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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка інтерактивного середовища “Портфоліо 3D-розробника”»

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

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

_____ Богдан МОМОТ
(підпис)

Виконав: здобувач вищої освіти групи ПД-43

Богдан МОМОТ

Керівник: Юрій БАЖАН
викладач
Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти Бакалавр
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри
Інженерії програмного забезпечення
_____Ірина ЗАМРІЙ
« ____» _____2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Момоту Богдану Дмитровичу

1. Тема кваліфікаційної роботи: «Розробка застосунку для організації спільних поїздок мешканців житлового комплексу»
керівник кваліфікаційної роботи викладач кафедри ІПЗ Юрій БАЖАН,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: опис технічних можливостей створення SPA-застосунку з тривимірною графікою, специфікації функціональних і нефункціональних вимог, архітектура клієнтської частини, схема структури проєкту, приклади інтеграції з бібліотеками Three.js та @react-three/fiber, опис логіки анімацій, макети інтерфейсу та принципи побудови користувацького досвіду.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз проблеми та огляд сучасних рішень для цифрової самопрезентації розробників у форматі інтерактивного вебпортфоліо.

2. Проєктування вебзастосунку з використанням 3D-графіки: архітектура, моделювання вимог, дизайн інтерфейсу, логіка взаємодії.
3. Програмна реалізація та опис функціонування інтерактивного середовища на базі React, Three.js та Tailwind CSS.
4. Тестування функціональності, адаптивності й інтерактивності застосунку, а також аналіз отриманих результатів та перспектив розвитку.

5.Перелік графічного матеріалу: *презентація:*

3. Аналіз аналогів.
4. Вимоги до програмного забезпечення.
5.Програмні засоби реалізації.
6.Use-case діаграма.
7.Алгоритм роботи.
8.Архітекутра проєкту.
9.Екранні форми.
10.Апробація результатів дослідження.

6. Дата видачі завдання«25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Аналіз проблеми та огляд сучасних рішень для цифрової самопрезентації розробників у форматі інтерактивного вебпортфоліо.	14.03 -27.03.2025	
4	Проєктування вебзастосунку з використанням 3D-графіки: архітектура, моделювання вимог, дизайн інтерфейсу, логіка взаємодії.	28.03 -09.04.2025	
5	Програмна реалізація та опис функціонування інтерактивного середовища на базі React, Three.js та Tailwind CSS.	10.04 -21.04.2025	
6	Тестування функціональності, адаптивності й інтерактивності застосунку, а також аналіз отриманих результатів та перспектив розвитку.	22.04 -28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Богдан МОМОТ
(підпис)

Керівник
кваліфікаційної роботи _____ Юрій БАЖАН
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 76 стор., 2 табл., 17 рис., 25 джерел.

Мета роботи – створення персоналізованого інтерактивного веб-застосунку у форматі 3D-портфоліо для презентації професійних навичок і проєктів розробника з акцентом на візуальну виразність, адаптивність та сучасні фронтенд-технології.

Об'єкт дослідження – процес цифрової самопрезентації спеціалістів у сфері IT.

Предмет дослідження – веб-застосунок для створення інтерактивного 3D-портфоліо розробника.

Короткий зміст роботи: У роботі досліджено сучасні підходи до створення веб-портфоліо, зокрема з використанням тривимірної графіки. Проведено аналіз існуючих рішень, зокрема портфоліо Bruno Simon, Renaud Rohlinger та інших. Обґрунтовано вибір SPA-архітектури та використання React, Three.js, Tailwind CSS.

Розроблено архітектурну модель клієнтського застосунку, що забезпечує візуалізацію тривимірної сцени, взаємодію з моделями (анімації, переміщення), перегляд розділів портфоліо (проєкти, навички, контакти), інтерактивні підказки та адаптацію інтерфейсу під різні пристрої.Фронтенд реалізовано з використанням React, маршрутизацію забезпечено через React Router, анімації – через Framer Motion. Тривимірні елементи побудовані за допомогою @react-three/fiber та @react-three/drei. Додано форму зворотного зв'язку з інтеграцією через EmailJS та можливість вмикання/вимикання музичного супроводу.

Застосунок протестовано в різних браузерах і на різних екранах, підтверджено відповідність функціональних і нефункціональних вимог. Проведено навантажувальне тестування та оцінку швидкодії.

Сфера застосування розробленого середовища – персональна технічна презентація для фахівців у сфері веброзробки, UI/UX-дизайну, а також демонстрація навичок у резюме або під час проходження співбесід.

КЛЮЧОВІ СЛОВА: PORTFOLIO, REACT, THREE.JS, SPA, ВЕБ-ЗАСТОСУНОК, 3D-ІНТЕРФЕЙС, ІНТЕРАКТИВНІСТЬ, TAILWIND CSS, EMAILJS, ВІЗУАЛІЗАЦІЯ.

ЗМІСТ

ВСТУП

Конкуренція на ринку інформаційних технологій потребує від фахівців не лише професійних навичок, а й уміння демонструвати власні результати у привабливій та доступній формі. Особливо це стосується frontend-розробників, для яких важливою є не лише функціональність продуктів, а й їх візуальна складова. Прості текстові резюме вже не забезпечують достатнього рівня інформативності та емоційного впливу. Натомість, інтерактивне портфоліо дозволяє поєднати в одному середовищі презентацію проєктів, опис навичок і елементи взаємодії з потенційним роботодавцем або замовником.

Портфоліо, реалізоване у вигляді односторінкового застосунку з інтеграцією тривимірної графіки, дозволяє фахівцю виділитися на тлі інших претендентів та водночас продемонструвати технічну обізнаність і креативність. Такий підхід розширює традиційні формати представлення навичок та досягає нової глибини взаємодії з глядачем. Користувач може не просто прочитати інформацію, а зануритися у віртуальний простір, у якому органічно поєднано функціональність, анімацію, візуальний стиль і логіку навігації.

Застосунок, що розробляється в межах цієї роботи, складається з кількох функціональних частин: головної сцени з 3D-об’єктами, сторінок «Про мене», «Навички», «Проєкти», а також форми зворотного зв’язку. У головній сцені представлені анімовані 3D-моделі — птах, лис, літак, острів, небо — які реагують на дії користувача. Це дозволяє не лише передати візуальні ефекти, але й додати інтуїтивну логіку керування, створюючи ефект взаємодії з живим цифровим середовищем.

В основі проєкту лежить використання React — бібліотеки для створення інтерфейсів, Three.js — засобу візуалізації 3D-графіки, а також Tailwind CSS — фреймворку для швидкої розробки адаптивного дизайну. Завдяки застосуванню цих

технологій реалізовано функціонал, що поєднує складні графічні елементи з високою продуктивністю та зручною структурою коду. Додаткові засоби — Framer Motion, @react-three/fiber, React Router — доповнюють інтерфейс плавними анімаціями, внутрішньою маршрутизацією та ефектами переходів.

Актуальність реалізованого підходу підтверджується тим, що більшість сучасних портфоліо залишаються статичними або малоінтерактивними. Вони рідко демонструють технологічні вміння автора у безпосередній взаємодії. На противагу цьому, запропоноване рішення дозволяє показати не лише результат, а й сам процес: анімації моделей, поведінкову логіку об'єктів, синхронізацію станів, адаптивну верстку, роботу з подіями тощо. Таке портфоліо не потребує окремих демонстрацій технічних навичок — воно вже є прикладом застосування всіх необхідних інструментів.

Крім того, проєкт реалізує нову комбінацію функціональних і нефункціональних можливостей, які в сукупності створюють продукт із підвищеною виразністю. Інтерактивна 3D-сцена, музичне оформлення, адаптивність, можливість масштабування, підключення до зовнішніх сервісів — усе це надає рішенням додаткову гнучкість. Проєкт орієнтований на IT-фахівців, які бажають представити себе у нестандартний спосіб, не вдаючись до шаблонних рішень.

Об’єкт дослідження — процес цифрової самопрезентації IT-спеціаліста за допомогою візуально-інтерактивних інтерфейсів.

Предмет дослідження — програмний засіб у вигляді SPA-додатку з тривимірною графікою для портфоліо розробника.

Мета роботи — підвищення ефективності представлення технічних навичок frontend-розробника шляхом створення інтерфейсу з 3D-сценою, який скорочує час на ознайомлення з інформацією, зменшує кількість кроків користувача до цільових дій і покращує емоційне сприйняття.

Досягнення цієї мети забезпечується за рахунок:

- використання актуального стеку технологій (React, Tailwind, Three.js, Vite);
- побудови сцени з анімованими 3D-об’єктами та логікою їх взаємодії;
- створення логічно структурованої навігації та компонентної архітектури;
- реалізації адаптивності для різних пристроїв;
- розробки контактного механізму з можливістю інтеграції сторонніх сервісів (наприклад, EmailJS).

Основні завдання роботи:

1. Дослідити предметну галузь та способи представлення цифрового портфоліо.
2. Проаналізувати існуючі засоби реалізації інтерактивних інтерфейсів.
3. Визначити обмеження аналогічних рішень і сформувати технічні вимоги до нового продукту.
4. Спроєктувати архітектуру SPA-додатку та визначити логіку 3D-сцени.
5. Реалізувати інтерфейс з інтерактивною взаємодією користувача та графічним середовищем.
6. Провести тестування на різних пристроях для оцінки зручності, стабільності та адаптивності.
7. Визначити напрями подальшого розширення функціоналу (аналітика, багатомовність, бекенд).

Методи дослідження, що застосовуються в роботі: моделювання архітектури інтерфейсів, компонування сцени, порівняльний аналіз аналогів, розробка програмних рішень за методами компонентного програмування, а також емпіричне тестування інтерактивності та адаптивності розробленого рішення.

Робота поєднує в собі проєктування, реалізацію та оцінку інтерфейсу, який одночасно виконує комунікативну, технічну та презентаційну функції. Отриманий результат може застосовуватись не лише як приклад портфоліо розробника, але й як основа для створення презентаційних інтерфейсів в інших професійних галузях, що потребують візуального самовираження.

1 АНАЛІЗ ПРОБЛЕМИ ТА СУЧАСНИХ ЗАСОБІВ ЇЇ ВИРІШЕННЯ

1.1. Опис предметної галузі: цифрова самопрезентація розробників

Сфера інформаційних технологій передбачає активну конкуренцію між фахівцями, а також постійне оновлення знань і навичок. У зв’язку з цим виникає потреба не лише у розвитку професійних компетенцій, а й у їх демонстрації в умовах швидкої комунікації, візуального сприйняття та стислого формату подання. Одним із інструментів, що вирішує це завдання, є цифрове портфоліо, призначене для представлення результатів професійної діяльності, технічного рівня, проєктів, сертифікацій, практичного досвіду та творчих підходів до вирішення завдань.

У випадку frontend-розробників, які працюють зі створенням інтерфейсів користувача, структура та вигляд цифрового портфоліо мають безпосередній зв’язок із практичними навичками. Демонстрація реалізованих проєктів, макетів, інтерактивних елементів та адаптивних рішень відображає не лише вміст, а й рівень володіння технологіями. Тому цифрове портфоліо розробника стає не просто презентаційним матеріалом, а прикладом конкретної реалізації функціональних можливостей, які можуть оцінити фахівці галузі.

Розглянута предметна галузь охоплює системи візуального самопредставлення, які функціонують в електронному форматі. До таких систем належать як персональні веб-сайти, так і профілі на спеціалізованих платформах (наприклад, GitHub, Behance, Dribbble, LinkedIn). Проте у випадку з вузькопрофільними спеціалістами, зокрема frontend-інженерами, найбільш повноцінним засобом репрезентації вважається індивідуально розроблений веб-ресурс. Такий підхід дозволяє уникнути обмежень, накладених стандартними шаблонами та сервісами, і реалізувати власне бачення взаємодії, структури, динаміки інтерфейсу, тематики оформлення тощо.

З погляду інформаційної архітектури, цифрове портфоліо виконує одночасно кілька функцій: інформативну, демонстраційну, репутаційну та комунікаційну. Інформативна функція полягає в поданні відомостей про кваліфікацію, досвід, навички, реалізовані проєкти, мови програмування, знайомство з бібліотеками, фреймворками та інструментами. Демонстраційна функція реалізується через інтерактивні приклади, навігаційні схеми, стилізацію та візуальні ефекти, які безпосередньо відображають професійні вміння. Репутаційна — у можливості підтвердження досягнень шляхом посилань на реалізовані проєкти, відгуків, сертифікатів, результатів тестувань тощо. Комунікаційна функція сприяє встановленню контакту з потенційними роботодавцями, колегами чи партнерами, завдяки формам зворотного зв’язку або інтеграції з професійними соціальними мережами.

Застосування цифрового портфоліо охоплює як статичні формати (PDF-документи, скриншоти), так і динамічні — односторінкові сайти, веб-додатки з інтерактивним наповненням, мобільні версії презентаційних середовищ. Серед останніх особливої уваги заслуговують SPA (Single Page Application), які забезпечують безперервну навігацію, високий рівень інтерактивності, зменшення часу завантаження та плавність переходів. Це дозволяє користувачу фокусуватися на змісті, а не на очікуванні або перевантаженні інформацією. Сутність цифрової самопрезентації полягає в тому, щоб у стислій формі, але з використанням доступних і ефективних технологічних засобів, донести інформацію про фахівця до цільової аудиторії. У випадку IT-розробників така форма самопрезентації повинна поєднувати технічну точність, візуальну виразність, структурованість даних та гнучкість подання. Це зумовлює потребу у створенні засобів, що виходять за межі типових шаблонів. Індивідуальні рішення дозволяють адаптувати формат під особисті особливості, стиль, технічну базу, а також відповідати вимогам певних ринків або напрямів у галузі розробки. Варто також зазначити, що цифрове портфоліо не є виключно презентаційним засобом. Воно також виконує функції професійного щоденника,

історії проєктів, порталу для демонстрації технологічних рішень, навіть платформи для обміну знаннями або відкритої документації. Особливо коли йдеться про повноцінні веб-додатки, реалізовані власноруч, що містять приклади коду, посилання на репозиторії, технічні описи реалізації, інтеграції з API, засоби керування станами та маршрутизацію.

Аспект візуалізації відіграє ключову роль у створенні ефективного цифрового портфоліо. Колірна палітра, типографіка, структура сторінки, анімаційні ефекти, інтерактивні блоки — усе це створює перше враження, яке часто визначає подальше зацікавлення. У frontend-середовищі очікування користувача стосуються саме якості візуального оформлення, що створює вимоги до дизайну на рівні професійних UI/UX-підходів. Необхідність дотримання принципів зручності, інтуїтивності та адаптивності є важливою складовою технічного завдання розробника.Зі зростанням підтримки WebGL у браузерах, а також появою бібліотек на кшталт Three.js, Babylon.js та A-Frame, стало можливим впровадження тривимірної графіки у веб-середовище без потреби використання сторонніх плагінів. Це дозволяє створювати віртуальні сцени, анімовані об'єкти, середовища з глибокою перспективою, які доповнюють візуальний образ і підкреслюють технічну майстерність розробника. Із використанням @react-three/fiber стає можливим об'єднання екосистем React і Three.js, що значно спрощує процес інтеграції 3D-сцен у структуру компонентного застосунку.

Самопрезентація через тривимірні засоби дозволяє фахівцю не лише виділитися візуально, а й продемонструвати знання в галузі графіки, математики, обробки подій, а також вміння синхронізувати різні середовища взаємодії. Наприклад, реалізація поведінкової логіки об'єкта на основі подій pointerMove або scroll може засвідчити обізнаність із концепцією реактивного програмування, роботою з анімаційними циклами, ефектами розгортання та оптимізації рендерингу.

Іншим важливим чинником є адаптивність. Цифрове портфоліо має функціонувати однаково ефективно на широкому спектрі пристроїв: від смартфонів до широкоформатних моніторів.

Це створює додаткові вимоги до структури, розмітки, стилізації, логіки завантаження ресурсів. Tailwind CSS як утилітарний CSS-фреймворк забезпечує швидку побудову адаптивного інтерфейсу з контролем поведінки елементів на різних брейкпоінтах, що особливо корисно в контексті портфоліо, де кожна секція виконує специфічну роль.Контактна складова також є важливою частиною системи цифрової самопрезентації. Вбудовані форми зворотного зв'язку, інтеграція з поштовими сервісами (EmailJS), месенджерами або соціальними платформами дозволяють швидко ініціювати контакт між користувачем і власником портфоліо. Це збільшує шанси на отримання зворотного зв'язку й формує прямий канал комунікації без потреби використання посередницьких сервісів.

Цифрова самопрезентація у вигляді динамічного застосунку є не лише інструментом для фрилансера чи шукача роботи, а й ефективним елементом персонального брендингу. Вона закріплює у свідомості відвідувача асоціацію між технічною майстерністю й візуальною якістю реалізації. Завдяки цьому цифрове портфоліо перетворюється на платформу, що представляє не просто факти про досвід і навички, а спосіб мислення, підхід до розв'язання завдань, здатність організувати проект і подати його грамотно.

Підсумовуючи, предметна галузь цифрової самопрезентації розробника охоплює створення програмних рішень, що виконують функції професійного подання, технічної демонстрації, комунікації та репутаційного позиціонування. Ефективність таких рішень визначається рівнем інтеграції актуальних технологій, візуального дизайну, зручності використання й загального користувацького досвіду. Індивідуальне цифрове портфоліо, побудоване на сучасних веб-технологіях, є не лише способом привернути увагу до фахівця, а й прикладом його технічних умінь у реальному застосуванні.

1.2. Огляд існуючих рішень та платформ для створення портфоліо

Створення персонального портфоліо стало поширеним інструментом для самопрезентації фахівців у різних галузях, зокрема у сфері веброзробки, дизайну, програмування, цифрового мистецтва та суміжних напрямків. Зважаючи на зростання попиту на візуальні, функціональні та інтерактивні засоби демонстрації професійного досвіду, з'явилася велика кількість онлайн-платформ та шаблонних рішень, орієнтованих на створення цифрових портфоліо. Варто детально проаналізувати ключові з них, їх функціональні можливості, гнучкість налаштувань, доступність для розробників і рівень відповідності потребам фахівців frontend-напрямку.

Серед найпоширеніших рішень умовно виділяють три великі групи:

1. спеціалізовані платформи для портфоліо,
2. конструктори веб-сайтів із шаблонами портфоліо,
3. самостійні вебзастосунки, створені розробниками з нуля або з використанням фреймворків.

До першої групи належать такі ресурси як Behance, Dribbble, GitHub Pages, Carbonmade, Coroflot, Adobe Portfolio. Їх ключовим призначенням є розміщення прикладів робіт з можливістю структурування за категоріями, додавання описів, тегів і візуального представлення через сітку або слайдери. Вони активно використовуються дизайнерами, художниками, UI/UX-фахівцями та почасти frontend-розробниками.Behance є продуктом компанії Adobe й орієнтований на креативні професії. Користувач може додати проєкт, розмістити зображення, відео, короткий опис, вказати використані технології, а також отримувати оцінки й коментарі від інших користувачів. Водночас, Behance не дає змоги створити інтерфейс, що повністю відповідає особистому стилю розробника або відображає структуру його коду чи логіку роботи SPA-додатку. Усе обмежено певним шаблонним підходом і стилістикою платформи.Dribbble має подібну логіку, але візуальний акцент зміщено на мініатюрні візуальні елементи (shot'и), що дозволяють швидко переглянути приклади UI або

графіки. Незважаючи на свою популярність серед дизайнерів, ця платформа недостатньо ефективна для розробників, які хочуть представити логіку інтерфейсу, структуру коду або інтерактивну поведінку елементів. Тут неможливо реалізувати кастомні анімації, динамічні взаємодії чи 3D-сцени.GitHub Pages дозволяє публікувати статичні сторінки безпосередньо з репозиторіїв GitHub. Це надає більше свободи розробнику, адже дозволяє вручну створювати сайт, розміщуючи його прямо у віртуальному середовищі. Однак обмеженням стає відсутність серверної частини, підтримки динамічних інтеграцій і потреба в самостійному адмініструванні кожного елемента. Водночас, GitHub Pages широко використовується для хостингу портфоліо, особливо серед студентів та молодих фахівців.Carbonmade і Coroflot орієнтовані на дизайнерів, і хоч надають певний рівень візуальної налаштовуваності, але не дозволяють працювати з кодом або реалізовувати індивідуальні функціональні блоки. Вони мають зручну панель керування, але в основному призначені для швидкого старту без потреби глибокого технічного втручання.

