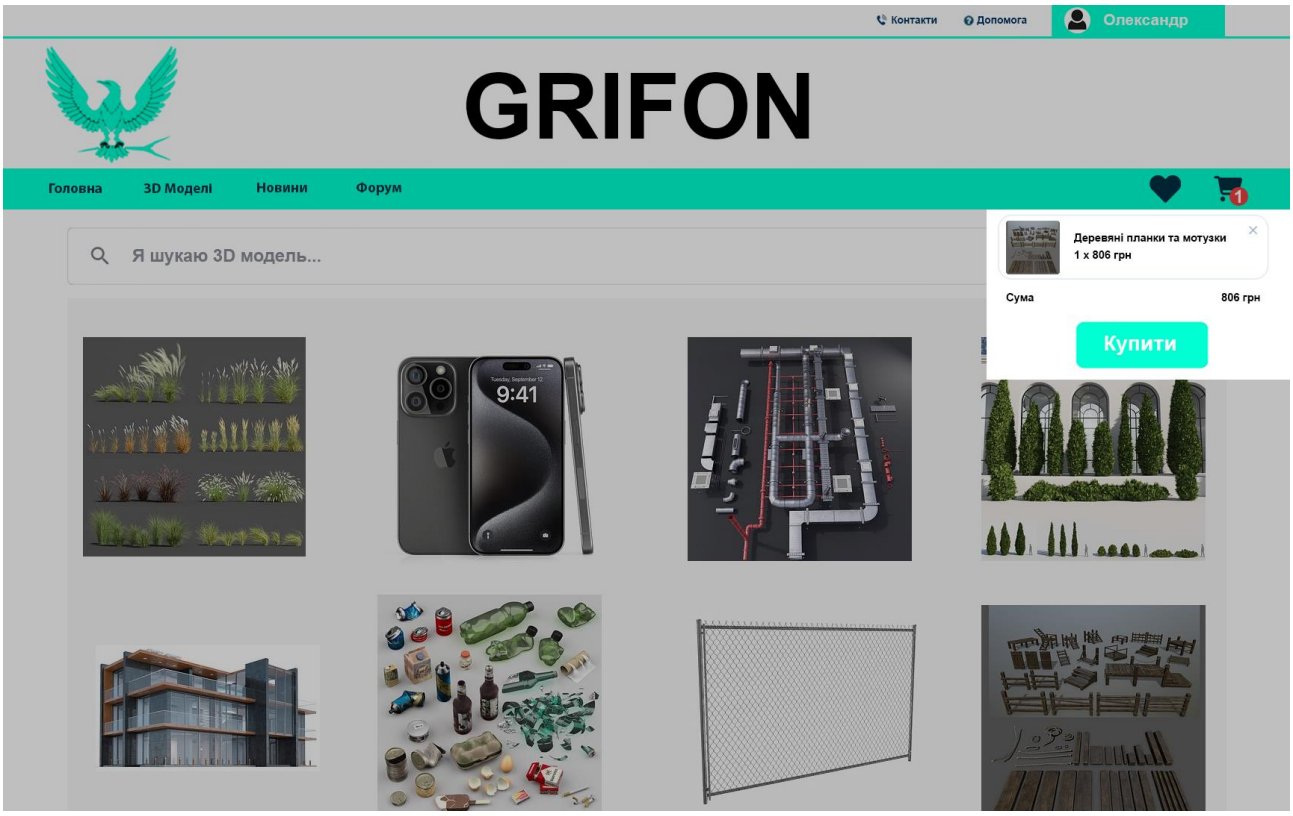


Рис. 4.8 Корзина користувача

Після введення платіжних даних користувач підтверджує оплату, натиснувши відповідну кнопку на сторінці оплати. Система передає дані до платіжного шлюзу для обробки транзакції. Платіжний шлюз перевіряє дані та здійснює транзакцію.

Після підтвердження оплати система автоматично генерує електронний лист, який містить посилання для завантаження купленої 3D моделі. Електронний лист надсилається на адресу електронної пошти, вказану користувачем при реєстрації акаунту.

