МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ
Циклова комісія комп’ютерної інженерії та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему

ІНТЕРАКТИВНИЙ ВЕБПОРТАЛ ПРО КОСМОС З ВІЗУАЛІЗАЦІЄЮ СОНЯЧНОЇ СИСТЕМИ ТА МОДУЛЕМ АКТУАЛЬНИХ НОВИН

Виконала: студентка групи 1П-22
Спеціальності
121 Інженерія програмного забезпечення
Діана ОВЧАРЕНКО
Керівник:
Вікторія НЕМЧЕНКО

Черкаси 2026

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ

Циклова комісія комп'ютерної інженерії та інформаційних технологій

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувачка циклової комісії КІ та ІТ

_____ Наталя ФАЛЬЧЕНКО
(підпис)

«_____» _____2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Овчаренко Діані Вікторівній

1. Тема кваліфікаційної роботи «Інтерактивний вебпортал про космос з візуалізацією Сонячної системи та модулем актуальних новин»

Керівник роботи Немченко Вікторія Юріївна, викладач другої категорії затверджені наказом закладу фахової передвищої освіти від «06» жовтня 2026 року №84/1-у

2. Строк подання студентом кваліфікаційної роботи 02.06.2026

3. Вихідні дані до кваліфікаційної роботи Технічне завдання на розробку вебпорталу; офіційна документація технологій веброзробки та 3D-графіки (React, Three.js, React Three Fiber); специфікація зовнішнього інтерфейсу Spaceflight News API. Вимоги до системи: архітектура односторінкового додатка з клієнтським рендерингом простору (WebGL); збереження продуктивності рендерингу не менше 30 FPS; адаптивний користувацький інтерфейс; наявність модуля отримання новин.

4. Зміст кваліфікаційної роботи аналіз астрономічних симуляторів та вебдодатків про космос; постановка задачі розроблення вебпорталу; визначення функціональних і нефункціональних вимог; вибір архітектури та технологічного стеку; проєктування структури додатка, просторового 3D-рендерингу та інтерфейсу користувача; реалізація тривимірної візуалізації планет, математичних моделей орбітального руху, адаптивного інтерфейсу та модуля новин; тестування сценаріїв і зручності використання, автоматизоване розгортання та експлуатація програмного продукту.

5. Дата видачі завдання 15.09.2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Терміни виконання етапів	Примітка про виконання
1	Вступ	14.10.2025	
2	РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ	09.12.2025	
3	РОЗДІЛ 2 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОЇ СИСТЕМИ	10.03.2026	
4	РОЗДІЛ 3 ТЕСТУВАННЯ ТА ВЕРИФІКАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	28.04.2026	
5	Висновки	12.05.2026	
6	Оформлення кваліфікаційної роботи (чистовий варіант)	26.05.2026	
7	Перевірка кваліфікаційної роботи на наявність ознак плагіату (за 10 днів до захисту)	02.06.2026	
8	Подання кваліфікаційної роботи на затвердження завідувачу кафедри (за 7 днів до захисту)	10.06.2026	

Студент _____ Діана ОВЧАРЕНКО
(підпис)

Керівник роботи _____ Вікторія НЕМЧЕНКО
(підпис)

АНОТАЦІЯ

Кваліфікаційна робота присвячена проєктуванню та розробці сучасного інтерактивного вебпорталу, головною метою якого є популяризація астрономічних знань через застосування передових технологій тривимірної візуалізації у браузері. У процесі виконання роботи було проведено комплексний аналіз предметної області та існуючих програмних рішень, що дозволило виявити їхні недоліки та сформувати оптимальну архітектуру власного програмного продукту у вигляді швидкодіючого односторінкового додатка (SPA).

Для реалізації клієнтської частини використано компонентний підхід на базі бібліотеки React, що забезпечило гнучку структуру коду та ефективне керування динамічними станами інтерфейсу. Тривимірна модель Сонячної системи побудована за допомогою бібліотеки Three.js у зв'язці з екосистемою React Three Fiber. Цей технологічний стек дозволив використати апаратне прискорення WebGL для плавного рендерингу складних космічних об'єктів, точного розрахунку орбітальних траєкторій, реалізації функції «розумної паузи» та обчислення освітлення в реальному часі.

Окрему увагу в роботі приділено вирішенню проблем кросплатформності та UX/UI дизайну: реалізовано адаптивний мобільний інтерфейс із застосуванням патерну «Bottom Sheet» та налаштовано коректну обробку сенсорних жестів, що запобігає конфліктам між скролом сторінки та обертанням 3D-сцени. Додаткова інформаційна цінність ресурсу забезпечується інтеграцією зовнішнього REST API (Spaceflight News API) для асинхронного завантаження актуальних новин аерокосмічної галузі. Програмний продукт успішно пройшов етапи модульного та функціонального тестування і розгорнутий у хмарному середовищі Vercel із налаштуванням конвеєра безперервної інтеграції та розгортання (CI/CD).

Ключові слова: ВЕБПОРТАЛ, REACT, THREE.JS, 3D-ВІЗУАЛІЗАЦІЯ, СОНЯЧНА СИСТЕМА, АДАПТИВНИЙ ДИЗАЙН, ІНТЕГРАЦІЯ API, WEBGL, CI/CD

ABSTRACT

The qualification work is devoted to the design and development of a modern interactive web portal, the main goal of which is the popularization of astronomical knowledge through the application of advanced three-dimensional browser visualization technologies. During the execution of the work, a comprehensive analysis of the subject area and existing software solutions was conducted, which made it possible to identify their shortcomings and form an optimal software product architecture in the form of a high-performance Single Page Application (SPA).

To implement the client side, a component-based approach using the React library was utilized, providing a flexible code structure and efficient management of dynamic interface states. The three-dimensional model of the Solar System was built using the Three.js library in conjunction with the React Three Fiber ecosystem. This technology stack allowed for the use of WebGL hardware acceleration for smooth rendering of complex space objects, precise calculation of orbital trajectories, implementation of a "smart pause" function, and real-time lighting computation.

Special attention in the work is paid to solving cross-platform compatibility issues and UX/UI design: an adaptive mobile interface was implemented using the "Bottom Sheet" pattern, and proper touch gesture handling was configured to prevent conflicts between page scrolling and 3D scene rotation. The additional informational value of the resource is provided by the integration of an external REST API (Spaceflight News API) for the asynchronous loading of current aerospace industry news. The software product successfully passed the stages of unit and functional testing and was deployed in the Vercel cloud environment with a continuous integration and continuous deployment (CI/CD) pipeline configuration.

Keywords: WEBPORTAL, REACT, THREE.JS, 3D VISUALIZATION, SOLAR SYSTEM, RESPONSIVE DESIGN, API INTEGRATION, WEBGL

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ

API	(Application Programming Interface) – прикладний програмний інтерфейс; набір інструментів і функцій для взаємодії між різними програмами.
CDN	(Content Delivery Network) – мережа доставки вмісту; географічно розподілена мережа серверів для швидкої передачі вебконтенту користувачам.
CI/CD	(Continuous Integration / Continuous Deployment) – безперервна інтеграція та безперервне розгортання; набір практик для автоматизації збірки, тестування та публікації програмного забезпечення.
CPU	(Central Processing Unit) – центральний процесор; головний обчислювальний пристрій комп'ютера.
DOM	(Document Object Model) – об'єктна модель документа; програмний інтерфейс для HTML- та XML-документів.
FPS	(Frames Per Second) – кількість кадрів на секунду; показник плавності та продуктивності відтворення комп'ютерної анімації або відео.
GPU	(Graphics Processing Unit) – графічний процесор; спеціалізований електронний пристрій для обробки та рендерингу графіки.
HTTP / HTTPS	(HyperText Transfer Protocol / Secure) – протокол передавання гіпертексту / розширення з підтримкою шифрування для безпечної передачі даних.
JSON	(JavaScript Object Notation) – легкий текстовий формат обміну даними, зручний для читання людиною та машиною.
JSX	(JavaScript XML) – розширення синтаксису мови JavaScript, що дозволяє використовувати HTML-подібну розмітку всередині JS-коду.
REST	(Representational State Transfer) – архітектурний стиль взаємодії компонентів розподіленого додатка в мережі.
SPA	(Single Page Application) – односторінковий вебдодаток; тип вебдодатків, що завантажує єдину HTML-сторінку та динамічно оновлює її контент.

UI	(User Interface) – інтерфейс користувача; сукупність засобів, за допомогою яких користувач взаємодіє з програмою.
URL	(Uniform Resource Locator) – уніфікований покажчик ресурсу (стандартизована адреса певного ресурсу в мережі Інтернет).
UX	(User Experience) – користувацький досвід; сприйняття та реакції користувача, що виникають під час використання програмного продукту.
WebGL	(Web Graphics Library) – кросплатформний API для рендерингу інтерактивної 3D- та 2D-графіки в браузері без використання плагінів.
XSS	(Cross-Site Scripting) – міжсайтовий скриптинг; тип вразливості вебдодатків, що дозволяє зловмисникам впроваджувати шкідливий код.

ВСТУП

Актуальність теми. Сучасний етап розвитку інформаційних технологій характеризується стрімким переходом від статичного вебконтенту до високоінтерактивних, імерсивних платформ. У сфері освітніх та науково-популярних ресурсів виникає гостра проблема: традиційні форми подачі матеріалу (тексти, двовимірні зображення, відеоролики) вже не здатні повною мірою задовольнити потреби сучасного користувача, який очікує інтерактивності та можливості самостійного дослідження. Особливо чітко ця проблема простежується в астрономії – науці, де розуміння масштабів, орбітальної механіки та просторового розташування об'єктів потребує саме тривимірної візуалізації. Ступінь дослідження цієї проблеми у вебсередовищі є достатньо високим з точки зору створення окремих 3D-симуляторів. Проте шляхом аналізу відомих розв'язань було виявлено суттєві недоліки існуючих рішень. Більшість з них є або перевантаженими науковими даними з вкрай незручним інтерфейсом для мобільних пристроїв, або ж являють собою статичні презентації, які не оновлюються в реальному часі. Отже, розроблення сучасного, кросплатформного односторінкового вебдодатка (SPA), який поєднує високопродуктивну 3D-візуалізацію космічних об'єктів з модулем безперервного оновлення актуальних новин, є надзвичайно актуальним завданням. Таке рішення є доцільним для розвитку галузі освітніх технологій, оскільки воно вирішує проблему фрагментованості інформації, пропонуючи єдиний інтуїтивно зрозумілий простір для вивчення Сонячної системи та відстеження світових космічних подій.

Об’єктом дослідження є процес розроблення та функціонування інтерактивних вебдодатків з елементами високопродуктивної тривимірної графіки.

Предметом дослідження є інструменти, методи та технології (зокрема React, Three.js, WebGL) створення клієнтської частини космічного вебпорталу з модулем актуальних новин.

Метою кваліфікаційної роботи є розроблення інтерактивного вебпорталу про космос із тривимірною візуалізацією Сонячної системи та модулем актуальних новин для покращення якості подачі науково-популярної інформації та підвищення рівня залученості користувачів.

Завдання кваліфікаційної роботи. Для досягнення поставленої мети та послідовного вирішення окресленої проблеми було визначено такі завдання:

1. Проаналізувати предметну область, дослідити існуючі програмні аналоги на ринку та виявити їхні архітектурні й функціональні недоліки.

2. Сформулювати функціональні та нефункціональні вимоги до розроблюваного вебпорталу.

3. Спроєктувати концептуальну модель та архітектуру клієнтської частини системи на базі компонентного підходу.

4. Реалізувати модуль високопродуктивної тривимірної візуалізації об'єктів Сонячної системи з використанням технологій WebGL та Three.js.

5. Розробити інтерфейс користувача та інтегрувати систему асинхронних запитів до зовнішнього API для отримання динамічної стрічки космічних новин.

6. Здійснити адаптацію інтерфейсу під мобільні пристрої та налаштувати конвеєр безперервного розгортання (CI/CD) у хмарному середовищі.

7. Провести верифікацію програмного забезпечення шляхом комплексного функціонального та автоматизованого модульного тестування.

Практичне значення одержаних результатів. Результати кваліфікаційної роботи мають високий потенціал для практичного застосування. Розроблений інтерактивний вебпортал може бути впроваджений у навчальний процес закладів освіти (школи, коледжі) як сучасний наочний посібник для вивчення астрономії та фізики. Гнучка компонентна архітектура додатка дозволяє легко масштабувати проєкт: додавати нові супутники, астероїдні пояси чи освітні вікторини. Крім

того, застосовані архітектурні рішення (інтеграція React Three Fiber із зовнішніми REST API та налаштування автоматизованого розгортання на платформі Vercel) можуть слугувати практичною рекомендацією та готовим шаблоном для веброзробників, які проєктують імерсивні освітні SPA-додатки.

Апробація кваліфікаційної роботи. Результати, наведені в межах поточної роботи, були представлені на XVIII Всеукраїнській науково-практичній конференції «Тенденції розвитку ІТ-технологій в Україні».

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Характеристика предметної області

1.1.1 Опис предметної області та бізнес-контекст

Предметною областю даної кваліфікаційної роботи є сфера розроблення інтерактивних освітніх та інформаційних вебдодатків, орієнтованих на тривимірну візуалізацію астрономічних даних. На концептуальному рівні ця предметна область лежить на перетині кількох галузей: освітніх технологій (EdTech), популяризації науки та сучасної веброзробки.

Бізнес-контекст функціонування майбутнього програмного продукту визначається стрімким зростанням попиту на імерсивні (занурювальні) технології в освіті та медіа. Традиційні підходи до вивчення астрономії або споживання новин про космос здебільшого спираються на статичні двовимірні схеми, текстові описи або лінійні відеоматеріали. Проте космічний простір є багатовимірним середовищем, де розуміння орбітальної механіки, просторового розташування об'єктів та їхніх масштабів є критично важливим для коректного засвоєння інформації. Відповідно, прикладна сфера застосування вебпорталу охоплює заклади освіти, платформи для самоосвіти, а також інформаційні ресурси для загальної аудиторії.

1.1.2 Ролі зацікавлених сторін та користувачів

Для коректного проєктування архітектури системи необхідно визначити основні суб’єкти взаємодії (користувацькі ролі) та їхні типові сценарії діяльності. В межах даної предметної області виділяються три ключові групи стейкхолдерів:

1. Здобувачі освіти (школярі, студенти). Основний інтерес цієї групи полягає в отриманні швидкого, наочного та інтерактивного доступу до інформації про будову Сонячної системи.

Типовий сценарій. Завантаження порталу, вільна навігація 3D-сценою (наближення, обертання камери), вибір конкретної планети для вивчення її характеристик (відстань від Сонця, тип, цікаві факти) через інформаційні модальні вікна.

2. Викладачі та популяризатори науки. Інтерес групи полягає у використанні порталу як допоміжного демонстраційного інструменту під час проведення лекцій або створення освітнього контенту.

Типовий сценарій. Використання функції «розумної паузи» для зупинки орбітального руху в конкретний момент часу з метою детального пояснення розташування небесних тіл без втрати контексту симуляції.

3. Ентузіасти та аматори астрономії. Цільова аудиторія, яка зацікавлена не лише в базових знаннях, а й в актуальних подіях аерокосмічної галузі.