У другу групу платформ - конструктори сайтів - входять Wix, Tilda, Webflow, Squarespace, WordPress. Ці сервіси дають змогу створювати повноцінні сайти без або з мінімальним програмуванням. Вони містять шаблони, які адаптуються до різних цілей, зокрема створення портфоліо. Часто користувач має змогу редагувати шрифти, кольори, розміщення елементів, додавати інтеграції з Google-аналітикою, CRM-системами, формами зворотного зв’язку.Tilda відзначається простотою створення лендингових сторінок. Платформа підтримує адаптивну верстку, дозволяє інтегрувати скрипти, але має обмеження у створенні складних анімацій, 3D-об'єктів або логіки взаємодії з динамічними API. У випадку frontend-розробника, який прагне показати рівень роботи з React, Three.js або Tailwind CSS, Tilda виявиться недостатньою.Webflow має більший потенціал з точки зору візуальної деталізації та анімації. Він дозволяє створювати складні взаємозв'язки між елементами, використовувати умовну видимість, тригери, а також створювати динамічні сторінки на основі CMS-контенту. Проте, подібно до інших конструкторів, Webflow не дає

змоги розробнику на повну реалізувати логіку SPA, побудовану на компонентній структурі з використанням React або його бібліотек.Wix і Squarespace мають зручний інтерфейс для початківців, проте мають чітко обмежені рамки кастомізації. Рівень інтерактивності обмежено готовими віджетами. Це робить їх зручними для загальних потреб, проте марними з точки зору демонстрації програмної архітектури чи логіки побудови інтерфейсу.

Окрему категорію становлять платформи, що орієнтовані на розробників. Тут варто згадати такі рішення, як Dev.to, Hashnode, Notion, Obsidian, які дозволяють вести блоги, структурувати проєкти, створювати системи знань або онлайн-документацію. Вони не є платформами для портфоліо у традиційному сенсі, але активно використовуються для підкріплення репутації, публікації кейсів і технічних розбірок. Наприклад, Hashnode дозволяє хостити персональний блог на власному домені, що часто поєднується з портфоліо-сайтом.

Водночас найширші можливості має третя група — індивідуально створені сайти, розроблені з нуля з використанням фреймворків та бібліотек. Найпоширенішими технологіями є React, Vue.js, Next.js, Gatsby, Svelte, а також підтримуючі засоби — Tailwind CSS, Framer Motion, React Router, Three.js, @react-three/fiber, Vite, Webpack, Zustand, Redux тощо. Цей підхід дає змогу створити сайт, який водночас є прикладом програмної роботи, візуального дизайну, логіки поведінки й знання фреймворків.У межах такого рішення розробник самостійно формує структуру проєкту, визначає маршрутизацію, організовує компоненти, керує станами, налаштовує стилізацію та анімацію. Ця гнучкість дозволяє реалізовувати не лише адаптивну верстку, але й складні 3D-сцени, анімовані об'єкти, інтерактивну взаємодію, зміну тем оформлення, багатомовність, інтеграцію з API, форму зворотного зв’язку та аналітичні сервіси.Наприклад, використання Three.js у зв’язці з React через @react-three/fiber дає змогу вбудувати у структуру компонентів тривимірну сцену, що взаємодіє з користувачем через події руху, кліку, наведення. Можна створити об’єкти з анімацією за допомогою useFrame, які реагують на стан додатку або позицію камери.

Це відкриває можливості не тільки для демонстрації візуальних ефектів, але й для вбудованої логіки на кшталт переходів між розділами при обертанні сцени, активації подій на основі взаємодії з 3D-об'єктами.Застосування Tailwind CSS забезпечує високий рівень контролю над стилями при мінімальному обсязі коду, що є корисним для швидкого розгортання інтерфейсу з динамічними класами. Інтеграція Framer Motion надає можливість створення плавних анімацій переходів, реакцій на hover, scroll, drag-and-drop події, що сприяє підвищенню інтерактивності портфоліо.

Порівнюючи зазначені підходи, можна зробити висновок, що лише самостійна розробка з використанням технологій frontend-розробки дозволяє повною мірою реалізувати цілі портфоліо-ресурсу для програміста. Це підтверджується як функціональною гнучкістю, так і можливістю точного відображення технічних навичок.

Для узагальнення даних розгляду представимо таблицю порівняльного аналізу:

Таблиця 1.1

Порівняння платформ

Платформа / Підхід	**Можливість кастомізації**	**Підтримка 3D / анімацій**	**Рівень технічного контролю**	**Підходить для фронтендера**
Behance / Dribbble	×	×	×	Обмежено
GitHub Pages	▲	▲	▲	Так, з обмеженнями
Tilda / Wix / Webflow	▲	▲	×	Частково
WordPress	▲	▲ (через плагіни)	×	Частково
Створення з нуля (React)	✔	✔	✔	Повністю

(✔ — повна підтримка; ▲ — часткова; × — відсутня)

У підсумку, огляд засвідчує, що незважаючи на велику кількість готових платформ, жодна з них не забезпечує достатнього рівня технічної демонстрації навичок для frontend-розробника. Для досягнення індивідуального результату, що поєднує візуальну привабливість, функціональність, інтерактивність та програмну логіку, доцільним є саме самостійне створення вебзастосунку на базі актуальних frontend-технологій.

1.3. Порівняльний аналіз інструментів і технологій (React, Three.js тощо)

Розробка інтерактивного односторінкового вебзастосунку, який містить тривимірні об’єкти, вимагає зваженого підходу до вибору технологій. Для досягнення балансу між функціональністю, продуктивністю та візуальною виразністю необхідно враховувати не лише популярність рішень, а й глибину їх інтеграції, гнучкість конфігурації та сумісність між собою. Серед великої кількості сучасних інструментів особливу увагу привертають React, Three.js, Tailwind CSS, Vite, Framer Motion і React Router. Їхня сукупність становить ефективний стек, який дозволяє реалізувати SPA-додаток з інтерактивною графікою.React є компонентною бібліотекою для побудови інтерфейсів користувача, що базується на декларативному підході та віртуальному DOM. Завдяки цьому React забезпечує ефективне оновлення стану та високу продуктивність. Його основною перевагою є можливість створення ізольованих компонентів, які можна комбінувати, повторно використовувати та тестувати. Розвинена екосистема, численні додаткові бібліотеки та активна спільнота підтримки роблять цю технологію найпоширенішим вибором серед frontend-розробників. У контексті побудови портфоліо React дозволяє формувати структуру інтерфейсу як систему незалежних блоків, що істотно спрощує підтримку і масштабування коду.

Для реалізації тривимірної графіки у вебсередовищі доцільно використовувати бібліотеку Three.js, яка надає високорівневий інтерфейс для роботи з WebGL. Вона підтримує побудову геометрій, використання матеріалів, текстурування, освітлення,

анімацію, камери та імпорт складних моделей у форматах .glb і .fbx. Вбудовані можливості анімації й управління сценою дозволяють досягти бажаного рівня деталізації та інтерактивності. Тісна інтеграція з React можлива за допомогою обгортки @react-three/fiber, яка забезпечує декларативне керування 3D-сценами всередині JSX-компонентів. Це значно полегшує логіку керування поведінкою об’єктів у межах одного застосунку.Альтернативами до React є Vue, Angular і Svelte. Vue приваблює простим синтаксисом і низьким порогом входження, проте має менш розвинуту екосистему для складних інтерактивних додатків. Angular вирізняється жорсткою структурою і надлишковою складністю, що не виправдано у разі персонального проєкту з кастомною візуалізацією. Svelte, попри інноваційний підхід до компіляції компонентів, все ще не набув достатнього рівня стабільності та підтримки у великих проєктах. З огляду на це, React залишається найбільш раціональним вибором.Three.js має також конкуренцію у вигляді Babylon.js і A-Frame. Babylon.js пропонує більшу кількість вбудованих інструментів, однак її інтеграція з React вимагає складнішого зв’язування логіки та сцен. Вона більше орієнтована на створення повноцінних 3D-ігор, що у випадку цифрового портфоліо створює зайву надмірність. A-Frame, зі свого боку, фокусується на VR/AR-додатках та використовує декларативний HTML-подібний синтаксис, який обмежує рівень кастомізації. Водночас @react-three/fiber дає змогу безпосередньо керувати об’єктами сцени в рамках React-компонентів, що робить цей підхід найбільш гнучким і ефективним.

Стилізація інтерфейсу відіграє важливу роль у створенні естетично привабливого, зручного і сучасного застосунку. Tailwind CSS є утилітарним CSS-фреймворком, який дозволяє створювати стилі без написання окремих класів у CSS-файлах. Він надає сотні попередньо визначених класів, що дозволяє розробнику безпосередньо в HTML або JSX-коді формувати зовнішній вигляд елементів. Це забезпечує швидкість розробки, компактність стилів та високу кастомізованість. Альтернативні рішення, такі як Bootstrap або Material UI, не дозволяють досягти такого рівня контролю над зовнішнім виглядом. Bootstrap має фіксовану структуру

компонентів і часто призводить до шаблонного вигляду, а Material UI реалізує стилізацію за принципами Material Design, які не завжди відповідають індивідуальним дизайнерським потребам.У контексті анімації користувацьких елементів Framer Motion вважається одним із найкращих інструментів для React. Він забезпечує декларативний синтаксис, просту інтеграцію, а також підтримує складні сценарії анімації, включно з drag-and-drop, layout transitions і presence-менеджментом. Інші бібліотеки, як-от React Spring або GSAP, мають власні переваги, але вимагають більш складної логіки конфігурації та не так добре поєднуються з компонентною структурою React. Framer Motion є оптимальним для інтерактивного портфоліо, де потрібні швидкі, але гнучкі анімації на події hover, tap, scroll тощо.Для маршрутизації у межах SPA React Router забезпечує повноцінну підтримку вкладених маршрутів, динамічних параметрів, умовної візуалізації компонентів і зручного керування переходами між сторінками. Альтернативні рішення не мають аналогічного рівня функціональності або вже застаріли. Завдяки інтеграції з useLocation і useNavigate, React Router дозволяє ефективно контролювати потік користувацької взаємодії у межах односторінкового застосунку.

Реалізація розробницького середовища та процесу збирання коду суттєво впливає на швидкість і ефективність роботи. Vite — сучасний інструмент, що забезпечує моментальне завантаження застосунку в режимі розробки завдяки використанню нативних ES-модулів. На відміну від Webpack, який потребує складної конфігурації і довгого часу збирання, Vite дозволяє зосередитися на розробці, не витрачаючи ресурси на конфігураційні обхідні шляхи. Parcel, хоч і є простим, поступається Vite за гнучкістю і продуктивністю в великих проєктах. Таким чином, вибір Vite як інструмента складання забезпечує швидке тестування змін, зручну інтеграцію з Tailwind CSS і ефективне збирання у фінальний продакшн-бандл.Синергія згаданих інструментів створює технологічну основу для реалізації портфоліо, яке не лише відображає технічні навички, а й є прикладом цих навичок у дії. Компонентна архітектура дозволяє реалізувати розділи "Про себе", "Проєкти",

"Контакти" та інтегрувати їх із головною 3D-сценою. Візуальне середовище формує цілісне враження, яке є результатом гармонійної взаємодії графіки, стилів, анімації та логіки.

Загалом, обраний стек демонструє високий рівень технічної узгодженості та ефективності. Кожен компонент цього рішення виконує конкретну роль: React відповідає за структуру, Three.js за графіку, Tailwind CSS — за стилі, Framer Motion — за анімацію, Vite — за швидкодію, а React Router — за маршрутизацію. Їх поєднання дозволяє реалізувати динамічний і технічно насичений застосунок, що відповідає найвищим стандартам у сфері frontend-розробки.

Таблиця 1.2

Порівняльний аналіз інструментів для реалізації SPA-додатку з 3D-графікою

Категорія технології	**Назва інструменту**	**Переваги**	**Недоліки**	**Доцільність використання**
Фреймворк інтерфейсу	React	Компонентність, велика екосистема, гнучкість, JSX, підтримка хуків	Вимагає налаштувань, складніший старт для початківців	Висока
Фреймворк інтерфейсу	Vue	Простий синтаксис, швидкий старт	Менша екосистема, обмежена підтримка 3D-інтеграцій	Середня
Фреймворк інтерфейсу	Angular	Повний стек, вбудовані модулі	Громіздкий, перевантажений для невеликих проєктів	Низька
Фреймворк інтерфейсу	Svelte	Висока швидкодія, компіляція в чистий JS	Менша підтримка, новизна	Середня
3D-графіка	Three.js + R3F	Повний контроль, інтеграція з React, підтримка .glb, анімацій	Потребує розуміння графіки	Висока
3D-графіка	Babylon.js	Вбудовані GUI, фізика, VR	Ускладнена інтеграція з React	Середня

Продовження таблиці 1.2

Порівняльний аналіз інструментів для реалізації SPA-додатку з 3D-графікою

Категорія технології	**Назва інструменту**	**Переваги**	**Недоліки**	**Доцільність використання**
3D-графіка	A-Frame	Простий синтаксис, орієнтація на VR	Обмежена кастомізація, HTML-підхід	Низька
CSS-фреймворки	Tailwind CSS	Гнучкість, утилітарний підхід, простота адаптації	Неінтуїтивний для новачків	Висока
CSS-фреймворки	Bootstrap	Готові компоненти, швидкий старт	Обмеження в дизайні, однотипність UI	Середня
CSS-фреймворки	Material UI	Стандарти Google, React-компоненти	Складність кастомізації, шаблонний вигляд	Середня
Анімація	Framer Motion	Простота інтеграції, декларативність, анімація layout, hover, drag	Залежність від React	Висока
Анімація	GSAP	Деталізовані таймлайни, контроль	Складний синтаксис, не декларативний	Середня
Анімація	React Spring	Фізичні анімації, хуки	Більш складне налаштування, менше підтримки	Середня
Маршрутизація	React Router	Вкладені маршрути, параметри, useNavigate	Крива навчання для складних проєктів	Висока
Маршрутизація	Reach Router	Легкість, мінімалізм	Застарілий, частково інтегрований у React Router	Низька
Системи складання	Vite	Швидкість, HMR, мінімальні конфігурації	Відносно новий, менше legacy-плагінів	Висока
Системи складання	Webpack	Потужність, плагіни, гнучкість	Складність конфігурації, повільне збирання	Середня
Системи складання	Parcel	Простота для старту	Менше підтримки, складна інтеграція великих проєктів	Середня

1.4. Недоліки наявних підходів та обґрунтування новизни проєкту

Аналіз існуючих рішень для створення цифрових портфоліо демонструє широке розмаїття підходів — від шаблонних сервісів і платформ до індивідуально реалізованих односторінкових додатків. Проте, попри зовнішнє різноманіття, більшість із цих рішень мають спільні функціональні, технічні та концептуальні обмеження, що унеможливлюють повноцінну реалізацію портфоліо розробника нового покоління. У даному підрозділі систематизовано ключові недоліки існуючих підходів та обґрунтовано необхідність створення інноваційного рішення, яке ці недоліки долає.

Насамперед, слід зазначити, що більшість популярних платформ на кшталт Behance, Dribbble, GitHub Pages або Adobe Portfolio надають користувачам обмежений простір для налаштувань та кастомізації. Поведінка, дизайн та логіка розміщення матеріалів у таких системах визначається заздалегідь. Користувач має змогу лише наповнювати контентом існуючі блоки, не впливаючи на взаємодію між ними або на структуру самої презентації. У випадку професійного frontend-розробника, який прагне не лише продемонструвати результат, а й сам процес побудови інтерфейсу, це позбавляє змоги показати ключову компетенцію — створення повнофункціональних інтерфейсів з нуля.Іншим аспектом є відсутність підтримки тривимірної графіки у переважній більшості рішень. Системи, що базуються на шаблонах, не передбачають інтеграції з WebGL чи використання бібліотек для 3D-візуалізації. Як наслідок, портфоліо, створене за допомогою таких інструментів, виглядає статичним, одноманітним і не має високої залученості користувача. Зважаючи на сучасний рівень розвитку веб-технологій і доступність засобів для реалізації тривимірного контенту (Three.js, Babylon.js, A-Frame), така обмеженість видається штучною та застарілою. Широко розповсюджені конструктори вебсайтів, такі як Tilda, Wix, Webflow чи Squarespace, пропонують більшу гнучкість у плані дизайну, однак також мають структурні межі. Їх інтерфейси орієнтовані на візуальне редагування без знання коду,

що зручно для початківців, проте водночас унеможливлює реалізацію нетипових логік поведінки, анімації на події користувача, керування станами або впровадження тривимірних сцен. Крім того, складні взаємозалежності між елементами в таких конструкторах або відсутні, або реалізуються за допомогою обмеженого набору тригерів і шаблонів. Це суперечить основним принципам програмного проєктування, зокрема таким, як повторне використання компонентів, інкапсуляція логіки, розділення відповідальностей.

Інші недоліки стосуються продуктивності, адаптивності та швидкості взаємодії. Сайти, побудовані на готових платформах, часто мають надмірну вагу за рахунок загальних бібліотек, не оптимізованих для конкретного проєкту. Це призводить до уповільненого завантаження, що особливо критично для користувачів із мобільних пристроїв або повільним інтернет-з'єднанням. Водночас, мобільна адаптація в таких системах зазвичай реалізується як окремий шаблон, а не як результат гнучкого компонування, що обмежує масштабованість і подальшу модернізацію.

Особливо слід підкреслити слабкість шаблонних платформ у контексті демонстрації програмної логіки. У портфоліо frontend-розробника важливо не лише показати візуальний результат, а й пояснити, яким чином побудовано систему, як працюють компоненти, як реалізовано маршрутизацію, де використано анімації, як керується стан, які бібліотеки інтегровано та яку структуру має проєкт на рівні коду. Усі ці елементи залишаються невидимими або недоступними в межах готових систем, що значно знижує цінність таких рішень у професійному контексті.

Варто також звернути увагу на обмеження у сфері взаємодії з користувачем. Переважна більшість існуючих портфоліо не враховують можливості персоналізації досвіду — користувач не має змоги взаємодіяти з елементами, змінювати перспективу, запускати анімації або отримувати зворотний зв'язок у режимі реального часу. У результаті портфоліо не виконує функцію середовища взаємодії, а залишається лише колекцією посилань або зображень.