4.2 Тестування веб застосунку

Тестування є невід'ємною частиною процесу розробки веб-застосунку. Воно забезпечує виявлення та виправлення помилок, а також гарантує, що всі функціональні та нефункціональні вимоги виконуються належним чином. У даній роботі тестування розробленого веб-застосунку здійснюється з використанням різних методів та інструментів, включаючи функціональне тестування, та тестування безпеки.

Таблиця 4.1

Функціональне тестування

Мета кейсу	**Тестовий сценарій**	**Результат**
Реєстрація та авторизація користувача	Користувач заповнює форму реєстрації з коректними даними та натискає кнопку "Зареєструватися".	Новий користувач успішно створений, відображається сторінка профілю.
Пошук 3D моделей	Користувач вводить ключове слово у поле пошуку та натискає кнопку "Пошук"	Відображаються правильні результати пошуку.

Продовження таблиці 4.1

Додавання 3D моделі до кошика	Користувач обирає 3D модель та натискає кнопку "Додати до кошика"	Модель успішно додана до кошика, відображається спливаюче повідомлення.
Додавання 3D моделі до вподобаних	Користувач натискає іконку серця на сторінці детального опису 3D моделі.	Модель успішно додана до вподобаних, іконка змінюється, відображається підтвердження.
Купівля 3D моделі	Користувач переходить до кошика, перевіряє вміст, натискає кнопку "Купити".	Користувач отримує підтвердження та саму модель вказану пошту при реєстрації акаунту.
Додавання користувачем коментарів	Користувач натискає кнопку "Коментарі" та пише коментар	Створений коментар зберігся та успішно відображається в владці Коментарі

Тестування безпеки перевіряє, наскільки веб-застосунок захищений від зовнішніх атак та забезпечує конфіденційність даних користувачів.

Таблиця 4.2

Тестування Безпеки

Мета кейсу	**Тестовий сценарій**	**Результат**
Перевірка аутентифікації та авторизації	Перевірка систем входу, реєстрації та управління доступом до захищених ресурсів.	Система коректно обробляє аутентифікацію та авторизацію, забезпечуючи захист даних користувачів.
Тестування на уразливості	Виконання тестів на уразливості бази даних з використанням OWASP ZAP.	Веб-застосунок не містить критичних уразливостей, всі виявлені проблеми виправлені.
Шифрування даних	Перевірка шифрування даних під час передачі між клієнтом та сервером (SSL/TLS) та зберігання чутливих даних у зашифрованому вигляді.	Дані користувачів надійно шифруються під час передачі та зберігання.

ВИСНОВКИ

Результатом виконаної дипломної роботи є розроблена торговельна площадки 3D-моделей "Grifon" з використанням JavaScript, Python, HTML, CSS та фреймворків React і Django. Метою даної роботи було спростити процес купівлі-продажу 3D-моделей за рахунок використання зручної та функціональної платформи, яка дозволяє користувачам легко знаходити, оцінювати, купувати та продавати 3D-моделі.

У рамках дипломної роботи було проведено всебічний аналіз ринку торговельної індустрії 3D моделей, що дозволило виявити основні тенденції та потреби цієї галузі. Було оцінено функціональні можливості веб-платформ конкурентів, зокрема таких як Sketchfab, TurboSquid та Quixel Megascans, та визначено їх сильні і слабкі сторони, що сприяло формуванню вимог до розробки власного веб-застосунку.

Визначено ключові вимоги до веб-застосунку з урахуванням необхідної функціональності та зручності використання для кінцевих користувачів. Проведено дослідження різних технологій та фреймворків, таких як JavaScript, Python, HTML, CSS, а також React та Django, для забезпечення оптимальної продуктивності та гнучкості при розробці веб-застосунку.

На основі отриманих результатів розроблено веб-застосунок "Grifon" для реалізації продажу 3D моделей, який інтегрує сучасні технології та забезпечує високий рівень користувацького досвіду.

Завершальним етапом було проведене тестування веб-застосунку, що підтвердило його функціональність, надійність та відповідність встановленим вимогам. Таким чином, робота підтверджує доцільність використання обраних технологій та підходів для створення ефективної торговельної платформи 3D моделей.