Типовий сценарій. Регулярне відвідування порталу для моніторингу новин через спеціалізований модуль, що асинхронно підтягує актуальну інформацію із зовнішніх баз даних (Spaceflight News API).

1.1.3 Об'єкти взаємодії та їхні функції

Основними об'єктами взаємодії в системах є:

1. Віртуальне 3D-середовище. Графова структура моделей небесних тіл, що реалізує розрахунок орбіт та освітлення.

2. Користувацький 2D-інтерфейс (UI). Набір навігаційних елементів, панелей та інформаційних карток, що слугують точками взаємодії користувача з системою. Зовнішні джерела даних: API-сервери, які постачають динамічний контент (новини) за запитом клієнта.

1.1.4 Ідентифікація ключових проблем та обмежень домену

Під час аналізу середовища функціонування було виявлено ряд критичних проблем, що знижують ефективність наявних рішень в обраній предметній області:

1. Фрагментованість інформації. На сьогодні користувач змушений використовувати кілька різних платформ: наукові симулятори для перегляду 3D-моделей, новинні агрегатори для читання статей та онлайн-енциклопедії для отримання довідкової інформації. Це розпорошує увагу та погіршує користувацький досвід (UX).

2. Низька доступність та високий поріг входження. Існуючі наукові десктопні симулятори вимагають встановлення важкого програмного забезпечення або мають перевантажений таблицями та параметрами інтерфейс, що є незручним для кінцевих користувачів, особливо на мобільних пристроях.

3. Статичність контенту. Більшість браузерних 3D-презентацій (наприклад, 100,000 Stars) є замкненими системами, які не актуалізуються після релізу. Відсутність динамічних оновлень призводить до швидкої втрати інтересу аудиторії.

4. Обмежена продуктивність у вебсередовищі. Рендеринг високополігональних моделей з текстурами високої роздільної здатності створює надмірне навантаження на процесор (CPU) пристрою, що призводить до падіння частоти кадрів (FPS) та зависання браузера.

1.1.5 Прикладні виклики

Виявлені проблеми формулюються у вигляді чітких прикладних викликів, які безпосередньо впливають на архітектурні рішення даної роботи і потребують технічного розв’язання:

1. Створення гібридної архітектури (SPA), що безшовно інтегруватиме апаратно-прискорену 3D-графіку зі стандартним DOM-деревом для виведення текстової інформації та новин.

2. Проєктування чуйного інтерфейсу, здатного адаптуватися під різні розміри екранів, з розмежуванням обробки подій дотику, щоб уникнути конфліктів між прокручуванням текстових панелей та навігацією по 3D-сцені.

3. Налаштування надійного механізму асинхронного зв'язку зі сторонніми API для подолання проблеми статичності контенту.

1.2 Аналіз наявних рішень

Для визначення конкурентних переваг та обґрунтування архітектурних рішень розроблюваного вебпорталу було здійснено аналітичний огляд існуючих програмних продуктів, що належать до відповідної предметної області. Метою даного аналізу є виявлення технологічних та функціональних особливостей наявних рішень, а також визначення їхніх сильних і слабких сторін з огляду на сучасні вимоги індустрії розроблення програмного забезпечення та потреби цільової аудиторії.

Як основні аналоги для аналізу було обрано два популярні вебресурси: TheSkyLive 3D Simulator та 100,000 Stars, які зображені на на рис. 1.1 та рис. 1.2.

TheSkyLive 3D Simulator – це комплексний інструмент для відстеження об'єктів Сонячної системи у браузері. Основне призначення продукту полягає у наданні точних астрономічних координат, орбітальних траєкторій та ефемерид для планет, комет і астероїдів. Сфера застосування охоплює здебільшого аматорську та професійну астрономію. Головною перевагою рішення є висока інформативність та використання реальних математичних моделей для симуляції руху.

100,000 Stars (Chrome Experiments) – це інтерактивна вебпрезентація, розроблена командою Google Data Arts для демонстрації можливостей сучасних браузерів. Призначення продукту – візуальне ознайомлення користувачів із найближчим зоряним оточенням Сонця. Сфера застосування – науково-популярна та розважальна. Рішення відзначається високою якістю кінематографічної 3D-візуалізації.

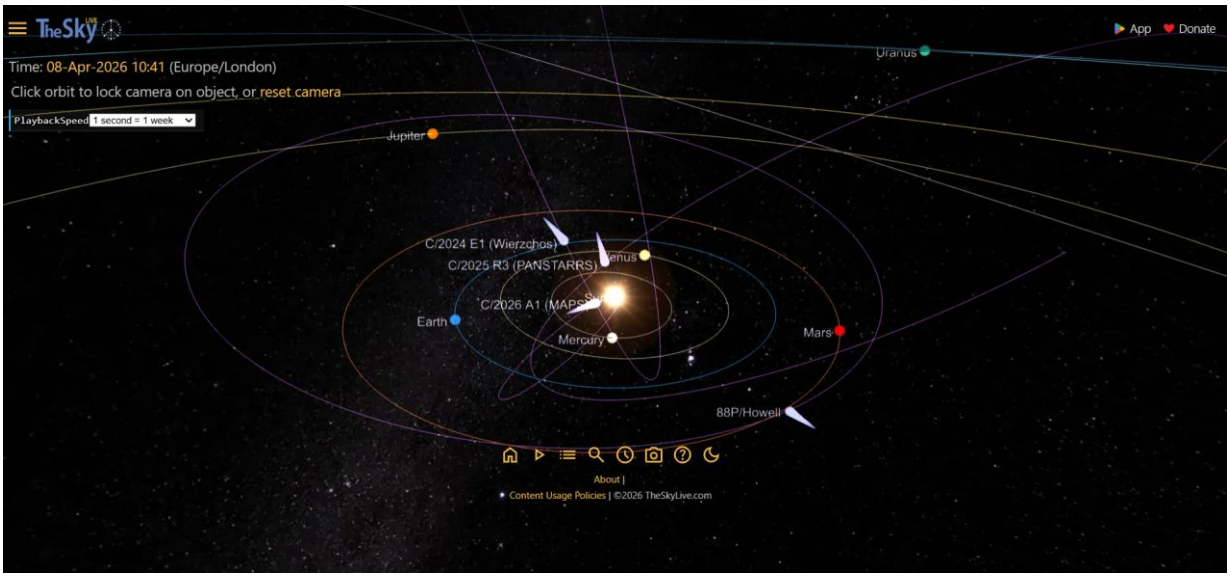

Рисунок 1.1 – Інтерфейс користувача TheSkyLive 3D

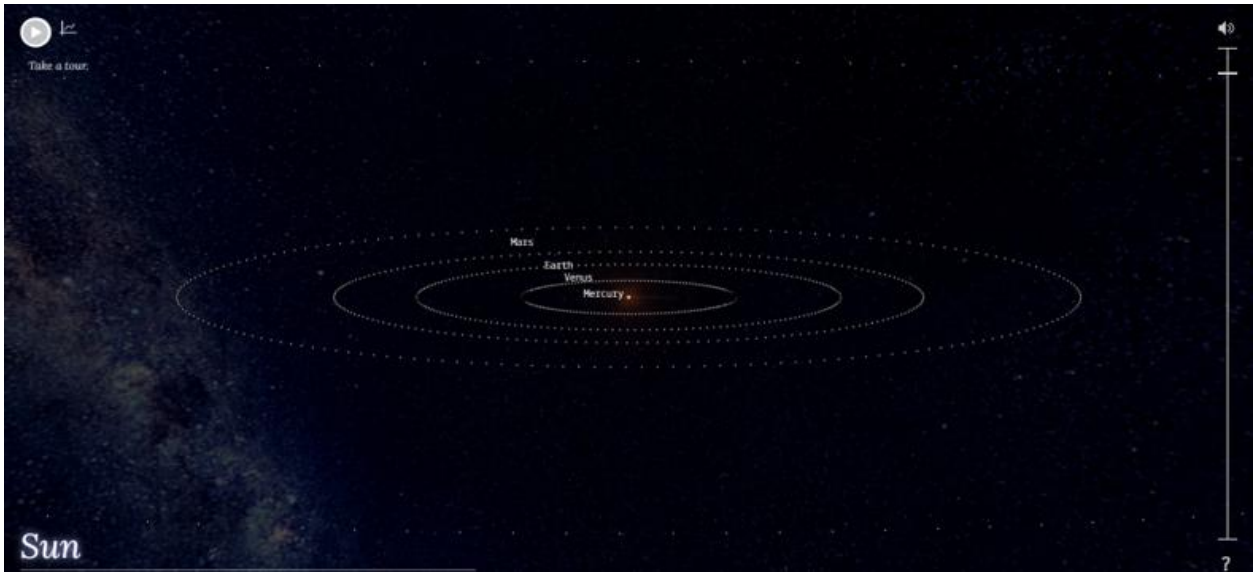

Рисунок 1.2 – Інтерфейс інтерактивної вебпрезентації 100,000 Stars

1.2.1 Порівняльний аналіз ключових функціональних можливостей

Для систематизації даних було проведено порівняння функціональних та технологічних параметрів обраних рішень у зіставленні з концепцією розроблюваного вебпорталу. Результати наведено у табл. 1.1.

Таблиця 1.1 – Порівняльний аналіз існуючих рішень

Критерії порівняння	TheSkyLive 3D Simulator	100,000 Stars	Майбутній вебпортал
Формат додатку	Багатосторінковий сайт із вбудованим 3D-вікном	Інтерактивна вебпрезентація	Односторінковий додаток (SPA)
Інтерфейс (UI/UX)	Застарілий, перевантажений таблицями та кнопками	Сучасний, кінематографічний (але лише як екскурсія)	Сучасний (Glassmorphism), інтуїтивно зрозумілий
Інтеграція стрічки актуальних новин (API)	Відсутня	Відсутня (статичний текст)	Реалізована (Spaceflight News API)
Основний технологічний стек	HTML5 Canvas / базовий WebGL	WebGL, CSS3D	React, Three.js, React Three Fiber
Керування часом симуляції	Ручне введення дати / кроку часу	Відсутнє	Синхронна «розумна» пауза зі збереженням дельти часу
Головний недолік	Незручний для мобільних пристроїв та звичайних користувачів	Проєкт давно не оновлювався, статичний контент	Менший обсяг астрономічних даних порівняно з базами симуляторів

1.2.2 Узагальнена оцінка нефункціональних характеристик

Окрім функціоналу, критично важливим є аналіз нефункціональних вимог наявних систем:

1. Продуктивність. Рішення 100,000 Stars демонструє високу продуктивність завдяки глибокій оптимізації WebGL. Натомість TheSkyLive часто стикається з падінням частоти кадрів (FPS) під час рендерингу великої кількості орбітальних ліній через застарілі підходи до управління пам'яттю.

2. Зручність користування. TheSkyLive має вкрай низькі показники UX для сучасного користувача. Інтерфейс є перевантаженим і майже не придатним для використання на мобільних пристроях. 100,000 Stars має відмінний UX на десктопі, але обмежену навігаційну свободу.

3. Масштабованість. Обидва проєкти мають монолітну або жорстко зв'язану структуру коду на ванільному JavaScript, що робить додавання нових модулів (наприклад, системи новин) дорогим і складним інженерним завданням.

4. Актуальність даних. 100,000 Stars містить виключно статичний текстовий контент, який не оновлювався роками, що швидко знижує залученість (retention rate) користувачів.

1.2.3 Ідентифікація недоліків та напрями диференціації

Проведений аналіз дозволяє виділити дві ключові проблеми наявних рішень, що ускладнюють їх використання:

1. Неможливість комфортної роботи на мобільних пристроях через відсутність адаптивних UI-патернів.

2. Відсутність інтеграції з динамічними джерелами даних (API), що перетворює симулятори на одноразові "музейні експонати", відірвані від реальних подій космічної галузі.

На основі виявлених недоліків, диференціація майбутнього програмного продукту будуватиметься на трьох архітектурних стовпах:

1. Гібридність контенту полягає в поєднанні потужної 3D-візуалізації з модулем динамічних актуальних новин, що стимулюватиме користувачів повертатися на портал.

2. Сучасна SPA-архітектура, що використовує компонентний підход React який забезпечить високу масштабованість системи та швидкість оновлення інтерфейсу без перезавантаження сторінки.

3. Mobile-First підхід, що впроваджує адаптивні патерни керування для забезпечення безшовного користувацького досвіду на смартфонах і планшетах.

Такий підхід дозволить створити конкурентоспроможний продукт, що вирішує проблеми фрагментованості інформації та незручності наявних аналогів.

1.3 Постановка завдання

На основі проведеного аналізу предметної області та виявлених недоліків існуючих програмних рішень, було визначано вектор подальшого проєктування системи через формування чітких вимог, обмежень та сценаріїв використання продукту.

Цільовим призначенням розроблюваної системи є створення кросплатформного інтерактивного вебпорталу про космос у форматі односторінкового додатка (SPA). Система призначена для візуалізації астрономічних даних (будови Сонячної системи) за допомогою засобів тривимірної графіки та надання користувачам актуальної текстової інформації і новин космічної галузі.

Основні сценарії використання (Use Cases) включають:

1. Ознайомлення з просторовим розташуванням планет Сонячної системи шляхом вільної навігації у 3D-просторі (обертання, наближення, панорамування).
2. Отримання довідкової інформації про конкретне небесне тіло шляхом взаємодії з його тривимірною моделлю (вибір об'єкта фокусування).
3. Управління часом симуляції для детального вивчення орбітальних позицій (пауза/відновлення руху).
4. Моніторинг актуальних подій у світі космонавтики через спеціалізовану панель новин.

1.4 Функціональні вимоги

Для формалізації користувацьких потреб на концептуальному рівні використано підхід гнучкої розробки з описом функціональних вимог у форматі користувацьких історій (User Stories). Основні функціональні вимоги до системи зведено у табл. 1.2.

Таблиця 1.2 – Перелік ключових функціональних вимог до вебпорталу

Назва функції	Опис у форматі User Story	Пріоритет
Ініціалізація 3D-сцени	Як користувач, я хочу бачити відрендерену 3D-модель Сонця та планет під час завантаження сторінки, щоб одразу почати взаємодію із системою.	Високий
Орбітальна анімація	Як користувач, я хочу спостерігати автоматичний розрахунок і рух планет по їхніх орбітах, щоб розуміти динаміку Сонячної системи.	Високий
Керування камерою	Як користувач, я хочу мати можливість обертати та наближати камеру за допомогою миші або сенсорного екрана, щоб розглядати об'єкти з різних ракурсів.	Високий
Інформаційні панелі	Як користувач, я хочу клікнути на планету і побачити вікно з її характеристиками (тип, відстань, цікавий факт), щоб отримати навчальну інформацію.	Середній
«Розумна» пауза	Як користувач, я хочу натискати кнопку паузи для зупинки орбітального руху без стрибків координат об'єктів, щоб зафіксувати поточний стан системи.	Середній
Модуль новин	Як користувач, я хочу відкривати окрему панель з новинами, щоб читати актуальні статті про дослідження космосу.	Високий

1.5 Нефункціональні вимоги

Нефункціональні вимоги визначають атрибути якості розроблюваного програмного забезпечення і формулюються наступним чином:

1. Продуктивність. Система повинна забезпечувати плавну тривимірну анімацію з частотою оновлення екрана не менше 30 кадрів на секунду (FPS) на середньостатистичних пристроях користувачів. Для досягнення цього необхідно використовувати апаратне прискорення через WebGL та бібліотеку React Three Fiber.