Недоліки традиційних рішень формують підґрунтя для обґрунтування новизни обраного проєкту. У межах цієї дипломної роботи реалізовано інтерфейс нового типу, що поєднує тривимірне середовище, інтерактивну логіку, анімаційні ефекти, компонентну архітектуру, адаптивність і персоналізацію. Кожен елемент портфоліо розроблений вручну за допомогою сучасних інструментів: React, Tailwind CSS, Three.js, @react-three/fiber, Framer Motion, Vite та інших. Завдяки цьому реалізація вийшла поза межі звичних шаблонів та наблизилася до поняття програмного середовища презентаційного типу.

Новизна полягає, передусім, у поєднанні кількох ключових функціональних елементів, які раніше не були реалізовані одночасно в межах одного проєкту. Зокрема, йдеться про головну сцену з інтерактивною 3D-анімацією, де об'єкти реагують на дії користувача; систему переходів між сторінками з використанням плавної анімації; контактну форму з можливістю інтеграції сторонніх сервісів; адаптивну логіку компонування; музику як фоновий супровід із керуванням з боку користувача. Ці компоненти об’єднані в одну SPA-структуру з використанням маршрутизації, управління станом, оптимізованого завантаження та модульного коду.

Іншою складовою новизни є концепція «портфоліо як продукт». Тобто портфоліо не лише демонструє виконані проєкти, але й само є прикладом програмного продукту, що відповідає професійним вимогам до якості, структури, архітектури та UX. Такий підхід формує новий тип взаємодії з цільовою аудиторією: роботодавець або клієнт не просто бачить результат, а занурюється в середовище, яке побудоване тими самими інструментами, які фахівець використовує у своїй повсякденній роботі.

Новизна підтримується також реалізацією індивідуального дизайнерського рішення, де кожен компонент стилізовано відповідно до загальної концепції візуального образу розробника. Tailwind CSS дозволив побудувати унікальну стилізацію без залежності від шаблонів, а Framer Motion додав гнучкість анімацій та логіки реакції інтерфейсу. Тривимірні моделі, завантажені у форматі .glb, оживлені за

допомогою useFrame та useGLTF, що продемонструвало навички роботи з анімаційними циклами, оптимізацією сцени та подієвим керуванням.

Завдяки використанню Vite як системи складання забезпечено надзвичайно швидке розгортання проєкту, ефективну HMR-підтримку та низький час старту сервера, що істотно покращило процес розробки та налагодження. У сукупності всі ці рішення формують практичну новизну, яка полягає не лише в технологічному поєднанні, але й у створенні нового рівня цифрової презентації — інтерактивного, адаптивного, модульного, і водночас персоналізованого.

Реалізований проєкт також відповідає принципам розширюваності: структура дозволяє легко додати CMS або систему керування контентом через markdown-файли, реалізувати багатомовність, підключити аналітичні сервіси, додати функції авторизації, зворотного зв’язку або інтеграцію з базами даних. Це дає змогу розвивати продукт поза межами дипломного завдання, перетворюючи його на платформу для власного бренду або візуального резюме.

Підсумовуючи, можна стверджувати, що реалізоване рішення відповідає критеріям новизни через:

- поєднання нестандартних функціональних можливостей;
- застосування технологій, які не представлені у типовому портфоліо;
- створення системи, що є одночасно демонстрацією технічних знань і діючим продуктом;
- акцент на інтерактивну взаємодію, а не лише на візуальне подання;
- орієнтацію на масштабування та подальший розвиток.

Проєкт долає обмеження шаблонних підходів і пропонує новий формат представлення розробника — через персональне інтерактивне середовище, побудоване власноруч, зі збереженням як програмної, так і естетичної цілісності.

1.5. Постановка задачі, визначення об’єктів, предмета та мети

Проєктування та реалізація інтерфейсів користувача для цифрової самопрезентації фахівців, зокрема frontend-розробників, є одним із актуальних напрямів у межах прикладної веброзробки. Зростаючі вимоги до якості візуального подання, інтуїтивності взаємодії, інтерактивності та персоналізації формують нові виклики для створення інформаційних середовищ, які не лише транслюють зміст, а й демонструють технічні компетенції розробника через саму форму подачі. Постановка задачі в межах цієї дипломної роботи спирається на необхідність подолання недоліків, виявлених у типових системах цифрових портфоліо, та обґрунтування створення такого рішення, яке би не лише містило необхідний функціонал, але й саме по собі слугувало прикладом розробницької майстерності.Вихідною позицією для формулювання задачі є виявлений розрив між реальними можливостями сучасних вебтехнологій та їхнім застосуванням у сфері персональної презентації фахівців. Більшість існуючих платформ, незалежно від їхньої орієнтації на дизайнерів чи програмістів, не забезпечують достатнього простору для вираження складних логік взаємодії, динамічної поведінки інтерфейсних компонентів або тривимірної візуалізації. У результаті, цифрове портфоліо розробника часто зводиться до статичної вітрини, що не передає реальний рівень компетенції у побудові сучасних вебінтерфейсів.

Потреба в автономному цифровому середовищі, яке би дозволяло повністю контролювати структуру, вигляд, поведінку, маршрутизацію, анімацію, тривимірні сцени, інтерактивність і адаптивність, стала ключовим фактором при постановці задачі цього проєкту. Йдеться не про використання шаблонів чи платформ, а про розробку повноцінного односторінкового застосунку (SPA), який був би результатом

ручного проєктування, кодування, тестування та оптимізації.Підхід до формулювання задачі базується на необхідності об'єднати технічні та естетичні вимоги в єдиному інформаційному продукті, який буде не лише засобом комунікації, але й інструментом оцінки професійного рівня. Цей підхід передбачає поділ задачі на низку підзадач, кожна з яких реалізує окремий аспект функціонування системи. Йдеться про побудову архітектури додатку, розробку компонентів інтерфейсу, проєктування тривимірної сцени, інтеграцію бібліотек для анімації, реалізацію маршрутизації, стилізацію відповідно до заданої візуальної концепції, забезпечення адаптивної поведінки на різних пристроях, а також створення контактної форми.Крім того, задачі, пов’язані із взаємодією користувача з додатком, передбачають імплементацію логіки поведінки елементів на основі подій: наведення курсора, натискання, руху камери, обертання сцени, перемикання анімаційних станів. Це означає, що система має підтримувати реактивне оновлення станів, що змінюється у відповідь на дії користувача, й водночас залишатися стабільною та продуктивною.

Під час постановки задачі важливо враховувати вимоги до архітектурної цілісності. Кожен компонент системи повинен бути взаємозв’язаний з іншими через чітко визначені API, дотримуватися принципів розділення відповідальності та мінімізації залежностей. Компонентна архітектура має бути організована в такий спосіб, щоб розширення або заміна частини функціоналу не вимагали радикального перегляду всієї системи. Цей принцип дозволяє підтримувати проєкт у майбутньому та інтегрувати нові можливості — від аналітики користувацьких дій до підключення сторонніх сервісів для комунікації.Із точки зору семантичного змісту, додаток повинен містити ключові розділи портфоліо: коротку інформацію про автора, перелік навичок, приклади проєктів, контактну форму. Водночас, кожен розділ має не лише бути інформаційно наповненим, але й демонструвати технологічні особливості реалізації — ефекти прокручування, анімації входу/виходу, тривимірні інтеракції. Зокрема,

центральною частиною додатку є 3D-сцена, яка динамічно реагує на зміну стану системи, дозволяє обирати напрями навігації та водночас візуалізує основний образ портфоліо як віртуального середовища.У зв’язку з поставленою задачею доцільно окреслити базові характеристики системи: інтерактивність, адаптивність, модульність, ефективність, масштабованість, естетична відповідність задуму. Реалізація таких характеристик неможлива без залучення відповідних технологій, що стало підставою для формування відповідного технологічного стеку, який поєднує React, Three.js, Tailwind CSS, Vite, Framer Motion та інші допоміжні інструменти.Визначення об’єкта дослідження здійснюється на основі загального характеру діяльності, яка підлягає автоматизації. У цьому випадку мова йде про побудову процесу цифрової самопрезентації frontend-розробника з використанням вебтехнологій. Об’єктом виступає система комунікації між фахівцем і цільовою аудиторією, реалізована через інтерактивне інформаційне середовище.

Предмет дослідження має вужчий характер і пов'язаний із засобами реалізації такого середовища. Зокрема, йдеться про сукупність програмних компонентів, логік, бібліотек, моделей побудови інтерфейсів, анімаційних сценаріїв, що використовуються для створення інтерактивного SPA-додатку з тривимірним візуальним супроводом. У фокусі дослідження знаходяться способи організації архітектури, побудови компонентів, структурування логіки поведінки інтерфейсу, обробки подій та впровадження візуальних ефектів.

Метою цієї бакалаврської роботи є підвищення якості цифрової самопрезентації frontend-розробника шляхом створення інтерфейсу, який забезпечує ефективну демонстрацію технічних навичок за допомогою інтерактивної логіки, тривимірної графіки, анімаційних ефектів і гнучкої маршрутизації у межах SPA-додатку. Ця мета не зводиться до формального подання інформації, а передбачає створення такого

середовища, у якому сама структура й функціональність свідчать про рівень професіоналізму розробника.

Досягнення поставленої мети реалізується через ряд конкретних цілей. По-перше, необхідно дослідити актуальні інструменти веброзробки, які дозволяють реалізовувати інтерактивну графіку, зокрема на базі WebGL. По-друге, слід оцінити можливості кожного інструмента щодо побудови компонентної архітектури, гнучкого стилізування, швидкого білдування, продуктивного рендерингу та інтеграції з іншими бібліотеками. По-третє, варто сформувати інформаційну структуру додатку та спроєктувати 3D-сцену, яка стане центральним елементом інтерфейсу. По-четверте, слід реалізувати логіку взаємодії користувача з інтерфейсом, забезпечити плавність анімацій, реакцію на дії користувача та оптимізувати продуктивність.

У межах розробки потрібно також забезпечити адаптацію додатку до різних типів пристроїв, реалізувати просту навігацію, надати можливість швидкого доступу до ключової інформації та створити умови для зворотного зв’язку. Після завершення реалізації має бути проведено тестування, зокрема функціональне, візуальне, адаптивне й продуктивне, для верифікації відповідності розробленої системи заданим вимогам. Сформульована задача охоплює повний цикл створення програмного продукту — від постановки мети та аналізу технологій до проєктування архітектури, реалізації функціоналу, забезпечення візуальної складової та виведення продукту в готовий вигляд. Це дає змогу не лише виконати навчальні цілі, але й створити дієвий інструмент самопрезентації, який має реальну практичну цінність.

2. ПРОЄКТУВАННЯ ІНТЕРАКТИВНОГО WEB-СЕРЕДОВИЩА

2.1. Архітектура системи та загальна структура

Проєктування архітектури вебзастосунку є критичним етапом, який визначає структурну цілісність, логіку взаємодії модулів, рівень модульності та розширюваність. У межах розробки інтерактивного SPA-портфоліо для frontend-розробника було обрано багаторівневий підхід, що поєднує логічні, функціональні та графічні компоненти в єдину взаємопов'язану систему. Структура реалізована з урахуванням принципів розділення відповідальностей, повторного використання компонентів, ефективного керування станами та візуальної незалежності шарів.В основі архітектури лежить клієнтська модель, оскільки система не використовує серверної частини у класичному розумінні, а всі обчислення, рендеринг і взаємодія виконуються безпосередньо у браузері користувача.

Це забезпечує швидкий відгук, низький час завантаження та можливість автономного функціонування, зокрема в офлайн-режимі для ознайомлення з локального середовища. Ключовими складовими архітектури виступають: інтерфейсна частина, шар тривимірної графіки, структура компонентів проєкту, інструменти стилізації та анімації, а також зовнішні інтеграції.

Центральним елементом системи є файл App.jsx, який виконує роль «кореневого» компонента. Він відповідає за ініціалізацію маршрутизації, підключення глобальних стилів, обгортання 3D-сцен, логіку перемикання сторінок та інтеграцію анімацій. Компонент App.jsx взаємодіє з React Router, що забезпечує SPA-структуру та дозволяє реалізовувати навігацію між сторінками без перезавантаження. Усі інші компоненти вбудовуються в нього як вкладені елементи згідно з обраними маршрутами.

Для стилізації застосунку використано Tailwind CSS, що імпортується через index.css. Tailwind дозволяє визначати стилі безпосередньо в JSX-компонентах завдяки

системі класів-утиліт. Такий підхід скорочує час розробки, забезпечує узгодженість дизайну та гнучку адаптацію інтерфейсу до різних брейкпоінтів. Основні стилі завантажуються у файлі main.jsx, який виконує роль точки входу у програму. Тут також відбувається монтування додатку до DOM-елемента та ініціалізація контексту, якщо він використовується.На рівні 3D-графіки реалізовано окремий шар, який побудований на основі бібліотеки Three.js. Для інтеграції з React використовується @react-three/fiber — спеціалізована обгортка, яка дозволяє описувати тривимірну сцену у вигляді JSX-компонентів. Це дає змогу безпосередньо вбудовувати 3D-об'єкти у дерево компонентів React, керувати їх станами, обробляти події й синхронізувати з іншими частинами інтерфейсу. Додатково використано @react-three/drei — набір утиліт і готових компонентів, які спрощують роботу з камерами, світлом, завантаженням моделей і контролерами навігації. У сукупності ці бібліотеки формують WebGL-шар системи, який відповідає за рендеринг динамічної сцени у реальному часі.

Тривимірна сцена містить кілька об’єктів — будинок, літак, птах, лис, острів, небо — кожен з яких представлений окремою моделлю, що зберігається в директорії models/. Завдяки використанню GLTFLoader (опосередковано через drei), моделі завантажуються з локальних ресурсів, після чого розміщуються в просторі сцени згідно з обраними координатами. Усі моделі підтримують анімацію: птах і лис мають кілька анімаційних станів, що керуються за допомогою useFrame та useRef. Рухи синхронізовані з діями користувача, наприклад, зміною кута огляду або взаємодією з інтерфейсними кнопками.Усі графічні компоненти взаємодіють із директорією components/, у якій містяться UI-елементи (наприклад, кнопки, поля вводу, картки проектів, заголовки), а також складніші контейнери для виведення логіки навігації, секцій з контентом, модальних вікон. Структура компонентів збудована за принципом атомарного дизайну: спочатку створюються прості компоненти, які потім об’єднуються у складніші блоки, придатні до повторного використання.Компоненти отримують дані з директорії constants/, де зберігаються

масиви зі списками навичок, проєктів, посилань, текстів секцій. Це дозволяє відокремити логіку від структури даних, забезпечуючи зручність масштабування. Директорія assets/ містить графічні ресурси — зображення, текстури, шрифти, музичні файли — що використовуються у тривимірній сцені або у візуальному інтерфейсі. У директорії pages/ розміщені логічно ізольовані сторінки: About, Projects, Contact, кожна з яких відповідає за свій контент і має власну структуру компонентів.
Для реалізації анімацій обрано Framer Motion, що дозволяє задавати ефекти появи, зникнення, переходів між сторінками та внутрішніх рухів елементів. Бібліотека працює декларативно, інтегрується з React та підтримує складні сценарії — у тому числі layout transition та motion variants. Анімації використовуються як у 2D-інтерфейсі (наприклад, hover на кнопках, плавне прокручування), так і в управлінні об'єктами 3D-сцени (наприклад, зміна положення камери, розворот острова або запуск анімації літака).

У проєкті передбачено зовнішні інтеграції, які підключаються через окремий функціональний блок. Це, зокрема, EmailJS, що забезпечує надсилання повідомлень через форму зворотного зв’язку, без потреби створення backend-сервера. Форма "Contact Form" розміщується в окремому компоненті, отримує дані від користувача, виконує перевірку введених значень та надсилає запит до EmailJS. Друга інтеграція — аудіосупровід, що реалізований як модульний компонент із можливістю ввімкнення/вимкнення. Користувач може персоналізувати досвід взаємодії з додатком, що створює атмосферність та підвищує емоційне сприйняття.Архітектура також передбачає можливість розширення. Структура директорій дозволяє підключити CMS або зчитування даних із markdown-файлів. Модулі можна інкапсулювати за допомогою контекстів або кастомних хуків, що вже частково реалізовано у директорії hooks/. Це забезпечує дотримання принципів DRY (Don’t Repeat Yourself) і SoC (Separation of Concerns).

На Рис. 2.1, що додається до цього пункту, представлено логічну архітектурну схему системи. У лівій частині зображено підсистему тривимірної графіки, що

побудована на основі Three.js, @react-three/fiber та @react-three/drei. Ці компоненти пов’язані з основним компонентом App.jsx, який є ядром додатку. У центральній частині схеми показано структуру клієнтського інтерфейсу — React, React Router, Tailwind CSS, Framer Motion — які взаємодіють із App.jsx та директорією components/. У нижній частині наведено логічні блоки структури проєкту: models/, pages/, constants/, assets/, hooks/, які підключаються до компонентів. У правій частині зображено блок функціональних інтеграцій: EmailJS і Аудіосупровід, які взаємодіють із формою Contact Form. Додатково зображено стрілку, що вказує на функцію рендерингу 3D-сцени з камерою, світлом, об’єктами та маршрутизацією.

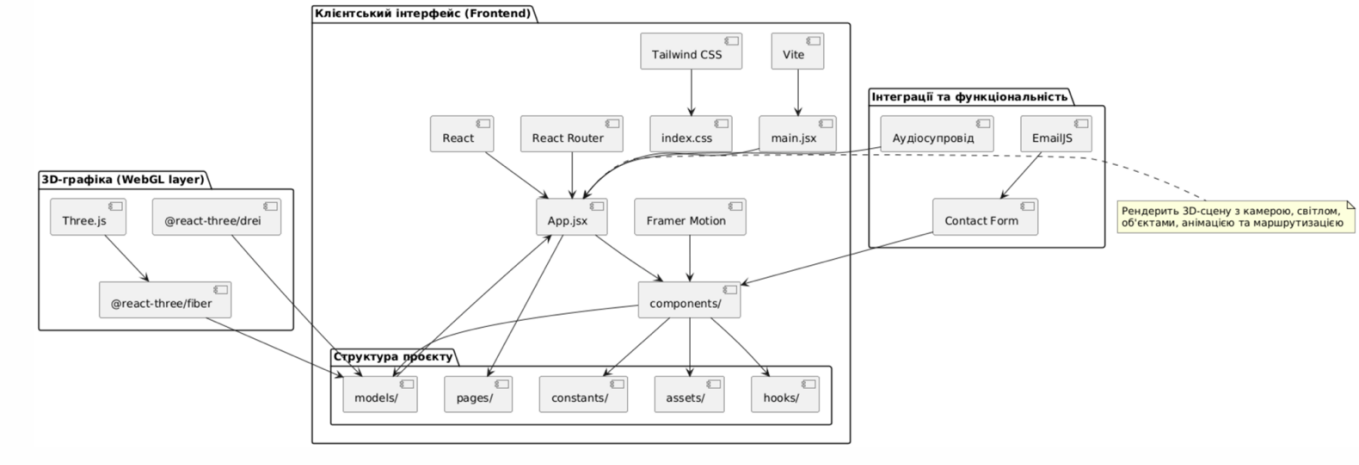


Рис. 2.1 Архітектура системи

Загалом, архітектура побудована за принципами модульності, ізоляції відповідальностей, логічного поділу структури та підтримки декларативного стилю програмування. Кожна частина проєкту є самодостатнім блоком, який може бути оновлений або замінений без порушення загальної логіки додатку. Такий підхід не лише спрощує підтримку, але й забезпечує гнучкість для подальшого розширення функціоналу. У межах одного додатку поєднано засоби візуального оформлення, логіку взаємодії, засоби обробки подій, а також графічну 3D-підсистему, що забезпечує вражаючий візуальний та технологічний результат.