Робота пройшла апробацію на наукових конференціях, за результатами апробації опубліковано тези доповідей:

1. Здоренко О.О., Золотухіна О.А. Аналіз сфери використання 3D-моделей: Матеріали IV Всеукраїнська Науково-практична конференція ‘‘ Сучасні інтелектуальні інформаційні технології в науці та освіті.’’Збірник тез. 17.05.2024, ДУІКТ, м. Київ. К.: ДУІКТ, 2024. (Подано до друку)

2. Здоренко О.О., Золотухіна О.А. Огляд веб-сервісів продажу 3D-моделей: Матеріали IV Всеукраїнська Науково-практична конференція ‘‘ Сучасні інтелектуальні інформаційні технології в науці та освіті.’’ Збірник тез. 17.05.2024, ДУІКТ, м. Київ. К.: ДУІКТ, 2024. (Подано до друку)

ПЕРЕЛІК ПОСИЛАНЬ

1. 3D-графіка: актуальність, напрям та думка експерта. [Електронний ресурс] : Режим доступу до ресурсу: https://univerpl.com.ua/blog/3d-grafika-aktualnist-napryami-ta-dumka-eksperta/
2. Що таке CGI. [Електронний ресурс] : Режим доступу до ресурсу: https://cgischool.ua/scho-take-cgi/
3. Як візуальзують архітектуру в 3D. [Електронний ресурс] : Режим доступу до ресурсу: https://skvot.io/uk/blog/yak-vizualizuyut-arhitekturu-v-3d
4. 3D моделювання та прототипування. [Електронний ресурс] : Режим доступу до ресурсу: https://www.smartart.space/3d-modeling-and-prototyping/
5. Wikipedia JavaScript [Електронний ресурс] - Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JavaScript
6. Turbosquid [Електронний ресурс] : Режим доступу до ресурсу: https://blog.turbosquid.com
7. Quixel Megascans [Електронний ресурс] : Режим доступу до ресурсу:https://forums.unrealengine.com/tags/c/development-discussion/quixel-megascans/200/unreal-engine3
8. SketchFab: [Електронний ресурс] : Режим доступу до ресурсу: https://sketchfab.com/blogs/community/articles/
9. Єгорова І.М. Проектування та розробка Web-документів: навч. Посібник. Харків: ХНУРЕ, 2018. 264 с.
10. W3Schools. HTML, CSS, JavaScript Tutorials. [Електронний ресурс] : Режим доступу до ресурсу: https://www.w3schools.com
11. React. [Електронний ресурс] : Режим доступу до ресурсу: https://de.react.dev/learn

12. Sequelize. [Електронний ресурс] : Режим доступу до ресурсу: https://sequelize.org/docs/v6/
13. TypeORM. [Електронний ресурс] : Режим доступу до ресурсу: https://docs.nestjs.com/recipes/sql-typeorm
14. Express. [Електронний ресурс] : Режим доступу до ресурсу: https://expressjs.com/ru/guide/routing.html
15. Які є конвенції в REST API та для чого їх дотримуватись. [Електронний ресурс]. — Режим доступу до ресурсу: https://dou.ua/forums/topic/34550
16. Node.js – Введеня [Електронний ресурс]. — Режим доступу до ресурсу: https://coderlessons.com/tutorials/kompiuternoeprogrammirovanie/uznaite-node-js/node-js-kratkoe-rukovodstvo
17. AngularJS [Електронний ресурс]. — Режим доступу до ресурсу: https://en.wikipedia.org/wiki/AngularJS
18. Software Testing Help [Електронний ресурс] – Режим доступу до ресурсу: https://www.softwaretestinghelp.com
19. Ranking of the most popular database management systems worldwide, as of February 2023 [Електронний ресурс]. — Режим доступу до ресурсу: https://www.statista.com/statistics/809750/worldwide-popularity

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка торговельної площадки 3D-моделей “Grifon” з використанням JavaScript, Python, HTML, CSS та фреймворків React, Django.

