МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії
програмного забезпечення

_____ Євген ДАВИДЕНКО

«___» _____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

Онлайн-сервіс для розробки та візуалізації шейдерів на 3D-модель

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

Здобувач _____ **Богдан ШМАЛЬКО**

«__» _____ 20__ р.

Керівник роботи

ст. викладачка _____ **Катерина ОБУХОВА**

«__» _____ 20__ р.

Миколаїв – 2026

Завдання на виконання кваліфікаційної роботи

Чорноморський національний університет імені Петра Могили

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

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«___» _____ 2025 р.

ЗАВДАННЯ

на кваліфікаційну бакалаврську роботу здобувача

Шмалько Богдан

1. Тема кваліфікаційної роботи «Онлайн-сервіс для розробки та візуалізації шейдерів на 3D-модель» затверджена наказом ректора ЧНУ ім. Петра Могили № 349 від «26» грудня 2025 р.
2. Строк представлення кваліфікаційної роботи «___» _____ 2026 р.
3. Очікуваний результат роботи та початкові дані якщо такі потрібні.

Розробка мінімально життєздатного прототипу (MVP) онлайн-сервісу для розробки та візуалізації шейдерів на 3D-моделях.

4. Перелік питань, що підлягають розробці:
   1) аналіз сучасних підходів та засобів для створення 3D-графіки у

вебсередовищі;

2) формулювання специфікації функціональних та нефункціональних вимог до програмного забезпечення;

3) проєктування програмного забезпечення, визначивши його архітектуру, структуру компонентів та взаємодію між ними із використанням діаграм варіантів використання та діаграм компонентів;

4) розробка механізмів параметризації, валідації текстур і шейдерів із керуванням через користувацький інтерфейс;

5) забезпечення можливості попереднього перегляду сцени та експорту результатів з використанням хмарної бази даних;

6) проведення тестування розробленого програмного забезпечення та оцінка його працездатності і відповідності сформульованим вимогам.

5. Перелік графічних матеріалів: Презентація
6. Консультанти:

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

Дата видачі завдання «____» _____ 20___ р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: **Онлайн-сервіс для розробки та візуалізації шейдерів на 3D-модель**

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КБР	01.09.2025	26.12.2025	Виконано
2.	Огляд літератури за темою роботи	30.12.2025	15.01.2026	Виконано
3.	Складання календарного плану КБР	18.01.2026	20.01.2026	Виконано
4.	Аналіз предметної області	01.02.2026	15.02.2026	Виконано
5.	Розробка проєктних рішень	18.02.2026	25.02.2026	Виконано
6.	Моделювання та конструювання ПЗ	02.03.2026	17.03.2026	Виконано
7.	Кодування, тестування та апробація розробленого ПЗ, аналіз результатів тестування, розробка керівництва користувача	18.03.2026	21.05.2026	Виконано
8.	Оформлення КБР та презентації	05.05.2026	03.06.2026	Виконано
9.	Попередній захист	25.05.2026	25.05.2026	Виконано
10.	Завершення оформлення звіту КБР	01.06.2026	04.06.2026	Виконано
11.	Рецензування	15.06.2026	15.06.2026	Виконано
12.	Відгук керівника КБР	15.06.2026	15.06.2026	Виконано
13.	Захист кваліфікаційної роботи	22.06.2026	22.06.2026	

Здобувач _____ **Богдан ШМАЛЬКО**

«__» _____ 20__ р.

Керівник роботи

ст. викладачка _____ **Катерина ОБУХОВА**

«__» _____ 2026 р.

АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи

«Онлайн-сервіс для розробки та візуалізації шейдерів на 3D-модель»

Здобувач 408 гр.: Шмалько Богдан

Керівник: ст. викладачка кафедри комп'ютерної інженерії Обухова Катерина

Актуальність роботи зумовлена зростаючою потребою у зручних та доступних інструментах для створення, навчання й тестування шейдерів та матеріалів у тривимірній графіці, в умовах зростаючого візуального різноманіття вебзастосунків. Сучасні технології реального часу, зокрема GPU-прискорення та шейдерне програмування, потребують високого рівня підготовки, що ускладнює їх використання початківцями та у навчальному процесі.

Метою кваліфікаційної бакалаврської роботи є розробка онлайн-сервісу, що поєднує можливості сучасних вебтехнологій з функціональністю професійних графічних редакторів шейдерів для створення, редагування та попереднього перегляду 3D-сцен у реальному часі.

Об'єктом роботи є процес створення та візуалізації тривимірних графічних сцен у вебзастосунках за допомогою шейдерного коду із використанням технологій реального часу.

Предметом роботи є методи та програмні засоби інтерактивного керування шейдерами, матеріалами й текстурами у веборієнтованих 3D-застосунках на основі бібліотек Three.js, React Three Fiber та мови GLSL.

Кваліфікаційна робота складається із вступу, чотирьох розділів, висновків та переліку джерел посилання.

У вступі обґрунтовано актуальність теми, визначено мету, завдання, об'єкт та предмет роботи, а також практичне значення отриманих результатів.

У першому розділі виконано аналіз предметної області, розглянуто сучасні підходи до створення 3D-графіки у вебсередовищі, проаналізовано існуючі програмні рішення та обґрунтовано доцільність розробки нової платформи.

Другий розділ присвячено аналізу сучасних технологій веброзробки та формуванню специфікації функціональних і нефункціональних вимог до програмного забезпечення.

У третьому розділі проводиться проєктування програмного забезпечення, визначення архітектури системи, структури її компонентів та взаємодії між ними з використанням UML-діаграм.

У четвертому розділі описано програмну реалізацію вебзастосунку для інтерактивного редагування шейдерів і матеріалів, механізми параметризації текстур, валідації користувацького коду, збереження проєктів, а також проводиться тестування розробленого програмного забезпечення.

У висновках наведено основні результати виконаної роботи, зроблено узагальнення та визначено напрями подальшого розвитку розробленого програмного продукту.

КБР викладена на 81 сторінки (без додатків), вона містить 4 розділи, 45 ілюстрацій, 8 таблиць, 29 джерел в переліку посилання та 2 додатків.

Ключові слова: *вебзастосунок, тривимірна графіка, шейдери, GLSL, Three.js, React Three Fiber, WebGL, 3D-об’єкти.*

ABSTRACT

to the Bachelor's Thesis

«Online service for developing and visualizing shaders on 3D model»

Student of 408 group: Shmalko Bohdan

Supervisor: Senior Lecturer, Department of Computer Engineering, Kateryna Obukhova

The relevance of this work stems from the growing need for user-friendly and accessible tools for creating, learning, and testing shaders and materials in 3D graphics, given the increasing visual diversity of web applications. Modern real-time technologies, particularly GPU acceleration and shader programming, require a high level of training, which makes them difficult for beginners to use and for teaching.

The goal of this bachelor's thesis is to develop an online service that combines the capabilities of modern web technologies with the functionality of professional shader editors to create, edit, and preview 3D scenes in real time.

The object of this work is the process of creating and visualizing three-dimensional graphic scenes in web applications using real-time technologies with shader code.

The subject of the work is the methods and software tools for interactive control of shaders, materials, and textures in web-oriented 3D applications based on the Three.js, React Three Fiber libraries, and the GLSL language.

The thesis consists of an introduction, four chapters, conclusions, and a list of references.

The introduction justifies the relevance of the topic, defines the purpose, objectives, scope, and subject of the study, as well as the practical significance of the results obtained.

The first chapter analyzes the subject area, examines modern approaches to creating 3D graphics in a web environment, analyzes existing software solutions, and justifies the feasibility of developing a new platform.

The second chapter is devoted to the analysis of modern web development technologies and the formulation of specifications for functional and non-functional software requirements.

The third chapter covers software design, the definition of the system architecture, the structure of its components, and the interactions between them using UML diagrams.

The fourth chapter describes the software implementation of a web application for interactive editing of shaders and materials, mechanisms for parameterizing textures, validating user code, and saving projects, as well as testing the developed software.

The conclusions present the main results of the work performed, provide a summary, and identify directions for the further development of the software product.

The thesis consists of 81 pages (excluding appendices); it contains 4 chapters, 45 figures, 8 tables, 29 references, and 2 appendices.

Keywords: *web application, three-dimensional graphics, shaders, GLSL, Three.js, React Three Fiber, WebGL, 3D objects.*

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

КБР – кваліфікаційна бакалаврська робота

ПЗ – програмне забезпечення

BaaS – Backend as a Service

DOM – Document Object Model

GLSL – OpenGL Shading Language

GPU – Graphics Processing Unit

GUI – Graphical User Interface

HDRI – High Dynamic Range Imaging

MPA – Multi-Page Application

MVP – Minimum Viable product

SPA – Single-Page Application

UML – Unified Modeling Language

WebGL – Web Graphics Library

ВСТУП

Сучасні комп’ютерні технологічні рушії тривимірної графіки починають відігравати все більшу роль у комп’ютерних іграх, візуалізації, вебдизайні, віртуальній та новітній, доповненій реальності, що розвиваються стрімкими темпами. Серед них, особливе місце займають технології реального часу, на прикладі використання графічних процесорів та шейдерного програмування як кросплатформного, користувацького досвіду. Проте створення та налаштування шейдерів потребує високого рівня підготовки, знань мови OpenGL Shading Language (GLSL) та розуміння принципів роботи графічного конвеєра.

Актуальною постає проблема від спрощення процесу створення, до налаштування та тестування шейдерів і навіть матеріалів, замість необхідності постійного редагування низькорівневого шейдерного коду, особливих програм та їх перевірки на існуючій користувацькій моделі. Особливо важливо це для вебзастосунків, де використовуються бібліотеки Three.js та React Three Fiber, які дозволяють поєднувати сучасні вебтехнології з 3D-візуалізацією.

Метою кваліфікаційної роботи є розробка онлайн-сервісу нового покоління, що поєднує зручність вебплатформи з функціональністю професійних графічних редакторів шейдерів із використанням власних 3D-моделей.

Для досягнення поставленої мети необхідно розв’язати наступні **завдання**:

– проаналізувати сучасні підходи та засоби для створення 3D-графіки у вебсередовищі;

– на основі проведеного аналізу сформулювати специфікацію функціональних та нефункціональних вимог до програмного забезпечення;

– спроєктувати програмне забезпечення, визначити архітектуру, структуру компонентів та взаємодію між ними із залученням діаграм варіантів використання та діаграм компонентів;

– розробити механізм параметризації, валідації текстур і шейдерів із керуванням через користувацький інтерфейс;

– забезпечити можливість попереднього перегляду сцени та експорту результатів з використанням хмарної бази даних;

– провести тестування розробленого програмного забезпечення та оцінити його працездатність і відповідність сформульованим вимогам.

Об’єктом роботи є процес створення та візуалізації тривимірних графічних сцен у вебзастосунках за допомогою шейдерного коду із використанням технологій реального часу.

Предметом роботи є методи та програмні засоби інтерактивного керування шейдерами, матеріалами й текстурами у веборієнтованих 3D-застосунках на основі Three.js, React Three Fiber та GLSL.

Практичне значення роботи має полягати у створенні програмного інструменту, яке зможе використовуватись здобувачами, розробниками та різноманітними дизайнерами для швидкого прототипування, експериментів із графічними ефектами, та збереження результатів користувача під виглядом готових проєктів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ СУЧАСНИХ ПІДХОДІВ ДО ВЕБОРІЄНТОВАНОЇ 3D-ВІЗУАЛІЗАЦІЇ

Стрімкий розвиток інтернет-технологій у поєднанні з безперервним зростанням обчислювальних потужностей як персональних комп'ютерів, так і мобільних пристроїв, докорінно змінили уявлення про подання візуальної інформації у вебсередовищі користувачеві. Браузери, що наразі еволюціонували від засобів перегляду статичних сторінок й до повноцінних платформ з виконанням складних інтерактивних застосунків – стали одним із найваріативніших та найперспективніших напрямків, зокрема для розгортання концепцій метавізуалізації [9].

У цьому розділі проводиться аналіз поточного стану технологій 3D-візуалізації в браузері, розглядаються існуючі рішення для роботи з шейдерами та обґрунтовується необхідність створення нової платформи, що поєднала б прозорість роботи коду, включно з текстуруванням та HDRI (англ. High Dynamic Range Imaging) мапами освітлення та фону із соціальною взаємодією користувачів [1].

1.1 Еволюція технологій 3D-візуалізації у вебсередовищі

Швидкість сучасних браузерів в поєднанні зі зручністю представлення інформації стали невід'ємним елементом у повсякденному житті звичайних користувачів. А взірцем еволюції є тривимірна комп'ютерна графіка, що обробляється та рендериться у реальному часі безпосередньо на стороні клієнта, стаючи неймовірним доповненням до візуального стилю середовища (рис. 1.1). Завдяки появі та стандартизації API WebGL (англ. Web Graphics Library), а згодом і компілятора WebGPU, веброзробники отримали низькорівневий доступ до апаратного кодування GPU (англ. Graphics Processing Unit) через інтерфейс браузера.

Рисунок 1.1 – Приклад ідеї сайту побудованого з допомогою 3D-графіки

Цей стандарт проклав шлях для створення складних візуальних рішень: від інтерактивних презентацій та браузерних ігор до професійних інструментів моделювання та редагування графіки, а згодом систем моніторингу [16]. На моменті виконання роботи майже не залишилось пристроїв що не підтримують наданих стандартів, а їх швидкодія вже далеко не така проблема, якою була ще 10 років тому. Кожен може зараз перевірити, чи підтримує його браузер стандарти 3D-рендеру, потрібно лише зайти на будь-який сайт перевірки WebGL (рис. 1.2). Хоча, пряма робота з низькорівневими API вимагає глибоких знань математики та принципів роботи графічного конвеєра, що створює високий поріг входження для нових розробників та дизайнерів. Більша частина роботи з базовим API WebGL передбачає написання значного обсягу шаблонного коду, де застосовується ручне управління буферами пам’яті відеокарти, поетапної компіляції шейдерів та виконання складних матричних перетворень для кожної зміни у сцені. Це суттєво сповільнює цикл розробки та зміщує фокус з процесу створення та оброблення на вирішення низькорівневих технічних завдань.

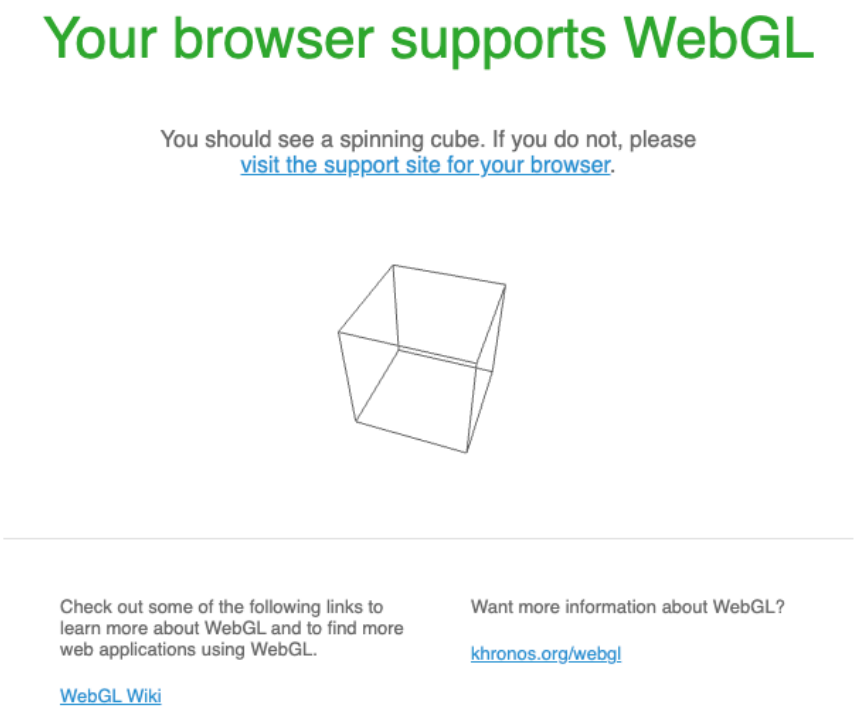

Рисунок 1.2 – Перевірка підтримки WebGL браузером користувача

З метою зменшення навантаження на нових розробників і кількості шаблонного низькорівневого коду, для спрощення та пришвидшення процесу розробки – створено низку високорівневих бібліотек та фреймворків, як Three.js [6]. Вони інкапсулюють у собі складні математичні обчислення та надають зручні об’єктно-орієнтовані інтерфейси. Завдяки цим змінам, оперування сирими масивами даних замінюється роботою зі зрозумілими сутностями: графами сцен, камерами, джерелами освітлення, полігональною геометрією та матеріалами.

Подальший розвиток екосистеми привів до інтеграції 3D-графіки з сучасними UI-бібліотеками. Інструменти, як R3F (англ. React Three Fiber), у поєднанні з декларативним підходом у побудові тривимірних сцен, забезпечуючи тісний зв’язок між загальним станом вебзастосунку та графічним конвеєром [26]. Ці зміни значно оптимізують управління життєвими циклами об’єктів та синхронізацію користувацького інтерфейсу з 3D-контентом.

Навіть із залученням високого рівня абстракції, вже існуючі стандартні бібліотеки та їх модулі при створенні візуалізації, ефектів та специфічної

постобробки не завжди мають усі потрібні матеріали та методи. Занадто важливою має залишатись можливість інтеграції користувацького GLSL-коду у конвеєр високорівневого рендерингу [8]. Одним із вагомих рішень поставатиме надання зручного інструментарію програмного забезпечення для високорівневої програмної архітектури що має взаємодіяти із викликами низькорівневого графічного процесора. Наведені проблеми мають вирішуватись у рамках створення новітніх систем розробки.

1.2 Об’єкт і предмет роботи

Будь-яка практична робота, що має комплексну архітектуру та застосовується у сфері розробки програмного забезпечення, її використання має обґрунтуватися на чітко визначених методологічних межах, в рамках яких вона проводитиметься. Процес, при якому охоплюється великий пласт роботи, від початку розробки інтерфейсів, до залучення серверної архітектури має враховувати безліч методик та практик подальшої розробки програмного забезпечення. Для створення застосунку із використанням шейдерного коду та інтерактивної графіки потрібно визначити об’єкт і предмет роботи, що забезпечить потрібний рівень контексту для розробки та дозволить сфокусуватися на можливих недоліках.

Об’єктом роботи є процес створення та візуалізації тривимірних графічних сцен у вебзастосунках за допомогою шейдерного коду із використанням технологій реального часу.

Даний процес охоплює широкий спектр взаємопов’язаних етапів: від завантаження та парсингу користувацьких 3D-моделей у середовище браузера до обчислення освітлення, накладання текстур та фінального рендерингу зображення на екран пристрою майбутнього користувача [21]. Об’єкт роботи також включатиме в себе архітектурний вибір до побудови застосунків, методів оптимізації продуктивності та забезпечення стабільної частоти кадрів при виконанні ресурсомістких графічних операцій на слабкій конфігурації вебклієнта.

Розвиток зазначеної частини тісно взаємодіє з швидким та невпинним розвитком стандартів веббраузера та апаратного прискорення заліза. Рендеринг тривимірної графіки у браузері проявляє себе специфічним у тому, що він виконується в інтерпретованому однопотоковому середовищі JavaScript. Ця особливість створює певні обмеження для розробників, коли будь-яке перевантаження основного потоку важкими обчисленнями геометрії або неконтрольоване спрацьовування збирача сміття, що може призводити до втрати кадрів, підторможувань та можливого зависання системи середовища користувача. А отже, певна розробка процесу візуалізації вимагатиме врахувань не лише з використанням математичного підходу при вимальовуванні пікселів, але й алгоритмів ефективного управління оперативною пам’яттю пристрою та відеопам’яттю графічного процесора.

Предметом роботи є програмні засоби інтерактивного керування шейдерами, матеріалами й текстурами у веборієнтованих 3D-застосунках на основі Three.js, React Three Fiber та GLSL.

На відміну від об’єкта роботи, предмет має робити акцент на обраних інструментах та механізмах реалізації поставлених завдань. Відповідно до контексту даної роботи, до предмета належать:

– підходи до динамічної компіляції, типізації та валідації користувацького GLSL-коду безпосередньо у браузері;

– практики розробки та інтегрування важких графічних ефектів, як у прикладі з HDRI-мапами оточення, при залученні багатошарового текстурування (англ. Image Layers), на завантажені користувацькі моделі;

– застосування строго типізованої мови програмування для побудови надійної архітектури взаємодії між компонентами React, React Three Fiber та графічним рушієм [15];

– механізми забезпечення збереження графічного контексту та управління станом сцени при взаємодії користувача з інтерфейсом редактора;

– вибір майбутньої системи керування даними користувачів, із можливістю збереження та публікації частин коду.

Виокремлення зазначених вище пунктів як предмету дослідження базується на сучасних парадигмах веброзробки. Перехід від імперативного управління DOM-елементами до декларативної реактивності вимагає створення надійних уніфікованих програмних мостів. Прикладом може стати React Three Fiber, що діє як такий міст, проте його інтеграція з динамічним користувацьким кодом створює нетривіальне інженерне завдання [26]. Необхідно розробити систему, яка б дозволяла стану компонентів безпечно та асинхронно оновлювати параметри WebGL-контексту без повного перемальовування полігональної сітки.

Об’єкт роботи визначатиме загальну сферу використання користувацької вебграфіки, тоді як предмет у свою чергу звужуватиме фокус на конкретних технологіях та рішеннях, що використовуватимуться при розробці платформи «Assimilate».

1.3 Огляд існуючих платформ розробки

На сьогоднішній час вже з’явилась певна, але невелика кількість достатньо популярних вебсервісів що спрямовані у фокусі на створення, тестування та презентацію візуального середовища з використанням шейдерного коду. Аналіз нижченаведених платформ має на меті опрацювати сильні сторони та цифрові обмеження. Інформація, що отримується в ході аналізу дозволить виокремити переваги та обґрунтувати процес подальшої розробки нового підходу у системі для шейдерної візуалізації.

Найбільш відомим інструментом у сфері вебкоду є ShaderToy [25] (рис. 1.3). Обрана платформа є однією з найширших сцен для опанування шейдер-коду, та дозволяє розробникам писати фрагментні шейдери мовою GLSL та ділитися ними зі спільнотою.

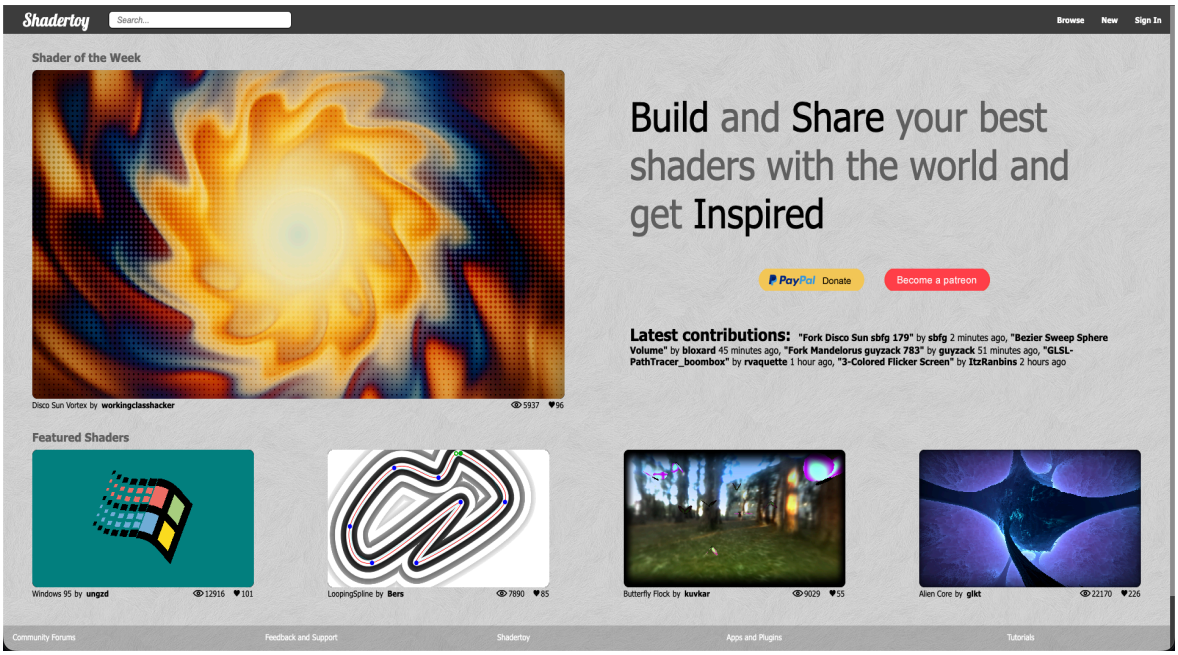

Рисунок 1.3 – Вебплатформа ShaderToy

Основною перевагою ShaderToy є велика база знань та активне ком'юніті. Одним з основних її недоліків перш за все є обмеження роботи лише у роботі з фрагментними шейдерами, що рендеряться на двовимірній площині (англ. Screen-space Quad). Для створення універсальних та глибоких 3D-сцен користувачам доводиться спиратися у роботі зі складними математичними методами, такими як реймарчинг або трасування променів у середовищі, а створення окремих об'єктів постає цілковитою математикою, що робить платформу складною для новачків. Крім того, ShaderToy не підтримує завантаження користувацьких 3D-моделей.

Наступним прикладом є популярне рішення – GLSL Sandbox [7] (рис. 1.4). Цей сервіс пропонує мінімалістичний інтерфейс для швидкого написання та перевірки фрагментних шейдерів. Хоча він є зручним інструментом для швидкого прототипування, йому вже бракує інструментів для роботи з тривимірною геометрією, текстуруванням, картами освітлення, а також відсутня повноцінна соціальна складова, як система профілів, коментарів, проєктів.