2.2. Моделювання вимог до функціоналу та інтерфейсу

Розширене моделювання вимог до функціоналу та інтерфейсу вебзастосунку 3D-портфоліо передбачає не лише декларативне визначення ключових сценаріїв взаємодії користувача, але й детальне осмислення ролей, зв'язків між компонентами, можливих виняткових ситуацій, гнучкості інтерфейсного рівня та відповідності вимогам до зручності користування. У центрі цієї взаємодії — користувач, що взаємодіє з візуально динамічним середовищем, яке поєднує традиційні текстові інтерфейси зі сценами тривимірної графіки та мультимедійними елементами.

Рис. 2.2 Use-case діаграма

Кожен варіант використання, позначений на діаграмі варіантів (Use Case), має під собою як технічну реалізацію, так і логічну мотивацію. Перегляд навичок — це не просто візуалізація іконок; це динамічна реакція на дію користувача, яка синхронізується з внутрішньою структурою масивів даних. У той же час перегляд головної сцени — це тривимірна композиція, яка оновлюється з урахуванням кута огляду, положення об'єктів, фону, а також із дотриманням фізики переміщення. Відтворення анімацій здійснюється у циклі useFrame(), а взаємодія об’єктів з користувачем відстежується через події pointerDown, pointerEnter та pointerMove.Вимкнення або ввімкнення музичного супроводу — окрема частина персоналізованої взаємодії. Реалізація передбачає інтеграцію з Web Audio API, керування відтворенням через кастомні кнопки та збереження поточного стану в localStorage. Користувач після першої взаємодії може автоматично отримати обране налаштування при повторному вході, що підвищує комфорт використання.Контактна форма, яка відображає ще один варіант використання, має критично важливу роль для взаємодії між автором портфоліо та потенційними замовниками. Вимоги до цієї функції передбачають обов'язкову валідацію на клієнтському боці, захист від XSS-ін'єкцій, повідомлення про статус запиту, а також можливість повторного надсилання при втраті з'єднання. Після заповнення форми дані передаються за допомогою EmailJS API, без використання серверної інфраструктури, що спрощує реалізацію без втрати функціональності.

Перегляд проєктів є інтерфейсною реалізацією динамічного виведення елементів масиву projects[], які містять метадані: назву, опис, стек технологій, зображення та посилання на GitHub. Галерея реалізується у вигляді адаптивної сітки карток, з lazy-loading зображень та модальними вікнами для детального ознайомлення. Для покращення продуктивності застосовується кешування зображень та попереднє завантаження найпопулярніших секцій при першій ітерації.Важливим елементом у діаграмі є роль адміністратора, яка хоч і не має окремого інтерфейсу в базовій реалізації, проте передбачає розширення додатку для внесення змін у вміст без

редагування коду. Оновлення навичок і контенту проєктів здійснюється через конфігураційні файли constants.js або JSON-ресурси. У перспективі ці функції можна винести у CMS (наприклад, Netlify CMS або Sanity), що дозволить змінювати інформацію навіть без участі розробника.

Узагальнюючи вимоги, можна згрупувати функціонал у кілька категорій. Перша — візуальна презентація, до якої належать головна сцена, навички, проєкти. Друга — взаємодія користувача: контактна форма, перемикач музики, навігація. Третя — адміністративний функціонал: оновлення контенту та навичок. Четверта — системні вимоги, які стосуються адаптивності, продуктивності, безпеки, доступності (WCAG 2.1). Зокрема, адаптивність інтерфейсу передбачає підтримку всіх основних брейкпоінтів: мобільні телефони, планшети, ноутбуки, монітори з високою роздільністю. Кожен блок має автоматично масштабуватись відповідно до розміру вікна браузера. Tailwind CSS забезпечує реалізацію цих вимог завдяки вбудованим класам sm:, md:, lg:, xl:.Продуктивність гарантується за рахунок використання Vite як збирача проєкту, оптимізації імпорту компонентів, лінійного рендерингу тривимірної сцени без зайвих draw calls, lazy-завантаження компонентів React. Основний скрипт має вагою до 1.5 МБ, що дозволяє завантаження за <1 секунду при середньому з’єднанні.Доступність (accessibility) забезпечується завдяки коректному застосуванню тегів aria-label, фокусуванню на полях введення, читабельним шрифтам, контрастній палітрі, можливості навігації за допомогою клавіатури. Усі зображення мають альтернативний текст, а анімовані компоненти — статуси для screen readers.

На завершення слід наголосити: модель варіантів використання, відображена на Рисунку 2.2, не є просто схемою ролей. Вона формує ядро логіки поведінки системи, що проектується. Виходячи з неї, структурується архітектура, розподіляються компоненти, визначаються ключові API та формуються очікувані реакції інтерфейсу. Кожен сценарій охоплює і технічну, і поведінкову частину, тому їх опрацювання є фундаментом для реалізації стабільного, масштабованого та функціонально завершеного 3D-портфоліо.

2.3. Проєктування тривимірного середовища та логіки взаємодії

Проєктування тривимірного середовища в межах реалізації цифрового портфоліо розробника передбачає створення цілісної сцени, яка є не лише візуальним тлом, але й функціональним інтерфейсом для взаємодії з користувачем. Така сцена повинна поєднувати в собі об'єкти, здатні до анімації, реагування на дії користувача, інтеракції з подіями DOM та синхронізації з внутрішнім станом додатку, реалізованого на React.Тривимірне середовище у цьому проєкті створено за допомогою стеку бібліотек Three.js, @react-three/fiber та @react-three/drei, які в сукупності забезпечують повноцінну підтримку WebGL-графіки у середовищі React. У центрі сцени розміщується головна структура — будинок, навколо якого відбувається логічне групування інших об'єктів: дерева, огорожа, літак, тварини, острів та небо. Всі об'єкти імпортуються у форматі .glb через useGLTF, позиціонуються за координатами та масштабуються з урахуванням відносних пропорцій сцени.Кожен об’єкт має анімаційні властивості та власну логіку відгуку на взаємодію. Наприклад, модель птаха рухається по заданій синусоїдальній траєкторії в межах трьох координат X, Y, Z. Положення обчислюється динамічно всередині циклу useFrame(), який дозволяє оновлювати стан сцени на кожен кадр. Такий підхід забезпечує плавність та органічність анімації без затримок. Птах виконує стандартну анімацію "Take 001", яка активується відразу після монтування компонента за допомогою useEffect. Для контролю руху використовується змінна стану, яка інкапсулює часовий параметр і коригується залежно від швидкості кадру.

Модель лиса реалізована з використанням кількох станів: "Idle", "Walk", "Run". Поточна анімація визначається змінною currentAnimation, яка оновлюється при певних подіях, наприклад, при натисканні або наведені курсора. Усі стани розділено на окремі канали анімації, які активуються з використанням AnimationMixer, що дозволяє плавне перемикання між станами без видимих збоїв або миготіння. Залежно від активного стану змінюється не тільки положення, а й напрямок руху.Літак у сцені

обертається по осі Y, при цьому логіка контролюється за допомогою прапорця isRotating, що зберігається в локальному стані компонента. Коли прапорець активовано, літак отримує плавну обертальну анімацію. Водночас користувач має можливість вручну зупинити або відновити анімацію натисненням певної клавіші або кліком на об’єкт. Такий сценарій поєднує у собі як візуальну привабливість, так і функціональність взаємодії.Небо (Sky) реалізується як об'єкт, що обертається безперервно по горизонтальній осі (Y). Це досягається шляхом додавання до кута повороту певного значення при кожному рендері кадру. Також можливо реалізувати зміну колірної температури або яскравості в залежності від стану дня, секції або зовнішніх тригерів.

Острів є логічною основою сцени: всі об’єкти, розміщені на ньому, мають координати, відносні до його позиції. Це дозволяє централізовано управляти сценою, обертати її або масштабувати, не порушуючи логіку розташування об'єктів. При натисканні стрілок клавіатури або при перетягуванні миші сцена може обертатись, змінюючи перспективу камери. Це досягається за допомогою обробки подій pointerDown, pointerMove, pointerUp та зміни кута огляду в об'єкті OrbitControls. Для реалізації камери застосовано PerspectiveCamera, яка надає реалістичну перспективу. Камера прив’язана до координат центру сцени, але має динамічну можливість зміни кута огляду залежно від взаємодії або зміни стану інтерфейсу. Наприклад, при відкритті секції "Про себе" камера може злегка зсунутись або нахилитись, щоб створити ефект глибини.Освітлення сцени включає в себе DirectionalLight — основне світло, яке забезпечує тіні, AmbientLight — м’яке фонове освітлення, та PointLight — джерела локального світла, які підкреслюють окремі об'єкти. Така комбінація дозволяє досягти глибини, контрасту та об'єму сцени без перевантаження.

Інтерактивність досягається шляхом призначення об'єктам властивостей onClick, onPointerEnter, onPointerLeave, onPointerDown. Наприклад, при наведені курсора на літак або лиса їхня прозорість або колір може змінюватися, що

сигналізує про активність. У подальшому це може бути доповнено текстовими підказками або аудіо супроводом.Під час проєктування логіки взаємодії також враховується залежність між подіями в сцені та станами компонентів інтерфейсу. Наприклад, при натисканні на певний об’єкт запускається функція navigate("/projects"), яка перемикає активний маршрут у SPA-додатку. При цьому сцена продовжує існувати у фоновому режимі, але візуально змінюється активна частина інтерфейсу.Кожен об’єкт сцени підключається як окремий React-компонент, який інкапсулює свою поведінку, анімацію, матеріали та геометрію. Це забезпечує модульність, повторне використання та ізольоване тестування. Для зручності управління всі об’єкти структуруються в директорії /models, де зберігаються імпортовані 3D-моделі у форматі GLB/GLTF, а також відповідні компоненти для їх рендерингу.Окрім того, реалізовано динамічне керування станами через глобальний контекст або хук useStore, що дозволяє синхронізувати взаємодію між об'єктами без потреби у пропсах. У разі затримки під час завантаження застосовується fallback-компонент — скелетон або обертаюче кільце. Для завантаження моделей застосовується Suspense та useLoader(GLTFLoader) або обгортка useGLTF() з пакету drei. У разі затримки під час завантаження застосовується fallback-компонент — скелетон або обертаюче кільце. Також застосовано useHelper() з drei для відображення gizmo-елементів (при потребі в дебагінгу) і Stats для аналізу FPS та навантаження на GPU.Інтеграція тривимірного середовища з іншими частинами інтерфейсу є ключовим завданням цього проєкту. Під час прокрутки, кліків або переходів між сторінками 3D-сцена зберігає послідовність, що створює ефект єдиного інтерактивного середовища. Користувач не просто переглядає розділи, а взаємодіє з ними через віртуальну оболонку.

Загалом, тривимірне середовище реалізоване як автономний візуальний модуль, який поєднує ефекти глибини, світла, руху та взаємодії. У сукупності з React-архітектурою воно забезпечує не лише інформативність, але й демонструє практичні

навички автора у роботі з тривимірною графікою, оптимізацією рендерингу, подієвим керуванням та побудовою інтерактивного інтерфейсу.

2.4. Вибір засобів реалізації та інформаційна структура проєкту

Розробка інтерактивного тривимірного портфоліо потребує обґрунтованого вибору технологічних засобів, здатних забезпечити одночасно високу продуктивність, гнучкість у розробці, а також підтримку адаптивної графіки та інтеграцій зі сторонніми сервісами. У процесі планування було проаналізовано низку сучасних фреймворків та бібліотек, після чого було сформовано стек технологій, який дозволяє побудувати повноцінний SPA-додаток з підтримкою 3D-графіки, маршрутизації, анімацій, валідації форм і гнучкого стилізування.Ключовим рішенням для побудови інтерфейсу було використання React. Ця бібліотека дозволяє ефективно управляти станом, будувати компоненти з високим рівнем повторного використання, а також дотримуватись принципів декларативного програмування. За допомогою React уся логіка поділяється на окремі ізольовані частини, кожна з яких відповідає за конкретну ділянку функціональності, що позитивно впливає на підтримуваність і масштабованість проєкту.

Для реалізації тривимірного середовища було обрано Three.js, як основу для роботи з WebGL. Оскільки стандартний синтаксис Three.js не враховує специфіки React-архітектури, у якості адаптера було застосовано @react-three/fiber, який дозволяє інкапсулювати тривимірні об’єкти у вигляді JSX-компонентів. Це спрощує інтеграцію з іншими React-компонентами, а також дає змогу управляти 3D-сценою за допомогою хуків і стейтів. Для пришвидшення розробки та уникнення рутинного коду також було застосовано бібліотеку @react-three/drei, яка містить набір утиліт для налаштування камери, освітлення, завантаження моделей і контролерів взаємодії.Стилізація інтерфейсу реалізована з використанням утилітарного CSS-фреймворку Tailwind CSS. Він дозволяє використовувати стилі безпосередньо в розмітці компонента, що значно

скорочує обсяг CSS-коду і підвищує консистентність візуальної мови всередині проєкту. Tailwind забезпечує гнучку адаптацію до різних екранів, високий рівень кастомізації теми та полегшує дотримання принципів responsive design. З метою оптимізації швидкості розробки і часу збірки застосовується інструмент Vite. На відміну від традиційного Webpack, Vite забезпечує миттєве оновлення модулів під час розробки та прискорене складання готового бандлу. Цей інструмент підтримує модульну структуру, налаштування за допомогою конфігураційного файлу, а також дозволяє працювати з TypeScript, JSX та CSS-модулями без додаткових налаштувань.

Для реалізації анімацій між сторінками, а також мікровзаємодій у рамках інтерфейсу, було обрано Framer Motion. Ця бібліотека дозволяє реалізовувати як прості ефекти появи чи зникнення, так і складні сценарії трансформації елементів з урахуванням фізичних параметрів. Завдяки сумісності з React і можливості використання декларативного синтаксису, Framer Motion добре інтегрується з існуючою структурою компонентів.Однією з вимог до функціоналу є наявність форми зворотного зв’язку, яка дозволяє користувачу залишити повідомлення розробнику. Для цього застосовується EmailJS — сервіс, який дозволяє надсилати поштові повідомлення безпосередньо з клієнтської сторони, без потреби розгортання серверної частини. Форма інтегрується через SDK, підтримує базову валідацію та обробку результатів зворотного запиту. У разі успішного надсилання користувач отримує підтвердження, а в разі помилки — відповідне повідомлення з поясненням.

Інформаційна структура проєкту сформована з урахуванням розділення відповідальностей між логікою, інтерфейсом, контентом і конфігурацією. Увесь проєкт поділено на декілька ключових папок, кожна з яких виконує окрему роль. Компоненти інтерфейсу розміщуються в окремому каталозі, який містить усі елементи, що використовуються для побудови візуальної частини системи. Кожен компонент є окремим файлом, який може бути повторно використаний у різних частинах застосунку. Основні сторінки, як-от інформація про автора, галерея проєктів і контактна форма, зберігаються у вигляді окремих логічних модулів. Це дозволяє

підтримувати чистоту структури та спрощує навігацію у проєкті.Окрема частина структури призначена для тривимірних моделей. Усі .glb-файли зберігаються централізовано, а для кожної моделі створюється окрема обгортка-компонент. Це дозволяє не лише інкапсулювати логіку завантаження, позиціонування та анімації, але й повторно використовувати її в інших контекстах, якщо буде потреба.Фіксовані масиви даних, такі як перелік навичок, опис проєктів або контактна інформація, зберігаються у вигляді об’єктів у спеціальній директорії, яка відповідає за constants. Завдяки цьому контент легко змінити без необхідності редагування основного JSX-коду, що підвищує зручність підтримки проєкту. Графічні ресурси, включно з іконками, зображеннями, аудіо та шрифтами, зберігаються в окремому сховищі, доступному для підключення через імпорти. Увесь застосунок функціонує в межах одного кореневого компонента, який відповідає за маршрутизацію, ініціалізацію глобальних стилів, завантаження сцени та синхронізацію між сторінками. Вхід у програму здійснюється через точку main.jsx, яка рендерить App у DOM. Власні хуки, які інкапсулюють логіку роботи з аудіо, формами або станом взаємодії користувача з об’єктами сцени, зберігаються в окремому модулі й підключаються лише там, де це необхідно.

Таким чином, вибір засобів реалізації та побудова структури проєкту були підпорядковані меті створення ефективного, інтуїтивно зрозумілого та гнучкого програмного середовища. Всі компоненти розміщено в межах логічно виділених блоків, що дозволяє легко орієнтуватись у кодовій базі, виконувати рефакторинг, додавати нові функції або адаптувати систему до нових платформ. Стек технологій не лише забезпечує реалізацію всього необхідного функціоналу, але й демонструє сучасний підхід до побудови складних frontend-рішень з використанням 3D-графіки та високої інтерактивності.

3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

3.1. Створення інтерфейсу з використанням React і Tailwind CSS

Створення інтерфейсу вебзастосунку, що поєднує сучасні технології візуалізації та інтерактивної взаємодії, починається з фундаментальної архітектурної платформи — React. У межах розробки цифрового 3D-портфоліо використання React виправдане його компонентною природою, яка дозволяє розділити весь інтерфейс на окремі ізольовані елементи, що взаємодіють між собою через чітко визначені пропси й стани. Це спрощує як початкову реалізацію, так і подальше масштабування, забезпечуючи контроль над логікою, зовнішнім виглядом, життєвим циклом елементів і реактивністю.Інтерфейс побудовано за принципами модульності та ізоляції. Кожен функціональний блок — чи то форма зворотного зв’язку, чи то блок навичок або карта проєкту — реалізовано як окремий функціональний компонент, який інкапсулює усю внутрішню логіку, стилі та поведінку. Такий підхід дозволяє легко змінювати окремі частини системи без ризику порушити загальну цілісність або виникнення конфліктів у стилях. Також це створює умови для розширення, коли нові компоненти можуть будуватися на основі вже створених, розширюючи їх функціональність через композицію.

Tailwind CSS, як утилітарний CSS-фреймворк, інтегрується з React шляхом вбудованих класів у JSX-розмітку. Це забезпечує швидке створення гнучких стилізованих компонентів, які відповідають принципам адаптивності, доступності та консистентності. Кожна частина інтерфейсу — від кнопок до повноекранних секцій — отримує точне стилістичне визначення, що робить структуру зрозумілою для підтримки та зміни. Усі стилі формуються безпосередньо у шаблоні компонента, що дозволяє зменшити потребу у зовнішніх CSS-файлах, спростити навігацію та скоротити кількість артефактів стилізації.Побудова інтерфейсу починається з

формування структури маршрутизації, яка дозволяє розділити логіку SPA на декілька логічно відокремлених сторінок: головна сцена, секція навичок, сторінка проєктів, форма контактів. React Router у цьому проєкті відповідає за визначення шляху та виведення відповідного компоненту в рамках базового компонента App. Усі сторінки мають єдиний layout, що передбачає наявність навігаційного меню, фону та анімованих переходів. Tailwind забезпечує адаптацію таких компонентів до різних розмірів екранів, дозволяючи точно контролювати положення, типографіку, кольори, тіні та відступи залежно від медіа-запитів.