Виконав студент 4 курсу
групи ПД-42
Здоренко Олексій Олександрович
Керівник роботи
К.т.н, доц, кафедри ІПЗ Золотухіна Оксана Анатоліївна

Київ – 2024

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи:** спростити процес купівлі-продажу 3D-моделей за рахунок використання торговельної площадки 3D-моделей “Grifon”
- **Об’єкт дослідження:** процес купівлі-продажу 3D-моделей
- **Предмет дослідження:** програмне забезпечення для купівлі-продажу 3D-моделей

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести аналіз ринку торговельної індустрії 3D моделей.
2. Оцінити веб-платформи конкурентів та їх функціонал для визначення сильних та слабких сторін.
3. Визначити вимоги до веб-застосунку з урахуванням його функціональності та зручності використання.
4. Розглянути можливості використання різних технологій та фреймворків для розробки веб-застосунку.
5. Розробити веб-застосунок для реалізації продажу 3D моделей.
6. Виконати аналіз зручності та відповідності вимогам користувацького інтерфейсу через тестування.

АНАЛІЗ АНАЛОГІВ

Критерій порівняння	Turbosquid	Quixel Megascans	SketchFab	**Розроблене ПЗ**
Відстежування вподобаних робіт авторів.	-	+	+	+
Підтримка, та оновлення вже проданих 3D моделей	-	+	-	+
Можливість написання рецензій та коментарів	-	+	+	+
Не прив'язаність до платформ візуалізації 3D моделей	+	-	+	+
Українська локалізація	-	-	-	+

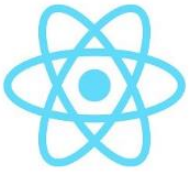
ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

- Реєстрація нового користувача: можливість створити новий обліковий запис з використанням електронної пошти.
- Вхід в систему: аутентифікація користувачів за допомогою електронної пошти/пароля.
- Перегляд каталогу: користувачі повинні мати можливість переглядати різноманітні 3D моделі, відфільтровані за категоріями, ціною, популярністю тощо.
- Детальний опис моделі: сторінка з детальним описом моделі, включаючи зображення, технічні характеристики, ціни та відгуки.
- Фільтрація: можливість фільтрувати результати пошуку за різними критеріями, такими як ціна, рейтинг, дата додавання тощо.
- Додавання до кошика: користувачі можуть додавати моделі до кошика для подальшої покупки.
- Додавання до Вподобаних: користувачі можуть додавати вподобані 3D моделі до 'Вподобані' для відслідковування оновлень або знижок.
- Оформлення замовлення: процес оформлення замовлення з вибором способу оплати та підтвердження покупки.
- Завантаження моделей: автори можуть завантажувати свої 3D моделі, додавати описи, встановлювати ціни та переглядати статистику завантажень.
- Оцінювання моделей: можливість користувачам залишати оцінки та відгуки про моделі.