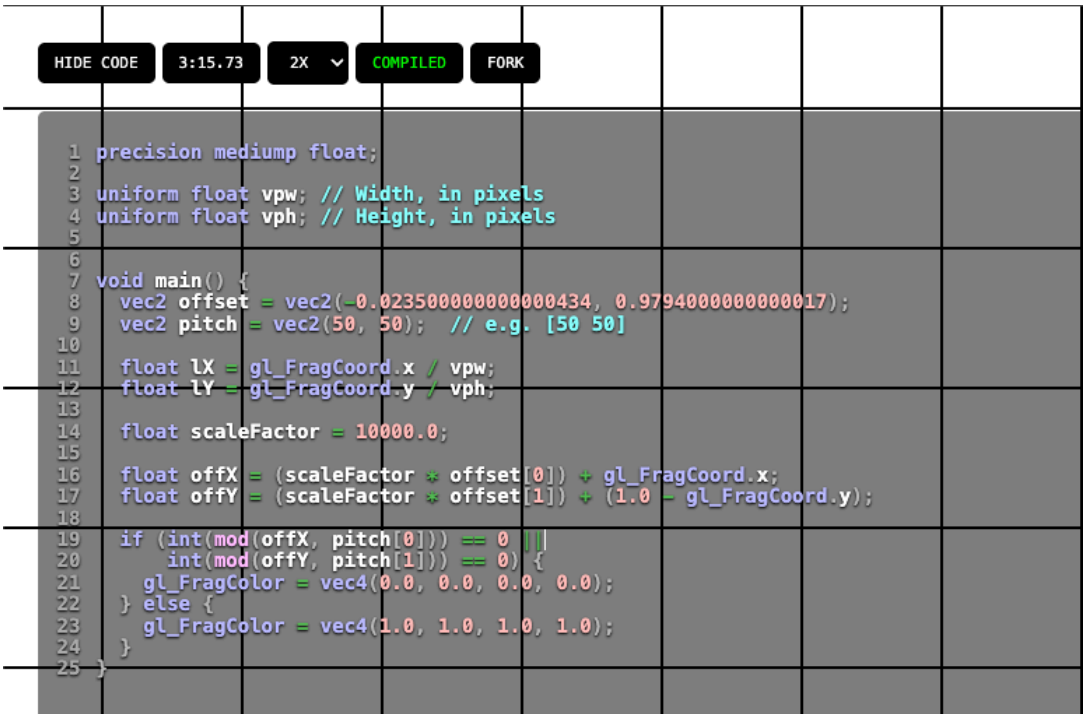

Рисунок 1.4 – Вебплатформа GLSL Sandbox

Існують також деякі комплексні браузерні 3D-редактори та рушії, такі як PlayCanvas [23] (рис. 1.5) або Spline [28]. Вони надають широкі можливості для імпорту 3D-моделей та створення інтерактивних сцен, проте їх функціонал орієнтований здебільшого на розробку ігор або загальний вебдизайн.

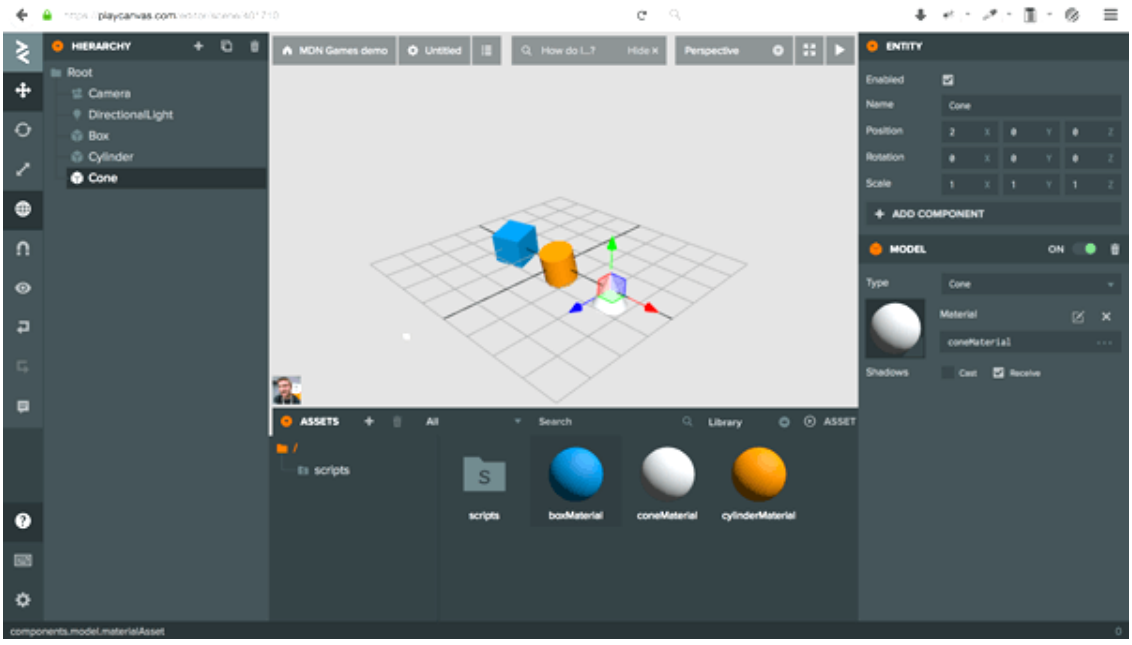

Рисунок 1.5 – Вебплатформа PlayCanvas

Вони є занадто перевантаженими функціонально для цілей вивчення та точкової розробки саме шейдерних ефектів, а доступ до низькорівневого коду шейдерів у них часто ускладнений або прихований за візуальними нодовими редакторами. Порівняльна характеристика розглянута нижче (табл. 1.1).

Таблиця 1.1 – Порівняльна характеристика існуючих платформ

Характеристика / Критерій	**ShaderToy**	**GLSL Sandbox**	**PlayCanvas / Spline**
Основне призначення	Навчання та обмін фрагментними шейдерами	Швидке прототипування фрагментних шейдерів	Розробка ігор та комплексний 3D-вебдизайн
Підтримка 3D-моделей	Відсутня	Відсутня	Повна підтримка імпорту
Доступ до GLSL коду	Прямий	Прямий	Ускладнений
Текстури та HDRI	Обмежена	Відсутня	Повноцінна підтримка
Соціальна екосистема	Розвинена	Відсутня	Командна розробка
Поріг входження	Високий	Вище середнього	Високий

Існує значна прогалина між простими 2D-пісочницями для фрагментних шейдерів та повноцінними, професійними 3D-рушіями. Відсутній спеціалізований інструмент, який би дозволяв зручно писати GLSL-код (як вершинний, так і фрагментний) та одразу застосовувати його до власних 3D-моделей із можливістю налаштування оточення.

1.4 Переваги системи, що розроблюється

Розроблювана онлайн-платформа Assimilate має на меті вирішити занотовані проблеми та запропонувати користувачам комплексне середовище оновленого бачення. До основних переваг системи порівняно з існуючими аналогами мають належати (табл. 1.2):

Таблиця 1.2 – Вимоги до переваг системи

№	Вимога	Опис вимоги
1	Робота з користувацькою 3D-геометрією	На відміну від аналогів, де рендеринг відбувається на площині, Assimilate в свою чергу дозволить завантажувати власні 3D-моделі та застосовувати написані шейдери безпосередньо до їхньої сітки та UV-координат, що будуть розраховуватись в залежності від складності та параметрів моделі, що завантажують
2	Комплексне налаштування сцени	Платформа має підтримувати розширені можливості візуалізації, включаючи завантаження користувацьких текстур, використання HDRI-карт для освітлення в сцені, а також налаштування базового кольору фону, в залежності від вибору користувача
3	Багаторівневий доступ до редагування	Система пропонуватиме три окремі редактори (вершин, площин та оточення) для широкого контролю над графічним конвеєром, а також спрощений режим із мінімальною панеллю керування для новачків, де можна буде перевірити базові налаштування коду для ознайомлення
4	Валідація та безпека	Кастомно написана система перевірки GLSL-коду має запобігати збоям WebGL-контексту в браузері, а Використання TypeScript у розробці системи забезпечить високу надійність архітектури та типізацію даних

Кінець таблиці 1.2

5	Розвинена соціальна взаємодія	Повноцінна екосистема з авторизацією, персональними профілями, можливістю публікувати проєкти, створювати новини, залишати коментарі та ставити лайки, це перетворить редактор на освітньо-творчу спільноту
6	Генерація медіаконтенту	Автоматичне, або за запитом створення користувацького прев'ю проєктів та вбудовані інструменти для запису gif-анімацій чи відео значно полегшить процес демонстрації результатів роботи

Зазначені вище переваги системи мають конкретизувати процес розробки та тестування платформи, а демонстрація шейдерів та сформований план розробки – оптимізувати програмну екосистему у вебсередовищі. Щодо інтеграції графічного інструментарію з соціальними елементами взаємодії між користувачами – воно має спростити залучення нових користувачів та функцій і забезпечити уніфікацію платформи в межах єдиного процесу розробки.

Реалізація підходу, що включає в себе багаторівневі задачі сприятиме значному зниженню порогу входження для початківців через спрощення режимів керування та інструментів автоматичної генерації медіаконтенту, а вся комплексна робота інкапсулюється. Водночас збережеться необхідна при роботі глибина налаштувань для досвідчених розробників. Гнучкість архітектури, що підтримує роботу з користувацькою 3D-геометрією, на рівні систем матеріалів та освітлення, перетворить веббраузер на готову до роботи та легкодоступну альтернативу стаціонарним графічним редакторам у контексті написання та налагодження коду.

З огляду на комплексність майбутньої розробки та поставлених завдань потрібно забезпечити безперебійну роботу багатокомпонентної системи шляхом формування технічного стеку проєкту. Для залучення сучасного інструментарію необхідно провести глибокий аналіз та сформувати деталізований перелік вимог до програмного забезпечення, яке розробляється.

Висновки до розділу 1

В рамках першого розділу проведено аналіз предметної сфери, що зв’язана з розробкою 3D-візуалізацій та шейдерних ефектів в веббраузері. Пройдено еволюцію технологій від залучення низькорівневих API, прямо до високорівневих абстракцій, використовуючи Three.js та React Three Fiber. Наведений із залученням прикладів перехід практик допускає розробникам не лише краще керувати життєвим циклом графічних об’єктів, але й спрощенням, інкапсуляції складної математики з використанням матричних перетворень та впровадженням 3D-сцен у декларативні вебінтерфейси із залученням апаратного прискорення браузера без втрати продуктивності.

Здійснено аналіз існуючих альтернативних рішень, таких як: ShaderToy, GLSL Sandbox та інших. Виявлено орієнтованість рішень до 2D-площин та генератора математичних об’єктів, що прибирає спеціалізацію роботи зі 3D-геометрією. Недоліки повного набору інструментів для завантаження полігональних сіток та налаштування UV-атласів робить наведені вище інструменти частково або повністю непридатними для тестування матеріалів фізично існуючих об’єктів, або як для тестування ассетів. З іншого боку, випередження показує, що багатокомлектні вебрушії представлені занадто перевантаженими середовищами за для вузьких цілей розробки шейдерних середовищ. Надмірна громіздкість програмного інтерфейсу з функціональністю систем відволікають від прямого програмування у графічному середовищі, що у свою чергу вимагає значних часових витрат на єдине налаштування системи.

Здобуті результати аналізу мають на меті довести актуальність створення проєкту, що має стати подальшою основою для вибору інструментарію розробки та формуванню специфікації вимог у наступному розділі кваліфікаційної роботи.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Перехід від етапу загального аналізу предметної області 3D-візуалізації у вебсередовищі до безпосередньої програмної реалізації оновлює необхідність ретельного планування технічної бази проєкту. Другий розділ кваліфікаційної роботи присвячено моделюванню об’єкта роботи, аналізу методів та технологій, а також формуванню широкої специфікації вимог до програмного забезпечення платформи «Assimilate». На основі особливостей предметної області здійснюватиметься вибір оптимального інструментарію середовища розробки, включаючи модулі, плагіни та інструментарій відлагодження, технологічний стек та розробляються базові інформаційні моделі.

Визначене повною мірою технологічне середовище має запобігти проблемному та неконтрольованому розширенню стеку розробки з його подальшим ускладненням. Під час пошуку потрібних технологічних рішень, їх якісний аналіз у поєднані з дослідженням технологічних ризиків унеможливить інтеграцію ненадійних рішень, або тих, що мають проблемну, неконтрольовану реалізацію. Навіть, якщо не брати до уваги стек технологій, подальше моделювання клієнтської та серверної частин, їх взаємодії – має гарантувати стійкість у поєднанні з цілісностю екосистеми. Даний підхід мінізуватиме проблеми можливого збою, відмови та помилок в обслуговувані та налагоджені конфігурації модулів під час додавання нових функцій, або відлагоджені старих із оновленням системи. Система інфраструктури бази даних, а також збереження файлів проєкту строго підпорядковуватиметься в майбутньому за принципами безпеки з використанням розмежування прав доступу. Що до взаємодії користувачів із сервісами – базуватиметься в першу чергу на можливості уніфікації та філософії відкритого коду шейдерів, при їх налаштуванні в межах широкої наукової та професійної спільноти.

Згідно з новими стандартами, вибір інструментарію та методів обробки шейдерного коду у вебсередовищі комп’ютерної графіки має здійснюватись на основі реальних методів розробки з науковою базою. Отримання інформації при

досліджені наукових статей дозволить обґрунтувати певний стек технологій, як WebGL, абстракцій Three.js [6] та реактивних підходів побудови інтерфейсів, при цьому опираючись на відомі вільні, ефективні практики розробки. Вся представлена база знань має стати гарантом, що обрані алгоритми рендерингу з функціями оптимізації та управління пам’яттю відповідають сучасним критеріям продуктивності.

Ці підходи якісно структурують функціональні можливості та нефункціональні обмеження майбутнього застосунка. Зазначені методи діють як фінальне технічне завдання, для проєктування архітектури та програмної реалізації.

2.1 Аналіз технологій 3D-візуалізації у вебсередовищі

Згідно з дослідженнями та аналізом у сфері комп’ютерної інженерії та взаємодії вебтехнологій, процес інтеграції інтерактивної тривимірної графіки у браузерне середовище зазнає суттєвих парадигмальних змін. Сьогоденний етап розвитку характеризується переходом до нативних браузерних стандартів, що забезпечують прямий доступ до апаратного прискорення графічного процесора. Проаналізувавши частину актуальних публікацій та відкритого коду з вільних ком’юніті, маємо, що фундаментом для створення додатків комп’ютерної графіки реального часу на сьогоднішній день представлений у якості базового джерела стандарт WebGL [5]. Ця технологія надає базову можливість виконання складних математичних обчислень та рендерингу полігональних об’єктів у режимі реального часу безпосередньо на стороні клієнта. Не залучуючи серверну частину – можна отримати швидкодійний застосунок для керування рендерингом, розділивши обрахункові спроможності від методів синхронізації та валідації. Серед варіантів також представлена ідея переносу бази для обчислення, якщо потрібна швидкодія без прив’язки до систем.

WebGL є низькорівневим прикладним програмним інтерфейсом, що базується на специфікації OpenGL ES. (рис. 2.1) При аналізі продуктивності

рендерингу, пряма робота з API WebGL вимагає значних зусиль для написання та компіляції низькорівневих шейдерних програм мовою GLSL [8]. Робота графічного конвеєра вимагає від розробника чіткого розуміння двох ключових етапів: вершинного (англ. vertex) та фрагментного (англ. fragment) компілювання коду. Першим етапом вершинний шейдер має обробити координати кожної полігональної точки на сітці моделі, що перетворить дані точки із локального простору, у простір екранного відсікання моделі. Цей процес можливий якщо помножити матрицю проєкції на матрицю виду. Далі за цими операціями йде процес растеризації пікселів, де після них фрагментний шейдер обчислюватиме фінальний колір кожного пікселя на екрані, із врахуванням освітлення [14], текстур та інших візуальних ефектів.

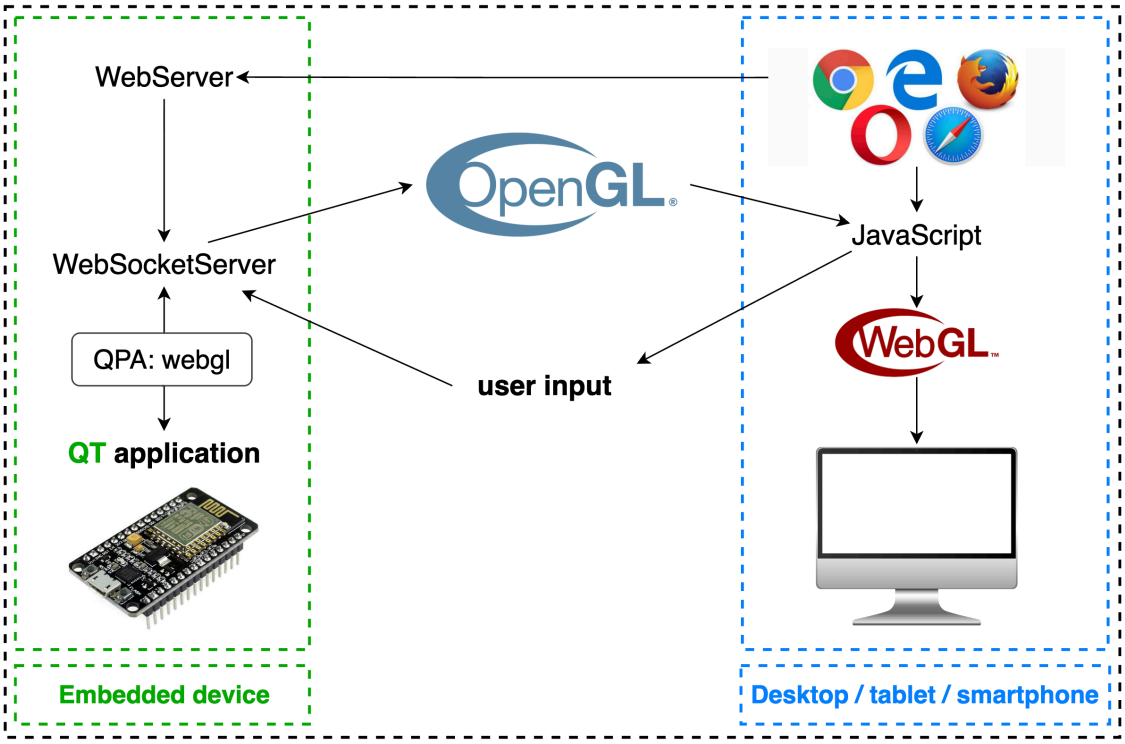

Рисунок 2.1 – Принцип роботи WebGL

Теоретично, розробнику б знадобилося самостійно ініціалізовувати графічний контекст кожного разу, виділяти буфера пам’яті для зберігання даних про вершини, та власноруч здійснювати множення матриць, щоб трансформувати певні об’єкти. Зазначений вище підхід має неймовірно високий поріг входження, й

сам вимагає глибоких математичних знань у прикладних сферах діяльності і є надто ресурсомістким, якщо розробляти об’ємні застосунки, або проводити обширні тести. Саме в процесі такої роботи фокус часто може мінятись від архітектури та планування, до низькорівневої математики.

З метою абстрагування від рутинних низькорівневих операцій вільна спільнота сформувала шар уніфікованих високорівневих інструментів. Стандартом де-факто у сфері веборієнтованої 3D-графіки стала бібліотека Three.js [6]. Цей інструмент реалізує об’єктно-орієнтований підхід, інкапсулюючи складні виклики WebGL-методів і надаючи розробникам вже готові класи для роботи з освітленням, полігональною геометрією та матеріалами. Однією з головних та незалежних переваг Three.js є реалізація архітектури графа сцени. Це представлена деревоподібна структура даних, що дозволяє об’єднувати об’єкти в ієрархічні групи. Коли розробник переміщує або обертає батьківський об’єкт у графі сцени – бібліотека перераховує локальні матриці трансформації автоматично, для всіх його дочірніх елементів, звільняючи завдання програміста від рутинної лінійної алгебри і складних математичних обчислень.

Окрім керування геометрією, Three.js бере на себе відповідальність за відсікання невидимих об’єктів, що значно зменшує кількість викликів малювання до графічного процесора. Важливою особливістю бібліотеки, що виділяє її як стандарт стає те, що вона надає можливість зберігати переваги високорівневого управління сценою, одночасно з цим залишаючи можливість прямої ін’єкції кастомного шейдерного коду у графічний конвеєр [17]. Через використання спеціальних класів, таких як на прикладі ShaderMaterial або RawShaderMaterial, розробник може передавати власні GLSL-програми та уніформи, зберігаючи контроль над конвеєром. Цей етап постає занадто важливо для процесів творчого програмування та створення складних візуальних ефектів у межах розроблюваної платформи.

Незважаючи на потужність Three.js, її імперативний стиль програмування вступає у конфлікт із сучасними декларативно-аналітичними підходами до

побудови користувацьких інтерфейсів (англ. User Interface, UI), зокрема з архітектурою популярної бібліотеки React [18] про яку йтиме мова нижче. В імперативній парадигмі розробник повинен вручну вказувати браузеру потрібну йому послідовність дій для мутації об’єктів у сцені. У масштабних застосунках із суворою типізацією та складним станом це часто призводить до втрати синхронізації між станом інтерфейсу та станом 3D-сцени, оскільки DOM-елементи та канвас існують у паралельних, не пов’язаних між собою життєвих циклах.

Для вирішення цієї архітектурної проблеми пропонується використання проміжного шару реконсиляції (як процедура контролю, звірка балансів та синхронізація даних) – бібліотеки React Three Fiber (рис. 2.2) [26].

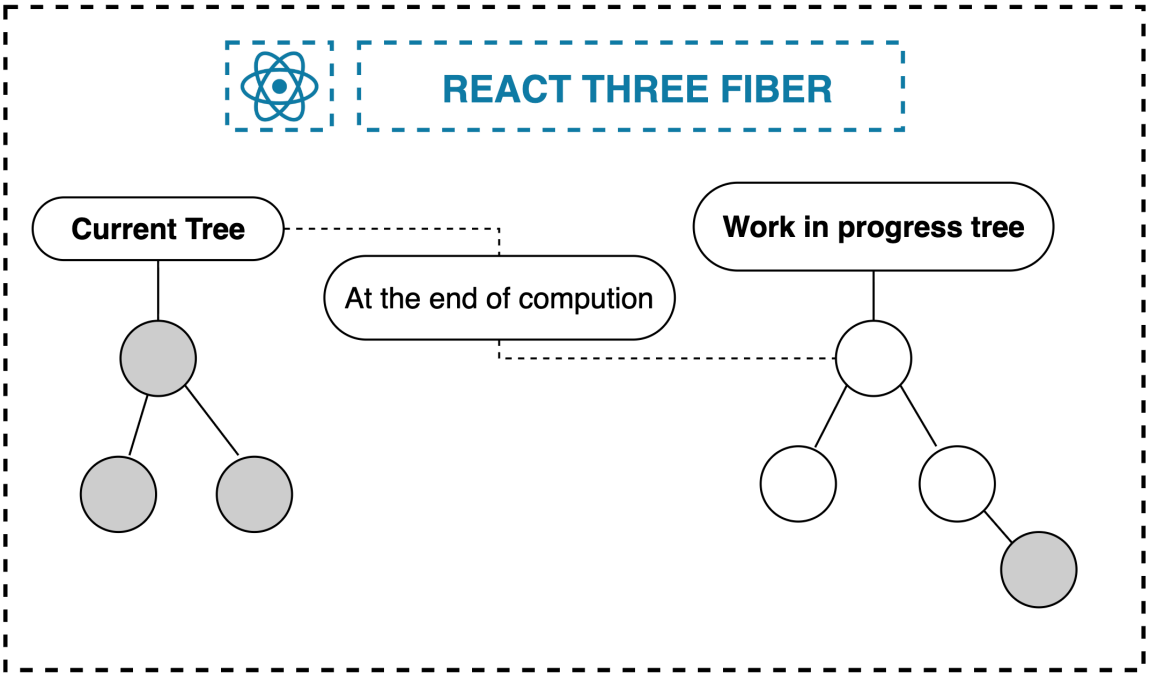

Рисунок 2.2 – Використання React Three Fiber

Цей інструмент дозволяє відносно легко інтегрувати інтерактивні 3D-моделі безпосередньо у React-застосунок. R3F виступає у ролі кастомного рендера, де подібно до React DOM вебсторінок або React Native, вже для мобільних пристроїв. Він транслює декларативні JSX-компоненти у відповідні екземпляри класів Three.js. Зазначений підхід гарно коригує парадигму веброзробки: 3D-об’єкти стають невід’ємною частиною загального дерева компонентів. Коли стан React-

компонента змінюється, як користувач, що змінює колір матеріалу через інтерфейс, R3F автоматично виявляє цю зміну за допомогою алгоритмів віртуального DOM і точково може оновлювати відповідну властивість об’єкта Three.js у пам’яті, що дозволяє автоматизувати управління їхнім життєвим циклом та значно оптимізувати рендеринг.

А отже, проведений аналіз технологій доводить підстави використання зв’язки React Three Fiber та Three.js як базового інструментарію графічного конвеєра. Подібна архітектура не лише розв’язуватиме проблему інтеграції 3D у сучасні інтерфейси, але й дозволить повністю сфокусуватися на логіці компіляції GLSL-коду, його валідації та розробці новітніх сервісів для роботи з шейдерами та 3D-моделями, таких як платформа «Assimilate» [1].

2.2 Аналіз сучасних підходів розробки клієнт-серверних застосунків

Паралельно з розвитком графічних технологій, новітні практики, еволюціонували й архітектурні підходи до побудови клієнтської частини комплексних вебплатформ. Для підтримки безперебійної роботи інтерактивних редакторів та соціальних механік сучасною індустрією остаточно затвердилася архітектура односторінкових застосунків SPA (англ. Single-Page Application). Перевагою наданого підходу стала відсутність необхідності повного перезавантаження сторінки під час навігації. У традиційних багатосторінкових застосунках (англ. Multi-Page Application, MPA) перехід за посиланням спричиняє повне знищення поточного документа і завантаження нового. У контексті із 3D-графікою це стало б катастрофічним рішенням із наслідками, що б значно скоротили варіативність застосунку, оскільки щоразу призводило б до втрати «важкого» контексту WebGL у пам’яті браузера користувача, примушуючи його наново завантажувати мегабайти текстур, парсити геометрію та компілювати шейдери. Натомість, SPA-архітектура використовує механізми маршрутизації на стороні клієнта, при цьому уникаючи повторної ресурсомісткої ініціалізації сцени під час навігації користувачем між розділами вебзастосунку.

Одним із стандартів інструментів для розробки SPA-інтерфейсів представлена бібліотека React (рис. 2.3). Як показує досвід створення сучасних вебінтерфейсів, компонентна архітектура React може ефективно декомпозитувати складні UI на незалежні модулі, що перевикористовуються [18].

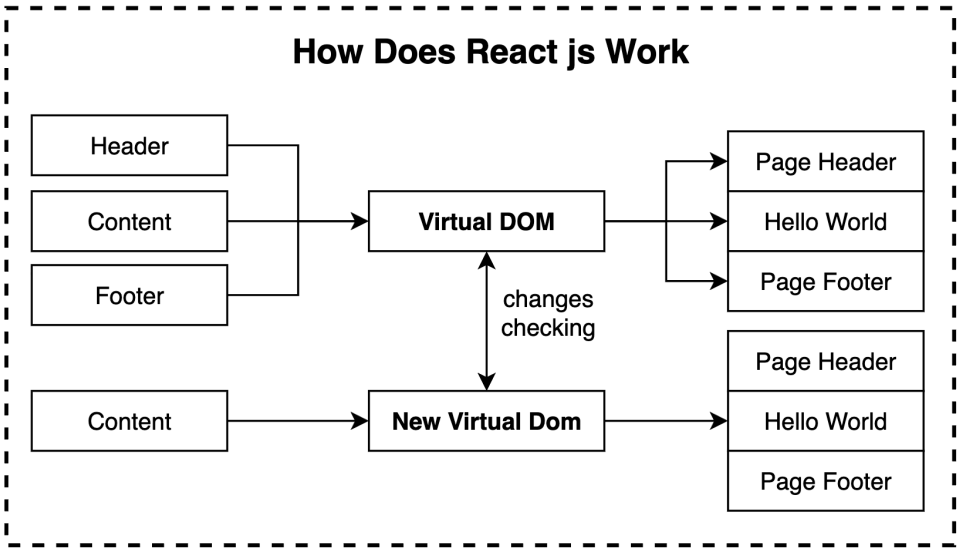

Рисунок 2.3 – Як працює React

Для підвищення надійності таких масштабованих систем стандартом є використання суворо типізованої мови TypeScript [24]. Впровадження TypeScript на етапі фронтенд-розробки забезпечує статичну перевірку типів, якісно знижує кількість логічних помилок [15]. Ця перевага є особливо актуальною під час передачі складних структур даних, прикладом може слугувати параметри шейдерів, векторів або масивів геометрії між компонентами застосунку, а також під час опрацювання відповідей від серверної частини бази даних.