Семантична структура інтерфейсу будується з урахуванням принципів доступності. Використання коректних HTML-елементів, а також aria-атрибутів, дозволяє взаємодіяти з додатком за допомогою клавіатури або скрінрідера. Це розширює потенційну аудиторію та забезпечує відповідність сучасним стандартам WCAG. Всі текстові блоки мають достатній контраст, елементи мають візуальне підтвердження фокусу, а взаємодія з формами супроводжується інформативними повідомленнями.Кольорова схема інтерфейсу базується на кастомізованій темі Tailwind, яка була налаштована згідно з дизайнерською концепцією проєкту. Це дозволяє уніфікувати вигляд елементів по всьому додатку, підтримуючи впізнаваність і візуальну цілісність. Вибір кольорів здійснювався з урахуванням не лише естетичних, але й ергономічних аспектів, таких як комфорт для очей під час тривалої взаємодії, читабельність тексту, ієрархія візуального контенту.

Типографіка була налаштована через Tailwind-шрифти, розміри тексту та висоту рядків. Заголовки, підзаголовки, абзаци і текст на кнопках мають узгоджену стилістику. Це дозволяє швидко сприймати структуру контенту, а також полегшує роботу з адаптацією під мобільні пристрої.Форми в додатку були реалізовані за допомогою кастомних компонентів, які включають поля вводу, текстові області та кнопки відправки. Tailwind дозволив забезпечити як візуальну адаптацію під всі типи екранів, так і інтуїтивну взаємодію: активний фокус, підказки, зміна кольору обводки при валідації тощо. Повідомлення про помилки реалізовано як умовні JSX-елементи,

які відображаються лише при порушенні логіки заповнення.Навігація між сторінками реалізована за допомогою компонентів Link з пакету react-router-dom. При натисканні відбувається миттєвий перехід до нового компонента без перезавантаження сторінки. Tailwind дозволяє стилізувати активний маршрут, підсвічуючи його кольором або змінюючи тінь.

Окрему увагу в інтерфейсі приділено кнопкам. Вони мають змінний стан: нормальний, наведений, активний, вимкнений. Завдяки Tailwind, всі ці стани легко контролюються через класи, без необхідності додаткового CSS. Елементи навігації та взаємодії мають плавні переходи, які забезпечують приємну реакцію системи на дії користувача.Позиціонування блоків реалізовано за допомогою Flexbox і Grid, які у Tailwind реалізуються за рахунок відповідних класів. Це дозволяє створити як вертикальні секції, так і адаптивні сітки, які змінюють свою структуру в залежності від розміру вікна. Наприклад, блоки з описами проєктів при ширині екрану понад 1024 пікселі відображаються в ряд по три, при ширині від 768 до 1024 — по два, а на мобільних — в одну колонку.Інтерфейсна частина підтримує ефекти переходу, які реалізуються через класи transition, duration, ease. Це дозволяє кожному елементу змінювати колір, розмір або положення з затримкою, що виглядає плавно та професійно. Крім того, Tailwind дозволяє реалізовувати темну тему або перемикач колірних схем.

У процесі реалізації кожен компонент ретельно тестувався на відповідність очікуваному дизайну, взаємодії та адаптивності. React Dev Tools дозволили перевірити життєвий цикл компонентів, а Tailwind CLI — згенерувати мінімалістичний та ефективний CSS-файл, який не містить зайвих класів.Завдяки поєднанню React і Tailwind CSS вдалося створити інтерфейс, що поєднує логіку, естетику, адаптивність і швидкодію. Компонентна модель дозволила побудувати інтерфейс з високою гнучкістю, а Tailwind CSS — забезпечити візуальну узгодженість і легкість масштабування. Усі реалізовані рішення демонструють сучасний підхід до проєктування вебінтерфейсів, орієнтований на UX та UI практики найвищого рівня.

3.2. Інтеграція 3D-графіки: використання Three.js та @react-three/fiber

Інтеграція тривимірної графіки у вебінтерфейс на базі React є технічно складним, але стратегічно доцільним кроком у побудові сучасного візуального досвіду. У межах проєкту 3D-портфоліо основним завданням було створення такої сцени, яка б не лише виконувала естетичну функцію, але й забезпечувала інтерактивну поведінку та просторову логіку взаємодії. Для досягнення цієї мети було обрано бібліотеку Three.js як головний рендеринг-движок та @react-three/fiber як засіб декларативної інтеграції з екосистемою React.Three.js надає повноцінний доступ до всіх можливостей WebGL, зокрема, управління камерами, матеріалами, текстурами, освітленням, тінями, анімаціями та геометрією. Його використання дозволяє розробнику працювати на нижчому рівні з графічним API браузера без необхідності писати шейдери вручну. Водночас, взаємодія з чистим Three.js у проєктах на React створює конфлікт у стилях написання коду, тому застосування @react-three/fiber дозволяє описувати усю 3D-сцену у вигляді звичного JSX. Це забезпечує не лише стильову узгодженість з іншими частинами додатку, але й значне спрощення логіки зв’язку між графічними об’єктами та станами React-компонентів.

Початкове налаштування сцени базується на ініціалізації елемента Canvas, що реалізується як кореневий обгортковий компонент із пакета @react-three/fiber. Саме в межах цього компонента реалізується сцена, камера, освітлення та всі моделі. Усі ці елементи — не окремі графічні блоки, а частини єдиного React-дерева, до якого можуть бути застосовані хуки, контексти та умови рендерингу.Візуальні 3D-об’єкти зберігаються у вигляді .glb-файлів, які були створені або адаптовані з відкритих джерел. Їх імпортування здійснюється за допомогою хука useGLTF із пакету drei, який спрощує роботу з GLTFLoader. Моделі використовуються не як інертні зображення, а як повноцінні об’єкти, до яких можна прив’язати анімації, події, перемикачі станів, поведінку на натискання або наведення.Кожна модель винесена в окремий React-компонент, який інкапсулює її геометрію, матеріали, положення у просторі та логіку

інтерактивності. Наприклад, для моделі птаха створено компонент Bird, в якому реалізовано рух по синусоїдальній траєкторії за допомогою useFrame — хука, що дозволяє оновлювати стан у кожному кадрі. Координати X, Y, Z обчислюються динамічно з урахуванням часу, що дозволяє досягти плавного руху, незалежно від частоти кадрів.

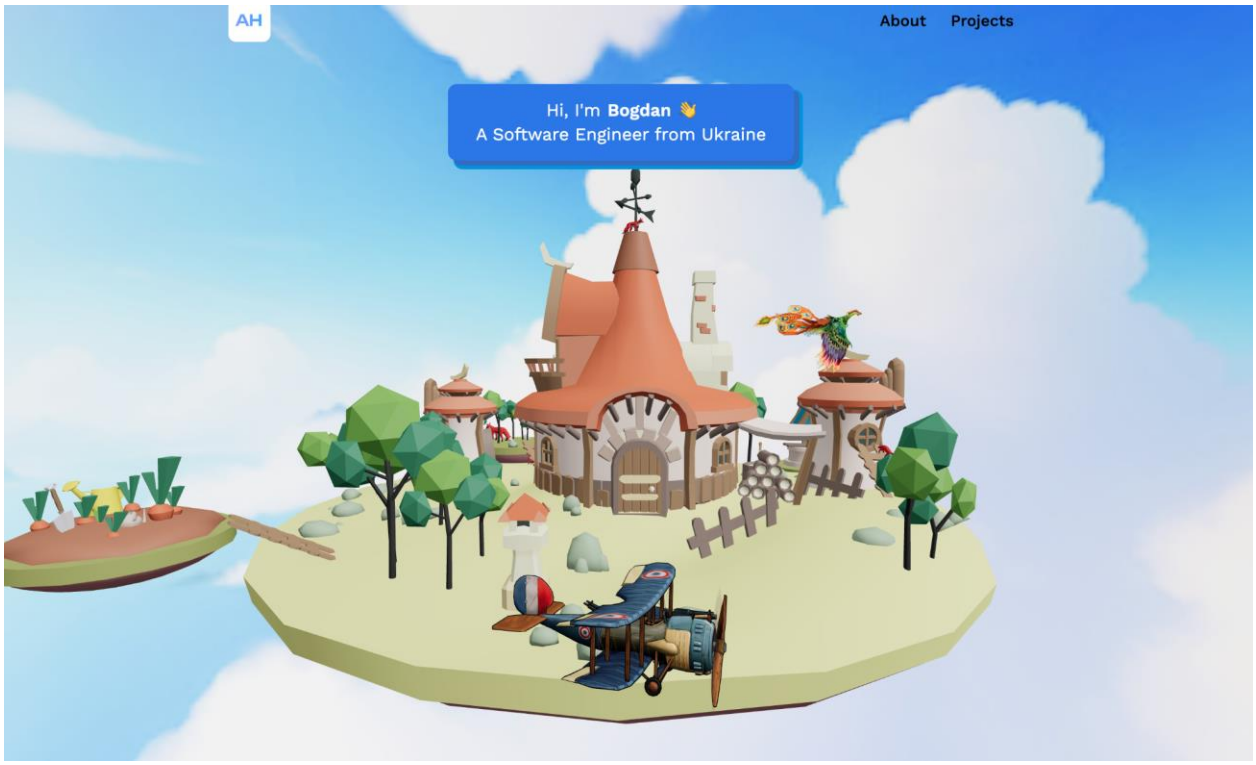


Рис. 3.1 Головна сцена 3D-портфоліо

Модель лиса включає анімації Idle, Walk і Run, які перемикаються на основі значення стану currentAnimation. Анімації реалізовано через AnimationMixer, що є частиною Three.js API, і повністю підтримуються всередині fiber. Користувач може вплинути на зміну анімації шляхом взаємодії з об'єктом, наприклад, при натисканні або утриманні клавіші. Змінна стану оновлюється за допомогою useState, а також може бути синхронізована з глобальним контекстом, якщо потрібно, щоби декілька компонентів реагували одночасно.Особливістю реалізації є сцена з островом, який виконує роль центральної платформи для розміщення інших моделей. Обертання

острова реалізовано через useRef, що вказує на mesh-компонент, і модифікацію кута його повороту в кожному кадрі. При взаємодії з клавішами або прокручуванні миші, обертання змінюється, створюючи ефект огляду середовища. Це рішення дає користувачеві відчуття контролю над камерою та сценою.Камера у сцені побудована на основі PerspectiveCamera, що забезпечує реалістичне відтворення простору. У поєднанні з OrbitControls, що реалізує орбітальне управління, камера дозволяє не лише обертати сцену, але й масштабувати її. Параметри цієї взаємодії можуть бути обмежені для збереження художньої цілісності композиції.

Система освітлення реалізована за допомогою комбінації AmbientLight для базової підсвітки, DirectionalLight для створення тіней та PointLight для локальних ефектів. Кожне джерело світла має свої координати, інтенсивність та колір, що дозволяє налаштувати сцену відповідно до дизайнерського задуму. Освітлення динамічно впливає на моделі, створюючи реалістичну гру тіней, об’ємність та глибину.Анімаційна логіка розміщена безпосередньо в компонентах моделей або у спільних модулях, які можна повторно використовувати. Наприклад, компонент Plane реалізує обертання навколо вертикальної осі при активному прапорці isRotating. Це досягається також через useFrame, а логіка перемикача керується через внутрішній стан.Ще одним елементом взаємодії є небо, яке реалізовано як сфера з налаштованою текстурою та обертанням. Воно обертається плавно навколо сцени, задаючи атмосферу. Небо також може змінювати колір залежно від вибраної теми, часу доби або взаємодії.

Для налагодження логіки взаємодії в проєкті використовуються засоби типу useHelper (з пакету drei), які дозволяють виводити гізмо-індикатори камери, світла, напрямків тощо. Це особливо важливо під час тестування сцен, коли необхідно точно візуалізувати положення об'єктів та джерел світла.Поведінка об'єктів визначається подіями pointerDown, pointerEnter, pointerLeave, click, які реалізуються на рівні кожного об'єкта. Таким чином, кожен елемент сцени здатен відповідати на дії користувача — змінювати колір, розмір, анімацію або спрацьовувати як навігаційна

точка. Наприклад, натискання на літак може перенаправити користувача до секції з проєктами, а клік по лису змінює його режим з очікування на рух.

Рис. 3.2 Основні технології та навички розробника

Завдяки використанню хуків, контекстів і компонентного підходу, уся логіка інтеграції 3D-сцени стала невід'ємною частиною архітектури React-застосунку. Всі елементи мають однакову стилістику програмування, однакову реактивну модель та синхронізуються зі станом інтерфейсу. Це дозволяє уникати дублювання коду, забезпечити контрольовану поведінку, а також спрощує подальше розширення.Інтеграція тривимірної графіки в межах цього проєкту — це не лише технічна реалізація, а й демонстрація того, як сучасні бібліотеки дозволяють створювати естетично привабливі, інтерактивні й продуктивні вебінтерфейси, що працюють у режимі реального часу. У результаті було створено середовище, де контент оживає, взаємодіє з користувачем, і водночас демонструє компетенції

розробника як у сфері візуального дизайну, так і в області програмування та оптимізації рендерингу.

3.3. Опис анімацій, логіки поведінки моделей та взаємодії користувача

Анімація та логіка поведінки моделей у 3D-середовищі мають не лише декоративну, але й функціональну роль. У межах реалізації 3D-портфоліо на базі React та бібліотек Three.js і @react-three/fiber, анімації використовуються для передачі станів, зміни поведінки об'єктів, керування увагою користувача та підтримки контексту переходів між інтерфейсними секціями. Саме через анімацію забезпечується візуальна цілісність, плавність переходів і створення враження живого середовища, що реагує на присутність користувача.

Для створення анімацій застосовується комбінація підходів. У межах 3D-графіки всі рухи моделей реалізовано через AnimationMixer із Three.js, а також через циклічні оновлення координат у hook useFrame з @react-three/fiber. Це дозволяє використовувати як заздалегідь підготовлені анімаційні цикли у GLTF-файлах, так і динамічно обчислювані траєкторії в режимі реального часу. У межах інтерфейсної частини застосовується Framer Motion, що дозволяє декларативно описувати появу, зникнення та перехід елементів. Такий поділ дозволяє досягти оптимального результату з точки зору розділення логіки — візуальна анімація в DOM та фізична анімація в WebGL.Модель птаха реалізована через комбіновану логіку. Після ініціалізації сцени, вона починає рух по синусоїдальній траєкторії, що імітує політ. Обчислення координат здійснюється в рамках useFrame з урахуванням часу, що минув з моменту монтування компонента. Таким чином, птах не просто переміщується в одному напрямку, а змінює висоту, нахил та напрямок руху. Це дозволяє створити ефект живої динаміки без використання складних алгоритмів. У візуальному плані, зміни положення супроводжуються також легким обертанням, що підсилює ефект реалістичності.

Лис реалізований як персонаж із трьома станами — очікування, хода і біг. Вибір анімації залежить від взаємодії користувача: при наведені або натисканні змінюється стан, відповідно до якого активується певна анімація. Перемикання між станами реалізоване через AnimationMixer і здійснюється плавно, без розриву між кадрами. При цьому використовується внутрішній стан компонента, зберігається поточний режим, і нова анімація активується лише при його зміні. Це забезпечує як керованість, так і візуальну послідовність.Літак в інтерфейсі має власну поведінку: при активації змінної isRotating він починає безперервно обертатись навколо вертикальної осі. Рух не залежить від положення камери і не синхронізується з курсором, що дозволяє зберегти незалежну логіку поведінки. Анімація обертання реалізується у hook useFrame і не вимагає зовнішніх тригерів. Водночас стан обертання може контролюватися через інтерфейс користувача або через події, згенеровані при взаємодії з іншими елементами.

Рис. 3.3 Алгоритм роботи

Острів, що є центральним елементом сцени, має властивість повільного обертання, яке активується при ініціалізації сцени. Така анімація забезпечує ефект занурення та дозволяє користувачу орієнтуватись у просторі. У разі потреби обертання можна синхронізувати з прокруткою або перемиканням між секціями. Позиція острова також може бути динамічно змінена відповідно до стану сторінки або вмісту компонента. Наприклад, при переході до секції «Проєкти» камера зміщується, а сцена

змінює кут огляду, створюючи ілюзію переміщення у віртуальному просторі.Небо реалізоване як сферична оболонка з текстурованою поверхнею, що обертається незалежно від інших об’єктів. Це обертання виконується за допомогою функції обчислення кута обертання в useFrame. Швидкість обертання може бути змінено залежно від теми оформлення, часу доби або на основі інтеракції користувача. Таким чином, навіть фонові елементи середовища підтримують концепцію динамічного живого простору.Крім анімацій, безпосередньо пов’язаних з 3D-об’єктами, інтерактивні елементи інтерфейсу мають власну логіку поведінки. Наприклад, карти проєктів анімовано з’являються при вході на сторінку, змінюють розмір при наведенні та забезпечують перехід до детального перегляду при натисканні. Всі ці ефекти реалізовано з використанням Framer Motion. Кожен елемент має початкову, анімовану та завершальну позицію. Це дозволяє контролювати ефект входу, затримку, тривалість та тип easing-функції.

Система взаємодії з користувачем включає в себе як прямі події (натискання, наведення), так і непрямі (скролінг, зміна секції, перебування на сторінці понад певний час). Усі ці події впливають на змінні стану, які в свою чергу викликають відповідні анімації. Наприклад, при прокручуванні сторінки відбувається зміна фону, масштабування об'єктів, а також активація нових сценічних елементів. Це дозволяє створити ефект послідовного розкриття контенту, що покращує юзер-експіріенс і підвищує інтерес до взаємодії.Логіка подій побудована навколо використання хуків useState, useEffect, useFrame, які дозволяють реагувати на події з високою точністю та стабільністю. Наприклад, події pointerDown і pointerEnter дозволяють реалізувати реакцію на наведення або клік. У кожному з компонентів ця логіка ізольована і не впливає на інші частини сцени, що підтримує модульність архітектури.