2. Зручність користування. Інтерфейс (UI) повинен бути розроблений у сучасній стилістиці (зокрема, Glassmorphism) та відповідати принципам чуйного дизайну (Responsive Web Design). На мобільних пристроях інформаційні панелі повинні відображатися з використанням патерну «Bottom Sheet».

3. Надійність. Додаток повинен коректно обробляти стани завантаження (loading states) під час виконання асинхронних мережевих запитів до API та мати механізми перехоплення помилок у разі відсутності інтернет-з'єднання.

4. Масштабованість. Архітектура проєкту має будуватися на основі незалежних React-компонентів, що дозволить у майбутньому легко додавати нові об'єкти (наприклад, супутники, пояси астероїдів) без необхідності переписування ядра симуляції.

1.6 Основні системні обмеження та припущення

Під час проєктування та реалізації системи враховуються наступні технічні та інтеграційні обмеження:

1. Платформа та середовище виконання. Програмний продукт є вебдодатком, отже, його виконання повністю залежить від клієнтського браузера. Головним технічним обмеженням є обов'язкова підтримка стандарту WebGL та увімкнений JavaScript на пристрої користувача.

2. Інтеграційні обмеження. Модуль новин жорстко залежить від доступності стороннього сервісу – Spaceflight News API. Будь-які зміни у структурі відповіді (JSON) з боку провайдера API вимагатимуть внесення відповідних змін до клієнтського коду порталу.

3. Розгортання. Проєкт розгортається у хмарному середовищі Vercel, що накладає певні обмеження на розмір статичних файлів (текстур планет з високою роздільною здатністю), які повинні бути оптимізовані перед додаванням у репозиторій.

Висновки до розділу 1

У першому розділі було здійснено комплексне дослідження предметної області розроблення імерсивних освітньо-інформаційних вебдодатків з елементами тривимірної графіки для візуалізації астрономічних даних. Визначено ключові ролі користувачів (здобувачі освіти, викладачі, ентузіасти) та їхні сценарії взаємодії. Ідентифіковано основні проблеми середовища функціонування, серед яких фрагментованість інформації, статичність контенту та обмежена продуктивність наявних рішень у вебсередовищі. Проведено ґрунтовний аналітичний огляд існуючих аналогів на ринку, зокрема TheSkyLive 3D Simulator та 100,000 Stars. Виявлено їхні архітектурні й функціональні недоліки, такі як перевантаженість інтерфейсу, відсутність сучасних адаптивних патернів для мобільних пристроїв та неможливість інтеграції з динамічними джерелами даних. На основі виявлених недоліків було обґрунтовано напрями диференціації розроблюваного проєкту, які полягають у створенні односторінкового додатка (SPA), поєднанні 3D-сцени з модулем динамічних новин та застосуванні Mobile-First підходу. Результатом проведеного концептуального аналізу стала чітка формалізація завдання кваліфікаційної роботи. Сформовано перелік ключових функціональних вимог, представлених у форматі користувацьких історій (ініціалізація сцени, орбітальна анімація,

керування камерою, «розумна» пауза, модуль новин). Крім того, визначено нефункціональні вимоги до системи з акцентом на продуктивність рендерингу (не менше 30 FPS), надійність та зручність користувацького інтерфейсу, а також ідентифіковано системні й інтеграційні обмеження щодо використання WebGL та зовнішніх API.

РОЗДІЛ 2 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОЇ СИСТЕМИ

2.1 Концептуальне моделювання системи

На аналітико-модельному рівні здійснюється логічний перехід від описової характеристики предметної області до формалізованого подання структури майбутньої програмної системи. Головною метою цього етапу є побудова узгодженої концептуальної моделі вебпорталу, яка абстрактно відображає основні сутності, їхню взаємодію, поведінкові сценарії та обмеження без жорсткої прив’язки до конкретних мов програмування, фреймворків чи апаратних рішень.

Концептуальна архітектура розроблюваної системи базується на принципі розділення відповідальності (Separation of Concerns) і складається з трьох ключових абстрактних підсистем: підсистеми просторової візуалізації, підсистеми управління користувацьким інтерфейсом та підсистеми інтеграції зовнішніх даних.

2.1.1 Структурна модель системи

Структурна модель відображає склад системи та основні взаємозв'язки між її абстрактними компонентами (рис. 2.1). Систему можна декомпозиувати на такі логічні блоки:

1. Блок просторової візуалізації (3D Engine). Відповідає за математичний розрахунок координат, перетворення локальних систем відліку об'єктів у глобальну сцену, рендеринг полігональних сіток та обчислення освітлення. Цей блок функціонує в окремому циклі відмальовування (Render Loop).

2. Блок стану (State Controller). Центральний координатор, який зберігає глобальні змінні системи (наприклад, стан паузи, ідентифікатор обраного космічного об'єкта, статус завантаження мережевих даних). Він забезпечує синхронізацію між тривимірним простором та двовимірним інтерфейсом.

3. Блок користувацького інтерфейсу. Відповідає за перехоплення користувацьких подій (кліки, сенсорні жести) та виведення візуальної інформації (навігаційні меню, модальні вікна, інформаційні панелі) у вигляді накладеного поверх 3D-сцени шару.

4. Шлюз даних (API Gateway). Абстрактний модуль, що відповідає за асинхронне виконання HTTP-запитів до зовнішніх джерел (серверів новин), обробку відповідей та передачу форматованих даних до Блоку стану.

2.1.2 Поведінкова модель та сценарії взаємодії

Поведінкова модель описує реакцію системи на зовнішні події та послідовність дій під час виконання ключових користувацьких сценаріїв. Розглянемо концептуальну логіку двох основних процесів:

1. Сценарій "Фокусування на об'єкті". Користувач ініціює подію вибору (натискання на віртуальну модель планети). Блок просторової візуалізації використовує метод трасування променів (Raycasting) для визначення точки перетину з 3D-об'єктом. Ідентифікатор об'єкта передається до Блоку стану. Блок стану сигналізує UI-шару про необхідність відкрити інформаційну панель, одночасно подаючи команду камері змінити координати для наближення до обраної планети.

2. Сценарій "Керування часом симуляції". Користувач натискає кнопку паузи в UI-шарі. Блок стану змінює глобальний прапорець перебігу часу. Цикл відмальовування (Render Loop) у Блоці візуалізації продовжує роботу для збереження можливості обертати камеру, але функція додавання дельти часу (Time Delta) до орбітальних кутів планет блокується. Це забезпечує "розумну" зупинку об'єктів без візуальних розривів (стрибків).

2.1.3 Логічна модель даних

Для формалізації інформаційних об’єктів предметної області без прив’язки до фізичного зберігання (баз даних) розроблено логічну модель даних. Вона включає дві основні абстрактні сутності:

1. Сутність «Небесне тіло». Містить атрибути, необхідні як для розрахунку фізики, так і для виведення інформації. Атрибути: ID (унікальний ідентифікатор), Name (назва), Type (класифікація), OrbitalRadius (радіус орбіти), OrbitalSpeed (швидкість обертання), TextureReference (посилання на візуальну карту поверхні), Description (текстовий опис).

2. Сутність «Новинний запис». Формалізує структуру даних, що надходить від зовнішнього API. Атрибути: ArticleID, Title (заголовок), Summary (короткий зміст), PublishedAt (дата публікації), SourceURL (посилання на першоджерело).

2.1.4 Модель дослідження та критерії оцінювання (для перевірки продуктивності)

Оскільки розробка імерсивних вебдодатків пов'язана з ризиками падіння продуктивності, в межах проєктування системи було сформовано концептуальну модель дослідження.

1. Гіпотеза дослідження. Ізоляція високочастотних обчислень рендерингу 3D-сцени від процесів оновлення дерева об'єктів документа (DOM) дозволить уникнути блокування головного потоку браузера.

2. Змінні параметри. Кількість полігонів у моделях планет, частота оновлення стрічки новин, роздільна здатність текстур.

3. Критерії оцінювання. Основним показником стабільності системи визначено збереження частоти кадрів на рівні не нижче 30 FPS (Frames Per Second) під час виконання складних анімацій (наближення камери) та відсутність затримок (Lag) при відкритті інформаційних панелей.

2.2 Архітектура та проєктування системи

З огляду на вимоги до інтерактивності та безперервності 3D-візуалізації, базовим архітектурним патерном обрано односторінковий додаток (Single Page Application, SPA) з клієнтським рендерингом. Цей підхід дозволяє завантажити оболонку додатка один раз, після чого всі зміни інтерфейсу відбуваються динамічно шляхом оновлення DOM-дерева, що критично важливо для уникнення переривань у циклі відмальовування WebGL-сцени.

Система логічно поділяється на такі архітектурні шари, що візуалізуються за допомогою UML-діаграми компонентів, яка подана на рис. 2.1.

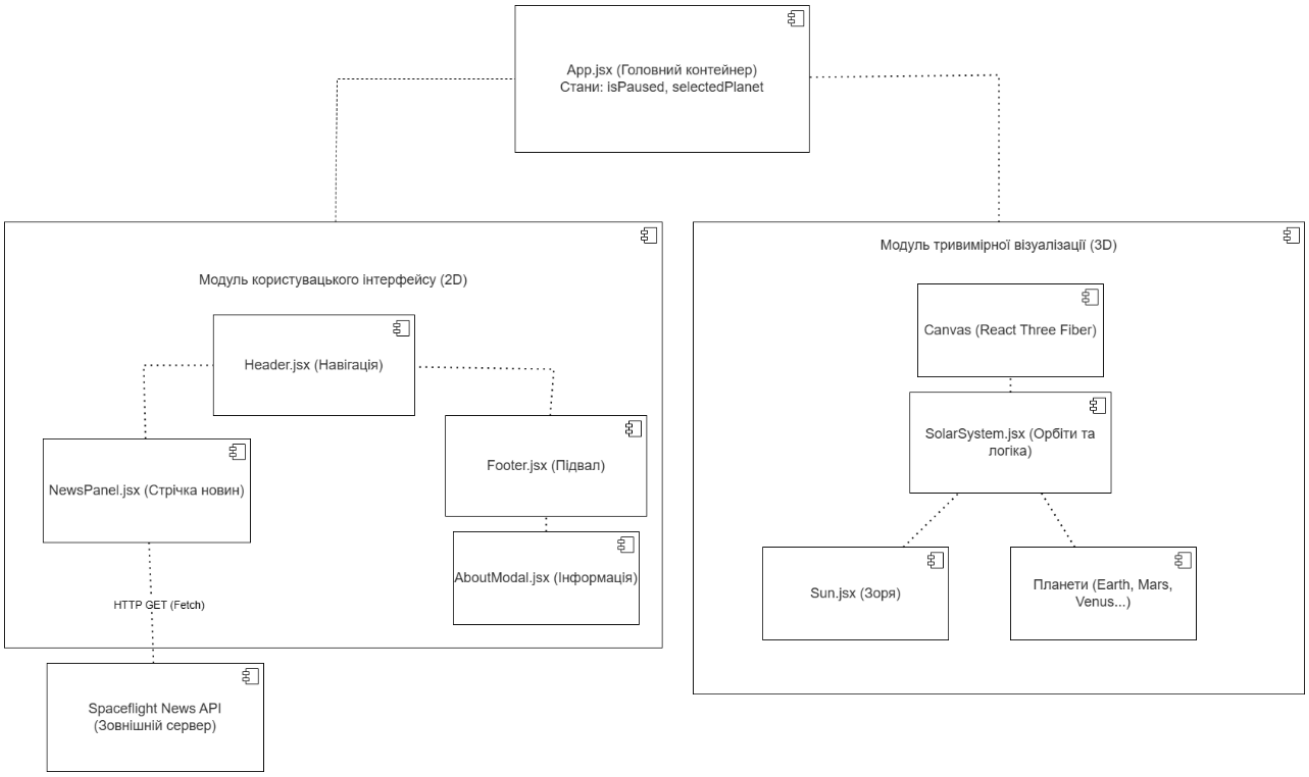

Рисунок 2.1 – UML-діаграма компонентів архітектури вебпорталу

1. Шар візуального інтерфейсу (2D UI Layer). Набір ієрархічних React-компонентів (Header, NewsPanel, AboutModal), що відповідають за навігацію та виведення текстового контенту.

2. Шар просторового рендерингу (3D Canvas Layer). Ізольоване середовище виконання просторової графіки. Використовує міст між React та Three.js для декларативного опису сцени.

3. Шар управління станом. Централізований механізм обміну даними між 2D та 3D шарами (наприклад, передача події кліку по планеті з 3D-сцени в UI-модальне вікно).

4. Інтеграційний шар (API Client). Модуль для виконання асинхронних HTTP-запитів до зовнішнього новинного REST API (Spaceflight News API).

2.2.2 Сценарно-поведінкові аспекти

Для моделювання динаміки взаємодії компонентів у часі побудовано UML-діаграму послідовності (рис. 2.2), яка описує критичний сценарій – завантаження актуальних новин.

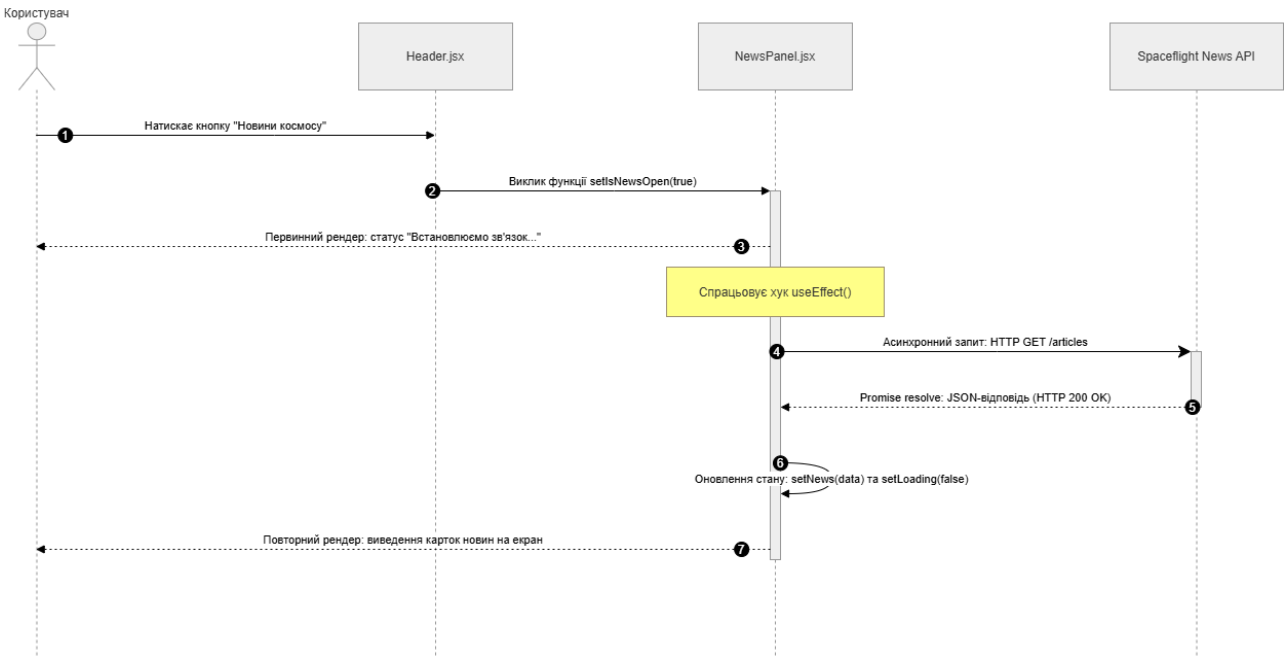

Рисунок 2.2 – UML-діаграма послідовності взаємодії з космічним API

1. Актор (Користувач) ініціює подію натисканням кнопки UI.