Важким аспектом розробки багатокомпонентних систем є управління потоками даних. У застосунках, де кожна зміна коду в редакторі повинна миттєво оновлювати 3D-модель, а авторизація – відкривати нові маршрути, використання стандартного передавання властивостей між рівнями компонентів є повністю неефективним (рис. 2.4). Передача через десятки проміжних компонентів, які його навіть не використовують, призводить до заплутаної архітектури та зниження загальної продуктивності застосунку.

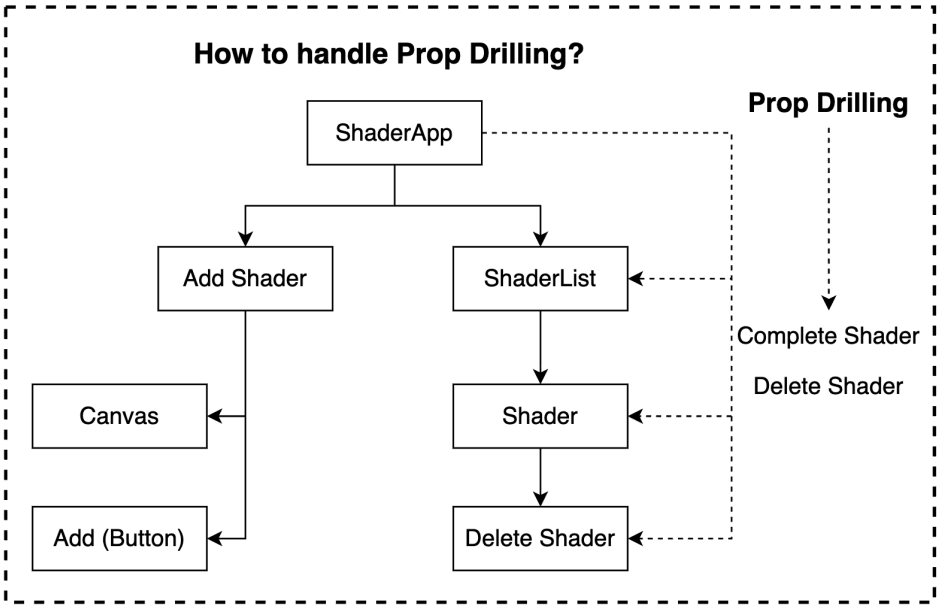

Рисунок 2.4 – Що робити зі шляхами передачі

Згідно з дослідженнями у сфері побудови масштабованих React-застосунків, оптимальним рішенням для цієї проблеми є архітектурний патерн централізованого стану на базі менеджера Redux (рис. 2.5) [4].

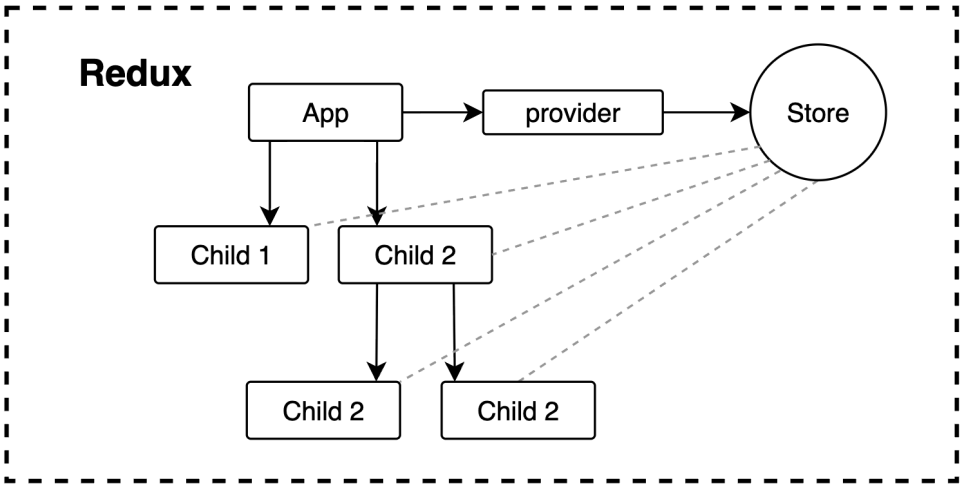

Рисунок 2.5 – Переваги взаємодії Redux

Сам по собі Redux формує єдине джерело істини, гарантуючи повну передбачуваність змін стану системи та спрощуючи синхронізацію складних UI-елементів, при цьому безпосередньо покращуючи показники юзабіліті та загального сприйняття користувачем [29]. Крім того, само використання прошарків

дозволяє елегантно обробляти асинхронні запити, такі як завантаження 3D-моделей з сервера.

Для реалізації соціальної складової платформи – необхідна надійна серверна інфраструктура. Сучасні тенденції розробки вже якісно перейшли від монолітних бекенд-серверів до використання хмарних рішень за моделлю Backend-as-a-Service, або BaaS, зокрема екосистеми Firebase [2]. Використання на прикладі документо-орієнтованих NoSQL баз даних дозволяє ефективно працювати з масивами інформації. Замість жорстких реляційних таблиць, дані зберігаються у вигляді гнучких колекцій та JSON-подібних документів. Ця система лаконічно підходить для зберігання конфігурацій шейдерних проєктів, які можуть мати різну кількість матеріалів, налаштувань освітлення та коментарів. Зазначена архітектура забезпечує масштабовану обробку даних і синхронізацію соціальних взаємодій між клієнтами в режимі реального часу [3].

Одним з базових критеріїв вибору хмарних рішень постає забезпечення високого рівня безпеки даних. Як зазначається у дослідженнях архітектури корпоративних систем на базі React і TypeScript, сучасні платформи повинні проєктуватися з урахуванням принципів нульової довіри [20]. У контексті, при використанні BaaS-платформ ця реалізація постає через строгі серверні правила безпеки, що при криптографічному шифруванні перевіряються на використання наданих ключів, до того моменту як будуть використані в якості транзакцій до JWT-токенів. Це допоможе уникнути несанкціонованого внесення змін до чужого користувацького шейдерного проєкту, або можливості перехопити чужі дані користувачів із персональною інформацією.

Зазначений набір технологій є відповідно надійним для забезпечення високої продуктивності у вебсередовищі, додаючи безпеку, масштабованість та повною мірою задовольняючи архітектурні вимоги до розробки платформи.

2.3 Формування вимог до програмного забезпечення

Для успішної майбутньої реалізації проєкту необхідно чітко визначити функціональні та нефункціональні вимоги до програмного продукту. Розробка комплексної архітектури вимагатиме детального опрацювання підсистем, кожна з яких відповідає за окрему бізнес-логіку вебзастосунку (табл. 2.1):

Таблиця 2.1 – Функціональні вимоги

№	Вимога	Опис вимоги
1	Підсистема авторизації та управління користувачами	Реєстрація та вхід через email/пароль або Google-акаунт, створення профілю у Firestore, розмежування прав доступу (користувач / адміністратор)
2	Підсистема управління контентом	Збереження, редагування, видалення та публікація 3D-проєктів; створення новин, залишення коментарів до проєктів, система оцінювання
3	Підсистема 3D-редактора	Завантаження користувацьких 3D-моделей, текстовий редактор коду для написання вершинних та фрагментних шейдерів, завантаження та застосування текстур і HDRI-карт, налаштування параметрів фону
4	Підсистема валідації та експорту	Перевірка синтаксису шейдерного коду, автоматична генерація прев’ю, експорт сцени у форматі .gif або відеофайлу
5	Адміністративна підсистема	Панель для модерації користувачів, проєктів, новин та коментарів

Наведені у таблиці функціональні вимоги мають на меті описати базовий процес розробки, інструментарій та модульну структуру системи. Вектор, що визначений за системою, має напрям: від моменту авторизації користувача на

платформі, до безпосереднього редагування шейдерного коду, а після – збереження та публікації проєкту. Для майбутнього проєктування системи необхідно в першу чергу розкрити сутності кожної з описаних підсистем.

1) Підсистема авторизації та управління користувачами є первинною точкою входу в систему. Сама система авторизації повинна забезпечувати безшовну та безпечну автентифікацію, при цьому відповідно підтримуючи як традиційний метод у випадку електронної пошти та пароля, так і авторизацію через сторонні сервіси на прикладі OAuth від Google. Після успішного входу система автоматично створить або оновить документ користувача у базі даних Firestore [2], де зберігається його публічна інформація, налаштування профілю та рівень доступу.

2) Підсистема управління контентом бере за основу формування соціальної структури. Вона інтегруватиме можливості зі збереження поточного стану шейдерного коду та базової системної конфігурації під виглядом комплексних JSON файлів у хмарному середовищі вебплатформи. Реалізація функцій коментарів та оцінювання опублікованих проєктів спрямована на формування відкритого ком’юніті авторів та симулювати освітній процес.

3) Підсистема 3D-редактора є центральним і найбільш технологічно складним елементом застосунку. Цей модуль повинен надавати користувачеві зручний текстовий редактор із підсвічуванням синтаксису GLSL. Одночасно інтерфейс має забезпечувати можливості завантаження зовнішніх ассетів: полігональних моделей у популярних форматах, таких як STL та інші. Різноманітних текстур для змінних каналів матеріалу (альбедо, нормалі, шорсткість) та карт оточення для забезпечення реалістичного глобального освітлення. Всі наведені параметри повинні миттєво синхронізуватися з графічним конвеєром.

4) Підсистема валідації та експорту поєднує у собі функції контролю за якістю коду, що йдуть суміжно при генерації медіа даних проєкту. Вона повинна призначатися для перехоплення викликів помилок за час компіляції шейдерів, що

має запускатись ще до виводу зображення на екран користувача, роблячи компіляцію у вигляді даних реального часу, реактивно виводячи повідомлення у зрозумілому форматі користувачам браузера на екран. Крім зазначеного, автоматизовані інструменти для генерації та запису анімацій надають користувачам можливості створювати демонстраційний матеріал прямо в вебсередовищі із збереженням результату на пристрої.

5) Адміністративна підсистема має забезпечувати життєздатність та чистоту контенту на платформі. За допомогою окремого спеціального інтерфейсу модератори матимуть можливість керувати користувацькими даними, видаляти порушення або публікувати глобальні новини про оновлення сервісу, що сприятиме підтриманню здорової екосистеми.

Враховуючи, що специфіка роботи з тривимірною графікою та рендерингом у реальному часі в середовищі браузера, потребує значних обчислювальних ресурсів графічного процесора – реалізації вказаних підсистем недостатньо для створення комерційно придатного та якісного продукту [8]. Розробка вебзастосунків такого класу суттєво відрізняється від створення нативних десктопних програм. Обмеження браузерного середовища, такі як суворі ліміти виділеної оперативної пам’яті, особливості фонової роботи збирача сміття в інтерпретаторі JavaScript та можливі потенційні затримки в мережевому середовищі, саме вони можуть створювати певні виклики при розробці. Ігнорування зазначених системних та апаратних обмежень з великим шансом призведе до деградації якості візуалізації, втрати кадрів або, можливо, до повного аварійного завершення роботи вкладки. Щоб гарантувати стабільність роботи системи, а також максимізувати комфорт користувачів для пристроїв різних типів – призначено встановлення набору деяких жорстких технічних критеріїв. Співвідносно зі сучасними стандартами інженерії коду, де успішність проєкту має вимірюватись не тільки наявністю необхідного зазначеного функціоналу, але й ефективністю його використання при нештатних, аварійних умовах.

Окрім функціонального компоненту в архітектурі платформи – необхідно забезпечити використання конкретних, нефункціональних вимог ціллю яких є регламент загальної частини продуктивності системи, її масштабованості, бази даних та надійності в умовах ненадійного користувацького коду. Архітектурні рішення, що мають заздалегідь механізми захисту па прикладі витоків пам’яті під час рендернгу складного шейдеру 3D-сцени. При цих зауваженнях, ці запроваджені алгоритми оптимізації та правила безпеки мають узгоджено працювати максимально прозоро для розробника. Користувачі у свою чергу не повинні відчувати жорсткі обмеження керування (табл. 2.2).

Таблиця 2.2 – Нефункціональні вимоги

№	**Вимога**	**Опис вимоги**
1	Продуктивність	Платформа повинна забезпечувати рендеринг 3D-сцен із прийнятною частотою кадрів (від 30 до 60 FPS) у сучасних веббраузерах на пристроях із середньою обчислювальною потужністю, як ноутбуки повсякденного користування [27]
2	Надійність	Застосунок має бути стійким до помилок у користувацькому коді шейдерів, не допускаючи критичного «падіння» всієї вебсторінки
3	Масштабованість	База даних та архітектура компонентів повинні підтримувати збільшення кількості користувачів та обсягу збереженого контенту без втрати швидкодії
4	Кросплатформність та адаптивність	Інтерфейс платформи (зокрема галерея робіт та профілі) має коректно відображатися на різних типах пристроїв, включаючи мобільні телефони та планшети [13]

При виконанні наведених нефункціональних вимог із таблиці проєкт отримає забезпечення життєздатності з більшою конкурентоспроможністю при розробці вебплатформи. Робота з рендерингом графіки в тривимірному просторі та компіляцією шейдерного коду напряму в браузері ставить на меті довести небезпеку створення значного та зазвичай неочікуваного впливу на програмний компонент, що зрештою додасть більше вимог у критеріях продуктивності. Певні модулі, як механізми знаходження та ізоляції помилок мають йти основою для гарантій безперебійної роботи користувацької системи, при некоректному синтаксисі GLSL, або компонентів від вводу користувача. Приведена проблема є типовим сценарієм в процесі розробки та експериментів програмного продукту.

Зокрема, механізми ізоляції помилок та управління контекстом мають гарантувати безперебійну роботу вебзастосунку користувача. Типовим сценарієм у процесі написання шейдерів є допущення синтаксичних помилок або створення випадкових нескінченних циклів на прикладі алгоритмів «Raymarching». Платформа повинна вміти перехоплювати події втрати контексту у WebGL та безпечно відновлювати його без необхідності повного перезавантаження сторінки користувачем. Безпека даних також вимагає налаштування політик CORS (англ. Cross-Origin Resource Sharing) для безпечного завантаження користувацьких 3D-моделей та текстур із хмарного сховища.

Сформований комплекс функціональних та нефункціональних вимог утворює вичерпну специфікацію програмного забезпечення. Наданий структурований перелік критеріїв стане базою для подальшого вибору конкретних технологічних рішень, побудови структури бази даних та проєктування архітектури платформи на наступних етапах розробки.

Висновки до розділу 2

За розділом проведено комплексний аналіз сучасного стану інструментарію та технологій, що стане основою, необхідною для реалізації вебплатформи. Із використанням актуальних наукових публікацій, досліджено еволюцію

веборієнтованої 3D-графіки. Доведена доцільність використання стандарту WebGL у поєднанні з високорівневими абстракціями у якості Three.js та React Three Fiber, що дозволить в подальшій роботі можливість інтегрувати складний графічний конвеєр у сучасні користувацькі інтерфейси без втрати продуктивності.

Проаналізовано сучасні підходи до побудови клієнт-серверних вебзастосунків. Визначено, що концепція односторінкового застосунку на базі бібліотеки React є оптимальним вибором при збереженні графічного контексту WebGL. Обґрунтовано необхідність використання жорсткої типізації TypeScript, менеджера глобального стану Redux та хмарної інфраструктури Firebase для забезпечення безпеки за принципом нульової довіри та миттєвої синхронізації соціальних взаємодій між користувачами.

На базі проведеного аналізу сформовано детальну специфікацію вимог до програмного забезпечення. Визначено п’ять ключових функціональних підсистем:

- авторизації;
- управління контентом;
- 3D-редактора;
- валідації;
- адміністрування.

Проведено аналіз та встановлено певні нефункціональні критерії із забезпеченням продуктивності у 30–60 FPS, для підвищення надійності платформи та оптимізації швидкодії обробки шейдерного коду застосунку.

Сформований технологічний стек та деталізована специфікація вимог створюють міцне підґрунтя для переходу до наступного етапу розробки проєкту – безпосереднього проєктування архітектури системи, візуального моделювання взаємодій та побудови UML-діаграм, що буде розглянуто у наступному розділі.

3 АРХІТЕКТУРА, МОДЕЛЮВАННЯ ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Етап проєктування архітектури має включати в себе побудову моделі взаємодії користувачів із системою за допомогою інструментарію UML-діаграм (англ. Unified Modeling Language), описом алгоритмів авторизації та маршрутизації в системі, з повноцінним моделюванням внутрішнього зв’язку між клієнтським інтерфейсом, графічним представленням та хмарною інфраструктурою бази даних.

Окремо спроєктована та затверджена архітектура йде основою у реалізації системи, задля її стабільності, стійкості до обчислювальних навантажень від 3D-рендерингу, стрес-тестів та зручності у підтримці програмного коду на етапі експлуатації.

3.1 Визначення архітектури системи

При забезпечені високої інтерактивності та швидкодії, у роботі з 3D-графікою у браузері, базовою архітектурною моделлю клієнтської частини після пошуку варіантів – обрано концепцію односторінкового вебзастосунку. Якщо порівнювати з класичними багатосторінковими моделями, що вважались сталими, ще декілька років тому, SPA завантажує основний HTML-документ лише один раз, а подальше оновлення контенту з можливими переходами між сторінками відбуваються динамічно за допомогою виконання JavaScript-коду (рис. 3.1).

Зазначений підхід як найкраще усуває необхідність перезавантаження сторінки, роблячи дію необхідною у крайніх випадках, це дозволяє зберігати та реактивно оновлювати активний контекст WebGL-сцени та забезпечувати плавний користувацький досвід, максимально наближений до нативних десктопних програм.

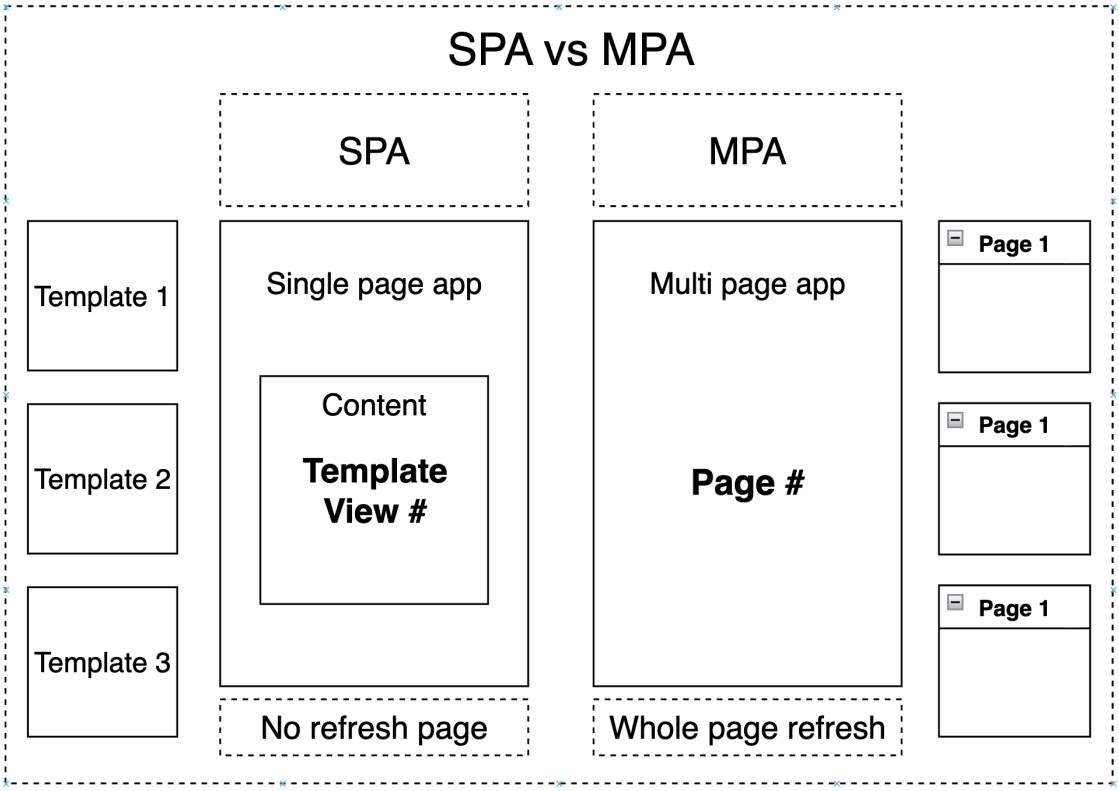

Рисунок 3.1 – Концепція односторінкового застосунку

Збереження контексту графіки WebGL постає якісною вимогою для подібних класів програмного забезпечення. Ініціалізація графічного контексту, компіляція початкового шейдерного коду та завантаження текстур у відеопам’ять GPU є ресурсомісткими операціями для більшої кількості персональних комп’ютерів або ж у випадку робочих станцій. При використанні класичної багатосторінкової архітектури кожен перехід між розділами вебзастосунку призводив би до повного знищення та подальшого перестворення графічного контексту з нуля. Ця особливість викликала б значні затримки в інтерфейсі, «зависання» та надмірне споживання системних ресурсів клієнтського пристрою, а найголовніше – втрату контексту редактора, що вело б до знищення користувацькі моделі на канвасі.

Побудова загально визначеної архітектури користувацької платформи має спиратись на три визначені основні взаємодіючі технічні рівні:

– рівень користувацького інтерфейсу та бізнес-логіки;
– рівень графічного рендерингу, 3D-конвеєр;
– рівень збереження даних та серверної взаємодії.

Рівень користувацького інтерфейсу реалізується на базі сталої бібліотеки React. Її компонентна архітектура дозволяє декомпозитувати комплексний інтерфейс застосунку на малі, незалежні, ізольовані модулі, які зможуть легко замінюватись, або перероблюватись у разі необхідності, як на прикладі майбутніх панелі налаштувань матеріалів, текстового редактору коду, системи навігації та галереї проєктів. Такий принцип спрощує розробку, тестування та подальше масштабування системи. Для забезпечення надійності архітектури та уникнення логічних помилок на етапі компіляції використовується раніше зазначена суворо типізована мова програмування TypeScript. Її глибока типізація структур даних гарантує передбачуваність взаємодії між компонентами при розробці комплексних модулів, які мають спиратись на відмовостійкість, це є важливою складовою при передачі масивів даних, складних конфігурацій 3D-моделей та параметрів користувацьких шейдерів.

Для ефективного управління даними на рівні з клієнтською частиною має застосовуватись архітектурний патерн централізованого стану за допомогою менеджера Redux. У представлені, коли застосунок містить у собі велику кількість взаємозалежних модулів, зміна коду має негайно відображатися редактором на 3D-моделі, а зміна статусу авторизації – у навігаційному меню для використання стандартного прокидання властивостей (англ. Props Drilling). Названа особливість може призвести до заплутаності із паралельним зниженням продуктивності системи. При застосуванні, Redux створює єдине джерело істини, кожен компонент якого може асинхронно отримувати окремо ті дані, що лише необхідні його контексту для коректного рендерингу.

Рівень графічного представлення концептуально відокремлений від стандартного дерева об’єктів документа браузера і базується на бібліотеці Three.js. Для її повної та стабільної інтеграції з реактивною екосистемою використовується якісний проміжний рівень – React Three Fiber. Бібліотека R3F виступає у ролі кастомного рендеру середовища, що транслює декларативний синтаксис JSX-компонентів у відповідні імперативні виклики до методів Three.js.

Використовуючи надані методики – розв’язується проблема управління життєвим циклом тривимірних об’єктів, на прикладі їх створення, оновлення параметрів, видалення з пам’яті, дозволяючи керувати 3D-сценою за тими ж принципами, що і звичайними HTML-елементами.

Рівень збереження даних спроєктовано за принципом безсерверної архітектурної платформи, а з використанням хмарної інфраструктури Firebase від Google, вона представлена за моделлю Backend-as-a-Service (рис. 3.2). Сама база даних виступатиме, як один з сервісів, коли розробник зможе доєднати та налаштувати незалежно від обраного середовища розробки.

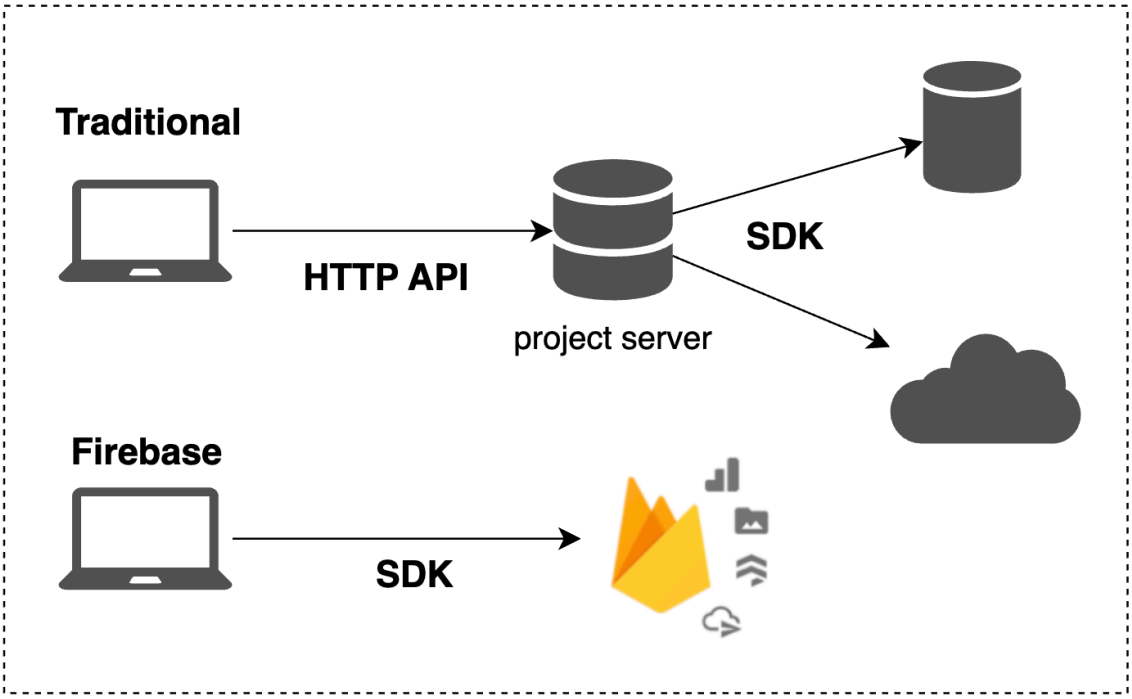

Рисунок 3.2 – Що таке Firebase

До складу бекенд-рівня входять два основні обрані сервіси:

1) Firebase Authentication відповідає за забезпечення безпечних протоколів реєстрації та авторизації з управлінням сесійними даними користувачів, також з підтримкою автентифікації через електронну пошту або сторонніх OAuth-провайдерів (рис. 3.3), що далі використовуватиметься. Окрім того, сервіс забезпечує централізоване управління обліковими записами клієнтів, підтримує багатофакторну аутентифікацію та автоматичне оновлення їх токенів,

представлене поєднання легко розширює безперервність роботи системи. Інтеграція з іншими компонентами Firebase поєднує комплексну логіку доступу до даних-подій у режимі реального часу, з гнучкістю при виборі методів аутентифікації, що сприятиме широкому спектру вебзастосунків та модулів.