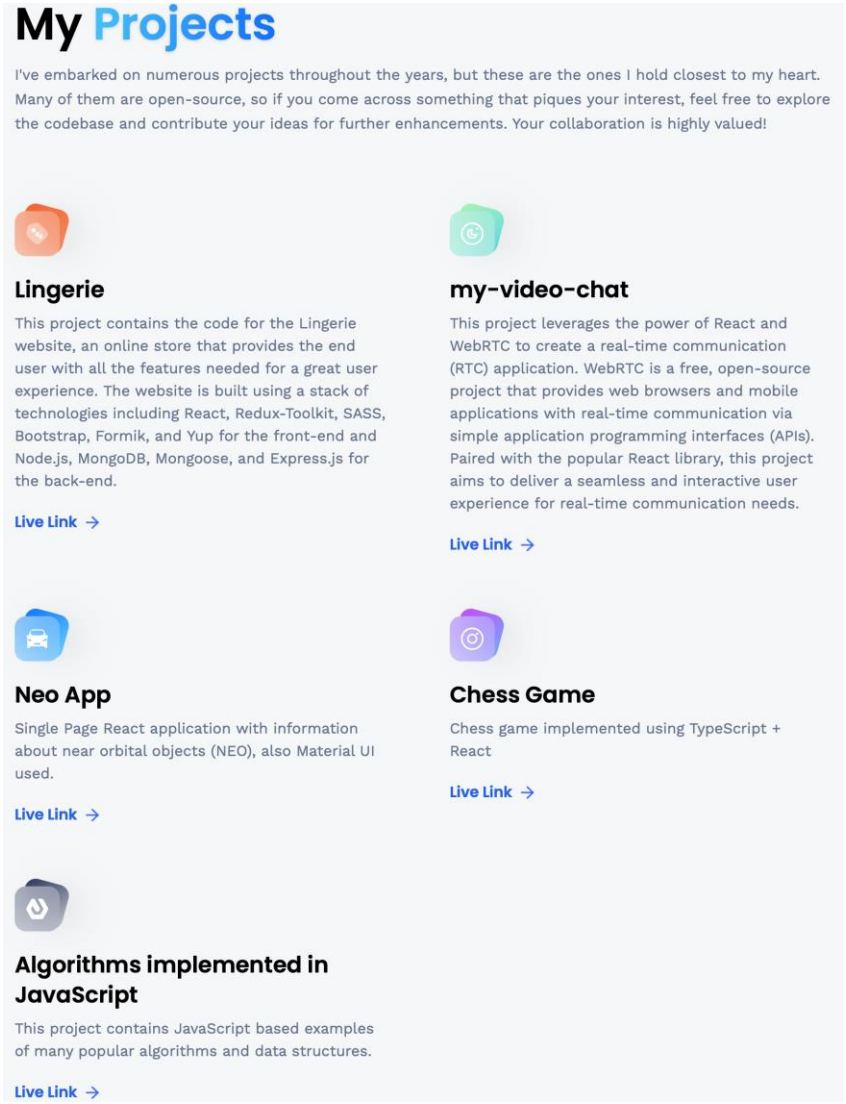


Рис. 3.4 Галерея реалізованих проєктів

Завдяки інтеграції анімаційних бібліотек та ретельному проєктуванню логіки поведінки, усі моделі та елементи інтерфейсу реагують на дії користувача у зрозумілий і приємний спосіб. Це створює ефект залучення, зменшує когнітивне навантаження та формує цілісний досвід навігації. Користувач не лише переглядає контент, а взаємодіє з ним як із частиною віртуального середовища, що змінюється у відповідь на його дії.

Загалом, застосовані методи забезпечують ефективне поєднання статичної структури з динамічним змістом, створюють багатий на події простір і дозволяють продемонструвати глибоке володіння інструментами frontend-розробки, 3D-графіки та UX-дизайну.

3.4. Тестування функціональності та адаптивності застосунку

Тестування є невід’ємною частиною життєвого циклу будь-якого програмного продукту, особливо у випадку систем з підвищеним рівнем візуальної інтерактивності та складною компонентною архітектурою. В межах реалізації вебзастосунку 3D-портфоліо було проведено багаторівневе тестування, яке охоплює функціональні, інтеграційні та адаптивні аспекти роботи системи. Основною метою таких перевірок було забезпечення стабільності роботи інтерфейсу, відповідності заявленим функціональним вимогам, узгодженості між різними модулями та гарантії комфортної взаємодії на різноманітних типах пристроїв.Функціональне тестування охоплювало перевірку коректності роботи кожного з компонентів. Було протестовано сценарії взаємодії з тривимірними об’єктами, активацію анімацій, перемикання між сторінками, рендеринг контенту, обробку форм зворотного зв’язку та логіку керування станами. Основна увага приділялася узгодженості реакції елементів на взаємодію: чи змінюється анімація відповідно до дії, чи зберігаються значення полів, чи оновлюється візуальний стан кнопок після натискання. Також перевірялась правильність логіки маршрутизації — чи правильно працюють переходи, чи зберігається стан між перемиканнями, чи не виникає помилок при безпосередньому введенні URL.

Особливо ретельно досліджувалась поведінка анімацій. Сцена з моделлю птаха мала стабільно оновлювати положення в кожному кадрі, незалежно від навантаження. Було перевірено, що значення координат, які обчислювались за допомогою тригонометричних функцій, залишались в межах очікуваних. Аналогічно, анімації лиса не повинні були конфліктувати при швидкій зміні станів. В окремих випадках анімаційні переходи тестувались на предмет дрифтів, втрати кадрової частоти або зациклення. Після численних перевірок підтверджено, що логіка перемикання між Idle, Walk та Run не створює конфліктів в AnimationMixer.Система інтерфейсу, побудована на React, також проходила перевірку на предмет ререндерингу компонентів. За допомогою React Developer Tools відстежувалось, чи оновлюються

тільки необхідні частини дерева компонентів, чи не відбувається зайвих перерендерів, які могли б негативно вплинути на продуктивність. У цьому контексті тестування охоплювало і правильність реалізації хуків — чи не відбувається втрати стану, чи не виникає умовна гонка при використанні useEffect, і чи підтримується ізоляція логіки при передачі даних через контексти.

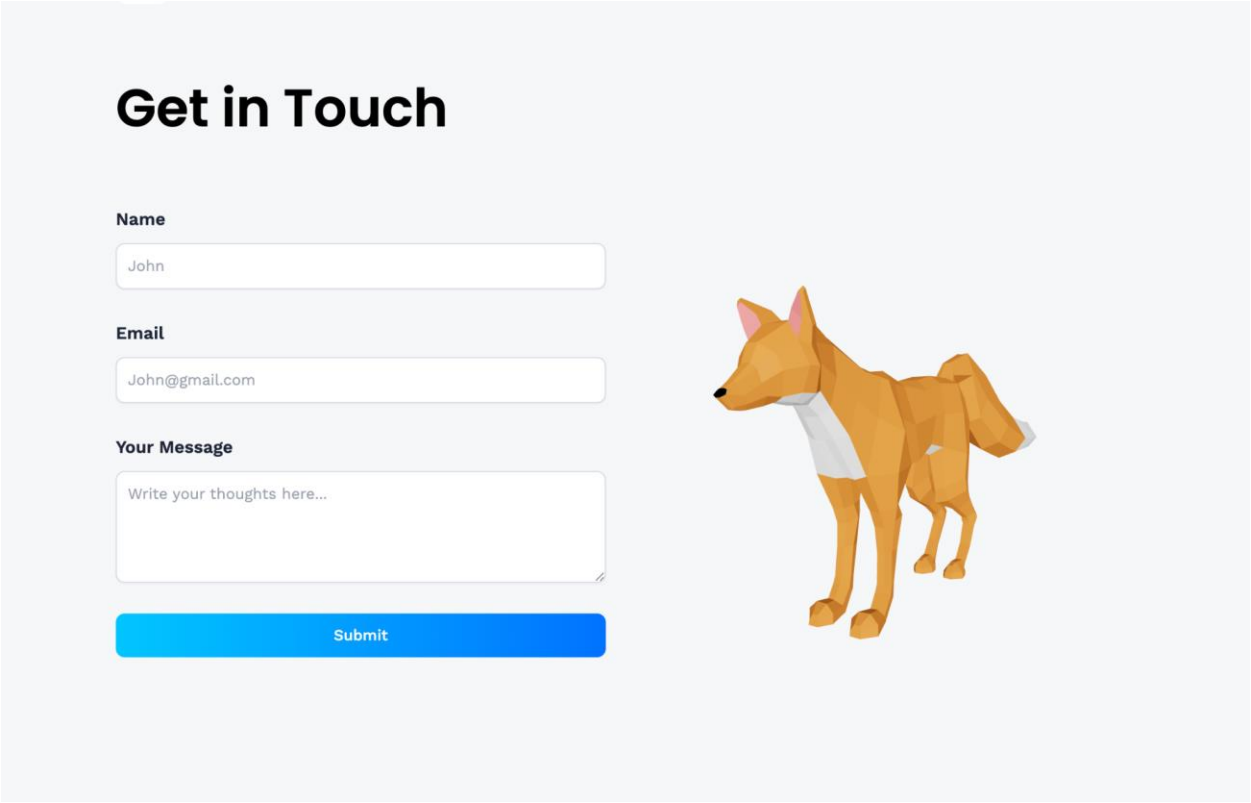


Рис. 3.5 Форма зворотного зв’язку

У процесі тестування форми зворотного зв’язку було перевірено коректність валідації, відповідність форматів введення та обробку виключних ситуацій. Визначалось, чи правильно обробляються порожні поля, як відображаються помилки, чи блокується кнопка надсилання при невірних даних. Окрім цього, відстежувалась робота EmailJS: чи надсилається запит, чи отримується відповідь, як поводить себе інтерфейс при втраті з’єднання або збої на стороні сервісу.Адаптивність інтерфейсу тестувалась у кілька етапів. Спочатку проводились перевірки за допомогою Chrome

DevTools, Firefox Responsive Design Mode та Safari Inspector. Емулювались різні розміри екранів, починаючи з мобільних телефонів діагоналлю від 4.7 дюймів і закінчуючи великими моніторами з роздільною здатністю 4K. У кожному випадку аналізувалась логіка розміщення елементів, коректність відступів, адаптація тексту та розміру кнопок, поведінка 3D-сцени при масштабуванні.

У процесі практичного тестування інтерфейс переглядався також на реальних пристроях: смартфонах з Android та iOS, планшетах, ноутбуках і десктопах. Було встановлено, що у всіх середовищах інтерфейс зберігає пропорційність, не обрізає контент, а сцена коректно реагує на зміни розмірів вікна. Тестувались реакції при обертанні екрану, переключенні між темною та світлою темою, масштабуванні браузера, а також при увімкненні функцій доступності на операційних системах.Окремо перевірялась адаптивність тривимірної сцени. Було зафіксовано, що всі моделі масштабуються пропорційно до розміру полотна canvas. Візуальна сцена автоматично змінює співвідношення сторін у залежності від ширини екрану, не спотворює об’єкти і зберігає перспективу. Камера коректно змінює свої параметри при зміні розміру контейнера, а OrbitControls не втрачає своєї чутливості. Це було особливо важливо у мобільному режимі, де замість миші використовуються жести.
Велика увага також приділялась перевірці продуктивності. За допомогою вбудованих засобів браузерів та бібліотеки drei-stats відслідковувалась кількість кадрів на секунду (FPS), навантаження на GPU, кількість об’єктів у сцені, частота викликів рендерингу. У всіх основних сценаріях — з увімкненими анімаціями, обертанням об’єктів, навігацією та взаємодією — було зафіксовано стабільний FPS на рівні 55–60 при середньому завантаженні. Це свідчить про ефективну оптимізацію рендерингу, грамотне використання useFrame і контрольовану кількість трикутників у сцені.

Для перевірки граничних ситуацій тестувались дії у випадках втрати мережі, блокування скриптів, некоректного введення. Наприклад, при вимкненому інтернеті форма зворотного зв’язку повинна відображати повідомлення про помилку, а 3D-сцена

має відображатись у своєму останньому відомому стані. Усі ці ситуації моделювались вручну, після чого вносились зміни у логіку обробки виключень.

Підсумком проведених тестів стало підтвердження стабільної, адаптивної та інтуїтивно зрозумілої роботи застосунку на широкому спектрі пристроїв. Система успішно демонструє не лише технічні навички розробника, а й відповідність практичним вимогам до якості вебпродукту. Ретельне тестування дозволило виявити й усунути низку потенційних багів ще до розгортання застосунку, що значно підвищує рівень готовності проєкту до публічного використання.

3.5. Аналіз отриманих результатів та перспективи розвитку

Аналіз отриманих результатів у межах реалізації 3D-портфоліо дозволяє зробити висновки не лише щодо технічної доцільності застосованого підходу, але й оцінити рівень інтегрованості функціоналу, стабільність взаємодії, ефективність використання графічних технологій та відповідність розробленого продукту початковим вимогам. Реалізоване середовище демонструє комплексність вирішення завдання — від архітектурної побудови до візуальної реалізації та взаємодії користувача з об'єктами у тривимірному просторі.Програма успішно виконує функції демонстрації особистих навичок, взаємодії з 3D-елементами, реалізації адаптивного інтерфейсу та передачі інтерактивного досвіду через сучасні фронтенд-технології. Високий рівень адаптивності забезпечує повну функціональність як на десктопах, так і на мобільних пристроях. Застосування анімаційної логіки підвищує залученість користувача і покращує сприйняття візуального контенту. Структурна організація коду дозволяє легко масштабувати рішення, що створює передумови для розширення функціоналу, покращення продуктивності, впровадження нових модулів або навіть міграції проєкту на інші платформи у майбутньому.

На етапі тестування було підтверджено стійкість системи до типових помилок взаємодії, некоректних запитів, змін у розмірах вікна браузера, втрати з'єднання. Усі ключові функції продемонстрували надійність: анімації працюють стабільно, навіть при зміні FPS, рендеринг сцени не супроводжується лагами, елементи інтерфейсу зберігають свою логіку взаємодії на різних пристроях.

Контактна форма успішно передає повідомлення, валідує дані та реагує на помилки з очікуваною поведінкою. Сторінки завантажуються швидко, компоненти оновлюються лише при потребі, а інтеграція 3D-графіки не створює додаткового навантаження, що негативно впливає на досвід користувача.Важливим фактором, що виявився у ході реалізації, є ефективність використання бібліотек @react-three/fiber та drei. Ці інструменти дозволили забезпечити декларативний стиль опису тривимірної сцени, зменшити обсяг ручного коду та створити інтерактивну логіку, яка інтегрується в загальну архітектуру React.

Анімації моделей, реалізовані через useFrame, AnimationMixer та хуки стану, демонструють, як можна досягти високої динамічності без використання складних графічних шейдерів або окремого рендер-сервера. Усі об'єкти в сцені, включаючи лиса, птаха, літак, острів та небо, мають повністю контрольовану поведінку, яка залежить від внутрішніх станів, контексту додатку та дій користувача.

Інтеграція інтерфейсної частини через Tailwind CSS забезпечила єдину стилістичну мову для візуальних елементів.

Кожен компонент легко налаштовується, змінюється та розширюється. У поєднанні з Framer Motion ця структура дала можливість реалізувати плавні переходи, анімації та мікровзаємодії, що в сукупності формують загальну якість UI/UX. Застосунок викликає відчуття завершеності: кожна дія користувача має візуальне підтвердження, зміна розділів супроводжується плавним анімованим переходом, а візуальні елементи реагують на взаємодію зворотним зв’язком.

З технічної точки зору, результат можна вважати оптимізованим. Усі компоненти проходять перевірку на продуктивність, зберігають ефективність навіть у

середовищах з обмеженими ресурсами. Завдяки використанню Vite як інструменту складання та Tailwind як фреймворку стилів, обсяг завантажуваного коду залишився мінімальним, що дозволяє досягти швидкого часу рендерингу першого екрана.
Важливим досягненням проєкту є підтвердження придатності фронтенд-екосистеми React до побудови не лише класичних інтерфейсів, але й повноцінних тривимірних середовищ.

Це відкриває нові можливості для реалізації інтерактивних платформ, віртуальних вітрин, персоналізованих навчальних модулів, ігор або систем візуалізації даних у просторі.Перспективи розвитку проєкту охоплюють декілька стратегічних напрямів.

Першим є впровадження бекенд-частини, яка дозволить динамічно оновлювати вміст сторінок, додавати нові проєкти, редагувати навички без необхідності втручання у вихідний код. Це дозволить перенести збереження даних до бази, синхронізувати портфоліо з іншими платформами (наприклад, GitHub або LinkedIn), а також автоматизувати деякі частини. Додатково, таке рішення зробить застосунок багатокористувацьким — кожен автор зможе створювати власну копію портфоліо з індивідуальним вмістом.

Другим напрямом розвитку є впровадження системи аналітики. Збір даних про перегляди, натискання, навігацію між сторінками дозволить аналізувати поведінкові патерни користувачів і оптимізувати структуру та контент. Такі метрики можуть збиратись за допомогою інтеграції з Google Analytics, Plausible або власної статистичної системи, що працює на бекенді.

Третім напрямком є локалізація та підтримка багатомовності. Завдяки впровадженню i18n, користувач зможе перемикатись між мовами, що суттєво розширює аудиторію і відкриває можливості для використання портфоліо в міжнародному контексті. Структура додатку вже зараз дозволяє легко реалізувати цю функціональність, оскільки більшість текстового вмісту винесено у зовнішні конфігураційні файли.

Четвертим можливим розширенням є автоматизація оновлень через систему керування контентом — CMS. У поточній реалізації всі дані зберігаються як об’єкти у файлах constants.js. Впровадження CMS, наприклад Sanity або Strapi, дозволить редагувати вміст з вебінтерфейсу без залучення розробника. Це зробить систему доступною для ширшого кола користувачів, які хочуть мати аналогічне портфоліо, але не мають технічного досвіду.

П’ятим напрямком розвитку може стати інтеграція з WebXR або WebGPU, що дозволить вивести інтерфейс у доповнену або віртуальну реальність. У цьому випадку сцена буде не лише на екрані, а і у фізичному просторі, доступному через VR/AR гарнітури. Таке розширення зробить проєкт унікальним з точки зору взаємодії і дозволить перетворити портфоліо у справжню віртуальну презентацію.

На завершення можна стверджувати, що розроблений застосунок є не просто технічним прикладом інтеграції React з Three.js. Це повноцінне цифрове середовище, яке поєднує візуальну виразність, функціональність, стабільність, продуктивність і високий рівень персоналізації. Проєкт не лише виконує початкові завдання, а й створює платформу для подальших інновацій, що може слугувати основою для навчання, комерційного просування або технологічних демонстрацій.

ВИСНОВКИ

У процесі реалізації бакалаврського проєкту було здійснено повний цикл розробки, починаючи з постановки завдання, аналізу сучасного стану предметної області, вибору технологічного стеку і закінчуючи безпосередньою реалізацією та тестуванням інтерактивного вебзастосунку 3D-портфоліо. Мета роботи полягала у створенні середовища для персональної презентації навичок розробника з використанням тривимірної графіки та сучасних засобів фронтенд-програмування. Ця мета була досягнута завдяки синергії таких технологій, як React, Three.js, Tailwind CSS, Vite, Framer Motion та @react-three/fiber.

У ході виконання дослідження було проведено глибокий аналіз існуючих рішень, зокрема, платформ для створення онлайн-портфоліо, а також специфіки візуальної презентації у контексті конкуренції на ринку IT. Визначено об’єктивну потребу в інструментах, які дозволяють не лише перелічити навички та проєкти, а й подати їх у більш інтуїтивній, динамічній та візуально привабливій формі. Запропоноване рішення дозволяє поєднати традиційний вміст портфоліо з інноваційними способами подання інформації через 3D-інтерфейс, який є інтерактивним і привертає увагу користувача.Технологічна реалізація інтерфейсу за допомогою React забезпечила гнучкість, модульність і зручність керування станами. Завдяки використанню Tailwind CSS було створено адаптивне середовище з уніфікованою стилістикою, що дозволяє легко масштабувати систему або модифікувати окремі компоненти.

Інтеграція тривимірної сцени за допомогою Three.js та @react-three/fiber показала високу ефективність у побудові об'єктно-орієнтованої структури взаємодії користувача з графічним простором. Усі елементи сцени взаємопов’язані зі станами та логікою додатку, що дозволяє досягти максимальної інтегрованості між візуальним середовищем та інтерфейсом.

Кожен етап проєктування та реалізації підтвердив доцільність застосування обраного інструментарію. Моделювання вимог дозволило чітко визначити сценарії користувача, основні функціональні блоки та порядок взаємодії з контентом. Архітектура проєкту виявилась стабільною, гнучкою та придатною до подальшого розширення. Було реалізовано чітке розділення між логікою, стилями, даними і візуальним представленням, що відповідає сучасним стандартам розробки SPA-додатків.