2. Шар управління станом змінює прапорець видимості панелі.

3. Інтеграційний шар формує асинхронний GET запит до зовнішнього API, переводячи UI у стан очікування (Loading State).

4. Зовнішній сервер повертає JSON-відповідь.

5. Система парсить дані, оновлює стан та рендерить стрічку новин.

2.2.3 Управління атрибутами якості та запис архітектурних рішень

Для забезпечення відповідності системи ключовим атрибутам якості (Performance, Usability), у процесі проєктування було використано практику фіксації архітектурних рішень, яка полягає в наступному.

Система повинна рендерити високополігональні об'єкти та обчислювати орбіти при стабільних 30+ FPS, одночасно підтримуючи складний 2D-інтерфейс. Класичний підхід (чистий Three.js + маніпуляції DOM) призводить до розсинхронізації станів та «спагеті-коду», тому варто використати екосистему React Three Fiber (R3F), яка будує граф сцени Three.js за допомогою алгоритму узгодження (reconciliation) React, що забезпечить значне підвищення стабільності архітектури та перенесення керування 3D-об'єктами в декларативну парадигму. Але існує ризик залежностні проєкту від сторонньої бібліотеки-моста.

2.2.4 Моделювання аспектів безпеки

Незважаючи на відсутність бази даних користувачів, система має зовнішні довірчі межі, що вимагають моделювання загроз. Головним вектором потенційної атаки є міжсайтовий скриптинг (Cross-Site Scripting, XSS) шляхом ін'єкції шкідливого коду через змінені відповіді від зовнішнього Spaceflight News API.

Заходи протидії: Архітектурно закладено відмову від прямої маніпуляції DOM (використання innerHTML). Всі вхідні текстові дані проходять через механізм автоматичного екранування (авто-ескейпінгу) бібліотеки React під час

побудови JSX-вузлів. Комунікація із зовнішніми сервісами дозволена виключно через захищений протокол HTTPS.

2.3 Реалізація програмного продукту

2.3.1 Обґрунтування вибору технологічного стека

Першим кроком на етапі програмної реалізації вебпорталу став вибір оптимальних інструментів для розробки клієнтської частини. Оскільки архітектура системи вимагає складної інтеграції 3D-середовища з динамічним 2D-інтерфейсом, було проведено порівняльний аналіз популярних підходів до побудови фронтенду.

Результати аналізу базового фреймворку наведено в табл. 2.1.

Таблиця 2.1 – Порівняльний аналіз технологій для реалізації клієнтської частини

Критерій порівняння	Vanilla JS (Чистий JavaScript)	Angular (Google)	Vue.js	React (Обраний варіант)
1	2	3	4	5
Архітектура додатка	Відсутня базова структура (імперативний підхід)	Строга архітектура MVC (важковагова)	Компонентна (декларативна)	Компонентна (гнучка, декларативна)
Інтеграція з 3D	Пряме використання (вимагає багато ручного та низькорівневого коду)	Обмежена інтеграція, відсутність популярних 3D-обгорток	Розвивається (через TresJS), але має меншу спільноту	Ідеальна інтеграція завдяки React Three Fiber (стандарт де-факто)
Керування станами	Ручне управління, високий ризик втрати синхронізації з DOM	Вбудоване, але має високий поріг входження (RxJS)	Зручне (вбудована реактивність)	Інтуїтивне та прогнозоване (за допомогою React Hooks)

Продовження таблиці 2.1

1	2	3	4	5
Робота з DOM (рендеринг)	Прямі маніпуляції (повільно при великій кількості даних)	Incremental DOM	Virtual DOM	Virtual DOM (висока продуктивність)
Синтаксис	HTML, CSS, JS у різних файлах	TypeScript + розширений HTML	Single-File Components (.vue)	JSX
Доцільність для даного проєкту	Низька (код швидко стане заплутаним через складність 3D-сцени)	Низька (фреймворк занадто громіздкий для SPA-порталу)	Середня (хороший фреймворк, але гірша підтримка 3D)	Висока (забезпечує ідеальний баланс між UI та 3D)

Як видно з табл. 2.1, бібліотека React є безальтернативним вибором для даного проєкту, оскільки вона забезпечує ідеальний баланс між продуктивністю (завдяки Virtual DOM), зручністю керування станами та має еталонну інтеграцію з інструментами 3D-графіки.

Окремої уваги заслуговує вибір базової графічної технології для модуля тривимірної візуалізації. Для визначення оптимального інструменту порівняно існуючі підходи до рендерингу графіки у вебсередовищі (табл. 2.2).

Порівняльний аналіз технологій (табл 2.2) свідчить, що використання повноцінних рушіїв (Unity) є недоцільним через великий розмір збірки, а чистий WebGL вимагає занадто низькорівневого коду. Оптимальним рішенням стало використання бібліотеки Three.js у зв'язці з React Three Fiber, що гарантує апаратне прискорення та збереження високого показника кадрів на секунду (FPS).

Таблиця 2.2 – Порівняльний аналіз технологій для реалізації 3D-графіки

Критерії порівняння	CSS3D (DOM-елементи)	Чистий WebGL (Native)	Unity / Unreal (WebAssembly)	Three.js + React Three Fiber (Обраний варіант)
Продуктивність (FPS)	Низька (гальмує при великій кількості об'єктів)	Дуже висока	Висока	Висока (зберігає апаратне прискорення WebGL)
Розмір збірки	Мінімальний	Мінімальний	Дуже великий (довге завантаження сторінки)	Середній (оптимізується через Tree-shaking у Vite)
Складність розробки	Низька	Дуже висока (потребує ручного написання шейдерів)	Середня (але потребує знання C# або C++)	Зручна (декларативний підхід, використання готових примітивів)
Інтеграція з React-інтерфейсом	Відмінна	Складна (ручна синхронізація станів)	Дуже складна (передача даних через мости повідомлень)	Ідеальна (3D-об'єкти працюють як звичайні React-компоненти)
Доцільність для SPA-порталу	Низька (не підходить для складних орбіт і текстур)	Низька (надмірна витрата часу на базові речі)	Низька (невиправдано важко для вебресурсу)	Висока (найкращий баланс між якістю, швидкістю та вагою)

2.4 Організація середовища розробки та система контролю версій

Основою для надійної реалізації програмного продукту є використання системи контролю версій Git. Це забезпечило збереження історії змін, можливість безпечного рефакторингу коду та підготовку фундаменту для подальшого автоматизованого розгортання (CI/CD).

Проєкт організовано у вигляді віддаленого репозиторію на платформі GitHub. Процес розробки супроводжувався осмисленим версіюванням та

логічною історією комітів (Commits). Коміти формувалися за принципом атомарності: кожна фіксація змін відповідала за реалізацію однієї конкретної функції (наприклад, feat: add Spaceflight News API integration або fix: resolve touch events conflict on mobile).

2.5 Структура первинного коду та відтворювана збірка

Для забезпечення відтворюваної збірки та миттєвого гарячого перезавантаження модулів під час написання коду використано сучасний інструмент збірки Vite. Налаштування збірки описані у конфігураційному файлі vite.config.js.

Структура директорій репозиторію була спроєктована у суворій відповідності до компонентної архітектури системи (додаток Б.1):

1. Директорія public/ – містить статичні медіаресурси, зокрема растрові зображення (Texture Maps) високої роздільної здатності для накладання на 3D-сфери планет.

2. Директорія src/ – є ядром проєкту та містить увесь вихідний код, логічно розділений на підкаталоги:

– assets/ – локальні стилі (index.css, App.css) та іконки.

– components/3d/ – інкапсулює логіку тривимірної графіки (наприклад, SolarSystem.jsx, Planet.jsx).

– components/ui/ – містить React-компоненти двовимірного інтерфейсу користувача (Header.jsx, NewsPanel.jsx, AboutModal.jsx).

– services/ – виділений логічний шар для програмної взаємодії із зовнішніми системами (функції-обгортки для роботи з Fetch API).

2.6 Реалізація прикладної логіки та математичних обчислень простору

Ключовим завданням реалізації стала розробка прикладної логіки 3D-візуалізації за допомогою екосистеми React Three Fiber. Кожне небесне тіло реалізоване як окремий функціональний React-компонент, що повертає геометричну сферу (<sphereGeometry />) та фізично коректний матеріал (<meshStandardMaterial />), на який накладається відповідна текстура за допомогою хука useTexture. Для реалізації орбітального руху використано хук useFrame, який викликається перед рендерингом кожного кадру (приблизно 60 разів на секунду). Математичний розрахунок нових координат планети у тривимірному просторі виконується за допомогою тригонометричних функцій. Якщо планета має радіус орбіти R, а кут її поточного відхилення дорівнює θ, то її координати в площині екліптики (осі X та Z) обчислюються як:

$$X = R \cdot cos(V \cdot T) \quad (1)$$

$$Z = R \cdot \sin(V \cdot T) \quad (2)$$

де R – радіус орбіти, V – швидкість, T – кастомний лічильник часу.

Кут θ постійно збільшується на величину, пропорційну швидкості планети та часу, що минув з попереднього кадру (Time Delta).

При активації користувачем сценарію «Розумна пауза», глобальний стан isPaused блокує додавання дельти часу до кута θ, що призводить до ідеально плавного зупинення орбітального руху без порушення координатної сітки.

2.7 Реалізація програмного інтерфейсу та взаємодії з API

Інтеграція динамічного контенту реалізована шляхом взаємодії із зовнішнім програмним інтерфейсом (Spaceflight News API) за протоколом HTTP. Прикладна логіка інкапсульована в компоненті NewsPanel.jsx.

Отримання даних ініціюється під час монтування компонента у DOM-дерево за допомогою хука useEffect. Для запобігання блокуванню головного

потоку застосовано асинхронні функції (async/await). Логіка обробки запиту включає три обов'язкові етапи:

1. Pending (Очікування). Активація стану isLoading = true, під час якого користувачеві демонструється індикатор завантаження або скелетний інтерфейс (Skeleton Screen).

2. Fulfilled (Успіх). Парсинг отриманого JSON-масиву даних, валідація полів (заголовок, посилання, дата) та збереження їх у локальний масив через setNews(data).

3. Rejected (Помилка). Перехоплення помилок мережі (блок try...catch) із виведенням відповідного сповіщення для користувача у разі недоступності зовнішнього сервера.

Такий підхід забезпечив створення надійного програмного застосунку зі зрозумілою структурою коду, повністю реалізованою бізнес-логікою та високим рівнем відмовостійкості інтеграційних механізмів.

Висновки до розділу 2

У другому розділі було здійснено повний цикл інженерного проєктування та програмної реалізації інтерактивного вебпорталу про космос, починаючи від концептуального моделювання до безпосереднього написання первинного коду.

На етапі аналітико-модельного проєктування розроблено абстрактні структурні, поведінкові та логічні моделі системи. Було обґрунтовано вибір архітектурного стилю SPA (Single Page Application), що забезпечило безперервність циклу відмальовування 3D-графіки без перезавантаження сторінки. Архітектуру додатка було декомпозировано на незалежні рівні: шар візуального 2D-інтерфейсу, шар просторового 3D-рендерингу, шар управління глобальним станом та інтеграційний шар.

На основі ґрунтовного порівняльного аналізу сучасних інструментів веброзробки визначено оптимальний технологічний стек. Бібліотеку React

обрано для побудови декларативного компонентного інтерфейсу, а Three.js у зв'язці з екосистемою React Three Fiber – як головний рушій для розрахунку тривимірної графіки з використанням апаратного прискорення WebGL. Для забезпечення зручності користування (Usability) було спроєктовано адаптивний дизайн з використанням патерну «Bottom Sheet» для мобільних пристроїв та візуальної стилістики «Glassmorphism». Окрім цього, на етапі проєктування закладено базові механізми безпеки, зокрема захист від міжсайтового скриптингу (XSS).

Під час програмної реалізації, яка супроводжувалася використанням системи контролю версій Git та сучасного інструменту збірки Vite, було успішно втілено прикладну бізнес-логіку порталу. Зокрема, імплементовано математичний розрахунок орбітального руху небесних тіл з використанням тригонометричних функцій, реалізовано алгоритм «розумної паузи» для зупинки часу симуляції без зміщення координатних осей, а також налаштовано надійний асинхронний зв’язок із зовнішнім REST API (Spaceflight News API) з належною обробкою станів очікування та помилок мережі.

РОЗДІЛ 3 ТЕСТУВАННЯ ТА ВЕРИФІКАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Оскільки розроблений продукт є односторінковим додатком (SPA) з акцентом на візуальну взаємодію, основною стратегією було обрано ручне функціональне тестування (Manual Testing) та тестування зручності використання (Usability Testing).

3.1 Функціональне тестування

Тестування проводилося у локальному середовищі розробки (на базі локального сервера Vite) з використанням браузерів Google Chrome та Safari. Головною метою було перевірити стабільність рендерингу 3D-сцени та коректність роботи логіки програми.

Основні тестові сценарії та їхні результати наведено у табл. 3.1.

Таблиця 3.1. – Опис тестових випадків та результати функціонального тестування

Назва тесту	Дія користувача	Очікуваний результат	Статус
1	2	3	4
Рендеринг 3D	Запуск додатка	Відображення Сонця та 8 планет з текстурами	Пройдено
"Розумна" пауза	Натискання «Зупинити час»	Миттєва зупинка об'єктів без стрибків координат	Пройдено
Фокус камери	Клік на модель планети	Плавний підліт та супроводження об'єкта	Пройдено
Синхронізація UI	Вибір планети	Автоматичне виведення даних із PLANET_DATA	Пройдено

Продовження таблиці 3.1

1	2	3	4
Інтеграція API	Відкриття панелі новин	Асинхронне отримання даних із зовнішнього сервера	Пройдено
Адаптивність (Mobile)	Запуск додатка на смартфоні (iOS/Android)	Коректне масштабування 3D-сцени, адаптація UI (Bottom Sheet), відсутність конфліктів сенсорного керування (свайп/скрол)	Пройдено
Зміна орієнтації екрана	Поворот смартфона з вертикального у горизонтальне положення	Коректна перемальовка 3D-сцени без спотворення пропорцій планет, адаптація розміщення панелей інтерфейсу	Пройдено
Масштабування жестами (Pinch-to-zoom)	Зведення та розведення двох пальців на екрані над 3D-сценою	Плавне наближення та віддалення камери від об'єктів без різких стрибків	Пройдено
Керування панеллю (Bottom Sheet)	Вертикальний свайп по інформаційній панелі	Панель плавно розгортається/згортається, не викликаючи випадкових натискань на 3D-сцену на задньому фоні	Пройдено

3.2 Тестування зручності та кросплатформність

Для перевірки коректної роботи на мобільних пристроях проводилося тестування на смартфонах з операційними системами iOS та Android. У ході тестування було виявлено конфлікт керування: при спробі прокрутити текст новин у панелі, браузер одночасно обертав 3D-сцену на задньому фоні.