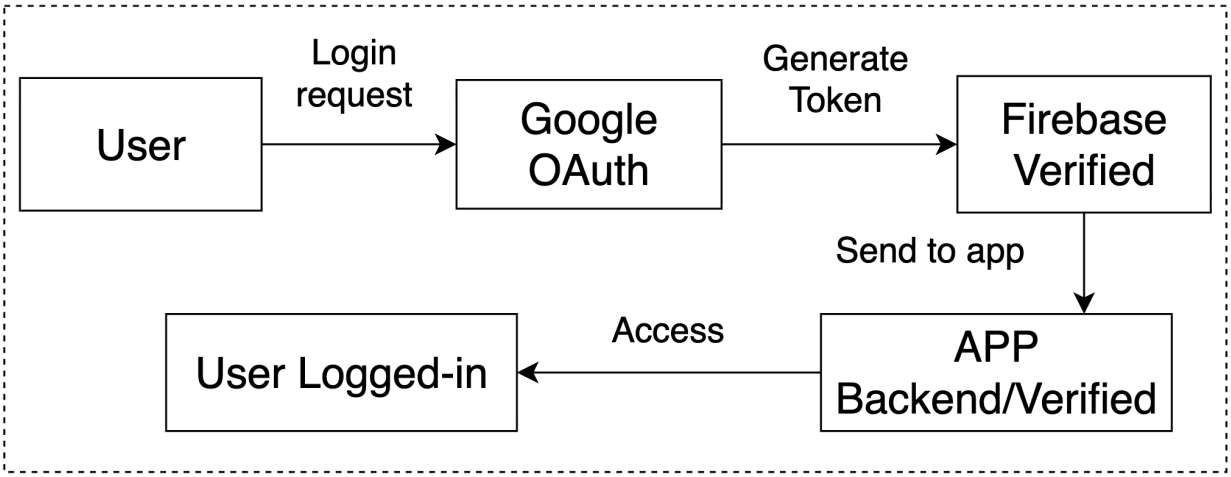

Рисунок 3.3 – Автентифікація за допомогою акаунта Google

2) Cloud Firestore – це документо-орієнтована NoSQL база даних в режимі реального часу, вона здатна забезпечувати збереження даних в ієрархічному потоку користувацького проєкту. Вона здатна забезпечити в подальшому структурування профілів користувачів бази, конфігурації 3D-сцен, JSON користувацького шейдерного коду, метадані проєктів, а також дії користувачів, представлені як коментарі, лайки, або новини (рис. 3.4).

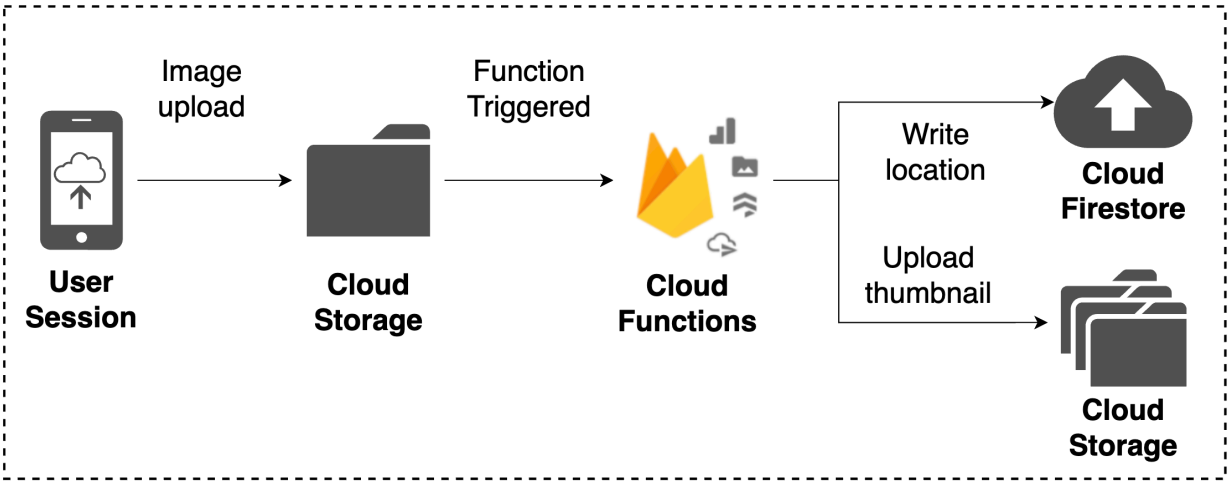

Рисунок 3.4 – Алгоритм роботи Firestore

Взаємодія між клієнтським застосунком та хмарною базою даних відбувається за умови використання асинхронних запитів. Їх взаємодія попереджує блокування основного потоку виконання програми та забезпечує стабільнішу частоту кадрів та безперервність рендерингу під час завантаження чи збереження масивів даних.

Безпека передачі даних зі захистом цілісності інформації реалізується на рівні серверних правил бази даних. В наведеній архітектурі SPA клієнтський код є відкритим, тобто безпосередній, односторонній контроль доступу не може бути цілком делегований фронтенд-частині клієнта. Серверні правила алгоритмічно перевіряють кожен вхідний запит до бази даних, автоматично зіставляючи криптографічні JWT-токени поточного користувача в системі, з ідентифікаторами власників ресурсу. Зазначена логіка гарантує, що операції модифікації або видалення проєктів здатні виконуватись виключно їхніми авторами чи системними адміністраторами, повністю відповідаючи парадигмі нульової довіри.

3.2 Формування структури застосунку

Формування логічної структури застосунку, надасть можливість візуалізувати архітектуру на рівні взаємодії користувачів із системою та підхопити алгоритми візуалізації та маршрутизації. Для самого візуального представлення функціональних вимог та за допомогою моделювання поведінки системи використано діаграми уніфікованої мови моделювання та блок-схеми алгоритмів.

UML-діаграми виступатимуть у ролі загальноприйнятих стандартів візуального моделювання середовища. Це має на меті демонстрації структури розроблюваного застосунку, включаючи в собі розподіл ролей, акторів та логіку бізнес-процесів ще до початку написання програмного коду (рис. 3.5). Їх застосування забезпечує повне розуміння функціональних можливостей системи та створює надійну документовану базу проєкту.

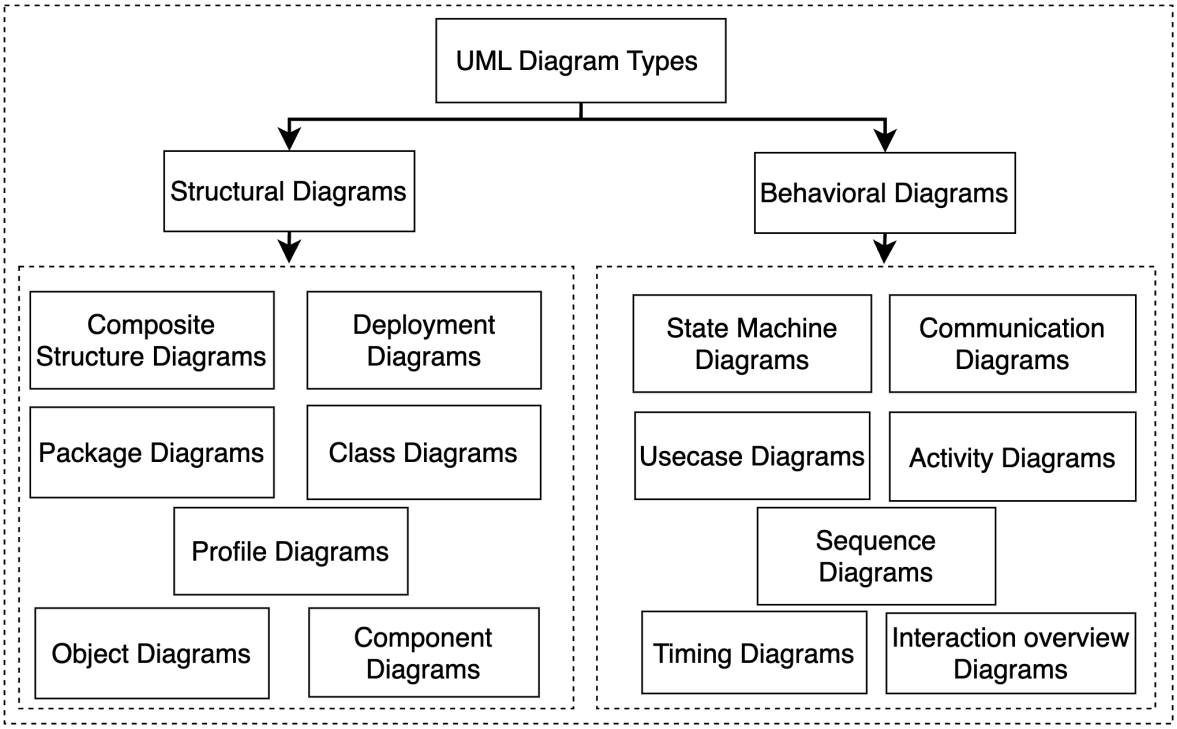

Рисунок 3.5 – Типи UML-діаграм

Діаграма варіантів використання є інструментом об’єктно-орієнтованого проєктування. Вона дозволяє абстрагуватися від технічних деталей програмної реалізації та починати розбор бізнес-логіки застосунку з точки кінцевого споживача. У якості розроблюваної платформи, складні інструменти редагування тривимірної графіки критично важливо поєднувати із механізмами соціальної взаємодії платформи, що до їх чіткого визначення та розмежування ролей. Наданий контекст дозволяє уникати конфліктів доступу до інформації, коректно спроєктувати структуру розроблюваної бази даних для зазначених вище груп користувачів та визначити необхідні обмеження із відображенням компонентів користувацького інтерфейсу. Візуалізація прецедентів ідентифікує базові функції системи, вони доступні відвідувачам, з розширенням можливостей, що стануть відкритими лише після проходження процедури ідентифікації.

За початком розробки структури потрібно з побудови діаграми прецедентів, що визначає межі системи «Assimilate» та опише можливі варіанти взаємодії різних груп користувачів (акторів) із закладеним функціоналом (рис. 3.6).

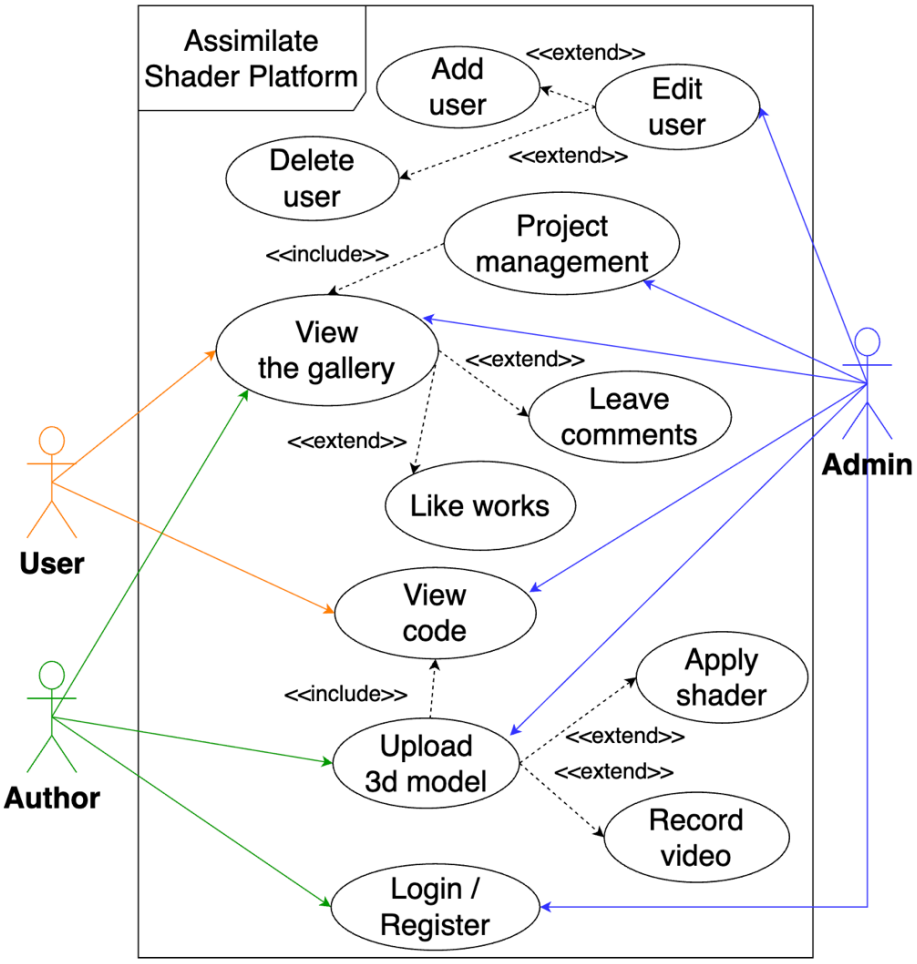

Рисунок 3.6 – Діаграма варіантів використання платформи

При створені моделі виділено три основні ролі акторів з різним рівнем привілеїв:

– користувач «User»: певна неавторизована роль (гість), що не надала даних для авторизації. Користувачу обмежується доступ до системи у певних діях: він лише може переглядати загальну галерею робіт, на сторінці відкривати код базових шейдерів лише для ознайомлення, та використовувати редактор без збереження;

– автор «Author»: зареєстрований та повністю авторизований користувач системи. Актор успадковує усі зазначені можливості базового користувача, але отримує більш розширений функціонал. Як залишати коментарі та оцінювати роботи інших авторів. Крім того, автор отримує повний доступ до завантаження власних 3D-моделей, що включає роботу з редактором коду та розширюється додатковими функціями застосування шейдерів і генерації;

– адміністратор «Admin»: користувач із повним доступом. Окрім відкритого функціоналу автора, адміністратор має ексклюзивні права на управління контентом та модерацію спільноти інших користувачів. До його прецедентів входить управління профілями користувачів та за необхідності розширюється їх додаванням або видаленням.

Захист приватних маршрутів, таких як панель модерації або сторінка редагування власного профілю, має виконуватися алгоритмічно, спираючись на поточний глобальний стан сесії користувача. Для деталізації процесу розмежування прав доступу та маршрутизації всередині застосунку розроблено алгоритм проходження авторизації та розподілу функціоналу (рис. 3.7).

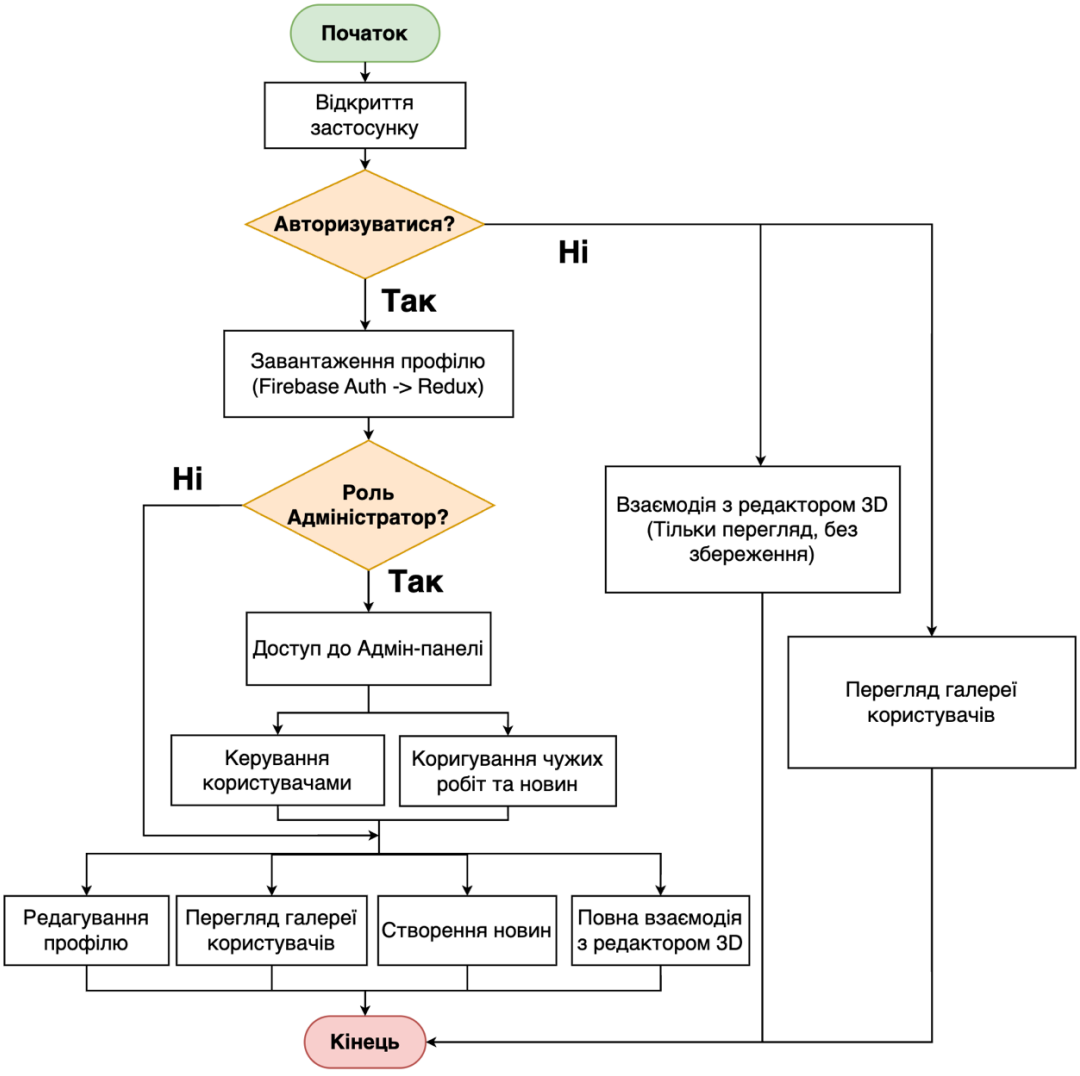

Рисунок 3.7 – Блок-схема алгоритму авторизації та маршрутизації користувачів

На початку завантаження система перевіряє наявність активної сесії користувача. Якщо користувач відхилятиме авторизацію або його сесія відсутня, маршрутизатор обмежить навігацію лише вільними публічними сторінками: переглядом галереї користувачів та взаємодією з 3D-редактором та унеможливить збереження проєкту у базу даних.

Якщо авторизація успішна – система виконує запит до сервісу Firebase Auth, де отримує дані профілю та зберігає їхній стан у глобальному сховищі, ця дія виконується за допомогою менеджера стану Redux. Використання Redux дозволяє компонентам системи миттєво отримувати інформацію про поточного користувача без необхідності повторних асинхронних запитів до сервера.

Перевірка ролі користувача на наявність прав адміністратора:

– якщо роль дорівнює «Адміністратор», користувач отримує доступ до прихованої до цього моменту адмін-панелі, де збережена логіка керування користувачами, на прикладі додавання/видалення, та коригування чужих робіт і новин. Окрім панелі модерації адміністратор також має доступ до всього базового функціоналу застосунку;

– якщо роль належить звичайному користувачеві, автору, система сама обходить панель адміністратора і надає доступ до особистого кабінету. Користувач може редагувати власноруч свій профіль, переглядати роботи, створювати новини та використовувати 3D-редактор із повним функціоналом збереження модифікацій.

Структура спирається на безпечному розмежуванні даних на стороні клієнта і логічно підготовлює застосунок до зв’язування графічного інтерфейсу з відповідними колекціями у базі даних.

3.3 Огляд взаємодії компонентів застосунку

Розробка вебзастосунку, у поєднанні інструментів різного типу рендерингу у реальному часі тривимірної графіки та соціальної взаємодії користувачів платформи, вимагає регламентації потоків даних. Для запобіганням конфліктів глобального стану зі зниженням продуктивності під час компіляції важких

шейдерних сцен, архітектура платформи «Assimilate» поділена на ізольовані логічні модулі один від одного. Огляд цих модулів, їх взаємодії йде з визначення головних компонентів системи, де кожен з них відповідає за окрему структуру бізнес-логіки. Вона йде від управління користувацьким інтерфейсом до обробки даних у хмарному середовищі.

Основні структурні компоненти системи та їхнє технологічне призначення наведено нижче (табл. 3.1).

Таблиця 3.1 – Основні компоненти програмної платформи

Модуль / Компонент	**Призначення в системі**	**Технологія реалізації**
Користувацький інтерфейс	Відображення візуальних елементів (меню, редактор коду, галерея), обробка подій введення від користувача	React, TypeScript
Менеджер глобального стану	Зберігання та синхронізація даних сесії, профілю авторизованого користувача та обмежених налаштувань проєкту	Redux
Графічний канвас	Парсинг та компіляція GLSL-коду, накладання текстур, HDRI-карт, та фінальний рендеринг сцени	React Three Fiber (R3F), Three.js
Хмарна база даних	Збереження даних в ієрархічній структурі проєктів, управління користувачами, обробка соціальних взаємодій	Firebase Auth, Cloud Firestore
Генератор медіаконтенту	Перехоплення кадрів з WebGL-контексту, створення статичного прев’ю, GIF-анімації або запис відео	Кастомні TS-скрипти, Web APIs

Архітектурна взаємодія між наведеними компонентами базується на принципах реактивності інтерфейсу та односпрямованого потоку даних. Головним вузлом клієнтської частини виступає зв’язок між інтерфейсом та графічним конвеєром. Якщо користувач вводить GLSL-код або змінює параметри матеріалів через UI-панелі, самі зміни фіксуються у локальному стані React-компонентів. За допомогою механізму передачі властивостей, оновлені дані надходитимуть до компонента Canvas R3F. Проміжний шар перехоплює декларативні зміни та може динамічно оновлювати властивості кастомно створеного матеріалу на полігональній сітці 3D-моделі, після цього викликаючи перекомпіляцію шейдера графічним процесором без перезавантаження сторінки користувача.

Для запобігання постійній передачі даних у глибоке дерево компонентів, взаємодія між незалежними модулями організована за допомогою менеджеру глобального стану Redux. Зазначене рішення дозволяє компонентам навігації, галереї та 3D-редактору працювати в одному просторі: оновлення аватара в профілі миттєво відображається у блоках коментарів та верхньому меню. Синхронізація клієнта з бекендом реалізується з використанням шару асинхронних сервісних функцій. Для соціальних функцій, як лайки, стрічка новин – використовуються механізми підписки на зміни від бази даних Firestore. Усі операції запису, для прикладу збереження проєкту, або публікація коментаря проходять попередню валідацію на стороні клієнта за допомогою TypeScript-інтерфейсів, що дає цілісність даних перед їх відправленням до хмарного сховища.

Практичне застосування жорстко типізованої структури компонентів в системі дозволяє їй обробляти помилки на різних етапах життєвого циклу. Важкі графічні обчислення ізольовані від логіки інтерфейсу, де вони забезпечують стабільну частоту кадрів. У випадку введення користувачем некоректного синтаксису в редакторі GLSL-коду, архітектура визначатиме та запобігатиме обробці помилки в межах компонента графічного конвеєра, запобігаючи «падінню» всього вебзастосунку та втраті незбережених даних.

3.4 Проєктування бази даних

При реалізації плану збереження даних на розроблюваній платформі «Assimilate» обрано сервіс Cloud Firestore, що поставляється у вигляді документо-орієнтованої NoSQL бази даних. Окрема нереляційна модель даних обиралась із необхідності обробки ієрархічних структур, під виглядом проєктів з вкладеними коментарями та оцінками, без складних масивів таблиць – JOIN-запитів, із пришвидшенням відповіді сервера на сторону клієнта. Структура бази спроєктована з урахуванням, забезпечити максимальну швидкодію під час читання загальнодоступного контенту та повну ізоляцію приватних даних користувачів із нативними правилами безпеки.

Логічна архітектура бази даних складається з колекцій та документообігу вузлів системи:

1) колекція «users»: коренева колекція системи, що зберігає у собі профілі всіх зареєстрованих осіб. Кожен документ ідентифікується з унікальним ключем (англ. User Identifier, UID), що генерується підсистемою в момент авторизації користувача. У представленому документі зберігаються метадані профілю та опціонально, прапорець привілеїв розширеного доступу до адмін-панелі, він визначає належність користувача до групи адміністраторів;

2) вкладена колекція «projects»: проєкти не мають зберігатись в кореневій колекції, вони мають йти у підколекцію конкретного користувача з приблизним шляхом «/users/{userId}/projects/{projectId}». Зазначена структура проєкта спирається на прив’язку шейдерного коду саме до його автора. Кожен документ проєкту містить у собі текст шейдерів, параметри 3D-моделі, налаштування оточення, прапорець публічності, а також звіти взаємодії.

Для відображення загальної галереї робіт, що існує незалежно від автора використовується механізм CollectionGroup запитів, що збирає всі доступні проєкти з полем «isPublic == true» по всій ієрархії робіт бази даних;

3) вкладені колекції «comments» та «likes»: у переліку кожного документа проєкту існують свої підколекції для збереження коментарів та відміток взаємодії.

Коментар містить ідентифікатор автора, а текст посилання та мітку часу його створення. У профілі кожного користувача існує власна підколекція likes «/users/{userId}/likes/{projectId}», що надає системі інформацію, що з проєктів оцінив користувач системи, без необхідності повністю сканувати базу даних;

4) колекція «news»: також є кореневою колекцією, але для ведення стрічки новин платформи. Поля новин містять контент, дату публікації та ідентифікатор автора.

Окремим правилом проєктування бази даних є розробка систем безпеки даних – серверних правил безпеки «Firebase Security Rules v2», вони виконують роль серверного пошарового слою авторизації запитів перед їх обробкою на стороні клієнта. Оскільки в базі застосунки SPA не є довіреним середовищем, правила безпеки відсікатимуть спроби несанкціонованого доступу чи ескалації привілеїв через підміну API-запитів, роблячи перевірку вже в контрольованому середовищі виконання політик.