Анімації та поведінка 3D-моделей реалізовані з урахуванням принципів плавності, інтуїтивності та контекстної активності. Кожен об’єкт у сцені має не лише візуальну, але й логічну функцію. Поведінка моделей — птаха, лиса, літака — керується внутрішнім станом і змінюється відповідно до дій користувача або зміни контексту сторінки. Це дозволило досягти ефекту занурення, який підсилює емоційне сприйняття проєкту та демонструє технічні можливості розробника.Інтерфейсна частина, що складається з секцій «Про себе», «Навички», «Проєкти» та «Контакти», реалізована у відповідності до кращих практик UI/UX дизайну. Інтерактивна навігація, адаптивне позиціонування елементів, ефекти наведення та анімації створюють позитивний досвід користування та підвищують рівень залученості. Форма зворотного зв’язку, яка працює через інтеграцію з EmailJS, забезпечує функціональну можливість контакту між відвідувачем і власником портфоліо без використання серверної частини.

Результати тестування підтвердили стабільність, надійність та адаптивність створеного продукту. Усі функції, включаючи навігацію, анімації, 3D-взаємодію, обробку форм та рендеринг, працюють відповідно до очікувань. Вебзастосунок показав високу продуктивність, швидкий час відгуку, стійкість до типових помилок і гнучкість у відображенні на різних типах пристроїв — від смартфонів до широкоформатних моніторів. Це свідчить про грамотну реалізацію архітектурних рішень і уважність до деталей під час розробки.

На основі виконаної роботи було підтверджено потенціал поєднання тривимірної графіки та frontend-технологій для створення інноваційного формату

самопрезентації. Такий підхід можна масштабувати до інших доменів: наприклад, в освітніх курсах, візуалізації продуктів, музеях, архітектурних презентаціях тощо. Таким чином, проєкт має потенціал не лише як особистий інструмент, а й як базова платформа для професійного використання в інших галузях.

У роботі також були визначені напрямки подальшого розвитку. Передусім, це впровадження серверної логіки для динамічного оновлення контенту, використання CMS або бази даних для зберігання інформації про проєкти, а також інтеграція з аналітичними сервісами для відстеження поведінки користувачів. Окрім цього, доцільною є реалізація багатомовності, що значно розширить потенційну аудиторію. На завершальному етапі розвитку можна передбачити інтеграцію WebXR, що дозволить вивести портфоліо у віртуальну або доповнену реальність, тим самим зробивши його максимально інноваційним.

Загалом, отриманий результат свідчить про повноцінну реалізацію поставлених цілей, високу якість реалізації функціональних вимог, технічну зрілість рішення та перспективність подальшого розвитку. Розроблений продукт повністю відповідає як вимогам бакалаврської роботи, так і актуальним тенденціям в IT-індустрії.

Робота пройшла апробацію. За її результатами опубліковані наступні тези доповідей:

1. Момот Б. Д., Бажан Ю.П. Інтеграця 3d-графіки та елементів штучного інтелекту в сучасні фреймворки react. V Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». 15.05.2025, ДУІКТ, м.Київ. Подано до друку.
2. Момот Б. Д., Бажан Ю.П. Використання інтелектуальних технологій у розробці інтерактивних 3d-середовищ на базі react з урахуванням світового досвіду та українських тенденцій. V Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». 15.05.2025, ДУІКТ, м.Київ. Подано до друку.

ПЕРЕЛІК ПОСИЛАНЬ

1. React Documentation. (n.d.). https://react.dev/learn
2. React – A JavaScript library for building user interfaces. (n.d.). https://legacy.reactjs.org/
3. Getting started with React. (n.d.). Mozilla Developer Network. https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started
4. React Tutorial. (n.d.). W3Schools. https://www.w3schools.com/REACT/DEFAULT.ASP
5. React example viability for efficient API learning (REVEAL): A tool to detect and fix common errors in React examples. (2023). ScienceDirect. https://www.sciencedirect.com/science/article/pii/S2590118423000114
6. Front-End Development in React: An Overview. (2023). ResearchGate. https://www.researchgate.net/publication/374154236_Front-End_Development_in_React_An_Overview
7. Review on React JS. (2021). International Journal of Trend in Scientific Research and Development. https://www.ijtsrd.com/papers/ijtsrd42490.pdf
8. REACT JS - A JAVASCRIPT LIBRARY. (2024). International Research Journal of Modernization in Engineering Technology and Science. https://www.irjmets.com/uploadedfiles/paper//issue_4_april_2024/52186/final/fin_irjmets1712488435.pdf

9. A Review on “React JS”. (2022). ADR Journals House. https://adrjournalshouse.com/index.php/computer-technology-software-app/article/view/1439

10. React – Hooks API Reference. (n.d.). https://legacy.reactjs.org/docs/hooks-reference.htmlReactMDNWebDocsW3SchoolsScienceDirectResearchGateIJTSRDIRJMETSadrjournalshouse.comReact

11. Three.js Documentation. (n.d.). https://threejs.org/docs/

12. Three.js Manual. (n.d.). https://threejs.org/manual/

13. Three.js Framework. (2016).

ResearchGate. https://www.researchgate.net/publication/302306483_Threejs_Framework

14. Three.js – JavaScript 3D Library. (n.d.). https://threejs.org/

15. Three.js Basics. (n.d.). Engineering

LibreTexts. https://eng.libretexts.org/Bookshelves/Computer_Science/Applied_Programming/Introduction_to_Computer_Graphics_(Eck)/05%3A_Three.js-_A_3D_Scene_Graph_API/5.01%3A_Three.js_Basics

16. A preliminary study of WebGL and Three.js. (2020). ACM Digital Library. https://dl.acm.org/doi/10.1145/3424616.3424726

17. Implementation of interactive three-dimensional visualization of air pollutants using Three.js. (2018). ScienceDirect.

https://www.sciencedirect.com/science/article/pii/S1364815218304195

18. Performance and Ease of Use in 3D on the Web. (2020). DiVA portal. https://www.divaportal.org/smash/get/diva2%3A1523176/FULLTEXT01.pdft

hreejs.orgResearchGateGitHub+1threejs.org+1Engineering LibreTextsACM Digital LibraryScienceDirectDIVA Portal

19. 7 fundamental UX design principles all designers should know. (2024). UX Design Institute. https://www.uxdesigninstitute.com/blog/ux-design-principles/

20. 10 Usability Heuristics for User Interface Design. (n.d.). Nielsen Norman Group. https://www.nngroup.com/articles/ten-usability-heuristics/

21. The 5 best academic journals in UX design and research. (2023). Anais Digital. https://www.anais.digital/5-best-academic-journals-in-ux-design-and-research/

22. Understanding The Single-Page Application Architecture. (2022). Ramotion. https://www.ramotion.com/blog/single-page-application-architecture/

23. Single-page application. (n.d.). Wikipedia. https://en.wikipedia.org/wiki/Single-page_application

24. SPA (Single-page application). (n.d.). MDN Web Docs. https://developer.mozilla.org/en-US/docs/Glossary/SPA

25. What is Single Page Application? Understanding SPA. (n.d.). OutSystems. https://www.outsystems.com/application-development/spa-single-page-app-defined-with-examples/uxdesigninstitute.comNielsen Norman Group+1Nielsen Norman Group+1Anais Digitalramotion.comWikipedia+1MDN Web Docs+1MDN WebDocs+2MDNWebDocs+2MDNWebDocs+2Medium+2OutSystems+2Geeksfor Geeks+2

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка інтерактивного середовища “Портфоліо 3D-розробника”

Виконав студент 4 курсу
групи ПД-43
Момот Богдан Дмитрович
Керівник роботи
Бажан Юрій Павлович, викладач кафедри

Київ – 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: розробка сучасного інтерактивного інтерфейсу для візуалізації професійних навичок 3D-розробника із застосуванням технологій React і Three.js з метою підвищення ефективності самопрезентації у сфері веб-розробки.

Об'єкт дослідження: процес створення інтерактивного портфоліо розробника.

Предмет дослідження: інструменти та технології веб-розробки і 3D-візуалізації, зокрема бібліотеки React, Three.js і засоби для реалізації анімації та інтерактивності.

2

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Ознайомлення з принципами роботи бібліотеки Three.js та фреймворку React.
2. Розробка структури та дизайну 3D-портфоліо.
3. Реалізація функціоналу для інтерактивної взаємодії користувача зі сценами.Інтеграція 3D-моделей та анімацій.

3

АНАЛІЗ АНАЛОГІВ

Технічні характеристики	Bruno Simon	Kenta Toshikura	Aidan Nelson	Renaud Rohlinger	My Project
SPA-архітектура	+	+	+	+	+
Кастомна логіка	+	+	+	+	+
Адаптивність під різні пристрої	+	+	+	+	+
UI/UX орієнтація на розробників	+	+	+	+	+
Наявність контактної форми	-	-	-	-	+
Інтерактивність 3D-сцени	+	+	+	+	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

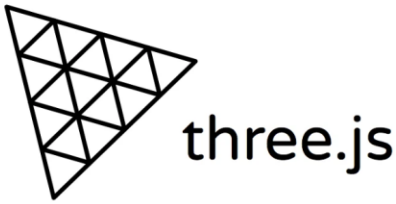
1. Адаптивна вебсторінка з інтерактивним 3D-інтерфейсом.
2. Можливість перегляду проєктів, технологій, контактної інформації.
3. Взаємодія користувача з елементами сцени (анімації, переміщення камери).
4. Анімація при переході між розділами та інтерактивні підказки для нових користувачів.

Нефункціональні вимоги:

1. Веб-застосунок повинен коректно працювати в основних сучасних браузерах (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari) без втрати функціональності чи якості відображення.
2. Забезпечується швидке рендерення сцени та елементів 3D-графіки, з мінімальною кількістю кадрів, що втрачаються (FPS ≥ 55 на середньому пристрої).
3. Дизайн користувацького інтерфейсу має бути мінімалістичним, логічно структурованим, із передбачуваною навігацією, що сприяє зниженню когнітивного навантаження при використанні.
4. Сторінка має завантажуватись менше ніж за 2 секунди при середній швидкості з'єднання (10 Мбіт/с). Розмір основних ресурсів (зображення, скрипти) не повинен перевищувати 1500 кБ у сукупності.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



6

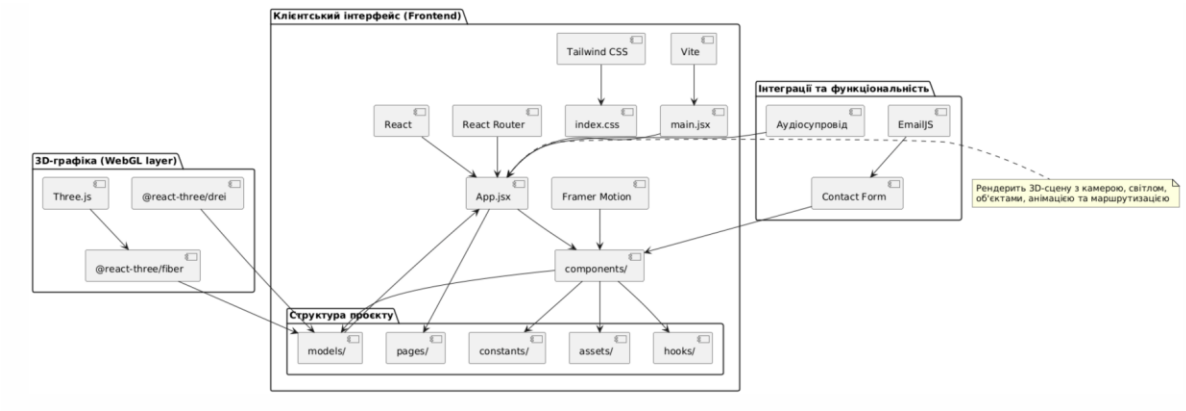
Use-case діаграма

7

Алгоритм роботи

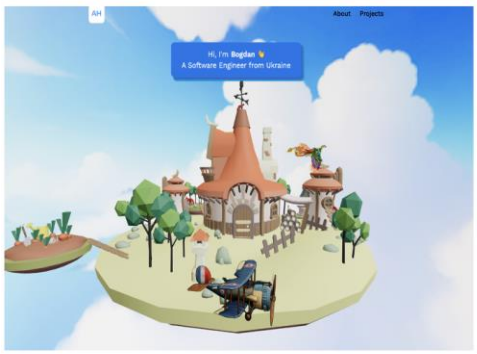
8

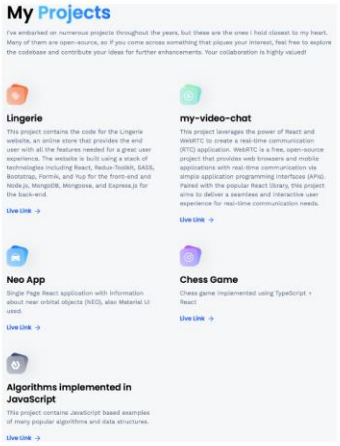
Архітектура проєкту

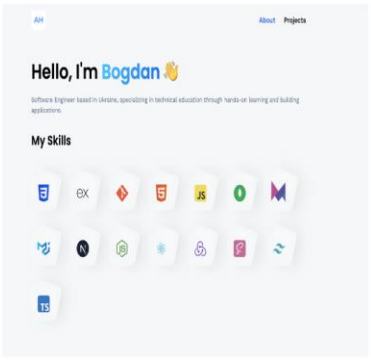


9

ЕКРАННІ ФОРМИ







10

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Момот Б. Д., Бажан Ю.П. Інтеграця 3d-графіки та елементів штучного інтелекту в сучасні фреймворки react. V Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». 15.05.2025, ДУІКТ, м.Київ. Подано до друку.
2. Момот Б. Д., Бажан Ю.П. Використання інтелектуальних технологій у розробці інтерактивних 3d-середовищ на базі react з урахуванням світового досвіду та українських тенденцій. V Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті». 15.05.2025, ДУІКТ, м.Київ. Подано до друку.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

Bird.jsx

```
import { useEffect, useRef } from "react";
import { useFrame } from "@react-three/fiber";
import { useAnimations, useGLTF } from "@react-three/drei";

import birdScene from "../assets/3d/bird.glb";

export function Bird() {
  const birdRef = useRef();

  // Load the 3D model and animations from the provided GLTF file
  const { scene, animations } = useGLTF(birdScene);

  // Get access to the animations for the bird
  const { actions } = useAnimations(animations, birdRef);

  // Play the "Take 001" animation when the component mounts
  // Note: Animation names can be found on the Sketchfab website where the 3D model is hosted.
  useEffect(() => {
    actions["Take 001"].play();
  }, []);

  useFrame(({ clock, camera }) => {
    // Update the Y position to simulate bird-like motion using a sine wave
    birdRef.current.position.y = Math.sin(clock.elapsedTime) * 0.2 + 2;

    // Check if the bird reached a certain endpoint relative to the camera
    if (birdRef.current.position.x > camera.position.x + 10) {
      // Change direction to backward and rotate the bird 180 degrees on the y-axis
      birdRef.current.rotation.y = Math.PI;
    } else if (birdRef.current.position.x < camera.position.x - 10) {
      // Change direction to forward and reset the bird's rotation
      birdRef.current.rotation.y = 0;
    }

    // Update the X and Z positions based on the direction
    if (birdRef.current.rotation.y === 0) {
      // Moving forward
      birdRef.current.position.x += 0.01;
      birdRef.current.position.z -= 0.01;
    } else {
      // Moving backward
      birdRef.current.position.x -= 0.01;
      birdRef.current.position.z += 0.01;
    }
  });

  return (
    // to create and display 3D objects
    <mesh ref={birdRef} position={[-5, 2, 1]} scale={[0.003, 0.003, 0.003]}>
      // use the primitive element when you want to directly embed a complex 3D
      model or scene
      <primitive object={scene} />
    </mesh>
  );
}

export default Bird
```

Fox.jsx

```
import React, { useRef, useEffect } from "react";
import { useGLTF, useAnimations } from "@react-three/drei";

import scene from "../assets/3d/fox.glb";

export function Fox({ currentAnimation, ...props }) {
  const group = useRef();
  const { nodes, materials, animations } = useGLTF(scene);
  const { actions } = useAnimations(animations, group);

  // This effect will run whenever the currentAnimation prop changes
  useEffect(() => {
    Object.values(actions).forEach((action) => action.stop());

    if (actions[currentAnimation]) {
      actions[currentAnimation].play();
    }
  }, [actions, currentAnimation]);

  return (
    <group ref={group} {...props} dispose={null}>
      <group name='Sketchfab_Scene'>
        <primitive object={nodes.GLTF_created_0_rootJoint} />
        <skinnedMesh
          name='Object_7'
          geometry={nodes.Object_7.geometry}
          material={materials.PaletteMaterial001}
          skeleton={nodes.Object_7.skeleton}
        />
        <skinnedMesh
          name='Object_8'
          geometry={nodes.Object_8.geometry}
          material={materials.PaletteMaterial001}
          skeleton={nodes.Object_8.skeleton}
        />
        <skinnedMesh
          name='Object_9'
```

```
            geometry={nodes.Object_9.geometry}
            material={materials.PaletteMaterial001}
            skeleton={nodes.Object_9.skeleton}
          />
          <skinnedMesh
            name='Object_10'
            geometry={nodes.Object_10.geometry}
            material={materials.PaletteMaterial001}
            skeleton={nodes.Object_10.skeleton}
          />
          <skinnedMesh
            name='Object_11'
            geometry={nodes.Object_11.geometry}
            material={materials.PaletteMaterial001}
            skeleton={nodes.Object_11.skeleton}
          />
        </group>
      </group>
    );
}

useGLTF.preload(scene);
```