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

HTML

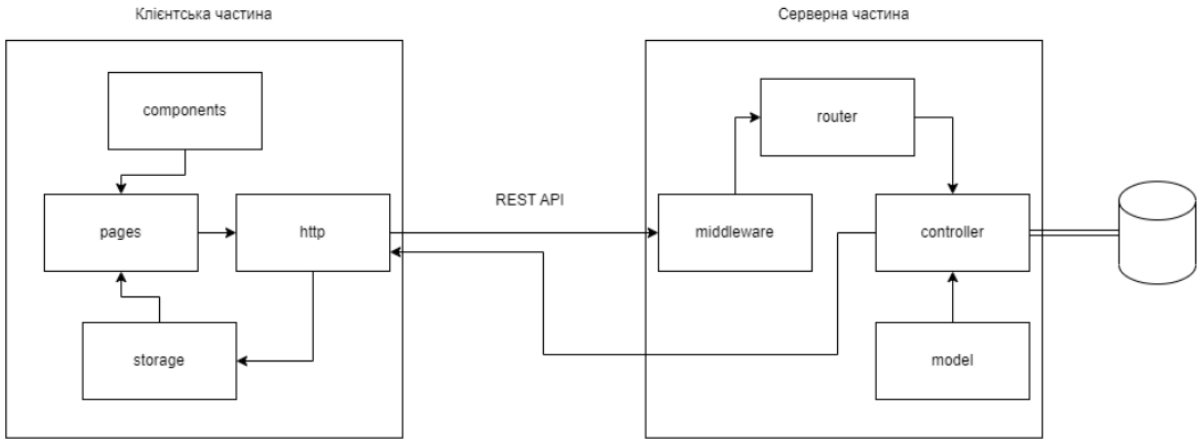
Python



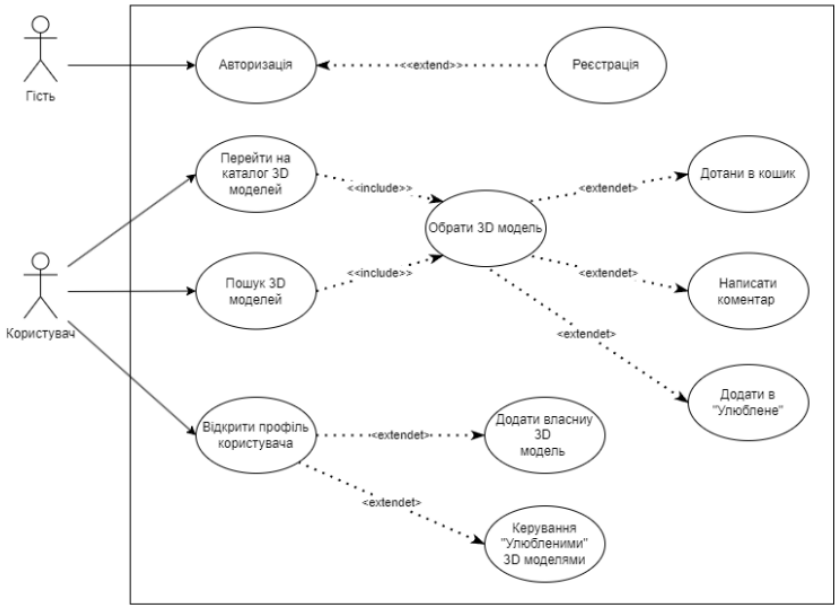
React

Sequelize

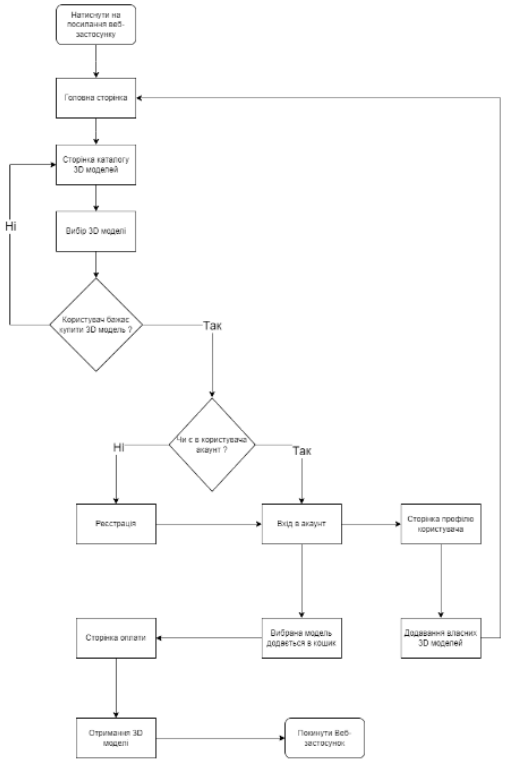
АРХІТЕКТУРА СИСТЕМИ



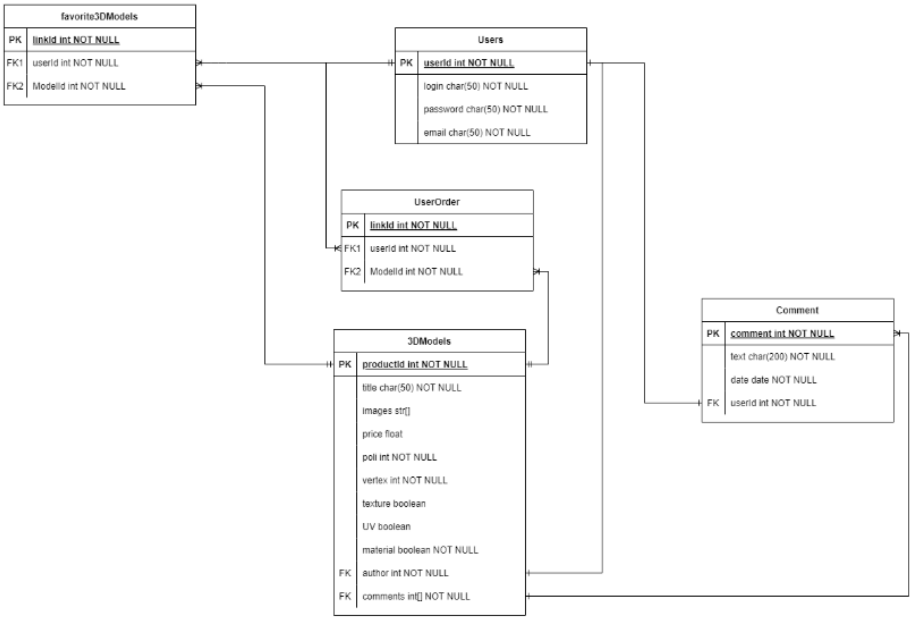
ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