Проєктування політик безпеки спирається на вже влаштованих функціях-валідаторах:

– *isAuthenticated()*: перевіряє на наявність валідного JWT-токена у запиті бази даних;

– *isOwner(userId)*: зіставляє ідентифікатор із токена з ідентифікатором документа, перевіряє права на доступ на ресурс;

– *isAdmin()*: виконує перший запит до профілю користувача у базі даних для перевірки статусу адміністратора.

На основі приведених валідаторів зіставлено логіку захисту колекцій:

– захист від підвищення привілеїв: при створенні нового профілю у колекції «users» правило «request.resource.data.isAdmin == false» має забороняти користувачам призначати права адміністратора під час реєстрації. Зміна цього статусу можлива виключно діючим адміністратором;

– ізоляція приватних проєктів: документи з колекції «projects» дозволено читати будь-кому лише за умови, якщо поле проєкту «isPublic» та має у собі

значення «true». Інакше, доступ до нього (read, update, delete) має виключно його власник або системний адміністратор;

– гранульоване оновлення даних: для роботи системи вподобань додано додаткове правило. Будь-який авторизований користувач має змогу оновити частину документу чужого проєкту, але виключно поле «likesCount», ця перевірка йде за ключами таблиці: «*affectedKeys().hasOnly(['likesCount'])*». Будь-яка інша спроба зміни шейдер проєкта користувача, або змінити автора буде алгоритмічно відхилена базою даних як порушення безпекових правил;

– модерація контенту: будь-які операції видалення в межах колекцій «users», чужих проєктів чи новин мають йти у спеціально представленій для таких дій ролі «Адміністратор», що надає модераторам платформи повний контроль над дотриманням правил спільноти.

Зазначена спроєктована структура Cloud Firestore з використаними декларативними правилами безпеки гарантує цілісність даних, високу швидкість зміни стану контенту для 3D-редактора та повну відповідність системи принципам нульової довіри.

3.5 Проєктування інтерфейсу користувача

Заключним етапом програмної архітектури та моделювання платформи йде проєктування та стилізація графічного інтерфейсу користувача відповідно до розроблених дизайн-макетів. Застосунок розробляється як інструментарій інтегрованого середовища розробки з частиною користувацької взаємодії, де головним завданням стилізації постає забезпечення високого рівня ергономіки, якісної навігації та зниження когнітивного навантаження за тривалою роботою з кодом [22].

Візуальною властивістю платформи визначається темна тема оформлення. Зазначений вибір є базовим для професійного програмного забезпечення у сфері розробки та комп’ютерної графіки, оскільки темне тло здатне знизити напруження на зір користувача, зменшуючи випромінювання екрана, особливо з технологією

OLED та створює необхідний високий контраст для яскравих 3D-об’єктів на рівні сцени. Архітектурні стилі опираються на поєднанні компонентного підходу ізольованих таблиць CSS, SCSS Modules та готових представлених UI бібліотек. Використання модульних таблиць гарантуватиме відсутність конфліктів класів між безліччю різних модулях односторінкового застосунку під час стилізації різних елементів вікна графічного редактора. Для оптимізації класів розробки та дотримання єдиної стандартизованої дизайн-системи, в архітектуру інтерфейсу частково інтегровано окрему бібліотеку компонентів під назвою «React MUI», або «Material UI». Її застосування пришвидшує процеси створення, тісно інтегруючись з оточенням, перевірені на кросбраузерну сумісність та доступність елементи базового рівня. Комбінований підхід забезпечує баланс, якщо порівнювати зі швидкістю імплементації інтерфейсних блоків та окремого кастомного оформлення складного робочого середовища платформи.

Окрема увага опирається на принципах адаптивного вебдизайну. За допомогою медіа-запитів сітка макета здатна динамічно трансформуватися залежно від роздільної здатності екрана пристрою, що представлено на окремому мокапі (рис. 3.8).

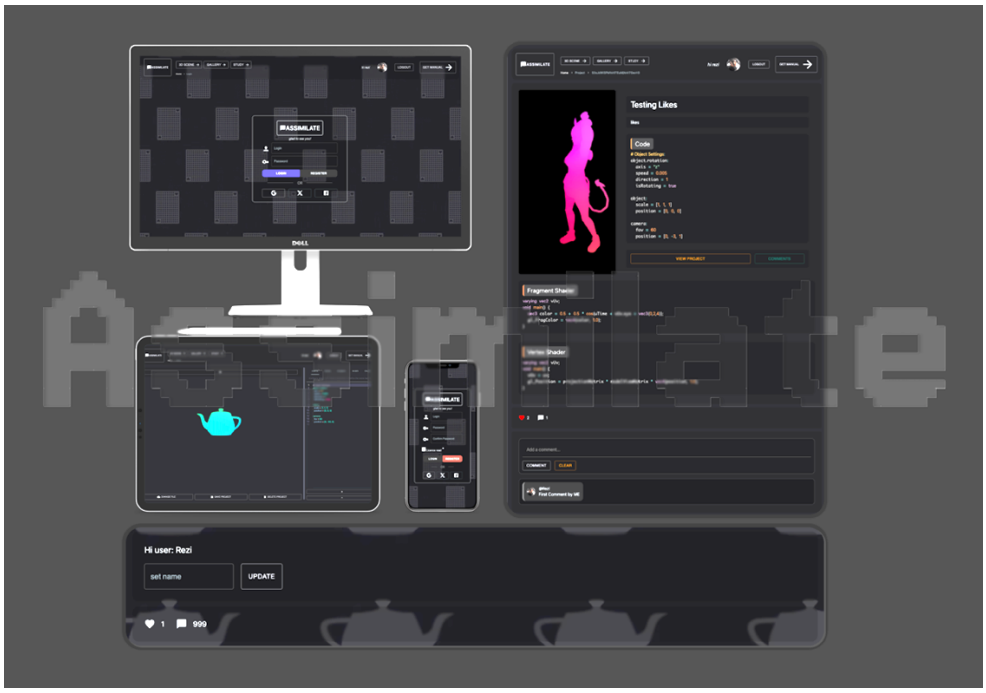

Рисунок 3.8 – Адаптивність користувацького інтерфейсу на різних пристроях

На настільних моніторах інтерфейс розгортається у робочий простір із паралельним розміщенням елементів. На планшетах і мобільних телефонах обрано до розміщення вертикального компонування, воно зберігає доступ без обмеження у функціоналі платформи.

Основний робочий простір, як сторінка редактора 3D-сцени, спроєктована за принципом поділу екрана. Ліву частину займає Canvas-контейнер із можливостями маштабування, бібліотеки React Three Fiber для відмальовування тривимірної графіки. Праву частину відведено під багатосторінкову панель керування з навігацією через вкладки:

1) Config – налаштування сцени через код;
2) Vertex – вертексні шейдери, їх налаштування через код;
3) Fragment – фрагментні шейдери, їх налаштування через код;
4) Images – додавання та менеджмент текстур;
5) Color – колір сцени (опціонально).

Основним елементом панелі керування є вбудований текстовий редактор коду (рис. 3.9).

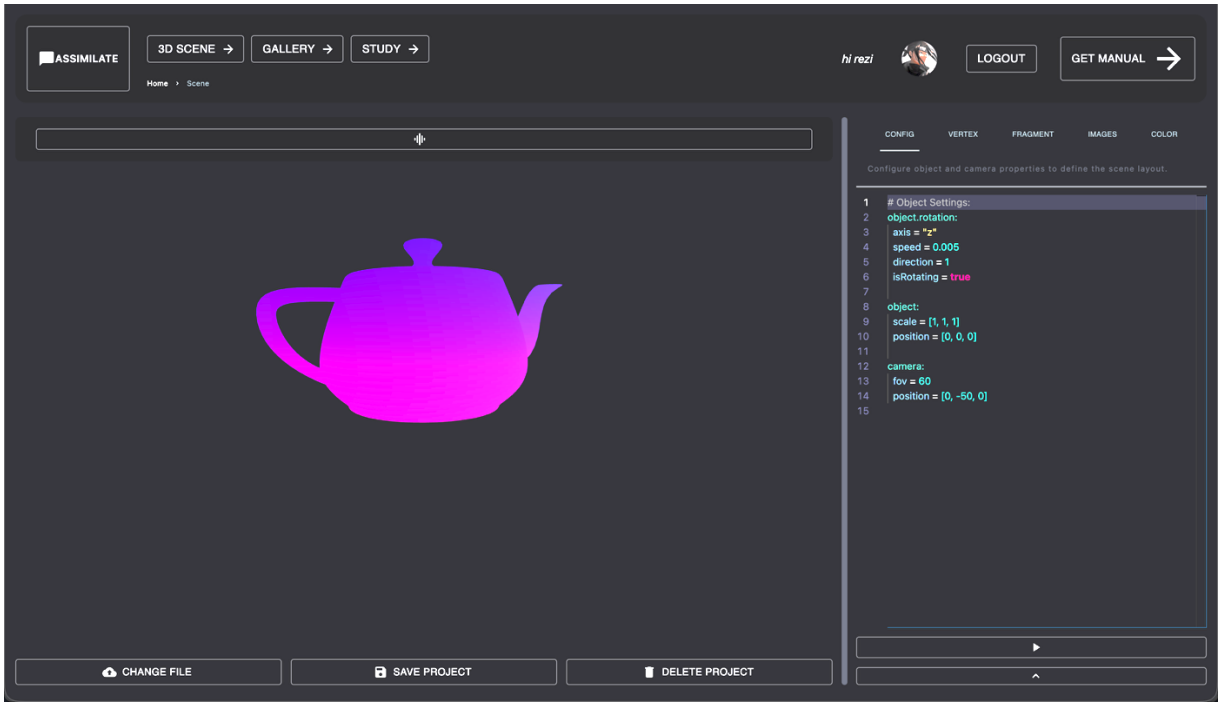

Рисунок 3.9 – Інтерфейс середовища розробки 3D-сцени та редактора коду

Для забезпечення комфортного та швидкого написання шейдерних програм відповідно до повноцінних редакторів – реалізовано окремий модуль механізму синтаксичного підсвічування. Алгоритм розпізнає специфічні ключові слова, типи даних, на прикладі: «vec3», «float», змінні та математичні функції мови GLSL, забарвлюючи їх у різні сталі кольори. Така робота значно полегшує візуальне сприйняття структури програми та допомагає розробникам швидше ідентифікувати синтаксичні помилки та структуру коду. Нижній блок редактора містить панель швидких дій для завантаження нових моделей, збереження або видалення поточного проєкту.

Центральним елементом для демонстрації результатів робіт спільноти виступає сторінка галереї (рис. 3.10).

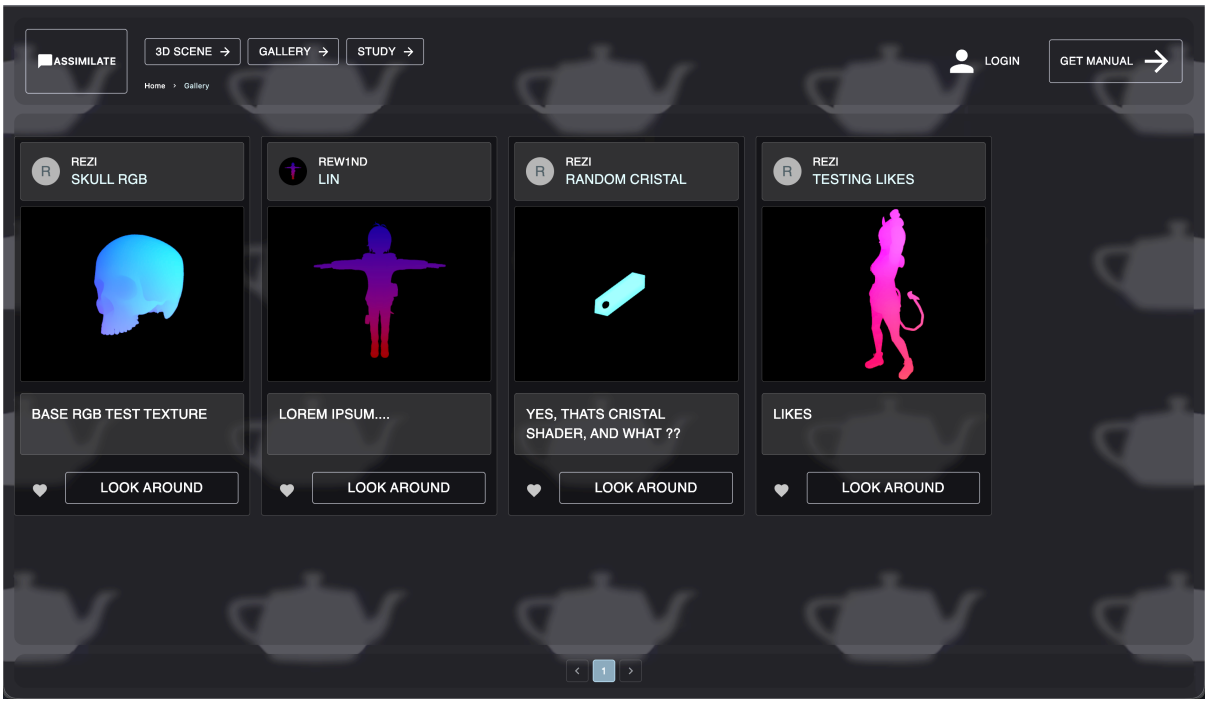

Рисунок 3.10 – Інтерфейс галереї проєктів

Інтерфейс секції реалізовано у вигляді Grid-сітки. Кожна індивідуальна картка проєкту користувачів спроєктована за модульним принципом, де розмежовано зони відповідальності: верхня частина містить у собі ідентифікаційні дані автора та назву роботи; центральний сегмент йде під статичний рендер прев'ю 3D-моделі. Інтерактивними елементами картки виступають іконка вподобання та

кнопка «LOOK AROUND» (Переглянути роботу), що ініціює перехід до детальної сторінки. Для забезпечення навігації між великими масивами даних у нижній частині сторінки інтегровано пагінацію, а паттерн із контурами чайників на фоні зберігає стилістичну цілісність усього ресурсу.

Сторінка детального перегляду проєкту (рис. 3.11) структурно поєднує графічні результати роботи з механізмами соціальної взаємодії.

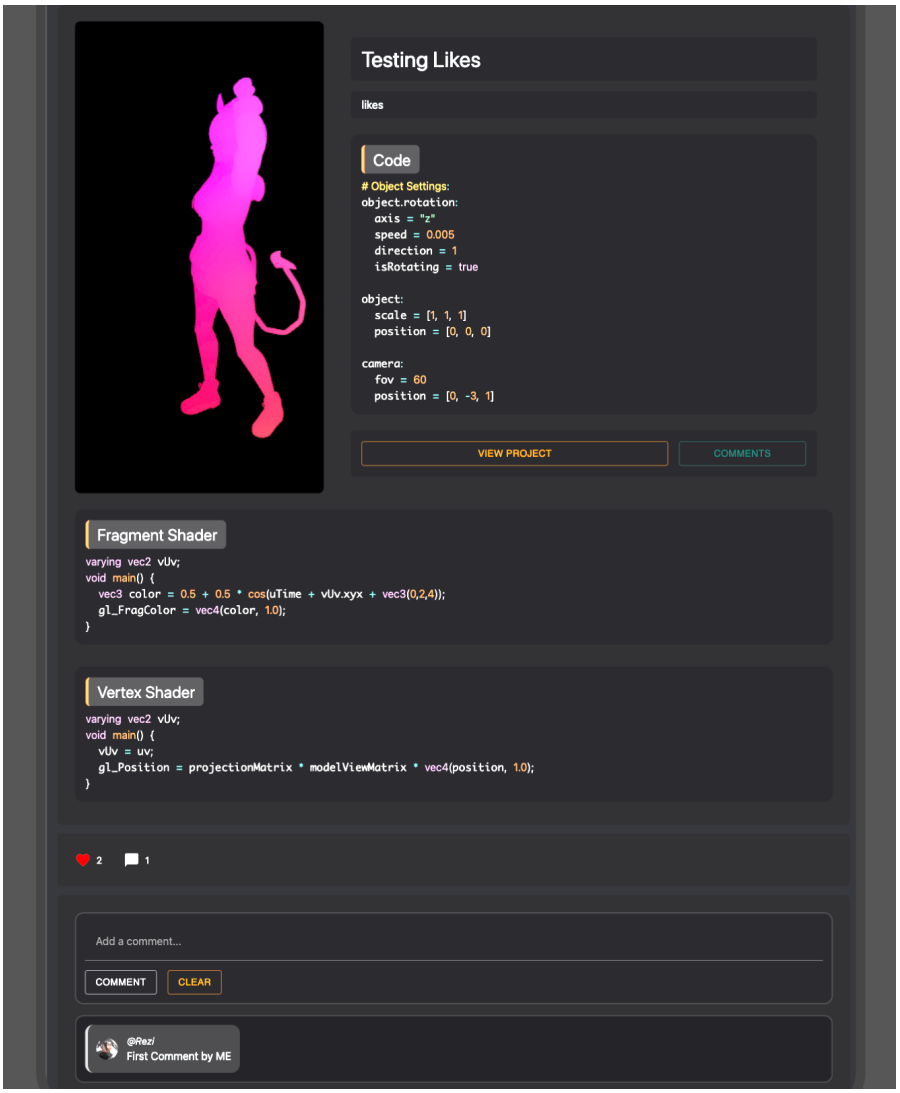

Рисунок 3.11 – Сторінка перегляду проєкту із соціальними функціями

Інтерфейс зазначеної сторінки містить велике прев'ю відрендереної 3D-моделі, під яким виставлено формат «тільки для читання» йде вивід блоків використаного конфігураційного та шейдерного коду цього проєкту. У нижній частині сторінки представлено модулі взаємодії проєкта: лічильники вподобань,

форму для введення нового коментаря та динамічний список залишених обговорень.

Висновки до розділу 3

У третьому розділі проведено моделювання логічної структури та інтерфейсу користувача платформи «Assimilate». Обґрунтовано вибір концепції односторінкового застосунку при використані трирівневої архітектури, що дозволятиме зберігати контекст WebGL-сцени під час навігації зі швидкістю системи. Для керування графічним рендерингом обрано модулі представлення Three.js та React Three Fiber, а для забезпечення серверної інфраструктури – безсерверну модель екосистеми Firebase.

За допомогою UML-діаграм розроблено модель варіантів використання, що зазначає сталі права доступу для трьох основних ролей: користувач, автор, адміністратор. Спроєктовано алгоритми маршрутизації, авторизації користувачів на платформу. Взаємодію між компонентами системи представлено за принципом односпрямованого потоку даних із використанням менеджера глобального стану Redux.

Спроєктовано структуру ієрархічної, документо-орієнтованої бази даних від Firebase – Cloud Firestore. Розроблено багаторівневі серверні правила безпеки, що перевіряють систему від несанкціонованого доступу, ізоляцію приватних даних та їх гранульоване оновлення, повністю реалізовано парадигму нульової довіри.

Окремо спроєктовано інтерфейсу користувача. Сформовано дизайн-систему, із темною темою оформлення, адаптивною сіткою та ергономічний поділ робочого простору в поєднанні з кастомним підсвічуванням синтаксису мови GLSL.

Оформлена архітектура, продумана логіка бази даних та візуальні макети повністю задовольняють вимоги специфікації та зазначає перехід до безпосередньої програмної реалізації вебзастосунку у наступному розділі.

4 РОЗРОБКА ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Четвертий розділ кваліфікаційної роботи обрано частині програмної реалізації вебплатформи «Assimilate». Головною метою поточного етапу є безпосередня розробка раніше спроєктованих алгоритмів у функціональний програмний продукт, на основі напрацювань попередніх розділів: візуального моделювання та визначення архітектури.

У розділі представлено процеси та модулі від створення клієнтської частини односторінкового застосунку на базі бібліотеки React із підтримкою в основі строго типізованої мови TypeScript. Основний акцент розробки полягає в проєктуванні функцій ядра платформи – методів інтеграції інструментів тривимірної графіки, з використанням Three.js та React Three Fiber, користувацьким інтерфейсом платформи. Послідовно описуються розроблені механізми імпорту 3D-геометрії, параметризації текстур, при імплементації комплексної, безпечної компіляції в віртуальному середовищі тестування та валідації користувацького шейдерного коду GLSL. Окремо можна зазначити, що усі приклади роботи 3D-геометрії протестовані, а основа роботи та тестів складає окремо створені користувацькі 3D-моделі, що використовуються для візуалізації та 3D-друку. Усі моделі створенні персонально та не є запозиченими об’єктами у сцені. Окрім візуальної складової, висвітлюється процеси програмної інтеграції модулів автентифікації та соціальних робіт із використанням глобального менеджера стану Redux.

4.1 Програмна реалізація ядра вебзастосунку

Функціональною особливістю платформи «Assimilate» є можливість застосування користувацького шейдерного коду до довільної користувацької 3D-геометрії, будь-якої складності, комплексної геометрії, за виключенням обмежень браузерного середовища. Основним обраним форматом для роботи з геометрією виступає файл стереолітографії (.stl), який є базовою структурою для тестування написаних користувачем шейдерів, оскільки файл зберігає виключно математичну

інформацію про поверхню об’єкта, роблячи його найлегшим із можливих для середовища (рис. 4.1).

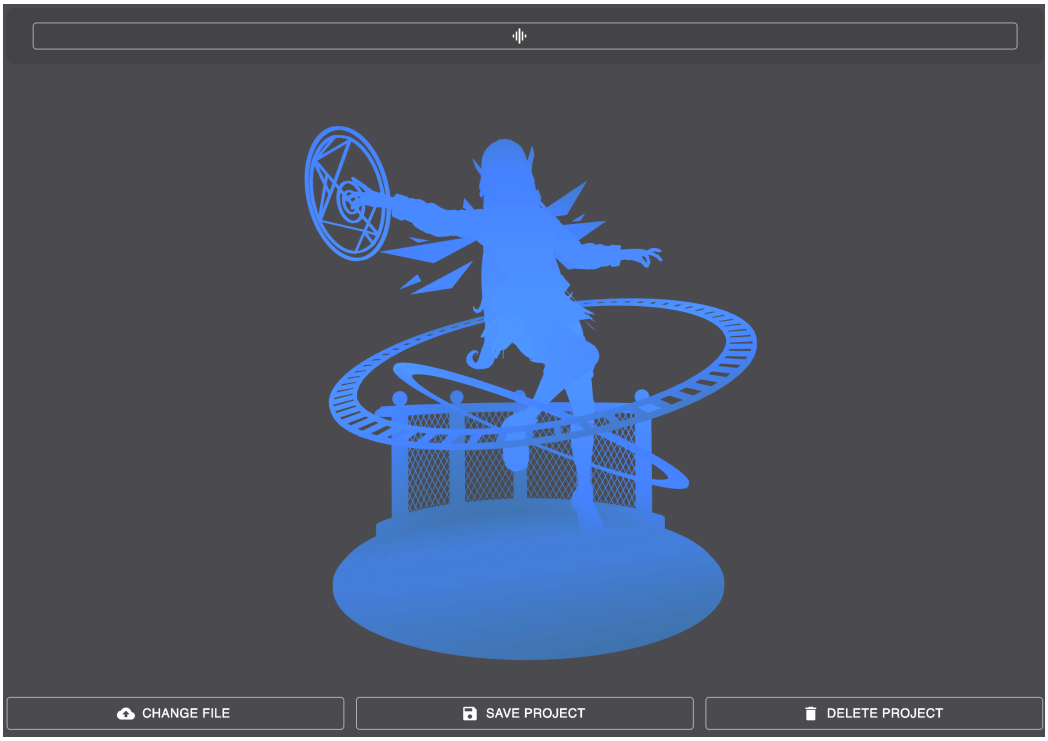

Рисунок 4.1 – Завантажена STL-модель у середовищі розробки

Обробка обраної 3D-моделі відбувається виключно на локальній стороні клієнта, а свідома відмова від завантаження моделей на сервер зумовлена перш за все безпекою користувачів. У той час, коли шейдерний код постає спільним набутком – 3D-моделі частіше за все залишаються персональними. Для дешифрування та парсингу файлу використовується модуль STLLoader з частини екосистеми Three.js. Оскільки формат .stl не містить інформації про текстурні координати, що є важливою складовою для роботи фрагментних шейдерів, у модулі імпорту застосовано програмно алгоритм автоматичної генерації UV-мапи (рис. 4.2). На основі комплексної Box-розгортки, та поєднань з алгоритмічною розгорткою. Програма обчислює обмежувальний паралелепіпед «Bounding Box» геометрії, обходить масив вершин та розраховує відповідні текстурні координати, записуючи їх у новий атрибут UV.

```
function generateUVs(geometry: THREE.BufferGeometry) {
  geometry.computeBoundingBox();
  const bbox = geometry.boundingBox!;
  const size = new THREE.Vector3();
  bbox.getSize(size);

  const uvAttr = new Float32Array(geometry.attributes.position.count * 2);

  for (let i = 0; i < geometry.attributes.position.count; i++) {
    const x = geometry.attributes.position.getX(i);
    const z = geometry.attributes.position.getZ(i);

    // Нормалізація координат відносно розмірів об’єкта
    uvAttr[i * 2] = (z - bbox.min.z) / size.z;
    uvAttr[i * 2 + 1] = (x - bbox.min.x) / size.x;
  }
  geometry.setAttribute('uv', new THREE.BufferAttribute(uvAttr, 2));
}
```

Рисунок 4.2 – Алгоритм генерації UV-координат для STL-моделі

Для компіляції комплексного візуального результату в систему додано модуль для обробки механізму параметризації текстур та їх оточення. Впроваджується підтримка завантаження HDRI-карт для різноманіття глобального освітлення IBL (рис. 4.3), в поєднанні із заповненням заднього фону сцени розробки. Також окремою складовою є окреме налаштування відтінку 3D-сцени, у випадках, коли HDRI-мапи не є доцільними, або потрібні лише, базові кольори.

Рисунок 4.3 – Застосування стилізованого освітлення до моделі та HDRI-мапи

Зв’язок між підготовленими текстурами, освітленням та шейдером реалізується через механізм глобальних обрахунку запечених змінних – uniforms. Завдяки декларативному підходу React Three Fiber, де зміна користувачем параметрів миттєво оновлює uniforms всередині ShaderMaterial. Процес написання

шейдерів безпосередньо у браузері пов’язаний із ризиком виникнення критичних помилок під час обробки GLSL-коду графічним процесором, що зазвичай може викликати ризики падіння сторінки з повною втратою контенту, чи зависань системи. Стандартна помилка в синтаксисі шейдера призводить до критичного збою у контексті WebGL рушія та повної зупинки рендерингу.