Island.jsx

```
import { a } from "@react-spring/three";
import { useEffect, useRef } from "react";
import { useGLTF } from "@react-three/drei";
import { useFrame, useThree } from "@react-three/fiber";

import islandScene from '../assets/3d/island.glb'

export function Island({
isRotating,
setIsRotating,
setCurrentStage,
currentFocusPoint,
...props
}) {
  const islandRef = useRef();
  // Get access to the Three.js renderer and viewport
  const { gl, viewport } = useThree();
  const { nodes, materials } = useGLTF(islandScene);

  // Use a ref for the last mouse x position
  const lastX = useRef(0);
  // Use a ref for rotation speed
  const rotationSpeed = useRef(0);
  // Define a damping factor to control rotation damping
  const dampingFactor = 0.95;

  // Handle pointer (mouse or touch) down event
  const handlePointerDown = (event) => {
    event.stopPropagation();
    event.preventDefault();
    setIsRotating(true);

    // Calculate the clientX based on whether it's a touch event or a mouse event
    const clientX = event.touches ? event.touches[0].clientX : event.clientX;

    // Store the current clientX position for reference
    lastX.current = clientX;
  };

  // Handle pointer (mouse or touch) up event
  const handlePointerUp = (event) => {
    event.stopPropagation();
    event.preventDefault();
    setIsRotating(false);
  };

  // Handle pointer (mouse or touch) move event
  const handlePointerMove = (event) => {
    event.stopPropagation();
    event.preventDefault();
    if (isRotating) {
      // If rotation is enabled, calculate the change in clientX position
      const clientX = event.touches ? event.touches[0].clientX : event.clientX;

      // calculate the change in the horizontal position of the mouse cursor or touch input,
      // relative to the viewport's width
      const delta = (clientX - lastX.current) / viewport.width;

      // Update the island's rotation based on the mouse/touch movement
      islandRef.current.rotation.y += delta * 0.01 * Math.PI;

      // Update the reference for the last clientX position
      lastX.current = clientX;

      // Update the rotation speed
      rotationSpeed.current = delta * 0.01 * Math.PI;
    }
  };

  // Handle keydown events
  const handleKeyDown = (event) => {
    if (event.key === "ArrowLeft") {
      if (!isRotating) setIsRotating(true);

      islandRef.current.rotation.y += 0.005 * Math.PI;
      rotationSpeed.current = 0.007;
    } else if (event.key === "ArrowRight") {
      if (!isRotating) setIsRotating(true);

      islandRef.current.rotation.y -= 0.005 * Math.PI;
      rotationSpeed.current = -0.007;
    }
  };

  // Handle keyup events
  const handleKeyUp = (event) => {
    if (event.key === "ArrowLeft" || event.key === "ArrowRight") {
      setIsRotating(false);
    }
  };

  useEffect(() => {
    // Add event listeners for pointer and keyboard events
    const canvas = gl.domElement;
    canvas.addEventListener("pointerdown", handlePointerDown);
    canvas.addEventListener("pointerup", handlePointerUp);
    canvas.addEventListener("pointermove", handlePointerMove);
    window.addEventListener("keydown", handleKeyDown);
```

```
    window.addEventListener("keyup", handleKeyUp);

    // Remove event listeners when component unmounts
    return () => {
      canvas.removeEventListener("pointerdown",
handlePointerDown);
      canvas.removeEventListener("pointerup",
handlePointerUp);
      canvas.removeEventListener("pointermove",
handlePointerMove);
      window.removeEventListener("keydown",
handleKeyDown);
      window.removeEventListener("keyup", handleKeyUp);
    };
  }, [gl, handlePointerDown, handlePointerUp,
handlePointerMove]);

  // This function is called on each frame update
  useFrame(() => {
    // If not rotating, apply damping to slow down the rotation
(smoothly)
    if (!isRotating) {
      // Apply damping factor
      rotationSpeed.current *= dampingFactor;

      // Stop rotation when speed is very small
      if (Math.abs(rotationSpeed.current) < 0.001) {
        rotationSpeed.current = 0;
      }

      islandRef.current.rotation.y += rotationSpeed.current;
    } else {
      // When rotating, determine the current stage based on
island's orientation
      const rotation = islandRef.current.rotation.y;

      /**
       * Normalize the rotation value to ensure it stays within the
range [0, 2 * Math.PI].
       * The goal is to ensure that the rotation value remains
within a specific range to
       * prevent potential issues with very large or negative
rotation values.
       * Here's a step-by-step explanation of what this code does:
       * 1. rotation % (2 * Math.PI) calculates the remainder of
the rotation value when divided
       *    by 2 * Math.PI. This essentially wraps the rotation
value around once it reaches a
       *    full circle (360 degrees) so that it stays within the range
of 0 to 2 * Math.PI.
       * 2. (rotation % (2 * Math.PI)) + 2 * Math.PI adds 2 *
Math.PI to the result from step 1.
       *    This is done to ensure that the value remains positive
and within the range of
       *    0 to 2 * Math.PI even if it was negative after the
modulo operation in step 1.
       * 3. Finally, ((rotation % (2 * Math.PI)) + 2 * Math.PI) %
(2 * Math.PI) applies another
       *    modulo operation to the value obtained in step 2. This
step guarantees that the value
       *    always stays within the range of 0 to 2 * Math.PI,
which is equivalent to a full
       *    circle in radians.
       */
      const normalizedRotation =
        ((rotation % (2 * Math.PI)) + 2 * Math.PI) % (2 *
Math.PI);

      // Set the current stage based on the island's orientation
      switch (true) {
        case normalizedRotation >= 5.45 &&
normalizedRotation <= 5.85:
          setCurrentStage(4);
          break;
        case normalizedRotation >= 0.85 &&
normalizedRotation <= 1.3:
          setCurrentStage(3);
          break;
        case normalizedRotation >= 2.4 && normalizedRotation
<= 2.6:
          setCurrentStage(2);
          break;
        case normalizedRotation >= 4.25 &&
normalizedRotation <= 4.75:
          setCurrentStage(1);
          break;
        default:
          setCurrentStage(null);
      }
    }
  });

  return (
    // {Island 3D model from: https://sketchfab.com/3d-
models/foxs-islands-163b68e09fcc47618450150be7785907}
    <a.group ref={islandRef} {...props}>
      <mesh

geometry={nodes.polySurface944_tree_body_0.geometry}
        material={materials.PaletteMaterial001}
      />
      <mesh
        geometry={nodes.polySurface945_tree1_0.geometry}
        material={materials.PaletteMaterial001}
      />
      <mesh
        geometry={nodes.polySurface946_tree2_0.geometry}
        material={materials.PaletteMaterial001}
      />
      <mesh
        geometry={nodes.polySurface947_tree1_0.geometry}
        material={materials.PaletteMaterial001}
      />
      <mesh

geometry={nodes.polySurface948_tree_body_0.geometry}
        material={materials.PaletteMaterial001}
      />
      <mesh

geometry={nodes.polySurface949_tree_body_0.geometry}
        material={materials.PaletteMaterial001}
      />
      <mesh
        geometry={nodes.pCube11_rocks1_0.geometry}
        material={materials.PaletteMaterial001}
      />
    </a.group>
  );
}
```

Plane.jsx

```
import { useEffect, useRef } from "react";
import { useGLTF, useAnimations } from "@react-three/drei";

import planeScene from "../assets/3d/plane.glb";

export function Plane({ isRotating, ...props }) {
  const ref = useRef();
  // Load the 3D model and its animations
  const { scene, animations } = useGLTF(planeScene);
  // Get animation actions associated with the plane
  const { actions } = useAnimations(animations, ref);

  useEffect(() => {
    if (isRotating) {
      actions["Take 001"].play();
    } else {
      actions["Take 001"].stop();
    }
  }, [actions, isRotating]);

  return (
    <mesh {...props} ref={ref}>
      model or scene
      <primitive object={scene} />
    </mesh>
  );
}
```

Sky.jsx

```
import { useRef } from "react";
import { useGLTF } from "@react-three/drei";
import { useFrame } from "@react-three/fiber";

import skyScene from "../assets/3d/sky.glb"

export function Sky({ isRotating }) {
  const sky = useGLTF(skyScene);
  const skyRef = useRef();

  useFrame((_, delta) => {
    if (isRotating) {
      skyRef.current.rotation.y += 0.25 * delta; // Adjust the rotation speed as needed
    }
  });

  return (
    <mesh ref={skyRef}>
      // use the primitive element when you want to directly embed a complex 3D
      model or scene
      <primitive object={sky.scene} />
    </mesh>
  );
}
```

About.jsx

```
import {
  VerticalTimeline,
  VerticalTimelineElement,
} from "react-vertical-timeline-component";
import { experiences, skills } from "../../constants/index.js";

import "react-vertical-timeline-component/style.min.css";
import CTA from "../../components/CTA/CTA.jsx";

const About = () => {
  return (
    <section className='max-container'>
      <h1 className='head-text'>
        Hello, I'm{" "}
        <span className='blue-gradient_text font-semibold drop-shadow'>
          {" "}
          Bogdan
        </span>{" "}
        □
      </h1>

      <div className='mt-5 flex flex-col gap-3 text-slate-500'>
        <p>
          Software Engineer based in Ukraine, specializing in technical
          education through hands-on learning and building applications.
        </p>
      </div>

      <div className='py-10 flex flex-col'>
        <h3 className='subhead-text'>My Skills</h3>

        <div className='mt-16 flex flex-wrap gap-12'>
          {skills.map((skill) => (
            <div className='block-container w-20 h-20' key={skill.name}>
              <div className='btn-back rounded-xl' />
              <div className='btn-front rounded-xl flex justify-center items-center'>
                <img
                  src={skill.imageUrl}
                  alt={skill.name}
                  className='w-1/2 h-1/2 object-contain'
                />
              </div>
            </div>
          ))}
        </div>
      </div>

      <div className='py-16'>
        <h3 className='subhead-text'>Work Experience</h3>
        <div className='mt-5 flex flex-col gap-3 text-slate-500'>
          <p>
            I've worked with these companies, leveling up my skills and
            teaming up with smart people. Here's the rundown:
          </p>
        </div>

        <div className='mt-12 flex'>
          <VerticalTimeline>
            {experiences.map((experience, index) => (
              <VerticalTimelineElement
                key={experience.company_name}
                date={experience.date}
                iconStyle={{ background: experience.iconBg }}
                icon={
                  <div className='flex justify-center items-center w-full h-full'>
```

```
                        <img
                          src={experience.icon}
                          alt={experience.company_name}
                          className='w-[60%] h-[60%] object-contain'
                        />
                      </div>
                    }
                    contentStyle={{
                      borderBottom: "8px",
                      borderStyle: "solid",
                      borderBottomColor: experience.iconBg,
                      boxShadow: "none",
                    }}
                  >
                    <div>
                      <h3 className='text-black text-xl font-poppins font-semibold'>
                        {experience.title}
                      </h3>
                      <p
                        className='text-black-500 font-medium text-base'
                        style={{ margin: 0 }}
                      >
                        {experience.company_name}
                      </p>
                    </div>

                    <ul className='my-5 list-disc ml-5 space-y-2'>
                      {experience.points.map((point, index) => (
                        <li
                          key={`experience-point-${index}`}
                          className='text-black-500/50 font-normal pl-1 text-sm'
                        >
                          {point}
                        </li>
                      ))}
                    </ul>
                  </VerticalTimelineElement>
                ))}
              </VerticalTimeline>
            </div>
          </div>

          <hr className='border-slate-200' />

          <CTA />
        </section>
    );
};

export default About;
```

Contact.jsx

```
import emailjs from "@emailjs/browser";
import { Canvas } from "@react-three/fiber";
import { Suspense, useRef, useState } from "react";
import { Fox } from "../../models/Fox.jsx";
import useAlert from "../../hooks/useAlert.js";
import Loader from "../../components/Loader/Loader.jsx";
import Alert from "../../components/Alert/Alert.jsx";

const Contact = () => {
  const formRef = useRef();
  const [form, setForm] = useState({ name: "", email: "", message: "" });
  const { alert, showAlert, hideAlert } = useAlert();
  const [loading, setLoading] = useState(false);
  const [currentAnimation, setCurrentAnimation] = useState("idle");

  const handleChange = ({ target: { name, value } }) => {
    setForm({ ...form, [name]: value });
  };

  const handleFocus = () => setCurrentAnimation("walk");
  const handleBlur = () => setCurrentAnimation("idle");

  const handleSubmit = (e) => {
    e.preventDefault();
    setLoading(true);
    setCurrentAnimation("hit");

    emailjs
      .send(
        import.meta.env.VITE_APP_EMAILJS_SERVICE_ID,
        import.meta.env.VITE_APP_EMAILJS_TEMPLATE_ID,
        {
          from_name: form.name,
          to_name: "All1in",
          from_email: form.email,
          to_email: "bm3023068@gmail.com",
          message: form.message,
        },
        import.meta.env.VITE_APP_EMAILJS_PUBLIC_KEY
      )
      .then(
        () => {
          setLoading(false);
          showAlert({
            show: true,
            text: "Thank you for your message ☐",
            type: "success",
          });

          setTimeout(() => {
            hideAlert(false);
            setCurrentAnimation("idle");
            setForm({
              name: "",
              email: "",
              message: "",
            });
          }, [3000]);
        },
        (error) => {
          setLoading(false);
          console.error(error);
          setCurrentAnimation("idle");

          showAlert({
            show: true,
            text: "I didn't receive your message ☐",
            type: "danger",
          });
```

```
        }
      );
  };

  return (
    <section className='relative flex lg:flex-row flex-col max-container'>
      {alert.show && <Alert {...alert} />}

      <div className='flex-1 min-w-[50%] flex flex-col'>
        <h1 className='head-text'>Get in Touch</h1>

        <form
          ref={formRef}
          onSubmit={handleSubmit}
          className='w-full flex flex-col gap-7 mt-14'
        >
          <label className='text-black-500 font-semibold'>
            Name
            <input
              type='text'
              name='name'
              className='input'
              placeholder='John'
              required
              value={form.name}
              onChange={handleChange}
              onFocus={handleFocus}
              onBlur={handleBlur}
            />
          </label>
          <label className='text-black-500 font-semibold'>
            Email
            <input
              type='email'
              name='email'
              className='input'
              placeholder='John@gmail.com'
              required
              value={form.email}
              onChange={handleChange}
              onFocus={handleFocus}
              onBlur={handleBlur}
            />
          </label>
          <label className='text-black-500 font-semibold'>
            Your Message
            <textarea
              name='message'
              rows='4'
              className='textarea'
              placeholder='Write your thoughts here...'
              value={form.message}
              onChange={handleChange}
              onFocus={handleFocus}
              onBlur={handleBlur}
            />
          </label>

          <button
            type='submit'
            disabled={loading}
            className='btn'
            onFocus={handleFocus}
            onBlur={handleBlur}
          >
            {loading ? "Sending..." : "Submit"}
          </button>
        </form>
      </div>

      <div className='lg:w-1/2 w-full lg:h-auto md:h-[550px] h-[350px]'>
        <Canvas
          camera={{
            position: [0, 0, 5],
            fov: 75,
            near: 0.1,
            far: 1000,
          }}
        >
          <directionalLight position={[0, 0, 1]} intensity={2.5} />
          <ambientLight intensity={1} />
          <pointLight position={[5, 10, 0]} intensity={2} />
          <spotLight
            position={[10, 10, 10]}
            angle={0.15}
            penumbra={1}
            intensity={2}
          />

          <Suspense fallback={<Loader />}>
            <Fox
              currentAnimation={currentAnimation}
              position={[0.5, 0.35, 0]}
              rotation={[12.629, -0.6, 0]}
              scale={[0.5, 0.5, 0.5]}
            />
          </Suspense>
        </Canvas>
      </div>
    </section>
  );
};

export default Contact;
```

Home.jsx

```
import { Canvas } from "@react-three/fiber";
import { Suspense, useEffect, useRef, useState } from "react";
import Loader from "../../components/Loader/Loader.jsx";
import HomeInfo from "../../components/HomeInfo/HomeInfo.jsx";

import { Island } from '../../models/Island.jsx'
import Bird from "../../models/Bird.jsx";
import { Sky } from "../../models/Sky.jsx";
import { Plane } from "../../models/Plane.jsx";

import sakura from "../../assets/sakura.mp3"
import { soundoff, soundon } from "../../assets/icons"


const Home = () => {
  const audioRef = useRef(new Audio(sakura));
  audioRef.current.volume = 0.4;
  audioRef.current.loop = true;

  const [currentStage, setCurrentStage] = useState(1);
  const [isRotating, setIsRotating] = useState(false);
```

```
  const [isPlayingMusic, setIsPlayingMusic] = useState(false);

  useEffect(() => {
    if (isPlayingMusic) {
      audioRef.current.play();
    }

    return () => {
      audioRef.current.pause();
    };
  }, [isPlayingMusic]);

  const adjustBiplaneForScreenSize = () => {
    let screenScale, screenPosition;

    // If screen width is less than 768px, adjust the scale and
position
    if (window.innerWidth < 768) {
      screenScale = [1.5, 1.5, 1.5];
      screenPosition = [0, -1.5, 0];
    } else {
      screenScale = [3, 3, 3];
      screenPosition = [0, -4, -4];
    }

    return [screenScale, screenPosition];
  };

  const adjustIslandForScreenSize = () => {
    let screenScale, screenPosition;

    if (window.innerWidth < 768) {
      screenScale = [0.9, 0.9, 0.9];
      screenPosition = [0, -6.5, -43.4];
    } else {
      screenScale = [1, 1, 1];
      screenPosition = [0, -6.5, -43.4];
    }

    return [screenScale, screenPosition];
  };

  const [biplaneScale, biplanePosition] =
adjustBiplaneForScreenSize();
  const [islandScale, islandPosition] =
adjustIslandForScreenSize();

  return (
    <section className='w-full h-screen relative'>
      <div className='absolute top-28 left-0 right-0 z-10 flex
items-center justify-center'>
        {currentStage && <HomeInfo
currentStage={currentStage} />}
      </div>

      <Canvas
        className={`w-full h-screen bg-transparent ${
          isRotating ? "cursor-grabbing" : "cursor-grab"
        }`}
        camera={{ near: 0.1, far: 1000 }}
      >
        <Suspense fallback={<Loader />}>
          <directionalLight position={[1, 1, 1]} intensity={2}
/>
          <ambientLight intensity={0.5} />
          <pointLight position={[10, 5, 10]} intensity={2} />
          <spotLight
            position={[0, 50, 10]}
            angle={0.15}
            penumbra={1}
            intensity={2}
          />
          <hemisphereLight
            skyColor='#b1e1ff'
            groundColor='#000000'
            intensity={1}
          />

          <Bird />
          <Sky isRotating={isRotating} />
          <Island
            isRotating={isRotating}
            setIsRotating={setIsRotating}
            setCurrentStage={setCurrentStage}
            position={islandPosition}
            rotation={[0.1, 4.7077, 0]}
            scale={islandScale}
          />
          <Plane
            isRotating={isRotating}
            position={biplanePosition}
            rotation={[0, 20.1, 0]}
            scale={biplaneScale}
          />
        </Suspense>
      </Canvas>

      <div className='absolute bottom-2 left-2'>
        <p style={{ marginBottom: '20px' }}> Turn on/off the
sound here: </p>
        <img
          src={!isPlayingMusic ? soundoff : soundon}
          alt='jukebox'
          onClick={() => setIsPlayingMusic(!isPlayingMusic)}
          className='w-10 h-10 cursor-pointer object-contain'
        />
      </div>
    </section>
  );
};

export default Home;
```

Project.jsx

```
import { projects } from "../../constants/index.js";
import { Link } from "react-router-dom";
import CTA from "../../components/CTA/CTA.jsx";
import { arrow } from "../../assets/icons/index.js";

const Projects = () => {
  return (
    <section className='max-container'>
      <h1 className='head-text'>
        My{" "}
        <span className='blue-gradient_text drop-shadow
font-semibold'>
          Projects
        </span>
      </h1>

      <p className='text-slate-500 mt-2 leading-relaxed'>
```

```
        I've embarked on numerous projects throughout the years, but these are
        the ones I hold closest to my heart. Many of them are open-source, so if
        you come across something that piques your interest, feel free to
        explore the codebase and contribute your ideas for further enhancements.
        Your collaboration is highly valued!
      </p>

      <div className='flex flex-wrap my-20 gap-16'>
        {projects.map((project) => (
          <div className='lg:w-[400px] w-full' key={project.name}>
            <div className='block-container w-12 h-12'>
              <div className={`btn-back rounded-xl ${project.theme}`} />
              <div className='btn-front rounded-xl flex justify-center items-center'>
                <img
                  src={project.iconUrl}
                  alt='threads'
                  className='w-1/2 h-1/2 object-contain'
                />
              </div>
            </div>

            <div className='mt-5 flex flex-col'>
              <h4 className='text-2xl font-poppins font-semibold'>
                {project.name}
              </h4>
              <p className='mt-2 text-slate-500'>{project.description}</p>
              <div className='mt-5 flex items-center gap-2 font-poppins'>
                <Link
                  to={project.link}
                  target='_blank'
                  rel='noopener noreferrer'
                  className='font-semibold text-blue-600'
                >
                  Live Link
                </Link>
                <img
                  src={arrow}
                  alt='arrow'
                  className='w-4 h-4 object-contain'
                />
              </div>
            </div>
          </div>
        ))}
      </div>

      <hr className='border-slate-200' />

      <CTA />
    </section>
  );
};

export default Projects;
```