ДІАГРАМА USER FLOW



СТРУКТУРА БАЗИ ДАНИХ



ЕКРАННІ ФОРМИ

Рис. 1 - Головна сторінка

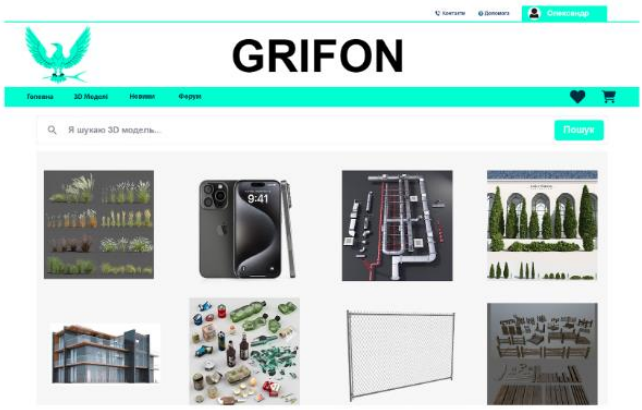


Рис. 2 - Каталог 3D моделей

ЕКРАННІ ФОРМИ

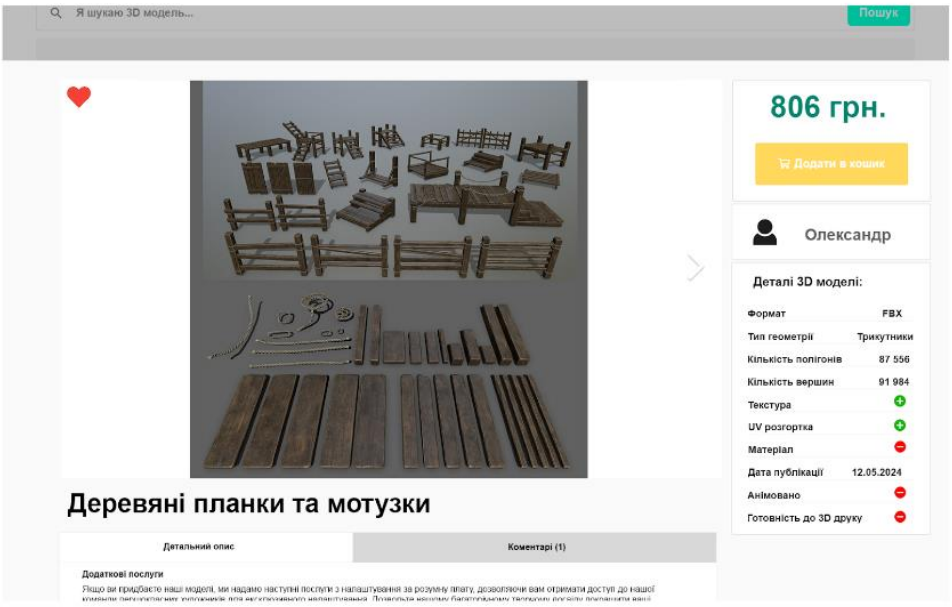


Рис. 3 - Інтерфейс сторінки 3D моделей

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Здоренко О.О., Золотухіна О.А. Огляд веб-сервісів продажу 3D-моделей. Розробка ігор та ігрофікація освітніх та бізнес процесів: Матеріали IV Всеукраїнська Науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті. Збірник тез. 17.05.2024 (Подано до друку)
2. Здоренко О.О., Золотухіна О.А. Аналіз сфери використання 3D-моделей. Розробка ігор та ігрофікація освітніх та бізнес процесів: Матеріали IV Всеукраїнська Науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті. Збірник тез. 17.05.2024 (Подано до друку)