Цю проблему було успішно вирішено шляхом застосування CSS-властивості touch-action: pan-y до контейнерів з текстом, що дозволило системі чітко розмежовувати скрол сторінки та жести керування 3D-простором.

3.3 Автоматизоване модульне тестування

Для підтвердження стабільності ключових логічних вузлів системи було розроблено набір автоматизованих модульних тестів (Unit Tests). Вони дозволяють ізольовано перевірити роботу окремих компонентів без необхідності ручного проклікування всього інтерфейсу.

Для перевірки UI-взаємодій здійснювалася програмна імітація подій користувача (наприклад, виклик fireEvent). Особливістю проєкту є наявність складної тривимірної графіки, тому для ізольованого тестування 3D-компонентів без запуску повноцінного браузерного оточення було застосовано спеціалізований інструмент @react-three/test-renderer.

Результати виконання ключових автоматизованих перевірок наведено у табл. 3.2. та рис. 3.1.

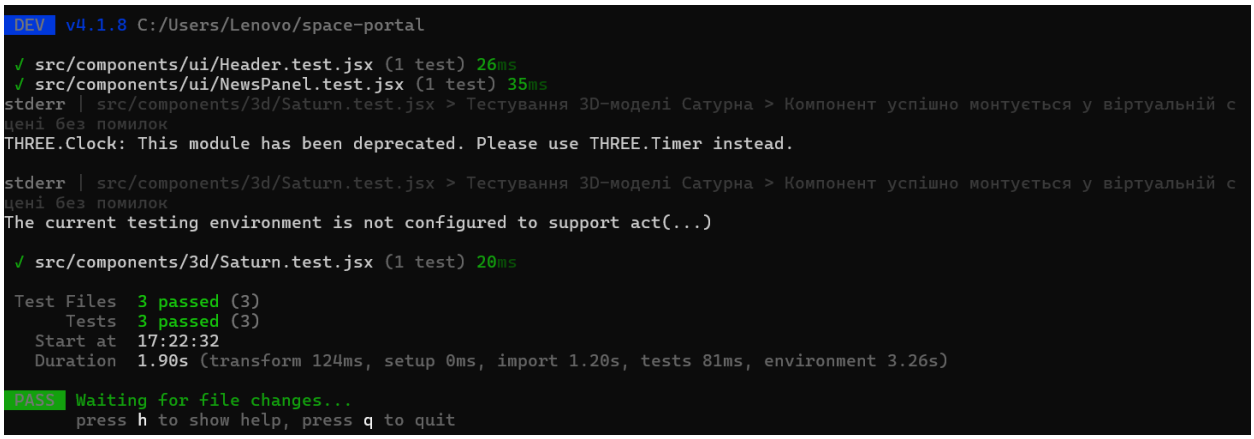

Рисунок 3.1. – Результати виконання автоматизованих тестів у середовищі Vitest

Таблиця 3.2 – Результати автоматизованого тестування компонентів

Модуль / Компонент	Тип перевірки	Умова або Дія (Тригер)	Очікуваний результат	Статус
UI-навігація	Обробка подій	Програмна імітація кліку (fireEvent) по кнопці «Новини космосу»	Коректний виклик функції setIsNewsOpen для зміни глобального стану	Успішно
Інтеграція API	Відображення станів	Рендеринг компонента з активним станом очікування даних	Поява індикатора статусу "Встановлюємо зв'язок..." у DOM-дереві	Успішно
3D-графіка	Smoke-тест (Віртуальний 3D-рендер)	Ізольований запуск 3D-моделі у віртуальному середовищі @react-three/test-renderer	Успішне монтування графа WebGL-сцени без критичних помилок рендерингу	Успішно

3.4 Розгортання програмного продукту

Для того щоб вебпортал був доступний кінцевим користувачам у мережі Інтернет, було налаштовано процес розгортання. Процес публікації проєкту складається з двох основних етапів: використання системи контролю версій та хостингу на хмарній платформі.

1. Система контролю версій. Увесь первинний код проєкту, конфігураційні файли та статичні ресурси (текстури планет) зберігаються у віддаленому репозиторії на платформі GitHub. Усі зміни в коді фіксуються за допомогою системи Git (команди git commit та git push), що забезпечує збереження історії розробки.

2. Хмарне розгортання на Vercel. Для публікації клієнтської частини додатка обрано платформу Vercel, яка ідеально оптимізована для роботи з React-додатками та збирачем Vite.

Процес розгортання є автоматизованим:

– Після завантаження оновленого коду на GitHub, платформа Vercel автоматично отримує доступ до нових файлів.

– Сервер Vercel запускає команду збірки (npm run build), яка оптимізує код та стискає зображення.

– Готовий продукт автоматично публікується на глобальній мережі доставки контенту (CDN), що забезпечує швидке завантаження 3D-моделей для користувачів з будь-якої точки світу.

– Системою автоматично генерується безпечний SSL-сертифікат для доступу до сайту за протоколом HTTPS.

3.5 Супровід та експлуатація

Після публікації проєкту на платформі Vercel розпочинається етап експлуатації. Для контролю стабільності роботи системи використовуються вбудовані інструменти аналітики та браузерні засоби розробника.

1. Моніторинг продуктивності. За допомогою панелі інструментів Vercel Analytics відстежується загальний трафік вебпорталу та швидкість завантаження сторінки користувачами.

2. Відстеження помилок. Для виявлення можливих проблем (наприклад, недоступність зовнішнього сервера Spaceflight News API або помилки завантаження текстур) використовується вкладка Network та Console у браузерних інструментах розробника (Chrome DevTools). Якщо API не відповідає, інтерфейс користувача коректно перехоплює помилку та виводить повідомлення про відсутність зв'язку.

Отримані під час тестування та експлуатації дані дозволяють планувати подальший розвиток вебпорталу: додавання нових супутників планет, реалізацію багатомовності (українська та англійська локалізації) та розширення інформаційної бази.

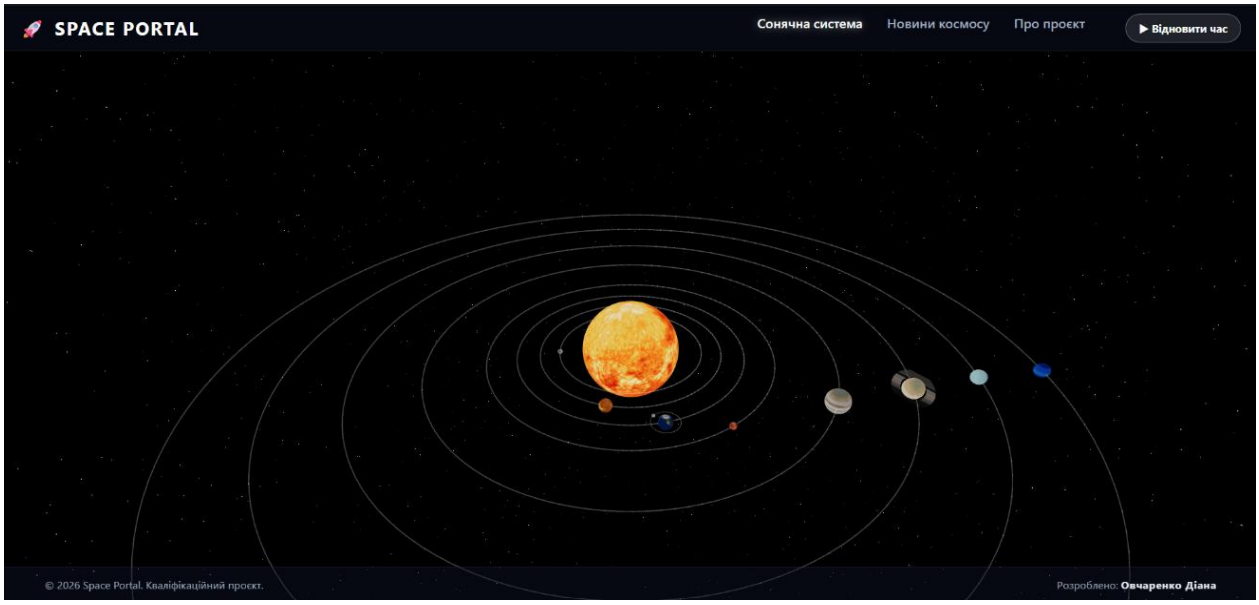

Рисунок 3.2 – Інтерфейс desktop сторінки

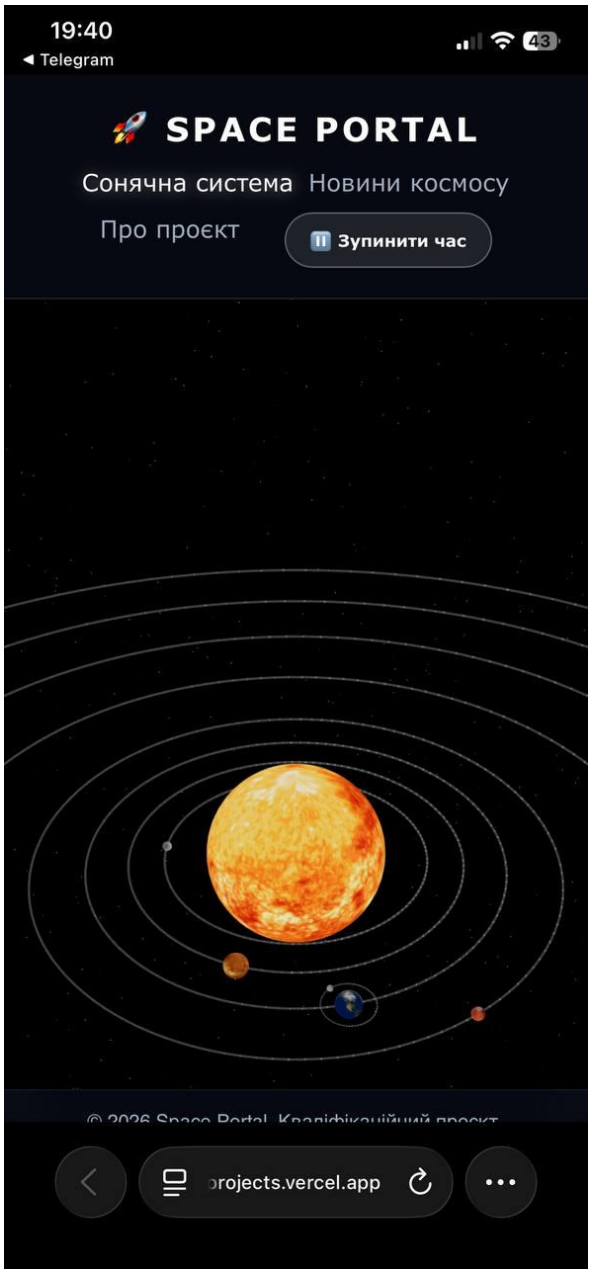

Рисунок 3.3 – Інтерфейс мобільної версії

Висновки до розділу 3

У третьому розділі було здійснено комплексний опис процесів тестування, розгортання та забезпечення експлуатації розробленого інтерактивного вебпорталу про космос.

Для забезпечення надійності програмного продукту було застосовано багаторівневу стратегію тестування. Завдяки ручному функціональному

тестуванню підтверджено коректність відображення 3D-моделей, безперебійність орбітальної анімації та стабільність асинхронного обміну даними зі Spaceflight News API. Високий інженерний рівень роботи підтверджується впровадженням автоматизованого модульного тестування за допомогою фреймворку Vitest та спеціалізованого інструменту @react-three/test-renderer, що дозволило ізольовано верифікувати монтування та стабільність складних тривимірних компонентів системи.

Окремо проведене тестування зручності використання (Usability Testing) на мобільних пристроях дозволило локалізувати та архітектурно усунути конфлікт сенсорних жестів між двовимірними інформаційними вікнами та просторовим керуванням 3D-камерою.

Процес публікації проєкту було оптимізовано шляхом використання системи контролю версій Git та хмарної платформи Vercel. Такий підхід дозволив реалізувати автоматизований конвеєр збірки (CI/CD), що гарантує миттєве оновлення продукту, швидке доставлення об'ємних 3D-ресурсів через глобальну мережу доставки контенту (CDN) та безпечний доступ користувачів за протоколом HTTPS. На етапі подальшої експлуатації для підтримки стабільності системи визначено використання засобів Vercel Analytics та інструментів моніторингу браузера.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне науково-прикладне завдання – спроєктовано та реалізовано сучасний інтерактивний вебпортал із тривимірною візуалізацією об'єктів Сонячної системи та модулем динамічного оновлення космічних новин. Виконане дослідження та програмна розробка повністю відповідають поставленій меті.

За результатами виконання роботи можна зробити такі висновки:

1. Проаналізовано предметну область та існуючі рішення. Виявлено, що більшість наявних астрономічних симуляторів (TheSkyLive 3D Simulator, 100,000 Stars) страждають від статичності контенту, перевантаженого інтерфейсу та відсутності якісної адаптації під мобільні пристрої. Це дозволило чітко сформулювати функціональні та нефункціональні вимоги до майбутнього програмного продукту з акцентом на кросплатформність та інтерактивність.

2. Спроєктовано архітектуру системи. Обґрунтовано доцільність створення односторінкового додатка (SPA) на базі компонентного підходу. Розроблено гібридну структурну модель, яка ізолює складні обчислення 3D-графіки від операцій з оновленням двовимірного інтерфейсу та гарантує збереження високої продуктивності системи.

3. Обґрунтовано вибір технологічного стека. Для реалізації клієнтської частини обрано бібліотеку React, яка забезпечує гнучке керування станами через Virtual DOM. В якості базової технології для рендерингу простору використано WebGL у поєднанні з бібліотекою Three.js та екосистемою React Three Fiber, що дозволило зберегти апаратне прискорення графіки.

4. Здійснено програмну реалізацію логіки вебпорталу. Успішно імплементовано математичні моделі орбітального руху з функціоналом «розумної паузи». Створено сучасний UI-інтерфейс, адаптований для мобільних пристроїв за допомогою патерну «Bottom Sheet». Вирішено проблему інформаційної ізольованості симуляторів шляхом інтеграції асинхронних запитів

до зовнішнього REST API (Spaceflight News API) для виведення актуальної стрічки новин аерокосмічної галузі.

5. Проведено комплексне тестування продукту. Виконано перевірку працездатності критичних функцій системи за допомогою ручного функціонального тестування. Надійність ключових логічних вузлів та стабільність монтування складних 3D-моделей підтверджено за допомогою автоматизованих модульних тестів (у середовищі Vitest з використанням @react-three/test-renderer). Окремо проведено Usability-тестування на мобільних пристроях, за результатами якого оптимізовано обробку сенсорних жестів і повністю усунено конфлікти між прокручуванням тексту та панорамуванням 3D-камери.