Для запобігання можливим помилкам в системі «Assimilate» реалізовано додатковий алгоритм безпечної компіляції (рис. 4.4). В першу чергу він спирається не на стандартну компіляцію вебрушія, що також може знаходити помилки контексту, а прописаний виключно, як додатковий шар логіки, що якісно перехоплює помилки, ще до етапу браузерної компіляції коду. На другому, внутрішньому етапі перехоплення помилок постає майже неможливим. Сам по собі рушій не передає контекст помилки та її причини. Логіка перехоплює подію зміни тексту та ініціює створення програми у віртуальному (ізольованому) WebGL-контексті браузера, що сам по собі обертає під себе компіляцію коду, обробляє результат, та при безпечному коді – передає його далі. Код валідації та застосування шейдерів на моделі наведено в додатку Б.

```
      function validateShaderInWebGL(vertex: string, fragment: string) {
        const canvas = document.createElement('canvas');
        const gl = canvas.getContext('webgl') || canvas.getContext('experimental-
webgl');
        if (!gl) return { valid: false, type: 'webgl', log: 'WebGL not available'
};

        const program = gl.createProgram();
        const vert = gl.createShader(gl.VERTEX_SHADER);
        gl.shaderSource(vert, patchedVertex);
        gl.compileShader(vert);

        if (!gl.getShaderParameter(vert, gl.COMPILE_STATUS)) {
          const log = gl.getShaderInfoLog(vert) || 'Unknown vertex shader error';
          cleanup(gl, program, vert, null);
          return { valid: false, type: 'vertex', log };
        }
        // Аналогічна перевірка виконується для фрагментного шейдера та лінкування
        return { valid: true };
      }
```

Рисунок 4.4 – Механізм ізольованої валідації GLSL-коду

У разі виявлення помилок, у прикладі з невідповідністю типів даних, алгоритм парсить системний лог за допомогою Regex операторів та виводить структуроване повідомлення з точною вказівкою на рядок помилки у користувача підставляючи у панель повідомлень (рис. 4.5). Основний рендеринг при цьому продовжує працювати на останній валідній версії коду.

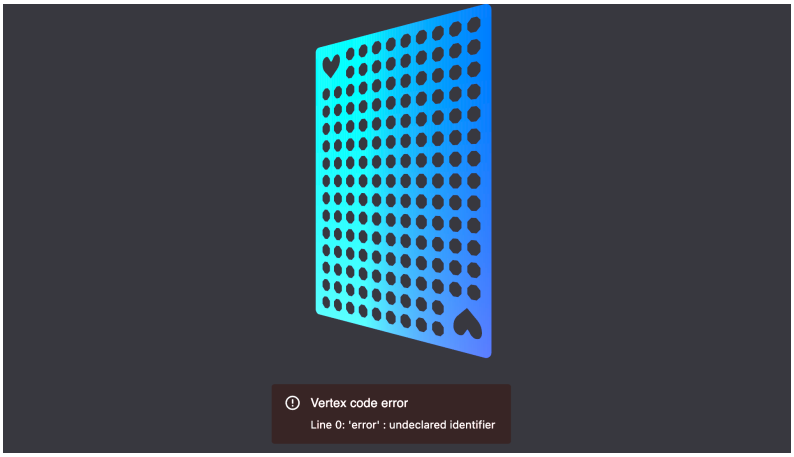

Рисунок 4.5 – Перехоплення та вивід синтаксичної помилки у редакторі

Для забезпечення комфортного написання користувачем коду на платформі реалізовано обрано та реалізовано кастомний модуль синтаксичного підсвічування на базі редактора Monaco. Розробка конфігуратора токенів мови GLSL перевіряє валідність синтаксису до окремо зазначених змінних, сталих слів, директив та чисел у різному представленні (рис. 4.6).

```
export const language: languages.IMonarchLanguage = {
  defaultToken: 'invalid',
  keywords: ['uniform', 'varying', 'attribute', 'void', 'float', 'vec2',
'vec3', 'vec4'],
  tokenizer: {
    root: [
      [/[a-zA-Z_]\w*/, {
        cases: {
          '@keywords': 'keyword',
          '@default': 'identifier'
        }
      }],
      [/^\s*#\s*\w+/, 'keyword.directive'], // Директиви препроцесора
      [/\d*\.\d+([eE][\-+]?\d+)?/, 'number.float'], // Числа з рухомою комою
    ]
  }
}
```

Рисунок 4.6 – Фрагмент конфігурації токенізатора мови GLSL

Він розпізнає специфічні типи даних (vec3, mat4), вбудовані математичні функції та директиви препроцесора. Окремо представлена робота зі строковими даними (текстом типу String). Зазначені вище конфігурації редактора значно знижують когнітивне навантаження на розробника під час немалих сесій розробки та тестування (рис. 4.7). Представлена робота повністю опирається на вже існуючі редактори коду, де представлені підсвічення коду, підказки та автодоповнення – повністю повторюють за ними базові принципи.

Для візуальної ідентифікації проєктів у галереї передбачено механізм генерації прев'ю за допомогою методу *usePreviewRenderer()*. Алгоритм створює окремий графічний контекст поза основним деревом документа (Offscreen Canvas), встановлюючи цільове, обране користувачем співвідношення сторін для камери та генерує статичне зображення у форматі Base64.

```
precision mediump float;

varying vec2 vUv;
varying vec3 vNormal;
varying vec3 vViewPosition;

void main() {
  vec3 baseColor = vec3(0.1, 0.4, 0.6); // Color

  vec3 lightDirection = normalize(vec3(0.1, .5, 0.7));

  vec3 normal = normalize(vNormal*4.0);
  float diffuse = max(dot(normal, lightDirection), 0.1);

  float ambient = 0.3;
  vec3 finalColor = baseColor * (diffuse + ambient);

  gl_FragColor = vec4(finalColor, 1.0);
}

// Switching to stylized
```

Рисунок 4.7 – Синтаксичне підсвічування коду GLSL

Крім візуальної та обчислювальної складової, інтегровано хмарні сервіси Firebase Authentication та Cloud Firestore. Управління поточним станом авторизації тісно синхронізовано з архітектурою глобального менеджера стану Redux. Оновлення соціальних взаємодій із вподобаннями та коментарями до робіт виконуються через атомарні транзакції сервера. Візуальні компоненти підписані на

зміни у Redux Store, завдяки чому при зміні стану бази даних в алгоритмах Virtual DOM точково перемальовуються лише змінені сегменти інтерфейсу, що забезпечує швидкий відгук платформи без повного перезавантаження сторінки і збереження контексту роботи користувача.

4.2 Тестування програмного забезпечення

Етап тестування розробленого програмного забезпечення є одним із етапів життєвого циклу розробки програмного забезпечення. Відповідаючи стандартам IEEE 829-2008 [10] та ISO/IEC/IEEE 29119-1:2022 [12], він гарантує відповідність створеного програмного продукту висунутим в попередніх розділах функціональним та нефункціональним вимогам відповідно, а також є частиною формалізованої перевірки якості. Для повної відповідності вебплатформи «Assimilate» розроблено методологію, що включає в собі перевірку модулів компіляції за допомогою підготовлених частин коду, основне навантажувальне тестування графічного редактора та тестування політик безпеки користувацької бази даних шляхом симуляції прямих несанкціонованих запитів до хмарного середовища.

Основною частиною етапу функціонального тестування, що проводилося відповідно до принципів Unit Testing та Integration Testing, є перевірка роботи кастомного модуля ізольованої компіляції GLSL-коду в браузері користувача. Оскільки навіть базові синтаксичні помилки на рівні графічного процесора можуть спричинити критичний збій WebGL-контексту та зависання вкладки браузера, механізм перехоплення винятків вирішено перевіряти стандартними методами тестування за допомогою двох підготовлених шаблонів користувацького шейдерного коду.

Для перевірки успішного сценарію виконання використано валідний фрагментний шейдер із розрахунком базового дифузного освітлення (рис 4.8).

```
precision mediump float;
varying vec2 vUv;
varying vec3 vNormal;
varying vec3 vViewPosition;
void main() {
  vec3 baseColor = vec3(0.1, 0.4, 0.6); // Базовий колір
  vec3 lightDirection = normalize(vec3(0.1, 0.5, 0.7));
  vec3 normal = normalize(vNormal * 4.0);
  float diffuse = max(dot(normal, lightDirection), 0.1);
  float ambient = 0.3;
  vec3 finalColor = baseColor * (diffuse + ambient);
  gl_FragColor = vec4(finalColor, 1.0);
}
```

Рисунок 4.8 – Валідний код фрагментного шейдера

При введенні шаблону шейдера до редактора користувацької системи, в ізольованому тестовому середовищі веббраузера компілятор успішно відтворив програму та миттєво оновив полігональну сітку на екрані користувача.

Для перевірки механізму перехоплення помилок у цей самий код внесено навмисні синтаксичні порушення: використано неправильне визначення типу точності. Замість базового типу «float» використано хибний «boolean» (рис. 4.9).

```
precision mediump boolean;
varying vec2 vUv;
varying vec3 vNormal;
varying vec3 vViewPosition;
void main() {
  vec3 baseColor = vec3(0.1, 0.4, 0.6); // Базовий колір
  vec3 lightDirection = normalize(vec3(0.1, 0.5, 0.7));
  vec3 normal = normalize(vNormal * 4.0);
  float diffuse = max(dot(normal, lightDirection), 0.1);
  float ambient = 0.3;
  vec3 finalColor = baseColor * (diffuse + ambient);
  gl_FragColor = vec4(finalColor, 1.0);
}
```

Рисунок 4.9 – Код із навмисними синтаксичними помилками

У якості результату – система відтворила помилку компіляції на відповідному першому рядку редактора, вивівши повідомлення з помилкою на екран користувача (рис. 4.10).

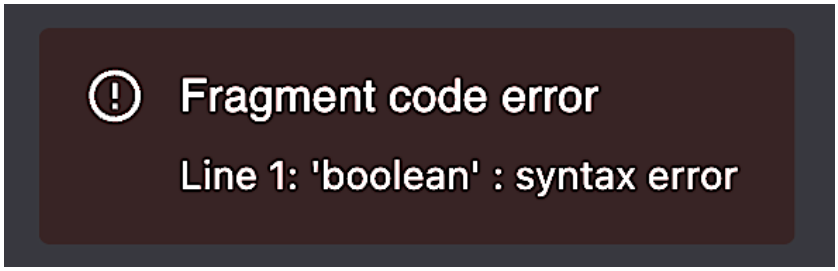

Рисунок 4.10 – Синтаксична помилка в рядку 1

Результати функціонального тестування (табл. 4.1) занотовані як успішні: графічний конвеєр не зазнав аварійного завершення. Розроблений модуль валідації перехопив системний лог від WebGL інтерпретатора, зупинив рендеринг пошкодженого коду матеріалу та вивів відформатоване повідомлення про синтаксичну помилку, вказівку на конкретні рядки безпосередньо в інтерфейс редактора та її вірогідне місце в рядку.

Таблиця 4.1 – Результати тестування модуля валідації шейдерів

Тестовий пресет	**Опис**	**Очікуваний результат**	**Статус**
Пресет 1	Використання шейдера з навмисними синтаксичними помилками	Зупинка рендерингу матеріалу. Вивід повідомлення про помилку у редактор	Успішно
Пресет 2	Введення валідного GLSL-коду після виникнення помилки	Успішна компіляція. Зникнення помилки, візуальне оновлення моделі	Успішно

Окрім задач перевірки бізнес-логіки, для коректної оцінки продуктивності графічного конвеєра з виявленням можливих вузьких місць в компіляторі проведено стрес-тестування, відповідно до методології Performance Testing від

ISO/IEC 25010:2023 [11]. У даних тестування враховувалися зазначені якісні метрики продуктивності: середня частота кадрів, використання GPU/CPU, затримка рендерингу та споживання пам’яті. Інструментарієм для профілювання виступив модуль відстеження метрик продуктивності вбудований в React-Drei під назвою «Stats». Враховуючи апаратне обмеження вертикальної синхронізації браузера V-Sync, максимальна частота оновлення екрана для тесту заблокована на рівні 60 кадрів на секунду. Профілювання проводилося з використанням 3D-моделей різної полігональної щільності у трьох розмірах, від середнього навантаження в 800 тис., до тестування граничних обмежень у 36 млн. трикутників (табл. 4.2).

Таблиця 4.2 – Заміри продуктивності рендерингу (Stress Test)

Сценарій тестування	**Об’єм геометрії, трикутників**	**Середній показник FPS**	**Результат навантаження**
Високе навантаження	~ 800 000	60 (V-Sync Limit)	Абсолютно стабільний рендеринг без втрат кадрів
Екстремальне навантаження	~ 11 000 000	60 (V-Sync Limit)	Стабільний рендеринг, ефективне використання BufferGeometry
Стрес-тест (Overload)	~ 36 000 000	20–25	Фізична деградація FPS через заповнення буфера GPU

1) Використання моделі найменшого з зазначених розмірів

Під час першого етапу тестування на сцену завантажено базову 3D-модель середньої тяжкості (рис. 4.11). Розмір файлу моделі складає 40 Мбайт.

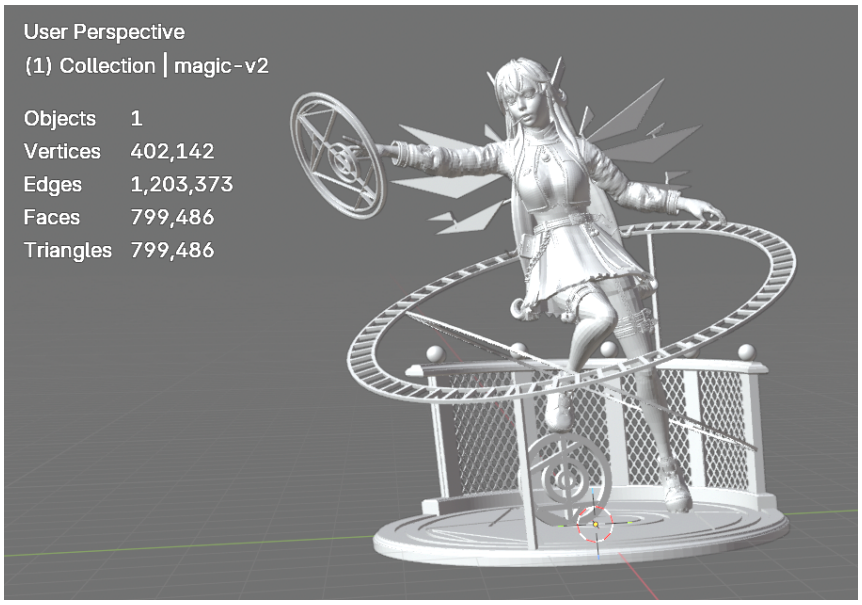

Рисунок 4.11 – Модель на 800 тис. полігонів

В типовому сценарії використання, обрану модель можна одразу прирівняти до завершеного проєкту через її достатню деталізацію. Як зазначено в інтерфейсі графічного редактора на рисунку, системний лічильник фіксує точну кількість, 799 486 трикутників, далі це значення будемо називати приблизно 800 тис.

Відповідний результат профілювання навантаження відображено в інтерфейсі користувача (рис. 4.12) із ввімкненим профілюванням редактора.

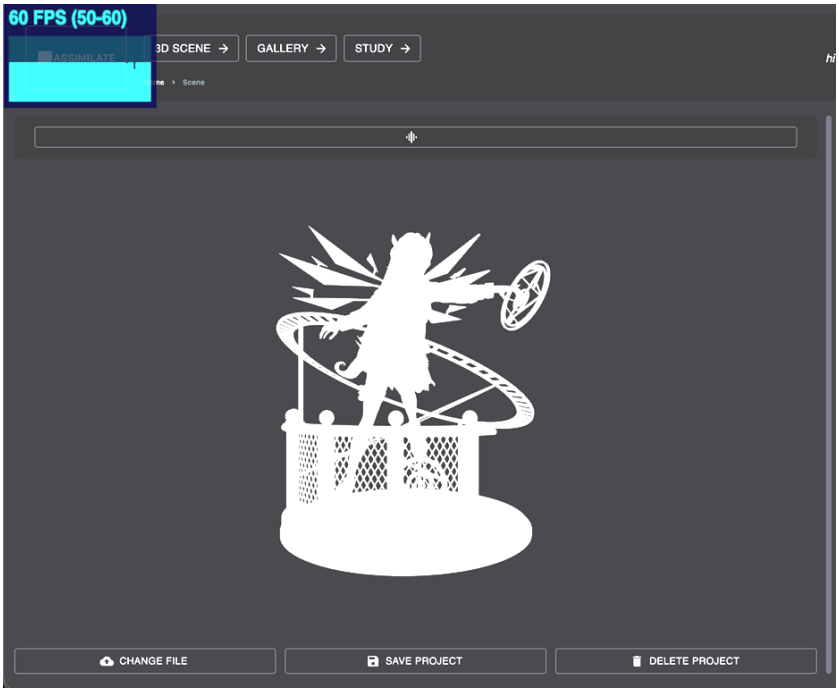

Рисунок 4.12 – Результат профілювання базової моделі

Обчислювальні потужності пристрою відмальовують геометрію без затримок, профілювання відображає стабільні 60 FPS із невеликою просадкою в момент завантаження 3D-моделі у вікно редактора.

2) Використання моделі середнього з зазначених розмірів

Наступним кроком є вкрай високе навантаження з використанням високополігональної моделі, що містить понад 11 мільйонів трикутників (рис. 4.13). Зазвичай моделі такого розміру використовуються лише для рендерів, або 3D-друку.

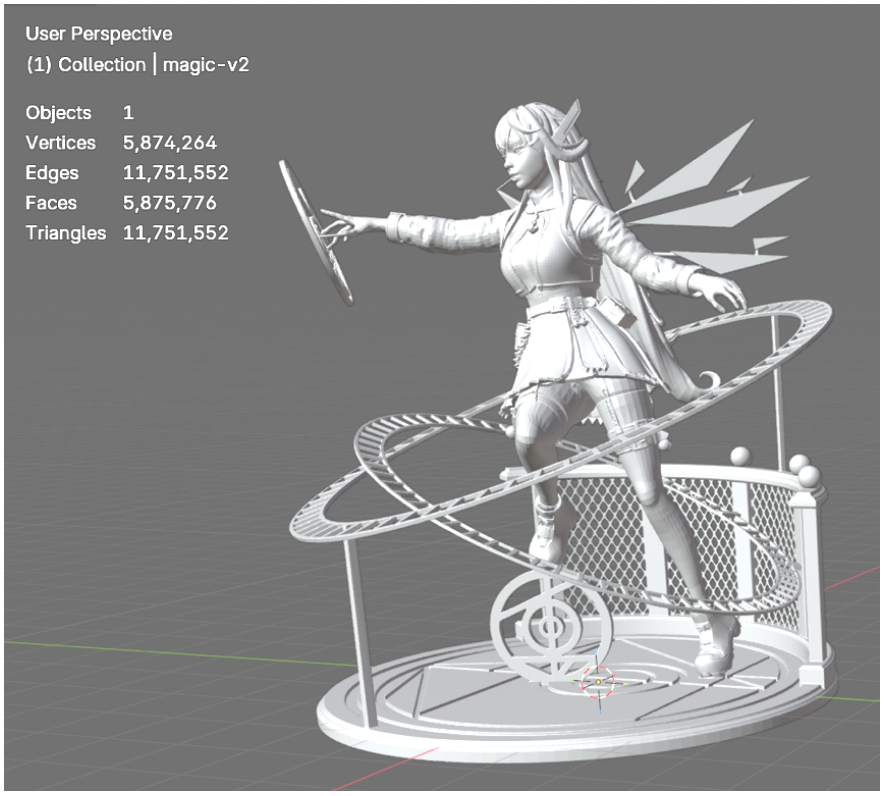

Рисунок 4.13 – Модель на 11 млн. полігонів

Незважаючи на приведений у тесті великий обсяг геометричних даних, показники продуктивності (рис. 4.14) свідчать, що частота кадрів залишилася заблокованою на максимальній позначці 60 FPS. Втрата продуктивності до 1 кадра однократно свідчить лише про великий обсяг моделі та час завантаження її до середовища.

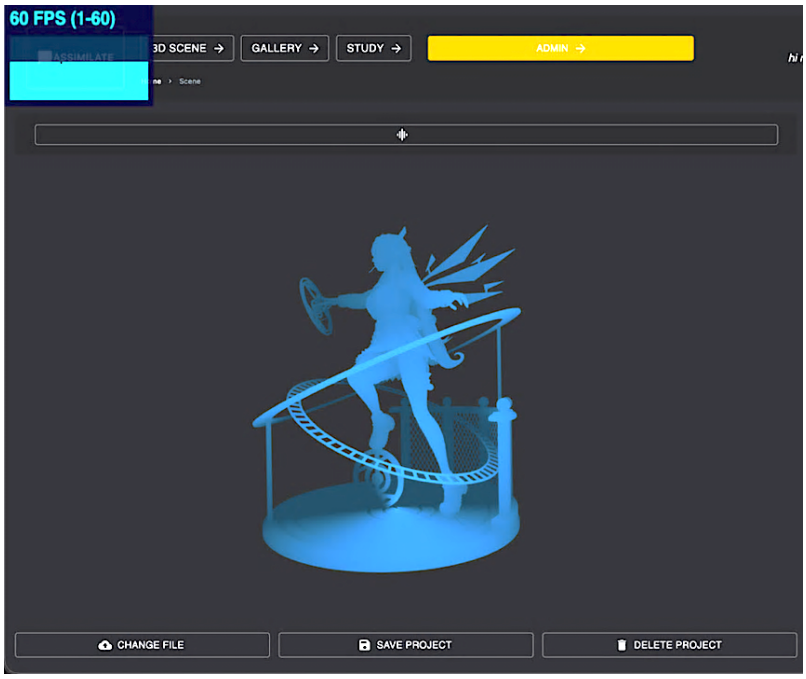

Рисунок 4.14 – Результат профілювання об’ємної моделі

Отримано підтвердження оптимізації платформи при обробці структури BufferGeometry за час компіляції та рендерингу 3D-моделі на полотні.

3) Використання моделі граничних розмірів зі зазначених варіантів

Заключний стрес-тест передбачає рендеринг геометрії обсягом близько 36 млн. полігонів (рис. 4.15), що є нетиповим прикладом у браузерній візуалізації. Тест проводиться для визначення лімітів платформи методом перевантаження.

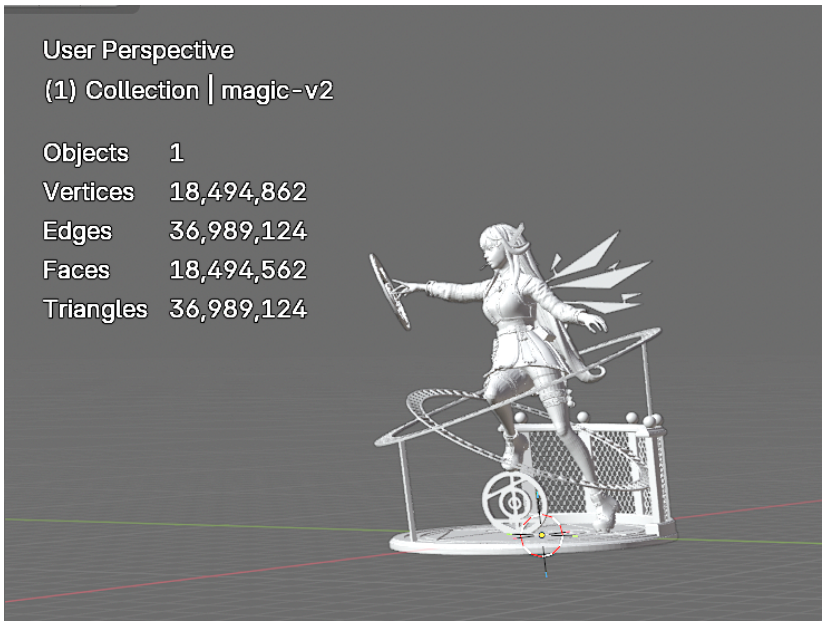

Рисунок 4.15 – Модель на 36 млн. полігонів

Лише за екстремальних умов, що є нетиповими для браузерного середовища користувача, спостерігається фізична деградація продуктивності. Частота кадрів знизилася до 20–25 FPS через апаратне заповнення буфера відеокарти (рис. 4.16).

Отримані в результатах тестування архітектурної платформи, зазначено стійкість до перевантаження браузерного середовища надважкими 3D-моделями.

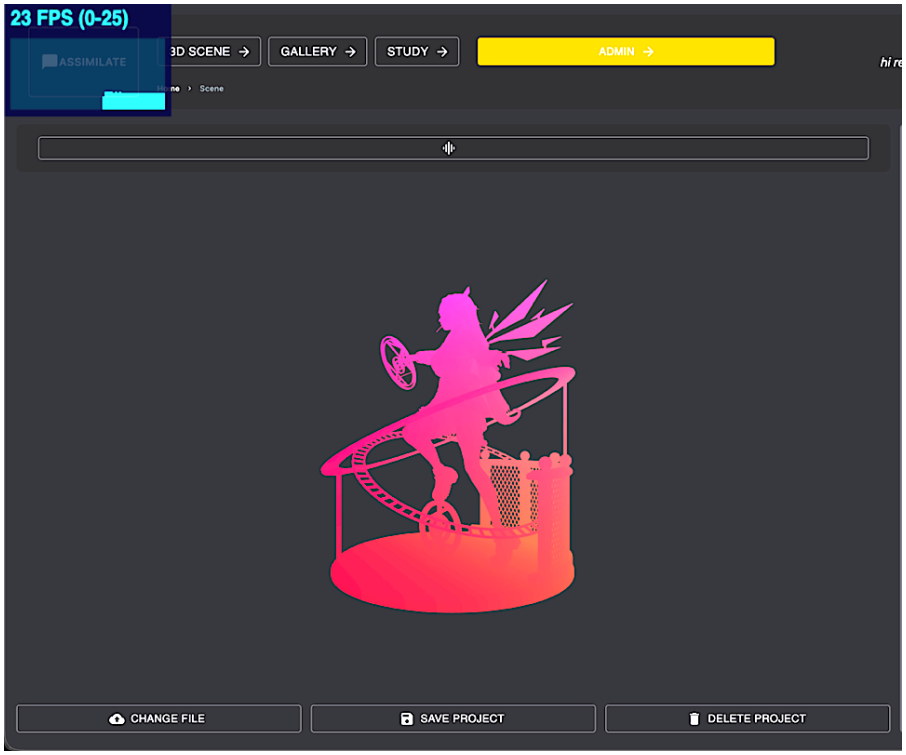

Рисунок 4.16 – Результат профілювання екстремальної моделі

Застосунок утримує стабільні 60 FPS навіть при важкому рендерингу від 11 млн. полігонів, що зазвичай значно перевищує потреби стандартних веборієнтованих 3D-сцен. Стандартні моделі для користувацьких тестів відповідають розмірам від 5 тис. до 100 тис. трикутників. Деградація кадрів спостерігалася лише під час екстремального стрес-тесту, хоча за таких умов WebGL-контекст не зазнав аварійного завершення роботи середовища, вкладки браузера чи критичних витоків оперативної пам’яті.

Останньою частиною тестування йде валідація архітектури нульової довіри за принципами Zero Trust архітектури, що відповідає рекомендаціям NIST SP 800-207 [19], та перевірка серверних правил Firebase Security Rules.