ВИСНОВКИ

1. Проаналізовано існуючий ринок торговельної індустрії 3D моделей.
2. Оцінено веб-платформи конкурентів та їх функціонал та визначені сильні та слабкі сторони.
3. Визначено вимоги до веб-застосунку з урахуванням його функціональності та зручності використання.
4. Досліджені різні технологій та фреймворки для розробки веб-застосунку.
5. Розроблено веб-застосунок для реалізації продажу 3D моделей.
6. Проведений аналіз зручності та відповідності вимогам користувацького інтерфейсу через тестування.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

Фрагменти коду серверу програми

```
const { Model, User, Favorite3DModel } = require('../models/models');
const ApiError = require('../error/ApiError');
const { Op } = require("sequelize");
const uuid = require('uuid');
const path = require('path');

class ModelController {
    async getById(req, res, next) {
        try {
            const { id } = req.params;
            const model = await Model.findOne({ where: { id } });
            return res.json(model);
        } catch (e) {
            next(ApiError.badRequest(e.message));
        }
    }

    async getAll(req, res, next) {
        try {
            let { search, limit = 21, page = 1 } = req.query;
            let offset = (page - 1) * limit;
            let models;
```

```
        if (search) {
            models = await Model.findAndCountAll({
                where: { title: { [Op.iRegexp]: search } },
                limit,
                offset
            });
        } else {
            models = await Model.findAndCountAll({ limit, offset });
        }

        return res.json(models);
    } catch (e) {
        next(ApiError.badRequest(e.message));
    }
}

async getProductsByDish(req, res, next) {
    try {
        const { id } = req.params;
        const model = await Model.findOne({
            where: { id },
            include: Product
        });
        return res.json(model.products);
    } catch (e) {
        next(ApiError.badRequest(e.message));
    }
}
```

```
async getFavoriteModelsByUser(req, res, next) {
    try {
        const { id } = req.params;
        let { search, limit = 21, page = 1 } = req.query;
        let offset = (page - 1) * limit;
        let models;

        if (search) {
            models = await Model.findAndCountAll({
                where: { title: { [Op.iRegexp]: search } },
                include: [{ model: User, where: { id } }],
                limit,
                offset
            });
        } else {
            models = await Model.findAndCountAll({
                include: [{ model: User, where: { id } }],
                limit,
                offset
            });
        }

        return res.json(models);
    } catch (e) {
        next(ApiError.badRequest(e.message));
    }
}
```

```
async createLinkUserModel(req, res, next) {
    try {
        const { userId, dishId } = req.body;
        await Favorite3DModel.create({ userId, dishId });
        return res.json({ message: 'Link created!' });
    } catch (e) {
        next(ApiError.badRequest(e.message));
    }
}

async deleteLinkUserModel(req, res, next) {
    try {
        const { userId, dishId } = req.query;
        const result = await Favorite3DModel.destroy({ where: { userId, dishId } });
        return res.json(result);
    } catch (e) {
        next(ApiError.badRequest(e.message));
    }
}

async checkLinkUserModel(req, res, next) {
    try {
        const { userId, dishId } = req.query;
        const link = await Favorite3DModel.findOne({ where: { userId, dishId } });
        return res.json(!!link);
    } catch (e) {
        next(ApiError.badRequest(e.message));
```

```
        }
    }

    async create(req, res, next) {
        try {
            let { title, steps, products } = req.body;
            let fileName = 'default.jpg';

            if (req.files && req.files.img) {
                fileName = uuid.v4() + '.jpg';
                const { img } = req.files;
                await img.mv(path.resolve(__dirname, '..', 'static', fileName));
            }

            const model = await Model.create({ title, steps, img: fileName });
            return res.json(model);
        } catch (e) {
            next(ApiError.forbidden(e.message));
        }
    }
}

module.exports = new ModelController();
```