6. Реалізовано розгортання та публікацію проєкту. Налаштовано процес збірки та публікації вебпорталу з використанням системи контролю версій Git, віддаленого репозиторію GitHub та хмарної платформи Vercel. Це забезпечило автоматизацію оновлень, швидке доставлення об'ємних 3D-ресурсів через глобальну мережу доставки контенту (CDN) та безпечний доступ до додатка за протоколом HTTPS.

Розроблений інтерактивний вебпортал є повністю працездатним програмним продуктом, готовим до експлуатації кінцевими користувачами. Його матеріали можуть бути впроваджені як інноваційний наочний посібник у закладах освіти для вивчення астрономії. Крім того, використані в роботі підходи до інтеграції React із Three.js можуть слугувати практичним шаблоном для веброзробників при створенні імерсивних освітніх застосунків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мартин Р. Чиста архітектура. Мистецтво розробки програмного забезпечення. Київ : Фабула, 2019. 416 с.

2. Норман Д. Дизайн звичних речей / пер. з англ. Київ : ArtHuss, 2021. 360 с.

3. Фрімен Е., Робсон Е. Вивчаємо програмування на JavaScript / пер. з англ. Київ : Фабула, 2021. 640 с.

4. Дакетт Д. HTML і CSS. Розробка та створення веб-сайтів / пер. з англ. Харків : Клуб сімейного дозвілля, 2017. 480 с.

5. Фленеган Д. JavaScript: Повний посібник / пер. з англ. Київ : Діалектика, 2021. 720 с.

6. Чорний М. Д., Коваль О. В. Сучасні підходи до розробки односторінкових вебзастосунків (SPA) на базі бібліотеки React. Комп'ютерно-інтегровані технології. 2023. № 4. С. 85–92.

7. Резніченко В. А. Моделювання та верифікація програмних систем імерсивного типу. Вчені записки ТНУ імені В.І. Вернадського. Серія: Технічні науки. 2024. Т. 35, № 2. С. 141–147.

8. React – A JavaScript library for building user interfaces. Офіційна документація. URL: https://react.dev/ (дата звернення: 10.03.2026).

9. Three.js – JavaScript 3D Library. Офіційна документація розробника. URL: https://threejs.org/docs/ (дата звернення: 12.03.2026).

10. React Three Fiber Documentation. Посібник з інтеграції Three.js та React. URL: https://docs.pmnd.rs/react-three-fiber/ (дата звернення: 16.04.2026).

11. MDN Web Docs. WebGL: 2D and 3D graphics for the web. Специфікація Mozilla Foundation. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API (дата звернення: 15.03.2026).

12. Spaceflight News API (SNAPI). Офіційна документація для розробників. URL: https://api.spaceflightnewsapi.net/v4/docs/ (дата звернення: 20.04.2026).

13. Vite – Next Generation Frontend Tooling. Керівництво з налаштування збірки. URL: https://vitejs.dev/ (дата звернення: 01.02.2026).

14. Vercel Documentation. Deploying standard front-end frameworks and SPAs. URL: https://vercel.com/docs (дата звернення: 10.05.2026).

15. GitHub Actions Documentation. Офіційний посібник з налаштування CI/CD. URL: https://docs.github.com/en/actions (дата звернення: 10.05.2026).

16. TheSkyLive: The Interactive Space Simulator. Астрономічний вебресурс. URL: https://theskylive.com/3d-solar-system (дата звернення: 05.04.2026).

17. Vitest – Blazing Fast Unit Test Framework. Офіційна документація для розробників. URL: https://vitest.dev/ (дата звернення: 12.02.2026).

18. NASA Open APIs. Solar System Dynamics and Exploration Data. URL: https://api.nasa.gov/ (дата звернення: 18.04.2026).

ДОДАТКИ

ДОДАТОК А

Посилання на Git-репозиторій: https://github.com/diatefi/my-space-portal/tree/main

Посилання на сайт: https://my-space-portal-diatefis-projects.vercel.app/

ДОДАТОК Б

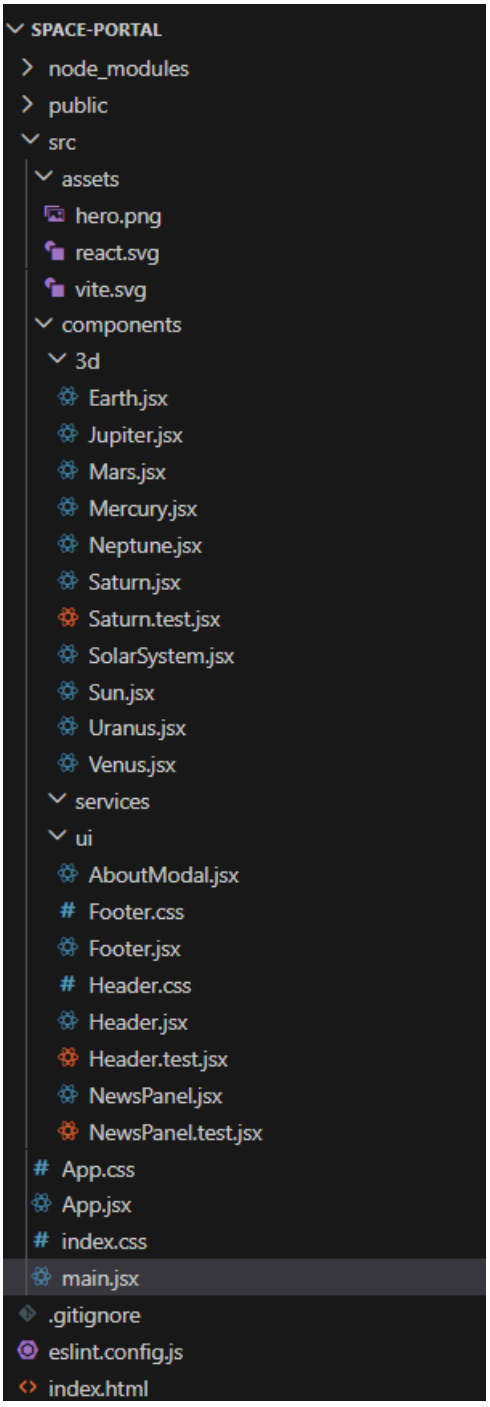

Рисунок Б.1 – Структура каталогів та компонентів клієнтської частини вебпорталу

ДОДАТОК В

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App.jsx';
// Підключаємо глобальні стилі
import './index.css';

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
);
```

Лістинг В.1 – main.jsx

```
import { useState } from 'react';
import { Canvas } from '@react-three/fiber';
import SolarSystem from './components/3d/SolarSystem.jsx';
import Header from './components/ui/Header.jsx';
import NewsPanel from './components/ui/NewsPanel.jsx';
import AboutModal from './components/ui/AboutModal.jsx';
import Footer from './components/ui/Footer.jsx';
import './App.css';



// База даних планет для інформаційної панелі
const PLANET_DATA = {
  sun: { title: "Сонце", Type: "Жовтий карлик", dist: "Центр системи", fact: "Зоря, навколо якої обертаються всі об'єкти Сонячної системи. Складається переважно з водню та гелію." },
  mercury: { title: "Меркурій", Type: "Планета земної групи", dist: "57.9 млн км від Сонця", fact: "Найменша і найближча до Сонця планета. Рік тут триває всього 88 земних діб." },
  venus: { title: "Венера", Type: "Планета земної групи", dist: "108.2 млн км від Сонця", fact: "Найгарячіша планета Сонячної системи через потужний парниковий ефект. Обертається у зворотний бік." },
```

```
        earth: { title: "Земля", Type: "Планета земної групи", dist: "149.6 млн км від Сонця", fact: "Єдина відома планета, на якій існує життя та рідка вода на поверхні." },
        mars: { title: "Марс", Type: "Планета земної групи", dist: "227.9 млн км від Сонця", fact: "Червона планета. Тут розташований Олімп — найвищий згаслий вулкан у Сонячній системі (26 км)." },
        jupiter: { title: "Юпітер", Type: "Газовий гігант", dist: "778.5 млн км від Сонця", fact: "Найбільша планета системи. Його Велика червона пляма — це гігантський шторм, що вирує сотні років." },
        saturn: { title: "Сатурн", Type: "Газовий гігант", dist: "1.43 млрд км від Сонця", fact: "Відомий своєю дивовижною системою кілець, що складаються з мільярдів частинок льоду та пилу." },
        uranus: { title: "Уран", Type: "Крижаний гіган", dist: "2.87 млрд км від Сонця", fact: "Єдина планета, яка обертається навколо Сонця, «лежачи на боці». Має блідо-блакитний колір." },
        neptune: { title: "Нептун", Type: "Крижаний гігант", dist: "4.5 млрд км від Сонця", fact: "Найделіша планета системи. Тут дмуть найсильніші вітри в Сонячній системі — до 2100 км/год." },
        moon: { title: "Місяць", Type: "Природний супутник", dist: "384 400 км від Землі", fact: "Єдиний природний супутник Землі та перше і поки єдине позаземне тіло, на якому побувала людина." }
      };

      function App() {
        const [isPaused, setIsPaused] = useState(false);
        const [selectedPlanet, setSelectedPlanet] = useState(null);

        // Новий стан для панелі новин
        const [isNewsOpen, setIsNewsOpen] = useState(false);
        const [isAboutOpen, setIsAboutOpen] = useState(false);

        return (
          <div style={{ width: '100vw', height: '100vh', position: 'relative', backgroundColor: '#000' }}>
```

```
            {/* 3. Передаємо керування у Header */}
            <Header
              isPaused={isPaused}
              setIsPaused={setIsPaused}
              setIsNewsOpen={setIsNewsOpen}
              setIsAboutOpen={setIsAboutOpen}
            />

            <NewsPanel isOpen={isNewsOpen} onClose={() => setIsNewsOpen(false)} />

            {/* 4. Виводимо модальне вікно */}
            <AboutModal isOpen={isAboutOpen} onClose={() => setIsAboutOpen(false)} />

            {selectedPlanet && (
           <div className="info-panel animate-slide">

            <button className="close-btn" onClick={() => setSelectedPlanet(null)}>✕</button>
            <h2>{PLANET_DATA[selectedPlanet].title || PLANET_DATA[selectedPlanet].title2}</h2>
            <div className="planet-badge">{PLANET_DATA[selectedPlanet].Type}</div>
            <p><strong>Відстань:</strong> {PLANET_DATA[selectedPlanet].dist}</p>
            <p className="planet-fact">{PLANET_DATA[selectedPlanet].fact}</p>
           </div>
          )}

            <Canvas camera={{ position: [0, 60, 150], fov: 60 }}>
              <SolarSystem isPaused={isPaused} selectedPlanet={selectedPlanet} setSelectedPlanet={setSelectedPlanet} />
            </Canvas>
            <Footer />
           </div>
          );
         }
```

```
export default App;
```

Лістинг В.2 – App.jsx

```
/* src/index.css */
* {
  box-sizing: border-box;
}

html, body {
  margin: 0;
  padding: 0;
  width: 100%;
  height: 100%;
  overflow: hidden;
  background-color: #000;
}
/* Адаптація для мобільних пристроїв */
@media (max-width: 768px) {
  .news-panel {
    width: 100%;
    right: 0;
    bottom: 0;
    border-radius: 20px 20px 0 0;
    height: 80vh;
    top: 150px;
  }

  .header {
    flex-direction: column;
    font-size: 14px;
  }
}
```

Лістинг В.3 – index.css

```
/* Інформаційна панель */
.info-panel {
  position: absolute;
  right: 20px;
  top: 100px;
  width: 350px;
  padding: 25px;
  color: white;
  z-index: 999;
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  padding-top: 40px;

  /* Ефект скла */
/*   background: rgba(15, 23, 42, 0.55);
  backdrop-filter: blur(12px);
  -webkit-backdrop-filter: blur(12px);
  border: 1px solid rgba(255, 255, 255, 0.1);
  border-radius: 16px;
  box-shadow: 0 10px 30px rgba(0, 0, 0, 0.5); */
  background: rgba(15, 23, 42, 0.95);

}

.info-panel h2 {
  margin-top: 0;
  font-size: 20px;
  letter-spacing: 1px;
    margin-bottom: 10px;
}

.planet-badge {
  display: inline-block;
  background: rgba(59, 130, 246, 0.3);
  border: 1px solid #3b82f6;
```

```
  padding: 4px 10px;
  font-size: 12px;
  border-radius: 20px;
  margin-bottom: 15px;
  font-weight: 600;
}

.planet-fact {
  line-height: 1.6;
  color: #e2e8f0;
  border-top: 1px solid rgba(255, 255, 255, 0.1);
  padding-top: 15px;
}

/* Кнопка закриття */
.close-btn {
  position: absolute;
  top: 30px;
  right: 30px;
  background: none;
  border: none;
  color: #a0aec0;
  font-size: 18px;
  cursor: pointer;
  transition: color 0.2s;
  z-index: 10;
}

.close-btn:hover {
  color: white;
}

/* Анімація плавної появи */
.animate-slide {
```

```
  animation: slideIn 0.4s cubic-bezier(0.16, 1, 0.3, 1) forwards;
}

@keyframes slideIn {
  from {
    transform: translateX(50px);
    opacity: 0;
  }
  to {
    transform: translateX(0);
    opacity: 1;
  }
}
/* Панель новин (виїжджає зліва) */
.news-panel {
  position: absolute;
  left: 20px;
  top: 100px;
  width: 380px;
  max-height: calc(100vh - 140px);
  padding: 25px;
  color: white;
  z-index: 999;
  font-family: 'Segoe UI', Tahoma, sans-serif;

  background: rgba(15, 23, 42, 0.7);
  backdrop-filter: blur(15px);
  -webkit-backdrop-filter: blur(15px);
  border: 1px solid rgba(255, 255, 255, 0.1);
  border-radius: 16px;
  box-shadow: 0 10px 30px rgba(0, 0, 0, 0.5);

  /* Щоб можна було скролити новини, якщо їх багато */
  overflow-y: auto;
```

```
}

/* Скролбар для панелі новин */
.news-panel::-webkit-scrollbar {
  width: 6px;
}
.news-panel::-webkit-scrollbar-thumb {
  background: rgba(255, 255, 255, 0.2);
  border-radius: 10px;
}

.news-panel h2 {
  margin-top: 0;
  font-size: 24px;
  border-bottom: 1px solid rgba(255, 255, 255, 0.1);
  padding-bottom: 15px;
  margin-bottom: 20px;
}

/* Картка окремої новини */
.news-card {
  background: rgba(0, 0, 0, 0.4);
  border-radius: 12px;
  margin-bottom: 15px;
  overflow: hidden;
  transition: transform 0.2s;
}

.news-card:hover {
  transform: translateY(-3px);
  background: rgba(255, 255, 255, 0.05);
}

.news-img {
```

```
  width: 100%;
  height: 140px;
  background-size: cover;
  background-position: center;
}

.news-text {
  padding: 15px;
}

.news-text h3 {
  margin: 0 0 10px 0;
  font-size: 15px;
  line-height: 1.4;
  color: #e2e8f0;
}