Тестування здійснювалося шляхом симуляції дій зловмисника безпосередньо з відтворенням у клієнтській частині застосунка, що відповідає принципам Security Testing. Для зазначеного тестування написано скрипт, що імітує спробу видалення чужого документа проєкту авторизованим користувачем, який не є його власником, або адміністратором (рис. 4.17).

```
const simulateAttack = async () => {
  try {
    const projectRef = doc(db, "projects", "project_1771764255022");
    await deleteDoc(projectRef);
    console.log("Error: Doc Deleted!");
  } catch (error: any) {
    console.error("Pass:", error.message);
  }
};
```

Рисунок 4.17 – Скрипт імітації несанкціонованого видалення документа

При виконанні скрипта, транзакція алгоритмічно перехоплена та відхилена на стороні сервера Cloud Firestore (рис. 4.18).

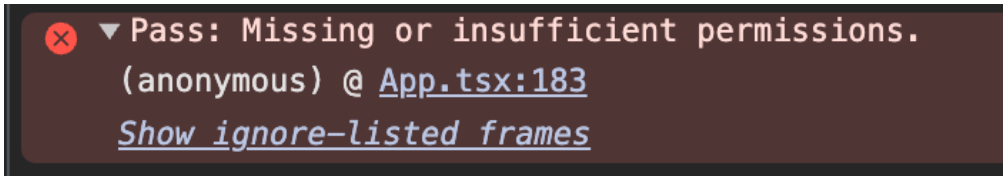

Рисунок 4.18 – Відхилений запит зі сторони сервера

У консолі розробника зафіксовано очікувану системну помилку, що повідомляє клієнту про коректну роботу декларативних правил безпеки «Pass: Missing or insufficient permissions» (табл. 4.3).

Усі приведені тести успішно пройдені та відповідають системним функціональним та нефункціональним вимогам, що визначені у попередніх розділах роботи. Результати підтверджують тести функціоналу алгоритмів валідації та стійкість хмарної інфраструктури до несанкціонованого доступу.

Таблиця 4.3 – Перевірка правил безпеки Cloud Firestore

Вектор перевірки	**Дія з клієнтської частини**	**Реакція системи безпеки**
Ізоляція приватних проєктів	Спроба зчитати документ проєкту, який має маркер isPublic: false	Запит відхилено. Отримано помилку «Missing or insufficient permissions»
Захист від модифікації	Спроба змінити текст коду в чужому публічному проєкті	Запит відхилено. Порушення правила *isOwner(userId)*
Гранульоване оновлення	Зміна поля likesCount у чужому проєкті через метод *arrayUnion()*	Транзакція підтверджена сервером. Оновлено атомарно

Розроблена вебплатформа «Assimilate» відповідає сучасним стандартам якості програмного забезпечення ISO/IEC 25010:2023 [11] – System and Software Quality Requirements та може розглядатися як надійне середовище для розробки й тестування шейдерного коду.

4.3 Керівництво користувача

Для використання застосунку новим користувачам на інтуїтивно зрозумілій взаємодії з платформою «Assimilate» розроблено керівництво користувача. Воно описує базові сценарії роботи: від першої авторизації на платформі до публікації готового шейдерного проєкту в загальній галереї та взаємодії з іншими проєктами користувачів.

1) Реєстрація та авторизація в системі

При першому відвідуванні користувач отримує так званий статус «Гостя» платформи, який дозволяє переглядати публічну галерею, проте обмежує можливості зі збереження проєктів та соціальної взаємодії з проєктами інших користувачів. Для отримання повного доступу, спочатку, необхідно пройти процес

авторизації. Натиснувши на кнопку профілю із надписом «Login» у навігаційному меню, користувач відкриває вікно входу (рис. 4.19).

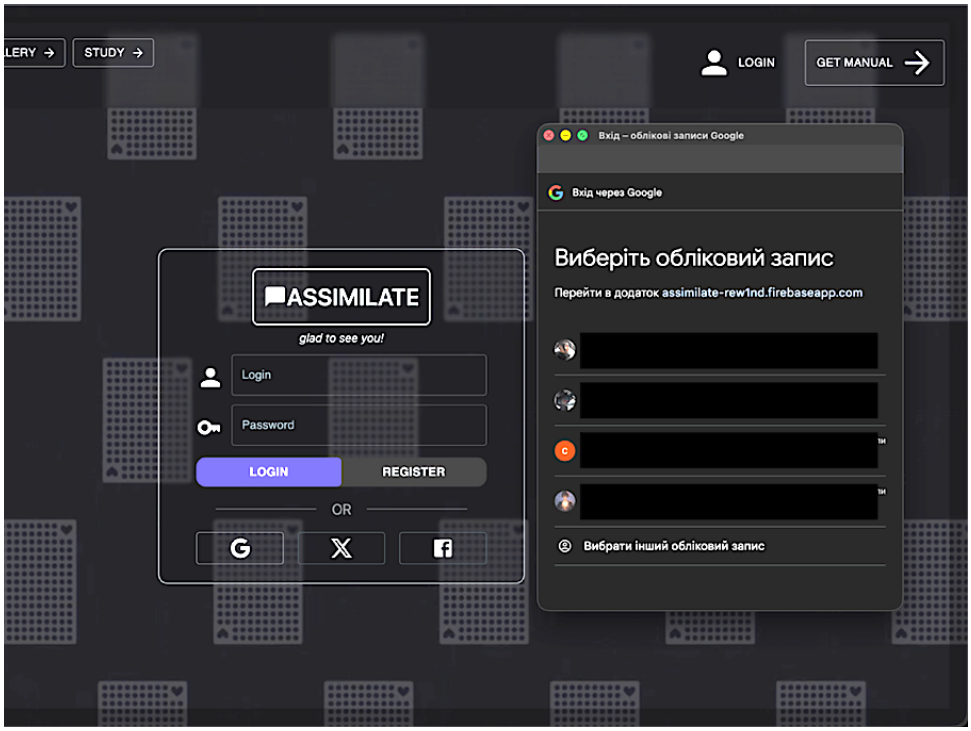

Рисунок 4.19 – Вікно авторизації користувача

Система пропонує швидку та безпечну авторизацію за протоколом OAuth через обліковий запис Google, що не вимагає створення нових паролів, або базова реєстрація при відсутності акаунту та небажанні використовувати прив’язку до Google-акаунт.

2) Створення проєкту та імпорт 3D-моделі

Після успішної авторизації користувача переводить на головну сторінку застосунку, звідти він може переходити до робочого простору графічного редактора, або за допомогою кнопки на панелі навігації. Для початку роботи необхідно завантажити полігональну сітку моделі. На панелі керування (вкладка Config) розташовано інтерфейс імпорту файлів (рис. 4.20). Користувач може обрати локальний файл формату .stl, .gltf або .glb. Завдяки обробці зображення на стороні клієнта, модель миттєво з’являється на інтерактивному Canvas-полотні без мережевих затримок.

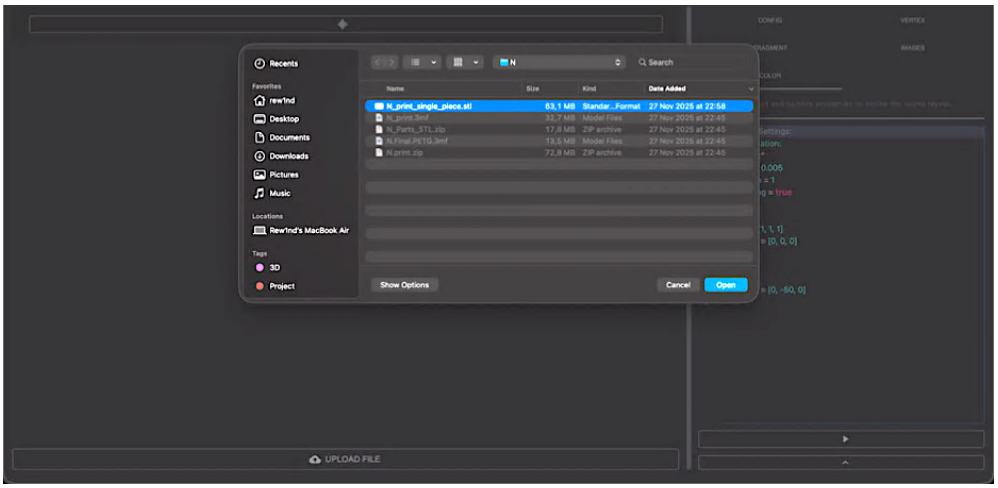

Рисунок 4.20 – Інтерфейс завантаження користувацької 3D-моделі

3) Налаштування сцени та написання шейдерного коду

Основний процес розробки відбувається за допомогою багатосторінкової бічної панелі. Вкладки на ній розділяють робочий процес на логічні етапи:

– Images / Color: дозволяють обрати колір фону або завантажити HDRI-карту для фонового зображення, або освітлення та відблисків на поверхні об’єкта;

– Vertex / Fragment: відкриває вбудований текстовий редактор коду вершинних, або фрагментних шейдерів (рис. 4.21).

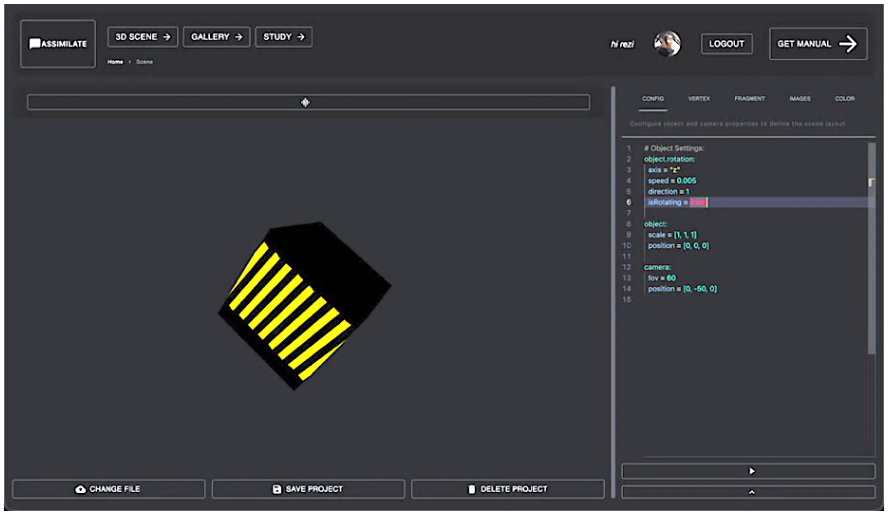

Рисунок 4.21 – Робочий простір

Користувач має змогу писати програми мовою GLSL у режимі реального часу, змінювати параметри сцени в вікні «Config», або використовувати панель швидких дій для сцени, щоб пришвидшити налаштування сцени. У разі допущення

синтаксичної помилки, при компіляції система автоматично виведе попередження із вказівкою на проблемний рядок у нижній частині панелі.

4) Збереження та соціальна взаємодія

Після завершення роботи над шейдером користувач має змогу зберегти проєкт, потрібно натиснути відповідну кнопку в нижній панелі дій. Під час збереження система автоматично згенерує статичне прев'ю поточної сцени проєкта та завантажить конфігурацію до бази даних (рис. 4.22).

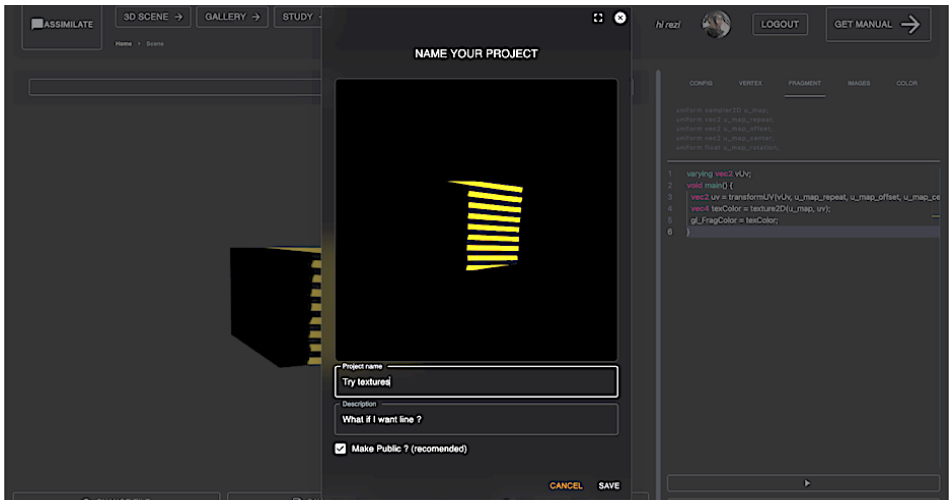

Рисунок 4.22 – Збереження проєкту

Опублікований проєкт з'являється у загальній галереї (рис. 4.23). Вона представлена у вигляді сітки проєктів від авторизованих користувачів платформи, що опублікували свої роботи, виставивши тип доступу «для всіх».

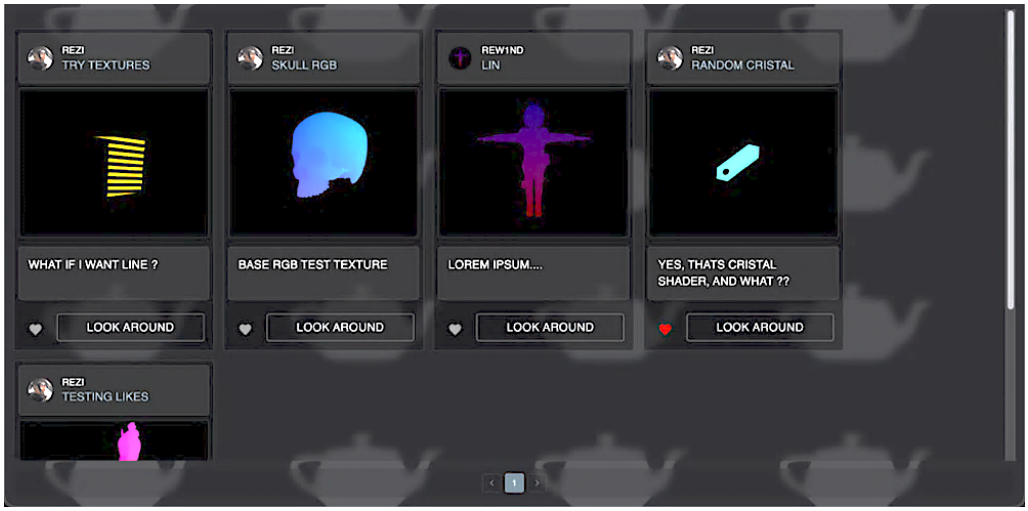

Рисунок 4.23 – Галерея проєктів

Будь-який відвідувач платформи може відкрити картку проєкту для детального перегляду відрендереної моделі сцени, ознайомитись з вихідним кодом шейдера, а також, якщо він авторизований – додати проєкт до вподобань та перейти до участі в обговоренні через систему коментарів (рис. 4.24).

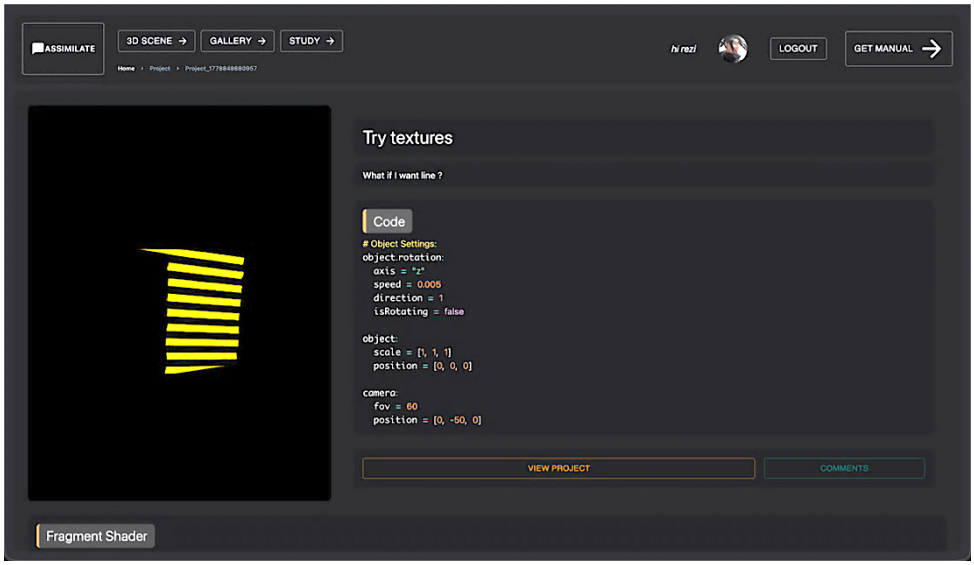

Рисунок 4.24 – Сторінка проєкту

Керівництво користувача представлене інтуїтивно зрозумілим та структурованим життєвим циклом роботи з вебплатформою «Assimilate». За залученням розділеного робочого простору модулі проєктів, налаштування сцени, розробки коду та публікації йде мінімізація порігу входження для нових користувачів проєкту. Комплексна інтеграція інструментів 3D-редактора коду сприяє індивідуальному створенню графічного контенту з активним обміном досвідом та колаборації спільноти розробників всередині платформи.

4.4 Апробація результатів розробки

Основні наукові та практичні результати кваліфікаційної роботи, архітектурні рішення, а також підходи до оптимізації рендерингу 3D-графіки у браузерному середовищі представлені та обговорені ще на науково-практичній конференції «Могилянські читання» (м. Миколаїв, 2025 рік). За результатами роботи конференції опубліковано тези доповіді на тему: «Assimilate: нове

покоління сервісів для роботи з шейдерами та 3D-моделями» [1]. Сторінки апробації роботи зазначені в додатку А.

Висновки до розділу 4

У четвертому розділі написано та протестовано програмну реалізацію клієнтської та її серверних частин у вебплатформі «Assimilate». Розроблено на базі екосистеми React, Three.js, React Three Fiber та ядра графічного редактора, що забезпечують імпорт та локальну обробку користувацької 3D-геометрії, математичну генерацію текстурних координат, параметризацію матеріалів та підключення карт фізично-коректного освітлення.

Розроблено та протестовано кастомний механізм ізольованої компіляції та перехоплення помилок від GLSL-коду, що підтверджує безперебійну, швидку взаємодію WebGL-контексту без ризику аварійного завершення роботи вкладок у браузері. Інтеграція платформи з хмарними сервісами Firebase у поєднанні з використаним менеджером глобального стану Redux, забезпечує надійну системну авторизацію користувачів та безпеку ключів доступів, збереження шейдерних проєктів та реактивне оновлення модулів під час публікації робіт користувачів.

Проведено повне тестування розробленого програмного забезпечення за міжнародними стандартами якості. Перевірка усіх складових вузлів платформи, що забезпечують її стабільність, на основі ряду підготовлених тестів на відмовостійкість, реактивність та безпеку користувачів за принципами нульової довіри.

Сформовано детальне керівництво користувача, з логічним та інтуїтивно зрозумілим життєвим циклом під час роботи із платформою. Надійність архітектурних рішень підтверджено апробацією результатів на науково-практичній конференції. Розроблений програмний продукт повною мірою задовольняє всі висунуті у специфікації функціональні та нефункціональні вимоги та готовий до базової експлуатації кінцевими користувачами.

ВИСНОВКИ

У кваліфікаційній роботі вирішено зазначене практичне завдання, що визначене як розробка та впровадження інтерактивної вебплатформи «Assimilate» – спеціалізованого програмного середовища для створення, тестування та соціальної інтеракції між користувачами з використанням 3D-моделей та шейдерним кодом у браузері. У результаті роботи поставлену мету досягнуто завдяки виконанню поставлених завдань:

1) проведено предметний аналіз з обґрунтуванням вибору технологічних рішень. Досліджено вже існуючі програмні застосунки на ринку комп'ютерної графіки та веброзробки. Обґрунтовано перевагу використання базової концепції односторінкового застосунку на основі екосистеми React, графічної бібліотеки Three.js, з використанням React Three Fiber та хмарної інфраструктури бази Firebase. Використання повної типізації у мові TypeScript дозволило забезпечити високу надійність архітектури в системі;

2) сформовано вимоги та проведено моделювання системи компонентів. Розроблено повну специфікацію функціонального та нефункціонального списку вимог. З використанням інструментів UML діаграм сконструйовано логічну модель поведінки системи, додатково розмежовано ролі користувачів на гостя, автора та адміністратора з формалізацією життєвого цикла графічних проєктів у системі;

3) спроєктовано архітектуру застосунку, базу даних та його інтерфейс. Створено архітектуру застосунку трирівневої взаємодії, що базується на принципі односпрямованого потоку даних із залученням менеджера Redux. Під'єднано та інтегровано документо-орієнтовану, нереляційну хмарну базу даних та налаштовано серверні правила безпеки, що реалізовують парадигму архітектурного стилю під назвою «нульова довіра». Розроблено користувацький графічний, адаптивний інтерфейс із впровадженням темної теми оформлення, елеіментами адмін-панелі, що орієнтовано на комфортну роботу з програмним кодом та 3D-сценами;

4) програмно реалізовано ядро модуля графічного редактора. Створено функціональну частину для імпорту користувацької 3D-геометрії на основі математичної генерації текстурних координат UV-розгортки. Додано систему параметризації матеріалів та підтримку карт освітлення HDRI. Основним технічним викликом стала реалізація кастомного модуля ізольованої компіляції GLSL-коду на принципах перехоплення помилок від графічного процесора, що гарантуватиме безперебійну роботу WebGL-контексту під час написання шейдерів користувачами;

5) інтегровано соціальні елементи платформи та перевірено відмовостійкість системи. Розроблено модулі автентифікації користувачів, включно з типами публічних/приватних проєктів, коментарів та лічильника вподобань з використанням атомарних транзакцій;

6) проведено комплексне тестування розробленого програмного застосунку. Навантажувальне тестування з оцінкою високої ефективності оптимізації системи: застосунок здатний обробляти об’ємні індустріальні моделі розміром понад 10 млн. полігонів зі стабільною частотою 60 кадрів на секунду. Тестування політик безпеки зазначило стійкий захист даних від несанкціонованого доступу.

Практичні результати розробки кваліфікаційної роботи пройшли апробацію на науково-практичній конференції «Могилянські читання». Розроблена платформа «Assimilate» повністю відповідає сучасним стандартам якості програмного забезпечення ISO/IEC 25010:2023 [11], вона задовольняє усі висунуті у специфікації вимоги. Застосунок готовий до експлуатації з залученням кінцевих користувачів. Отримані результати мають використовуватись як інструментарій для розробників комп’ютерної графіки, так і в якості освітньої платформи для вивчення програмування шейдерів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Обухова К. О., Шмалько Б. І. Assimilate: нове покоління сервісів для роботи з шейдерами та 3D-моделями. *Могилянські читання – 2025* : тези доп. XXVIII Всеукр. наук.-практ. конф., Миколаїв, 10–14 листоп. 2025 р. Миколаїв : Чорном. нац. ун-т ім. Петра Могили, 2025. С. 179–182. URL: https://dspace.chmnu.edu.ua/jspui/handle/123456789/3066 (дата звернення: 22.05.2026).

2. Cloud Firestore Documentation : website. URL: https://firebase.google.com/docs/firestore (Accessed: 03.01.2026).

3. Chandra B. Real-Time Data Processing in ERP Systems: Benefits and Challenges. *Journal of Information Systems Engineering and Management*. 2025. Vol. 10, No. 45s. P. 42–54. DOI: 10.52783/jisem.v10i45s.8889.

4. Dao P. Scalable React application with Redux and TypeScript : Bachelor's Thesis / South-Eastern Finland University of Applied Sciences. 2022. URL: https://urn.fi/URN:NBN:fi:amk-2022060114249 (Accessed: 21.01.2026).

5. Dave S. An Introduction to Creating Real-Time Interactive Computer Graphic Applications. *SA ’23 : SIGGRAPH Asia 2023 Courses*. Sydney, Australia, Dec. 12–15, 2023. P. 1–93. DOI: 10.1145/3610538.3614630.

6. Dirksen J. *Learn Three.js: Program 3D animations and visualizations for the web with JavaScript and WebGL*. Birningham, UK : Packt Publishing Ltd., Feb. 2023. 532 p. ISBN: 978-1-80323-387-1. URL: https://surl.li/kadiud (Accessed: 27.01.2026).

7. GLSL Sandbox : website. URL: https://glslsandbox.com/ (Accessed: 12.03.2026).

8. Hamzaturrazak M., Jonemaro E. M. A., Pinandito A. Performance analysis of 3D rendering method on web-based augmented reality application using WebGL and OpenGL shading language. *SIET ’23 : Proceedings of the 8th International Conference on Sustainable Information Engineering and Technology*. Badung, Indonesia, Oct. 24–25, 2023. P. 637–643. DOI: 10.1145/3626641.3626949.

9. Hatami M., Qu Q., Chen Y., Kholidy H., Blasch E. A survey of the real-time metaverse: Challenges and opportunities. *Future Internet*. 2024. Vol. 16, Is. 10. P. 1–52. DOI: 10.3390/fi16100379.

10. IEEE Std 829-2008. IEEE Standard for Software and System Test Documentation. Jul. 18, 2008. 150 p. DOI: 10.1109/IEEESTD.2008.4578383.

11. ISO/IEC 25010:2023. Systems and software engineering – Systems and software quality requirements and evaluation (SQuaRE) – Product quality model. Nov. 2023. 22 p. URL: https://www.iso.org/standard/78176.html (Accessed: 12.03.2026).

12. ISO/IEC/IEEE 29119-1:2022. Software and systems engineering – Software testing – Part 1: General concepts. Jan. 2022. 47 p. URL: https://www.iso.org/standard/81291.html (Accessed: 12.03.2026).

13. Jantan A. H., Norowi N. M., Yazid M. A. UI/UX Fundamental Design for Mobile Application Prototype to Support Web Accessibility and Usability Acceptance: A Scenario-based Design of Mobile Application for Visually Impaired Users. *ICSCA ’23 : Proceedings of the 2023 12th International Conference on Software and Computer Applications*. Kuantan, Malaysia, Feb. 23–25, 2023. P. 105–111. DOI: 10.1145/3587828.3587845.

14. Kuge T., Yatagawa T., Morishima S. Real-time Shading with Free-form Planar Area Lights using Linearly Transformed Cosines. *The Journal of Computer Graphics Techniques*. 2023. Vol. 10, № 1. P. 869–883. URL: https://www.jcgt.org/published/0011/01/01/ (Accessed: 15.02.2026).

15. Mattila T. Professional development journal of a software developer working with React and TypeScript: A reflection of the lessons gained from a front-end software development project : *Bachelor's Thesis / Turku University of Applied Sciences*. 2025. URL: https://urn.fi/URN:NBN:fi:amk-2025061723180 (Accessed: 16.02.2026).

16. Minxi D. 3D Visualization Interactive Design for Smart Home Remote Monitoring Using WebRTC and Three.js. *IoTCCT ’25 : Proceedings of the 2025 3rd International Conference on Internet of Things and Cloud Computing Technology*. Haikou, China, Dec. 15, 2025. Vol. 25. P. 153–159. DOI: 10.1145/3776865.3776891.