.read-more {
  display: inline-block;
  color: #3b82f6;
  text-decoration: none;
  font-size: 13px;
  font-weight: 600;
  margin-top: 5px;
}

.read-more:hover {
  color: #60a5fa;
  text-decoration: underline;
}

.loading-text {
  text-align: center;
  color: #a0aec0;
```

```
  font-style: italic;
}

/* Анімація появи зліва */
.animate-slide-left {
  animation: slideInLeft 0.4s cubic-bezier(0.16, 1, 0.3, 1) forwards;
}

@keyframes slideInLeft {
  from {
    transform: translateX(-50px);
    opacity: 0;
  }
  to {
    transform: translateX(0);
    opacity: 1;
  }
}
/* Вікно "Про проєкт" */
.modal-overlay {
  position: fixed;
  top: 0;
  left: 0;
  width: 100vw;
  height: 100vh;
  background: rgba(0, 0, 0, 0.6);
  backdrop-filter: blur(8px); /* Розмиває всю сцену на фоні */
  -webkit-backdrop-filter: blur(8px);
  z-index: 2000; /* Має бути вище за всі інші панелі */
  display: flex;
  align-items: center;
  justify-content: center;
}
```

```
.about-modal {
  background: rgba(15, 23, 42, 0.85);
  border: 1px solid rgba(255, 255, 255, 0.2);
  border-radius: 16px;
  padding: 40px;
  width: 500px;
  max-width: 90%;
  color: #e2e8f0;
  position: relative;
  box-shadow: 0 20px 50px rgba(0, 0, 0, 0.6);
  font-family: 'Segoe UI', Tahoma, sans-serif;
}

.about-modal h2 {
  color: white;
  margin-top: 0;
  font-size: 28px;
  border-bottom: 1px solid rgba(255, 255, 255, 0.1);
  padding-bottom: 15px;
  margin-bottom: 20px;
}

.about-modal h3 {
  color: white;
  margin-top: 25px;
  font-size: 18px;
}

.tech-list {
  padding-left: 20px;
  line-height: 1.8;
}

.tech-list li {
```

```
  margin-bottom: 8px;
}

.author-info {
  margin-top: 30px;
  padding-top: 20px;
  border-top: 1px solid rgba(255, 255, 255, 0.1);
  text-align: center;
  color: #a0aec0;
}

.author-info strong {
  color: white;
  font-size: 18px;
}

.author-info p {
  margin: 5px 0;
}

/* Плавна поява вікна */
.animate-fade-in {
  animation: fadeIn 0.3s ease-out forwards;
}

@keyframes fadeIn {
  from {
    opacity: 0;
    transform: scale(0.95);
  }
  to {
    opacity: 1;
    transform: scale(1);
  }
```

```
}
@media (max-width: 768px) {
  .info-panel {
    position: absolute;
    bottom: 0 !important;
    left: 0;
    width: 100% !important;
    top: auto !important;
    right: auto;

    border-radius: 20px 20px 0 0;
    padding: 25px 20px 50px 20px !important;
    max-height: 50vh !important;
    box-sizing: border-box !important;

    overflow-y: auto !important;
    overscroll-behavior: contain !important;
    -webkit-overflow-scrolling: touch !important;
    pointer-events: auto !important;
    touch-action: pan-y !important;
    z-index: 9999 !important;
  }

  /*після останнього тексту завжди буде порожнє місце */
  .info-panel > *:last-child {
    margin-bottom: 40px !important;
  }
}

  /* Адаптація панелі планети для ГОРИЗОНТАЛЬНОГО режиму телефонів */
@media (max-height: 500px) and (orientation: landscape) {
  .info-panel {
    position: absolute;
    bottom: 0;
```

```
  left: 0;
  width: 100% !important;
  height: 60dvh !important;

  /* Відступ з урахуванням безпечної зони айфона: */
  padding: 15px 20px calc(15px + env(safe-area-inset-bottom)) 20px !important;
  border-radius: 15px 15px 0 0;

  /* Дозволяємо системі самій розібратися зі скролом: */
  overflow-y: auto !important;
  -webkit-overflow-scrolling: touch !important;
  z-index: 9999 !important;

}

/* Зменшуємо шрифти всередині панелі, щоб більше влазило */
.info-panel h2 {
  font-size: 18px !important;
  margin-bottom: 5px !important;
}

.planet-badge {
  padding: 2px 8px !important;
  font-size: 10px !important;
  margin-bottom: 8px !important;
}

.planet-fact {
  font-size: 12px !important;
  line-height: 1.4 !important;
  padding-top: 8px !important;
}

/* Зменшуємо та підсуваємо кнопку закриття (хрестик) */
```

```
.close-btn {
  top: 15px !important;
  right: 15px !important;
  font-size: 16px !important;
}

/* --- Адаптація панелі НОВИН для горизонтального режиму --- */
.news-panel {
  top: 45px !important; /* Підтягуємо під саму вузьку шапку */
  left: env(safe-area-inset-left, 15px) !important; /* Відступ від лівого краю екрана з урахуванням "чубчика" айфона */
  width: 300px !important; /* Робимо панель акуратною, щоб не перекривала весь космос */
  max-height: calc(100dvh - 60px) !important; /* Висота до самого низу екрана */
  padding: 15px !important;
  z-index: 9999 !important;

  /* Запобіжник для iOS Safari: вимикаємо розмиття, щоб скрол новин не глючив */
  background: rgba(15, 23, 42, 0.98) !important;
  backdrop-filter: none !important;
  -webkit-backdrop-filter: none !important;
}

/* Зменшуємо головний заголовок новин */
.news-panel h2 {
  font-size: 18px !important;
  margin-bottom: 12px !important;
  padding-bottom: 8px !important;
}

/* Робимо картинки новин нижчими, щоб більше інформації помістилося на екрані */
.news-img {
  height: 80px !important;
}
```

```
        /* Зменшуємо відступи та текст у самих картках новин */
        .news-text {
          padding: 10px !important;
        }

        .news-text h3 {
          font-size: 13px !important;
          margin-bottom: 5px !important;
        }

        .read-more {
          font-size: 11px !important;
        }

        /* --- Адаптація вікна "Про проєкт" для горизонтального режиму --- */
        .about-modal {
          max-height: 85dvh !important; /* Обмежуємо висоту, щоб вікно не вилазило за екран */
          width: 85% !important;
          padding: 20px 20px calc(20px + env(safe-area-inset-bottom)) 20px !important; /* Зменшуємо величезні відступи */

          /* Вмикаємо скрол для контенту */
          overflow-y: auto !important;
          -webkit-overflow-scrolling: touch !important;
        }

        /* Зменшуємо головний заголовок */
        .about-modal h2 {
          font-size: 20px !important;
          margin-bottom: 12px !important;
          padding-bottom: 10px !important;
        }
```

```
  /* Зменшуємо підзаголовки ("Використані технології:") */
  .about-modal h3 {
    font-size: 16px !important;
    margin-top: 15px !important;
  }

  /* Робимо список технологій компактнішим */
  .tech-list {
    font-size: 13px !important;
    line-height: 1.5 !important;
  }

  .tech-list li {
    margin-bottom: 6px !important;
  }
  .author-info {
    margin-top: 15px !important;
    padding-top: 15px !important;
  }

  .author-info strong {
    font-size: 15px !important;
  }
}
```

Лістинг В.4 – App.css

```
export default function AboutModal({ isOpen, onClose }) {
 if (!isOpen) return null;

 return (
  // Темний фон, який перекриває весь екран
  <div className="modal-overlay" onClick={onClose}>


   <div className="about-modal animate-fade-in" onClick={(e) => e.stopPropagation()}>
    <button className="close-btn" onClick={onClose}>✕</button>

    <h2>Про портал</h2>

    <div className="about-content">
     <p>Цей інтерактивний космічний вебпортал розроблено як кваліфікаційний проєкт.</p>
     <p>Головна мета - візуалізація Сонячної системи у 3D-просторі з використанням реальних текстур та надання актуальних новин у сфері дослідження космосу.</p>

     <h3>Використані технології:</h3>
     <ul className="tech-list">
      <li><strong>React</strong> - побудова інтерфейсу та управління станами.</li>
      <li><strong>Three.js & React Three Fiber</strong> - 3D-графіка, математика орбіт та анімація.</li>
      <li><strong>Spaceflight News API</strong> - інтеграція стрічки новин у реальному часі.</li>
     </ul>

     <div className="author-info">
      <p>Розробник: <strong>Овчаренко Діана</strong></p>
      <p>2026 рік</p>
     </div>
    </div>
   </div>
```

```
    </div>
  );
}
```

Лістинг В.5 – AboutModal.jsx

```
import './Header.css';

// Приймаємо setIsAboutOpen
export default function Header({ isPaused, setIsPaused, setIsNewsOpen, setIsAboutOpen }){
  return (
    <header className="main-header">

      <div className="logo">
        <span className="logo-icon">🚀</span> SPACE PORTAL
      </div>

      <nav className="nav-links">
        <a href="#solar-system" className="active">Сонячна система</a>

        <a href="#news" onClick={(e) => {
          e.preventDefault();
          setIsNewsOpen(true);
        }}>
          Новини космосу
        </a>

        {/* Додаємо onClick сюди */}
        <a href="#about" onClick={(e) => {
          e.preventDefault();
          setIsAboutOpen(true);
        }}>
          Про проєкт
        </a>
```

```
      <button className="pause-btn" onClick={() => setIsPaused(!isPaused)}>
        {isPaused ? '▶ Відновити час' : '⏸ Зупинити час'}
      </button>

    </nav>

  </header>

 );
}
```

Лістинг В.6 – Header.jsx

```
/* Робимо хедер фіксованим поверх екрана */
.main-header {
 position: absolute;
 top: 0;
 left: 0;
 display: flex;
 justify-content: space-between;
 align-items: center;
 position: relative !important;
 padding: 10px 20px;
 margin: 0 auto !important;
 box-sizing: border-box !important;
 width: 100% !important;

 /* Ефект матового скла */
 background: rgba(15, 23, 42, 0.4); /* Напівпрозорий темно-синій фон */
 backdrop-filter: blur(10px);       /* Розмиття того, що під хедером */
 -webkit-backdrop-filter: blur(10px);
 border-bottom: 1px solid rgba(255, 255, 255, 0.1);

 /* Щоб хедер завжди був поверх 3D сцени */
 z-index: 1000;
 box-sizing: border-box;
```

```
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;



}

.header-wrapper {
  width: 100%;
  display: flex;
  justify-content: space-between;
  align-items: center;

  padding: 0 20px !important;
  box-sizing: border-box;
}

.logo {
  font-size: 22px;
  font-weight: 700;
  color: #ffffff;
  letter-spacing: 2px;
  display: flex;
  align-items: center;
  gap: 10px;
}

.nav-links {
  display: flex;
  gap: 30px;

}

.nav-links a {
  color: #a0aec0;
```

```
  text-decoration: none;
  font-size: 16px;
  font-weight: 500;
  transition: color 0.3s ease;
}

.nav-links a:hover,
.nav-links a.active {
  color: #ffffff;
  text-shadow: 0 0 10px rgba(255, 255, 255, 0.5);
}

.pause-btn {
  background: rgba(255, 255, 255, 0.1);
  color: #fff;
  border: 1px solid rgba(255, 255, 255, 0.3);
  padding: 8px 16px;
  border-radius: 20px;
  cursor: pointer;
  font-family: inherit;
  font-weight: bold;
  transition: all 0.3s ease;
  margin-left: 20px; /* Відступ від звичайних посилань */
}

.pause-btn:hover {
  background: rgba(255, 255, 255, 0.25);
  box-shadow: 0 0 10px rgba(255, 255, 255, 0.2);
}
/* 1. Адаптація для звичайних вертикальних телефонів (Portrait) */
@media (max-width: 768px) {
  .main-header {
    flex-direction: column !important; /* Ставить елементи один під одним */
    height: auto !important;          /* Дозволяє шапці розтягнутися */
```

```
    padding: 15px 20px !important;
    gap: 10px !important;              /* Відстань між логотипом і меню */
  }

  .nav-links {
    display: flex;
    flex-wrap: wrap;
    justify-content: center;
    gap: 10px;
    padding: 0 10px !important;
  }

  .pause-btn {
    margin-left: 0 !important; /* Забираємо зайвий відступ на телефонах */
  }
}

/* 2. Адаптація ДЛЯ ГОРИЗОНТАЛЬНОГО екрана (Landscape) */
@media (max-height: 500px) and (orientation: landscape) {
  .main-header {
    flex-direction: row !important; /* ставимо все в 1 рядок */
    justify-content: space-between !important;
    padding: 2px 15px !important; /* Відступи мінімальні */
    height: 40px !important; /* Дуже вузька шапка */
  }

  .header-wrapper {
    flex-direction: row !important;
  }

  /* Зменшуємо логотип */
  .logo {
    font-size: 14px !important;
    margin: 0 !important;
```

```
  white-space: nowrap !important; /* Забороняємо розривати слова */
 }

 /* Вибудовуємо меню в рядок */
 .nav-links {
  display: flex !important;
  flex-direction: row !important;
  flex-wrap: nowrap !important; /* Жодних переносів на новий рядок */
  align-items: center !important;
  gap: 12px !important;
  padding: 0 !important;
 }

 /* Зменшуємо текст посилань */
 .nav-links a {
  font-size: 11px !important;
  white-space: nowrap !important;
 }

 /* Зменшуємо кнопку паузи */
 .pause-btn {
  padding: 4px 8px !important;
  font-size: 11px !important;
  margin-left: 5px !important;
  white-space: nowrap !important;
 }
}
```

Лістинг В.7 – Header.css

```
import { useState, useEffect } from 'react';

export default function NewsPanel({ isOpen, onClose }) {
  const [news, setNews] = useState([]);
  const [loading, setLoading] = useState(true);

  // useEffect запустить завантаження лише тоді, коли панель відкриється
  useEffect(() => {
    if (isOpen && news.length === 0) {
      // Звертаємося до реального API для отримання космічних новин
      fetch('https://api.spaceflightnewsapi.net/v4/articles/?limit=5')
        .then((response) => response.json())
        .then((data) => {
          setNews(data.results);
          setLoading(false);
        })
        .catch((error) => {
          console.error("Помилка при завантаженні новин:", error);
          setLoading(false);
        });
    }
  }, [isOpen, news.length]);

  if (!isOpen) return null; // Якщо панель закрита, не рендеримо нічого

  return (
    <div className="news-panel animate-slide-left">
      <button className="close-btn" onClick={onClose}>✕</button>
      <h2>🚀 Актуальні новини</h2>

      <div className="news-list">
        {loading ? (
          <p className="loading-text">Встановлюємо зв'язок із Землею... 📡</p>
        ) : (
```

```
        news.map((article) => (
          <div key={article.id} className="news-card">
            {/* Картинка новини */}
            <div
              className="news-img"
              style={{ backgroundImage: `url(${article.image_url})` }}
            ></div>
            <div className="news-text">
              <h3>{article.title}</h3>
  <a href={article.url} target="_blank" rel="noopener noreferrer" className="read-more">
                Читати джерело ➔
              </a>
            </div>
          </div>
        ))
      )}
    </div>
  </div>
 );
}
```

Лістинг В.8 – NewsPanel.jsx