17. Misjuns A. Guiding the Creative Process of Shader Programming to Meet Output Requirements for Virtual Reality on the Web. *In book: HCI International 2025 Posters*. Cham : Springers, 2025. Vol. 2522. P. 121–129. DOI: 10.1007/978-3-031-94150-4_13.

18. Nguyen N. Creating a modern web user interface using React and TypeScript : *Bachelor's Thesis / Vaasa University of Applied Sciences*. 2022. URL: https://urn.fi/URN:NBN:fi:amk-202204265988 (Accessed: 23.02.2026).

19. NIST Special Publication 800-207. Zero Trust Architecture. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf (Accessed: 12.03.2026).

20. Onishchenko S. Building Cross-Platform Enterprise Systems Using React.js and TypeScript with Consideration of Zero Trust Principles. *World Conference on Emerging Science, Innovation and Policy 2025*. Poland, Jun. 1 – Jul. 20, 2025. Vol. 3. P. 1–6. DOI: 10.5281/zenodo.15664966.

21. Panchal S., Raval P., Shetty S., Ambadekar S. College 3D model rendering using Three.js. *2022 5th International Conference on Advances in Science and Technology (ICAST)*. Mumbai, India, Dec. 2–3, 2022. P. 142–147. DOI: 10.1109/ICAST55766.2022.

22. Paszkiewicz A., Salach M., Ganzha M. et al. Implementation of UI Methods and UX in VR in Case of 3D Printer Tutorial. *Frontiers in Artificial Intelligence and Applications*. 2022. Vol. 355. P. 460–471. DOI: 10.3233/FAIA220275.

23. PlayCanvas : website. URL: https://playcanvas.com/ (Accessed: 12.03.2026).

24. Rippon C. *Learn React with TypeScript: A beginner’s guide to reactive web development with React 18 and TypeScript*. Packt Publishing Limited, 2024. ISBN: 978-1-80461-105-0.

25. Shadertoy : website. URL: https://www.shadertoy.com/ (Accessed: 12.03.2026).

26. Smelov A. Integration of interactive 3D models into React-based application : *Bachelor's Thesis / Seinäjoki University of Applied Sciences*. 2024. URL: https://urn.fi/URN:NBN:fi:amk-2024120933986 (Accessed: 20.03.2026).

27. Steeger S., Atzberger D., Scheibel W. Instanced Rendering of Parameterized 3D Glyphs with Adaptive Level-of-Detail using Three.js. *Web3D ’24 : Proceedings of the 29th International ACM Conference on 3D Web Technology*. Guimarães, Portugal, Sep. 25, 2024. Vol. 1. P. 1–11. DOI: 10.1145/3665318.3677171.

28. Suni S. Creating an interactive 3D map prototype using Spline : *Bachelor's Thesis / Tampere University of Applied Sciences*. 2024. URL: https://urn.fi/URN:NBN:fi:amk-2024112931402 (Accessed: 09.04.2026).

29. Sunjaya D. R., Shibghatullah A. S. bin, Anjum S. S. The use of heuristic evaluation on UI/UX design: A review to anticipate web app’s usability. *In book: Intelligent Communication Technologies and Virtual Mobile Networks*. Singapore : Springer, Jun. 2023. Vol. 171. P. 119–128. DOI: 10.1007/978-981-99-1767-9_9.

ДОДАТОК А

Матеріали апробації роботи

XXVIII Всеукраїнська науково-практична конференція «Могилянські читання – 2025»

Міністерство освіти і науки України
Чорноморський національний університет імені Петра Могили
ДНУ «Інститут модернізації змісту освіти»
Південний науковий центр НАН та МОН
Інститут української археографії та джерелознавства
імені М. С. Грушевського НАН України
Первинна профспілкова організація ЧНУ ім. Петра Могили

**«МОГИЛЯНСЬКІ ЧИТАННЯ – 2025:
досвід та тенденції розвитку суспільства в Україні: глобальний,
національний та регіональний аспекти»**

XXVIII Всеукраїнська науково-практична конференція

ТЕЗИ ДОПОВІДЕЙ

ТЕХНІЧНІ НАУКИ

Миколаїв, 10–14 листопада 2025 року

Миколаїв – 2025

Підсекція:

- **МОДЕЛІ, МЕТОДИ ТА ЗАСОБИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ**

УДК 004.45:004.925

Обухова. К. О.
старша викладачка кафедри комп'ютерної інженерії,
Чорноморський національний університет ім. Петра Могили, м. Миколаїв, Україна
Шмалько Б. І.
здобувач першого (бакалаврського) рівня вищої освіти,
Чорноморський національній університет ім. Петра Могили, м. Миколаїв, Україна

ASSIMILATE: НОВЕ ПОКОЛІННЯ СЕРВІСІВ ДЛЯ РОБОТИ З ШЕЙДЕРАМИ ТА 3D-МОДЕЛЯМИ

Сучасні вебсервіси для роботи з графікою та візуальними ефектами, зокрема ShaderToy, GLSL Sandbox тощо, стали важливими інструментами як для дослідження технологій комп'ютерної графіки та практики в написанні шейдерів для великої кількості платформ, так і для навчання програмуванню. Проте більшість подібних застосунків обмежені демонстрацією шейдерів на простих поверхнях (площина, сфера, куб) і не забезпечують повноцінної інтеграції з користувацькими 3D-моделями розробників і дизайнерів, текстурами чи складними постпроцесами (рис. 1). Деякі сервіси, навпаки, відійшли від практики роботи з об'єктами, зосереджуючись на оточенні.

Отже, мета даної роботи – розробити онлайн-сервіс нового покоління, що поєднує зручність вебплатформи з функціональністю професійних графічних редакторів шейдерів із використанням власних 3D-моделей.

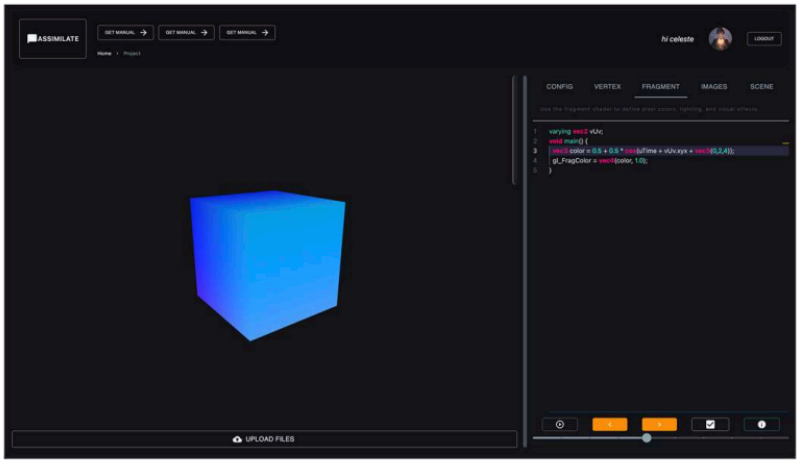

Рисунок 1 – Приклад застосування шейдера до примітива

Проєкт **Assimilate** розробляється як комплексне середовище, що поєднує зручність розробки оточення з функціональністю професійних графічних редакторів шейдерів: підтримує роботу з сіткою та UV-координатами об'єктів, дозволяючи змінювати властивості відображення та накладання елементів. Основна концепція полягає у можливості завантаження користувацьких 3D-моделей, на які можна накладати шейдери в реальному часі, експериментувати з освітленням, матеріалами, текстурами та ефектами постобробки, окремо налаштовуючи елементи оточення.

Для досвідчених користувачів передбачено повний доступ до функціоналу сервісу, включно з трьома редакторами – вершин, площин та оточення (рис. 2). Крім того, користувачі зможуть виконувати постобробку, застосовувати матеріали з оточенням та експериментувати з їх параметрами.

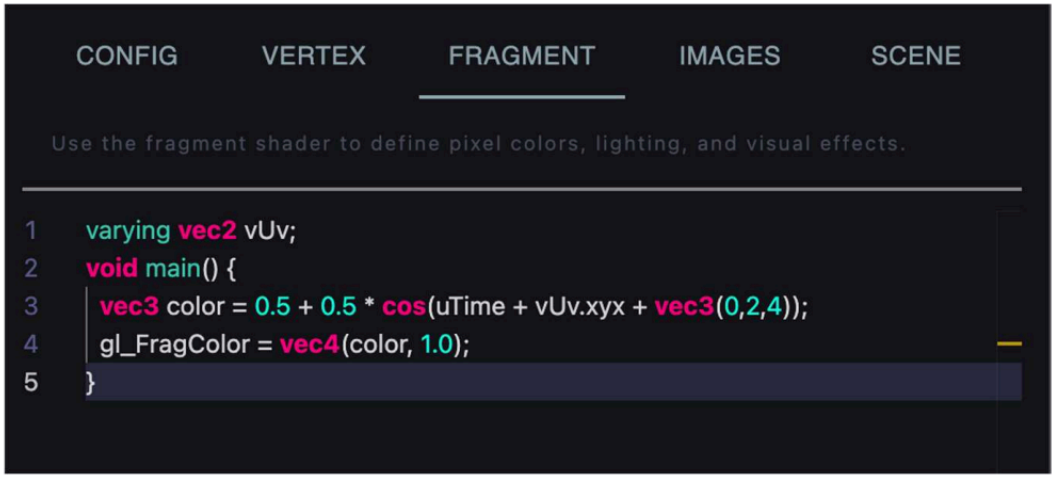

Рисунок 2 – Приклад GLSL-фрагментного коду

Для недосвідчених користувачів передбачено спрощений режим із мінімалістичною панеллю керування для ознайомлення, що дасть змогу візуально змінювати компоненти текстур, або додавати нові без використання складних інструментів та професійної термінології.

Важливим елементом є **валідація коду шейдерів**: перед змінами система перевіряє код на помилки, забезпечуючи стабільність виконання. Також, впроваджена система облікових записів, що дозволяє зберігати, публікувати роботи у профіль користувача та відстежувати їх популярність. При збережені роботи, система автоматично генерує прев'ю проєкту, що зберігається у вигляді іконки і буде використовуватись для відображення іншим користувачам. Крім

того, можна створити GIF-анімацію або відео, що демонструє динаміку шейдера (рис. 3).

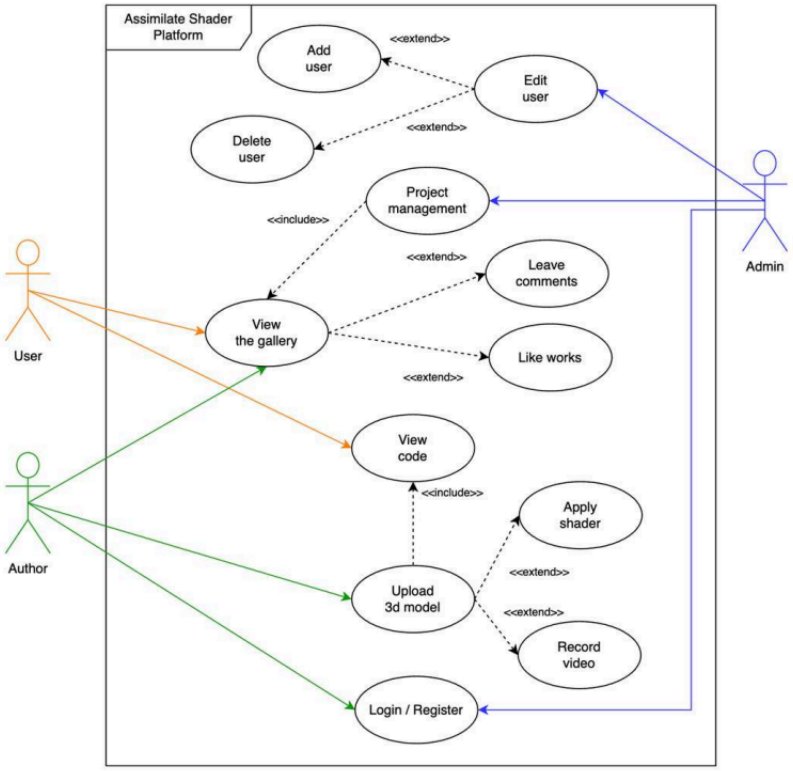

Рисунок 3 – Діаграма варіантів використання

Користувацький профіль у системі дозволяє зберігати створені проєкти, ділитися ними з іншими учасниками спільноти та відстежувати статистику переглядів і рівень популярності. Кожна робота, збережена користувачем, отримує власний унікальний ідентифікатор, який через систему «індексів» може бути винесений на головну сторінку робіт, якщо автор цього забажає. Завдяки цьому інші користувачі отримують змогу переглядати, оцінювати та коментувати представлені матеріали, що сприяє активному обміну думками.

Кожен опублікований проєкт супроводжується відкритим кодом шейдера, що забезпечує прозорість усіх модифікацій і налаштувань. Такий підхід формує середовище для розвитку спільноти авторів, де важливими є не лише кінцеві технічні результати, але й сам процес співпраці, можливість навчання один в одного, обміну досвідом та творчого змагання. У результаті система стає не просто майданчиком

для демонстрації робіт, а й простором для колективного зростання та натхнення.

Assimilate Shader Platform поєднує навчальну та творчу складові, забезпечуючи розширені можливості для роботи з шейдерами та 3D-графікою без потреби у встановленні додаткового програмного забезпечення. У перспективі платформа може стати як інструментом для освіти, так і базою для створення інноваційних графічних ефектів, сприяючи розвитку цифрових технологій та інтерактивної візуалізації.

Список використаних джерел

1. Kessenich J., Baldwin D., Rost R. The OpenGL Shading Language, Version 4.60.8. URL: https://registry.khronos.org/OpenGL/specs/gl/GLSLangSpec.4.60.pdf (Last accessed: 13.10.2025).

2. Gonzalez Vivo P., Lowe J. The Book of Shaders. URL: https://thebookofshaders.com/ (Last accessed: 13.10.2025).

3. de Vries J. Learn OpenGL. URL: https://learnopengl.com/About (Last accessed: 13.10.2025).

ДОДАТОК Б

Лістинг коду валідації шейдерів на канвасі

```
type ObjectProps = {
  object: {
    rotation: {
      isRotating: boolean;
      axis: 'x' | 'y' | 'z';
      speed: number;
      direction: number;
    };
  };
}
export type ShaderError = {
  type: 'vertex' | 'fragment' | 'link' | 'validate' | 'webgl' | 'success';
  message: string;
  raw?: string;
  valid: boolean;
}
function validateShaderInWebGL(vertex: string, fragment: string) {
  const canvas = document.createElement('canvas');
  const gl: any = canvas.getContext('webgl') || canvas.getContext('experimental-webgl');
  if (!gl) return { valid: false, type: 'webgl', log: 'WebGL not available' } as const;
  const program = gl.createProgram();
  if (!program) return { valid: false, type: 'webgl', log: 'Failed to create program' } as
const;
  const patchedVertex = `
      precision mediump float;
      uniform mat4 projectionMatrix;
      uniform mat4 modelViewMatrix;
      uniform mat3 normalMatrix;
      attribute vec3 position;
      attribute vec3 normal;
      attribute vec2 uv;
      ${vertex}
  `;
  const patchedFragment = `
    precision mediump float;
    ${fragment}
  `;
  const vert = gl.createShader(gl.VERTEX_SHADER);
  gl.shaderSource(vert, patchedVertex);
  gl.compileShader(vert);
  if (!gl.getShaderParameter(vert, gl.COMPILE_STATUS)) {
    const log = gl.getShaderInfoLog(vert) || 'Unknown vertex shader error';
    cleanup(gl, program, vert, null);
    return { valid: false, type: 'vertex', log } as const;
  }
  const frag = gl.createShader(gl.FRAGMENT_SHADER);
  gl.shaderSource(frag, patchedFragment);
  gl.compileShader(frag);
  if (!gl.getShaderParameter(frag, gl.COMPILE_STATUS)) {
    const log = gl.getShaderInfoLog(frag) || 'Unknown fragment shader error';
    cleanup(gl, program, vert, frag);
    return { valid: false, type: 'fragment', log } as const;
  }
  gl.attachShader(program, vert);
  gl.attachShader(program, frag);
```

```
  gl.linkProgram(program);
  if (!gl.getProgramParameter(program, gl.LINK_STATUS)) {
    const log = gl.getProgramInfoLog(program) || 'Linking failed';
    cleanup(gl, program, vert, frag);
    return { valid: false, type: 'link', log } as const;
  }
  gl.validateProgram(program);
  if (!gl.getProgramParameter(program, gl.VALIDATE_STATUS)) {
    const log = gl.getProgramInfoLog(program) || 'Validation failed';
    cleanup(gl, program, vert, frag);
    return { valid: false, type: 'validate', log } as const;
  }
  cleanup(gl, program, vert, frag);
  return { valid: true } as const;
}
function cleanup(
  gl: WebGLRenderingContext,
  program: WebGLProgram,
  vert: WebGLShader | null,
  frag: WebGLShader | null
) {
  if (vert) gl.deleteShader(vert);
  if (frag) gl.deleteShader(frag);
  gl.deleteProgram(program);
}
function parseGlslError(log: string): string {
  const lineMatch = log.match(/ERROR:\s*(\d+):(\d+)/);
  const message = log.split(':').slice(3).join(':').trim();
  if (lineMatch) {
    const line = parseInt(lineMatch[1], 10);
    return `Line ${line+1}: ${message}`;
  }
  return message || 'Shader error';
}
const SafeShaderMaterial = ({
  textures,
  vertex,
  fragment,
  uniforms,
  onError,
}: {
  textures: any;
  vertex: string;
  fragment: string;
  uniforms: Record<string, any>;
  onError: (error: ShaderError) => void;
}) => {
  const [material, setMaterial] = useState<THREE.ShaderMaterial | null>(null);
  const lastValidRef = useRef({ vertex, fragment });

  useEffect(() => {
    const validation = validateShaderInWebGL(vertex, fragment);

    if (!validation.valid) {
      const err: ShaderError = {
        type: validation.type || 'webgl',
        message: parseGlslError(validation.log),
        raw: validation.log,
        valid: false
      };
```

```
      onError(err);
      const fallback = new THREE.ShaderMaterial({
        vertexShader: lastValidRef.current.vertex,
        fragmentShader: lastValidRef.current.fragment,
        uniforms: { ...uniforms },
      });
      setMaterial(fallback);
      return;
    } else {
      const err: ShaderError = {
        type: 'success',
        message: 'Shader compiled successful',
        raw: validation.log,
        valid: true
      };
      onError(err)
    }
    lastValidRef.current = { vertex, fragment };
    const newMaterial = new THREE.ShaderMaterial({
      vertexShader: vertex,
      fragmentShader: fragment,
      uniforms: {
        ...uniforms,
      }
    });
    setMaterial(newMaterial);
    return () => {
      if (material && material !== newMaterial) {
        material.dispose();
      }
    };
  }, [vertex, fragment, uniforms, onError]);
  return material ? <primitive object={material} attach="material" /> : null;
};
export function ModelObject({
  url,
  objectProps,
  vertexProps,
  fragmentProps,
  shadeError,
  textures,
  useImportType,
  hdriUrl,
  hdriProps,
  sceneProps
}: {
  url: string;
  objectProps: ObjectProps;
  vertexProps: string;
  fragmentProps: string;
  shadeError: (error: ShaderError) => void;
  textures: Record<string, TextureProps>;
  useImportType: (uniforms: any) => void;
  hdriUrl?: string;
  hdriProps?: HdriProps;
  sceneProps: { sceneColor: string | null }
}) {
  const geometry = useLoader(STLLoader, url);
  const meshRef = useRef<THREE.Mesh>(null);
  const [loadedTextures, setLoadedTextures] = useState<Record<string, THREE.Texture>>({});
```

```
  const { scene } = useThree();
  useHdriEnvironment(hdriUrl ?? null, hdriProps ?? null);
  useEffect(() => {
    if (sceneProps?.sceneColor === "#00000000") {
      scene.background = null;
    } else {
      scene.background = new THREE.Color(sceneProps?.sceneColor)
    }
    console.log(scene.background)
  }, [sceneProps])
  useEffect(() => {
  console.log('hdriProps:', hdriProps);
}, [hdriProps]);
  useEffect(() => {
    if (geometry) {
      generateUVs(geometry);
    }
  }, [geometry]);
  useEffect(() => {
    if (textures) {
      const loadAllTextures = async () => {
        const entries = await Promise.all(
          Object.entries(textures).map(async ([name, tex]) => {
            const texture = await createTexture(tex);
            return [name, texture] as const;
          })
        );
        setLoadedTextures(Object.fromEntries(entries));
      };
      loadAllTextures();
    }
  }, [textures]);
  const textureUniforms = Object.entries(loadedTextures).reduce((acc, [name, tex]) => {
    acc[`u_${textures[name].slot}`] = { value: tex };
    return acc;
  }, {} as Record<string, { value: THREE.Texture }>);
  const uniforms = {
    uTime: { value: 0 },
    ...textureUniforms,
  };
  Object.entries(textures).forEach(([name, tex]) => {
    const slot = tex.slot;
    const props = tex.props;

    uniforms[`u_${slot}_repeat`] = { value: new THREE.Vector2(...props.repeat) };
    uniforms[`u_${slot}_offset`] = { value: new THREE.Vector2(...props.offset) };
    uniforms[`u_${slot}_center`] = { value: new THREE.Vector2(...props.center) };
    uniforms[`u_${slot}_rotation`] = { value: props.rotation };
  });
  useEffect(() => {
    useImportType(uniforms);
  }, [uniforms, useImportType]);

  const handleError = useCallback((err: ShaderError) => {
    shadeError(err);
  }, [shadeError]);

  useFrame(({ clock }) => {
    const mesh = meshRef.current;
    if (mesh && objectProps?.object.rotation.isRotating) {
```

```
      const { axis, speed, direction } = objectProps.object.rotation;
      mesh.rotation[axis] += speed * direction;
    }
    if (hdriProps) {
      if (hdriProps.isRotating)
        scene.backgroundRotation.set(0, clock.getElapsedTime() * hdriProps.rotationSpeed /
100, 0);
    }
    if (mesh?.material && 'uniforms' in mesh.material) {
      const mat = mesh.material as THREE.ShaderMaterial;
      if (mat.uniforms.uTime) {
        mat.uniforms.uTime.value = clock.getElapsedTime();
      }
    }
  });
  return (
    <>
      <mesh ref={meshRef} geometry={geometry}>
        <SafeShaderMaterial
          textures={textures}
          vertex={vertexProps}
          fragment={fragmentProps}
          uniforms={uniforms}
          onError={handleError}
        />
      </mesh>
    </>
  );
}
async function createTexture(textureProps: TextureProps): Promise<THREE.Texture> {
  const url = URL.createObjectURL(textureProps.file);
  const texture: any = await loadTextureAsync(url);
  URL.revokeObjectURL(url);
  const props = textureProps.props;
  texture.wrapS = THREE[props.wrapS as keyof typeof THREE];
  texture.wrapT = THREE[props.wrapT as keyof typeof THREE];
  texture.encoding = THREE[props.encoding as keyof typeof THREE];
  texture.magFilter = THREE[props.magFilter as keyof typeof THREE];
  texture.minFilter = THREE[props.minFilter as keyof typeof THREE];
  texture.mapping = THREE[props.mapping as keyof typeof THREE];
  texture.format = THREE[props.format as keyof typeof THREE];
  texture.type = THREE[props.type as keyof typeof THREE];
  texture.anisotropy = props.anisotropy;
  texture.flipY = props.flipY;
  texture.needsUpdate = true;
  texture.matrixAutoUpdate = true;

  return texture;
}

function loadTextureAsync(url: string): Promise<THREE.Texture> {
  return new Promise((resolve, reject) => {
    new THREE.TextureLoader().load(
      url,
      (texture) => resolve(texture),
      undefined,
      (err) => reject(err)
    );
  });
}
```

```
function generateUVs(geometry: THREE.BufferGeometry) {
  geometry.computeBoundingBox();
  const bbox = geometry.boundingBox!;
  const size = new THREE.Vector3();
  bbox.getSize(size);
  const uvAttr = new Float32Array(geometry.attributes.position.count * 2);
  for (let i = 0; i < geometry.attributes.position.count; i++) {
    const x = geometry.attributes.position.getX(i);
    const z = geometry.attributes.position.getZ(i);
    const u = (z - bbox.min.z) / size.z;
    const v = (x - bbox.min.x) / size.x;
    uvAttr[i * 2] = u;
    uvAttr[i * 2 + 1] = v;
  }
  geometry.setAttribute('uv', new THREE.BufferAttribute(uvAttr, 2));
}
export function useHdriEnvironment(
  hdriUrl: string | null,
  hdriProps: { intensity: number; rotation: number } | null
) {
  const { scene, gl } = useThree();
  const pmremGeneratorRef = useRef<THREE.PMREMGenerator | null>(null);
  const envMapRef = useRef<THREE.Texture | null>(null);
  const hdrTextureRef = useRef<THREE.Texture | null>(null);
  useEffect(() => {
    pmremGeneratorRef.current = new THREE.PMREMGenerator(gl);
    pmremGeneratorRef.current.compiler = true;
    return () => {
      if (pmremGeneratorRef.current) {
        pmremGeneratorRef.current.dispose();
      }
    };
  }, [gl]);
  useEffect(() => {
    if (!hdriUrl || !pmremGeneratorRef.current) {
      scene.environment = null;
      scene.background = null;
      if (envMapRef.current) {
        envMapRef.current.dispose();
        envMapRef.current = null;
      }
      if (hdrTextureRef.current) {
        hdrTextureRef.current.dispose();
        hdrTextureRef.current = null;
      }
      return;
    }
    const loader = new RGBELoader();
    loader.load(
      hdriUrl,
      (hdrTexture) => {
        hdrTexture.mapping = THREE.EquirectangularReflectionMapping;
        const envMap = pmremGeneratorRef.current!.fromEquirectangular(hdrTexture).texture;
        scene.environment = envMap;
        scene.background = envMap;
        gl.outputEncoding = THREE.sRGBEncoding;
        gl.toneMapping = THREE.ACESFilmicToneMapping;
        gl.toneMappingExposure = hdriProps?.intensity || 1.0;
        scene.backgroundRotation.set(0, (hdriProps?.rotation || 0) * Math.PI / 180, 0);
        envMapRef.current = envMap;
```

```
        hdrTextureRef.current = hdrTexture;
        console.log('HDRI loaded successfully:', hdriUrl);
      },
      undefined,
      (err) => {
        console.error('HDRI loading failed:', err);
      }
    );
    return () => {
      if (envMapRef.current) {
        envMapRef.current.dispose();
        envMapRef.current = null;
      }
      if (hdrTextureRef.current) {
        hdrTextureRef.current.dispose();
        hdrTextureRef.current = null;
      }
      scene.environment = null;
      scene.background = null;
    };
  }, [hdriUrl, scene, gl]);
  useEffect(() => {
    if (hdriProps) {
      gl.toneMappingExposure = hdriProps.intensity;
      scene.backgroundRotation.set(0, hdriProps.rotation * Math.PI / 180, 0);
      console.log('HDRI updated: intensity =', hdriProps.intensity, 'rotation =',
hdriProps.rotation);
    }
  }, [hdriProps, scene, gl]);
```