```
// @vitest-environment jsdom
import { describe, test, expect, vi } from 'vitest';
import { render, screen, fireEvent } from '@testing-library/react';
import '@testing-library/jest-dom/vitest';
import Header from './Header';

describe('Тестування компонента Header', () => {
  test('клік по кнопці "Новини космосу" викликає функцію відкриття панелі', () => {
    // 1. Створюємо "фейкову" (mock) функцію для імітації зміни стану
const mockToggleNews = vi.fn();

    // 2. Рендеримо шапку сайту і передаємо їй цю функцію
    render(<Header setIsNewsOpen={mockToggleNews} />);

    // 3. Знаходимо кнопку за її текстом
    const newsButton = screen.getByText('Новини космосу');

    // 4. Імітуємо клік користувача по цій кнопці
    fireEvent.click(newsButton);

    // 5. Перевіряємо, чи була викликана наша функція рівно 1 раз
    expect(mockToggleNews).toHaveBeenCalledTimes(1);
  });
});
```

Лістинг В.9 – Header.test.jsx

```
// @vitest-environment jsdom
import { describe, test, expect } from 'vitest';
import { render, screen } from '@testing-library/react';
import '@testing-library/jest-dom/vitest';
import NewsPanel from './NewsPanel'

describe('Тестування компонента NewsPanel', () => {
  test('показує статус "Встановлюємо зв\'язок..." під час завантаження даних', () => {

    render(<NewsPanel isOpen={true} isLoading={true} news={[]} />);

    // 2. Шукаємо текст на екрані
    const loadingText = screen.getByText(/Встановлюємо зв'язок/i);

    // 3. Перевіряємо (expect), що цей текст дійсно з'явився в документі
    expect(loadingText).toBeInTheDocument();
  });
});
```

Лістинг В.10 – NewsPanel.test.jsx

```
import { Suspense, useRef } from 'react';
import { useFrame } from '@react-three/fiber';
import { OrbitControls, Stars, Line } from '@react-three/drei';
import * as THREE from 'three'; // Імпортуємо базовий Three.js для векторів математики

import Sun from './Sun.jsx';
import Mercury from './Mercury.jsx';
import Venus from './Venus.jsx';
import Earth from './Earth.jsx';
import Mars from './Mars.jsx';
import Jupiter from './Jupiter.jsx';
import Saturn from './Saturn.jsx';
import Uranus from './Uranus.jsx';
import Neptune from './Neptune.jsx';
```

```
// Словник параметрів орбіт для розрахунку позиції камери
const PLANET_CONFIGS = {
  mercury: { radius: 22, speed: 1.2, size: 0.8 },
  venus: { radius: 28, speed: 0.7, size: 1.9 },
  earth: { radius: 35, speed: 0.5, size: 2 },
  mars: { radius: 44, speed: 0.35, size: 1 },
  jupiter: { radius: 62, speed: 0.15, size: 3.5 },
  saturn: { radius: 82, speed: 0.08, size: 3 },
  uranus: { radius: 102, speed: 0.04, size: 2.2 },
  neptune: { radius: 122, speed: 0.02, size: 2.1 },
};

function OrbitLine({ radius }) {
  const points = [];
  const segments = 128;
  for (let i = 0; i <= segments; i++) {
    const angle = (i / segments) * Math.PI * 2;
    points.push([radius * Math.cos(angle), 0, radius * Math.sin(angle)]);
  }
  return <Line points={points} color="#ffffff" lineWidth={1.5} transparent opacity={0.3} />;
}

export default function SolarSystem({ isPaused, selectedPlanet, setSelectedPlanet }) {
  const timeRef = useRef(0);

  useFrame((state, delta) => {
    if (!isPaused) timeRef.current += delta;
    const time = timeRef.current;

    if (state.controls) {
      if (selectedPlanet) {
        let targetX;
        let targetZ;
```

```
        let cameraOffset;

        if (selectedPlanet === 'sun') {
          // Сонце стоїть у центрі (0,0,0)
          targetX = 0;
          targetZ = 0;
          // Камера відлітає трохи далі, бо Сонце велике (радіус 15)
          cameraOffset = new THREE.Vector3(0, 25, 55);
        }
        else if (selectedPlanet === 'moon') {
          // Обчислюємо позицію Землі
          const earthConfig = PLANET_CONFIGS['earth'];
          const earthX = earthConfig.radius * Math.cos(time * earthConfig.speed);
          const earthZ = earthConfig.radius * Math.sin(time * earthConfig.speed);

          // Обчислюємо локальну орбіту Місяця (радіус 4, швидкість 1.5)
          const moonLocalX = 4 * Math.cos(time * 1.5);
          const moonLocalZ = 4 * Math.sin(time * 1.5);

          // Абсолютна позиція Місяця в просторі
          targetX = earthX + moonLocalX;
          targetZ = earthZ + moonLocalZ;

          // Камера підлітає дуже близько, бо Місяць маленький (радіус 0.5)
          cameraOffset = new THREE.Vector3(targetX, 2, targetZ + 6);
        }
        else {
          // Стандартна логіка для всіх інших планет
          const config = PLANET_CONFIGS[selectedPlanet];
          targetX = config.radius * Math.cos(time * config.speed);
          targetZ = config.radius * Math.sin(time * config.speed);
          cameraOffset = new THREE.Vector3(targetX, config.size * 3 + 4, targetZ + config.size
* 4 + 6);
        }
```

```
    // Визначаємо, чи відкритий сайт на мобільному пристрої (ширина менше 768px)
    const isMobile = window.innerWidth < 768;

    // Налаштовуємо зміщення по осі Y
    let offsetY = 0; // Для комп'ютера фокус залишається по центру

    if (isMobile) {
      // Якщо це Сонце (воно велике), зміщуємо фокус камери значно нижче
      if (selectedPlanet === 'sun') {
        offsetY = -12;
      } else {
        // Для звичайних планет зміщуємо фокус трохи нижче
        offsetY = -4;
      }
    }

    // Створюємо нову точку фокусу з урахуванням нашого зміщення
    const targetPlanetPos = new THREE.Vector3(targetX, offsetY, targetZ);
    state.controls.target.lerp(targetPlanetPos, 0.05);
    state.camera.position.lerp(cameraOffset, 0.05);
   } else {
    const center = new THREE.Vector3(0, 0, 0);
    state.controls.target.lerp(center, 0.05);
    const defaultCameraPos = new THREE.Vector3(0, 60, 150);
    if (state.camera.position.distanceTo(center) < 80) {
      state.camera.position.lerp(defaultCameraPos, 0.03);
    }
   }
  }
});

return (
  <>
```

```
            <OrbitControls makeDefault minDistance={5} maxDistance={300} />
            <ambientLight intensity={1.5} color="#ffffff" />
            <Stars radius={100} depth={50} count={5000} factor={4} saturation={0} fade
speed={1} />

            <Sun isPaused={isPaused} onSelect={() => setSelectedPlanet('sun')} />

            {Object.values(PLANET_CONFIGS).map((p, idx) => (
             <OrbitLine key={idx} radius={p.radius} />
            ))}

            <Suspense fallback={null}>
             {/* Передаємо функцію вибору у кожну планету */}
             <Mercury isPaused={isPaused} onSelect={() => setSelectedPlanet('mercury')} />
             <Venus isPaused={isPaused} onSelect={() => setSelectedPlanet('venus')} />
             <Earth isPaused={isPaused} onSelect={() => setSelectedPlanet('earth')}
onSelectMoon={() => setSelectedPlanet('moon')} />
             <Mars isPaused={isPaused} onSelect={() => setSelectedPlanet('mars')} />
             <Jupiter isPaused={isPaused} onSelect={() => setSelectedPlanet('jupiter')} />
             <Saturn isPaused={isPaused} onSelect={() => setSelectedPlanet('saturn')} />
             <Uranus isPaused={isPaused} onSelect={() => setSelectedPlanet('uranus')} />
             <Neptune isPaused={isPaused} onSelect={() => setSelectedPlanet('neptune')} />
            </Suspense>
           </>
          );
         }
```

Лістинг В.11 – SolarSystem.jsx

```
import { useRef } from 'react';
import { useFrame } from '@react-three/fiber';
// Додаємо Line до імпорту з drei
import { useTexture, Line } from '@react-three/drei';

export default function Earth({ isPaused, onSelect, onSelectMoon }) {
  const systemRef = useRef();
  const earthRef = useRef();
  const moonRef = useRef();
  const timeRef = useRef(0);



  const earthMap = useTexture('/textures/earth.jpg');
  const moonMap = useTexture('/textures/moon.jpg');

  // Параметри орбіт
  const earthOrbitRadius = 35;
  const earthOrbitSpeed = 0.5;

  const moonOrbitRadius = 4;
  const moonOrbitSpeed = 1.5;

  // Генеруємо точки для кола орбіти Місяця (навколо локального центру 0,0,0)
  const moonOrbitPoints = [];
  const segments = 64; // Для маленького кола 64 сегментів цілком достатньо
  for (let i = 0; i <= segments; i++) {
    const angle = (i / segments) * Math.PI * 2;
    moonOrbitPoints.push([
      moonOrbitRadius * Math.cos(angle),
      0,
      moonOrbitRadius * Math.sin(angle)
    ]);
  }
```

```
 useFrame((state, delta) => {
  // Якщо пауза натиснута – просто виходимо з функції, нічого не обчислюючи!
  if (isPaused) return;

  // Якщо паузи немає, додаємо час, що минув з минулого кадру
  timeRef.current += delta;
  const time = timeRef.current; // Використовуємо НАШ час замість state.clock

  // 1. Рух усієї системи навколо Сонця
  if (systemRef.current) {
   systemRef.current.position.x = earthOrbitRadius * Math.cos(time * earthOrbitSpeed);
   systemRef.current.position.z = earthOrbitRadius * Math.sin(time * earthOrbitSpeed);
  }

  // 2. Обертання Землі
  if (earthRef.current) {
   earthRef.current.rotation.y += 0.002;
  }

  // 3. Рух Місяця навколо Землі
  if (moonRef.current) {
   moonRef.current.position.x = moonOrbitRadius * Math.cos(time * moonOrbitSpeed);
   moonRef.current.position.z = moonOrbitRadius * Math.sin(time * moonOrbitSpeed);
   moonRef.current.rotation.y += 0.005;
  }
 });

return (
  <group ref={systemRef}>

   {/* Земля */}
   <mesh
    ref={earthRef}
    // ДОДАЄМО КЛІК:
```

```
            onClick={(e) => {
              e.stopPropagation(); // Зупиняє подію, щоб клік не летів крізь планету далі
              onSelect();          // Викликає функцію вибору
            }}
            // ДОДАЄМО ЗМІНУ КУРСОРУ:
            onPointerOver={(e) => {
              e.stopPropagation();
              document.body.style.cursor = 'pointer'; // Стає ручкою при наведенні
            }}
            onPointerOut={() => {
              document.body.style.cursor = 'default'; // Повертається назад
            }}
          >
            <sphereGeometry args={[2, 64, 64]} />
            <meshStandardMaterial map={earthMap} />
          </mesh>

          <Line points={moonOrbitPoints} color="#ffffff" lineWidth={1} transparent opacity={0.25} />

          {/* Місяць */}
          <mesh ref={moonRef}
            onClick={(e) => {
              e.stopPropagation();
              onSelectMoon(); // Викликаємо функцію для Місяця
            }}
            onPointerOver={(e) => {
              e.stopPropagation();
              document.body.style.cursor = 'pointer';
            }}
            onPointerOut={() => {
              document.body.style.cursor = 'default';
            }}
          >
```

```
      <sphereGeometry args={[0.5, 32, 32]} />
      <meshStandardMaterial map={moonMap} />
    </mesh>

  </group>
 );
}
```

Лістинг В.12 – Earth.jsx

```
import { useRef } from 'react';
import { useFrame } from '@react-three/fiber';
import { useTexture } from '@react-three/drei';

export default function Saturn({ isPaused, onSelect }) {
 const saturnGroupRef = useRef();
 const timeRef = useRef(0);

 const radius = 82;
 const speed = 0.08;

 const planetMap = useTexture('/textures/saturn.jpg');
 const ringMap = useTexture('/textures/saturn_ring.png');

 useFrame((state, delta) => {
  if (isPaused) return;

  timeRef.current += delta;
  const time = timeRef.current;

  if (saturnGroupRef.current) {
   saturnGroupRef.current.position.x = radius * Math.cos(time * speed);
   saturnGroupRef.current.position.z = radius * Math.sin(time * speed);
   saturnGroupRef.current.rotation.y += 0.003;
  }
```

```
});

return (
  <group
    ref={saturnGroupRef}
    rotation={[0.45, 0, 0]}
    onClick={(e) => {
      e.stopPropagation();
      onSelect();
    }}
    onPointerOver={(e) => {
      e.stopPropagation();
      document.body.style.cursor = 'pointer';
    }}
    onPointerOut={() => {
      document.body.style.cursor = 'default';
    }}
  >
    <mesh>
      <sphereGeometry args={[3, 64, 64]} />
      <meshStandardMaterial map={planetMap} />
    </mesh>
    <mesh rotation={[-Math.PI / 2, 0, 0]}>
      <ringGeometry args={[3.5, 6, 64]} />
      <meshBasicMaterial
        map={ringMap}
        transparent={true}
        opacity={0.8}
        side={2}
        depthWrite={false}
      />
    </mesh>
  </group>
);
```

```
}
```

Лістинг В.13 – Saturn.jsx

```
// @vitest-environment jsdom
import { describe, test, expect } from 'vitest';
import ReactThreeTestRenderer from '@react-three/test-renderer';
import Saturn from './Saturn';

describe('Тестування 3D-моделі Сатурна', () => {
  test('Компонент успішно монтується у віртуальній сцені без помилок', async () => {
    let renderer;

    await ReactThreeTestRenderer.act(async () => {
      renderer = await ReactThreeTestRenderer.create(<Saturn />);
    });

    // перевіряємо, що віртуальна сцена успішно створилася
    expect(renderer.scene).toBeDefined();
  });
});
```

Лістинг В.14 – Saturn.test.jsx

```
import { useRef } from 'react';
import { useFrame } from '@react-three/fiber';
import { useTexture } from '@react-three/drei';

export default function Sun({ isPaused, onSelect }) {
  const sunRef = useRef();
  const sunMap = useTexture('/textures/sun.jpg');

  useFrame(() => {
    if (isPaused) return;
    if (sunRef.current) {
      sunRef.current.rotation.y += 0.001;
    }
```

```
  });

  return (
    <mesh
      ref={sunRef}
      // Додаємо обробники подій
      onClick={(e) => {
        e.stopPropagation();
        onSelect();
      }}
      onPointerOver={(e) => {
        e.stopPropagation();
        document.body.style.cursor = 'pointer';
      }}
      onPointerOut={() => {
        document.body.style.cursor = 'default';
      }}
    >
      <sphereGeometry args={[15, 64, 64]} />
      <meshBasicMaterial map={sunMap} />
      <pointLight intensity={200} color="#ffffff" distance={300} decay={1.5} />
    </mesh>
  );
}
```

Лістинг В.15 – Sun.jsx