МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ ВІЗУАЛІЗАЦІЇ КОСМІЧНИХ ДАНИХ ЗАСОБАМИ API NASA, ANGULAR ТА LARAVEL»**

Виконав: студент 2 курсу, групи 8.1213-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
Д.І. Литвин
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, к.ф.-м.н. Кривохата А.Г.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Литвину Данилу Ігоровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка вебзастосунку для візуалізації космічних даних засобами API NASA, Angular та Laravel

керівник роботи Кривохата Анастасія Григорівна, к.ф.-м.н.
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » березня 2024 року № 385-с

2. Строк подання студентом роботи 25.11.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі, аналіз предметної області.
2. Проєктування.
3. Реалізація та тестування.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 07.03.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану виконання кваліфікаційної	22.03.2024	
	роботи магістра.		
2.	Збір вихідних даних та аналіз предметної	19.04.2024	
	області.		
3.	Обробка методичних та теоретичних	24.05.2024	
	джерел.		
4.	Специфікація вимог до системи. Робота над	16.08.2024	
	першим розділом.		
5.	Моделювання та проєктування системи.	20.09.2024	
	Робота над другим розділом.		
6.	Реалізація та тестування системи. Робота над	08.11.2024	
	третім розділом.		
7.	Оформлення та нормоконтроль	20.11.2024	
	кваліфікаційної роботи магістра.		
8.	Захист кваліфікаційної роботи.	12.12.2024	

Студент _____ Д.І. Литвин
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.Г. Кривохата
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка вебзастосунку для візуалізації космічних даних засобами API NASA, Angular та Laravel»: 61 с., 25 рис., 16 джерел, 4 додатка.

ANGULAR, API, FRAMEWORK, LARAVEL, NASA, RXJS, TYPESCRIPT, UML.

Об’єкт дослідження – вебзастосунок, інструменти для взаємодії з API, засоби взаємодії системи з користувачем.

Мета роботи – розробити вебзастосунок для візуалізації космічних даних.

Методи дослідження – моделювання, проєктування, програмний, аналітичний.

Ця робота присвячена розробці вебзастосунку для візуалізації космічних даних та взаємодії з API NASA з використанням фреймворків Angular та Laravel. Використання Angular дозволяє створити динамічний та інтерактивний інтерфейс користувача, тоді як Laravel забезпечує надійну та ефективну роботу серверної частини додатка. Такий підхід дозволяє об’єднати переваги обох технологій та створити систему, здатну задовольнити потреби користувачів у отриманні, аналізі та візуалізації космічних даних.

Таким чином, за результатами роботи створений зручний та ефективний вебзастосунок для візуалізації космічних даних засобами API NASA, Angular та Laravel.

SUMMARY

Master’s qualifying paper “Development of Web Application for Visualizing Space Data Using NASA APIs, Angular and Laravel”: 61 p., 25 figures, 16 references, 4 supplements.

ANGULAR, API, FRAMEWORK, LARAVEL, NASA, RXJS, TYPESCRIPT, UML.

The object of the study is a web application, tools for visualizing space data using NASA APIs, Angular and Laravel.

The aim of the study is to develop a web application for visualizing space data.

The methods of research are modeling, design, software, analytical.

This work is devoted to the development of a web application for visualizing space data and interacting with NASA APIs using the Angular and Laravel frameworks. The use of Angular allows to create a dynamic and interactive user interface, while Laravel ensures reliable and efficient operation of the server side of the application. This approach allows combining the advantages of both technologies and creating a system that can meet the needs of users in obtaining, analyzing and visualizing space data.

Thus, based on the results of the work, a convenient and efficient web application for visualizing space data using NASA API, Angular and Laravel was created.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 8
1 Технічне завдання 10
1.1 Терміни та визначення 10
1.1.1 Загальні терміни 10
1.1.2 Технічні терміни 10
1.2 Функціональні вимоги 11
1.2.1 Призначення і цілі створення системи 11
1.2.2 Загальні функціональні можливості системи 11
1.3 Нефункціональні вимоги 11
1.4 Опис предметної області 12
1.5 Опис системи 13
1.6 Огляд та порівняння аналогів 14
2 Проєктування 17
2.1 Використання UML під час розробки системи 17
2.2 Діаграма варіантів використання 18
2.2.1 Опис варіантів використання 21
2.3 Діаграма класів 27
2.4 Діаграма послідовності 29
2.5 Діаграма діяльності 31
2.6 Діаграма розгортання 33
3 Реалізація та тестування 36
3.1 Опис інструментів розробки 36
3.2 Основні класи системи 36
3.3 Отримання API-ключа NASA 38

3.4 Тестування проєкту 40
3.5 Керівництво користувача 42
3.5.1 Головна сторінка 42
3.5.2 Інструмент APOD 43
3.5.3 Інструмент Mars Rover Photos 43
3.5.4 Інструмент NASA Image Search 45
3.5.5 Інструмент Near-Earth Objects (NEO) 46
3.5.6 Інструмент Solar Activity (CME) 47
Висновки 49
Перелік посилань 50
Додаток А Angular-компонент 52
Додаток Б Angular-сервіс 55
Додаток В Laravel-контролер 57
Додаток Г Тестування NasaService 60

ВСТУП

Сучасний світ стрімко розвивається, і технології відіграють у цьому процесі ключову роль. Однією з найдинамічніших галузей є космічні дослідження, де великі обсяги даних генеруються та обробляються щодня. Використання цих даних може надати унікальну можливість для вивчення космосу, створення нових технологій та розширення наукових знань.

Водночас, з розвитком вебтехнологій з’являються нові можливості для створення інтерактивних та зручних для користувача вебдодатків, які можуть забезпечити доступ до космічних даних широкій аудиторії. Важливим завданням стає розробка таких додатків, які можуть ефективно візуалізувати космічні дані та забезпечити зручну взаємодію з API NASA, що є одним з найпотужніших джерел космічної інформації.

Виходячи з цього задача створення вебзастосунку для взаємодії користувачів з космічними даними є актуальною.

Метою кваліфікаційної роботи є розробка вебзастосунку зі зрозумілим інтерфейсом та гнучким функціоналом.

Для досягнення сформульованної мети необхідно вирішити такі завдання:

– сформувати вимоги до вебзастосунку;
– спроєктувати та побудувати архітектуру вебзастосунку;
– реалізувати вебзастосунок для візуалізації космічних даних;
– протестувати роботу вебзастосунку.

Об’єкт дослідження: вебзастосунок, інструменти для взаємодії з API, засоби взаємодії системи з користувачем.

Предмет дослідження: фреймворки веброзробки, NASA API.

Методи дослідження: моделювання, проєктування, програмний, аналітичний.

Перший розділ присвячено збору та аналізуванню вимог до вебзастосунку, огляду аналогів та опису застосунку.

У другому розділі розглянуто етапи проєктування вебзастосунку, наведено детальний опис прецедентів.

Третій розділ присвячено реалізації та тестуванню роботи вебзастосунку, наведено детальне керівництво користувача, яке описує процес роботи з вебзастосуноком для візуалізації космічних даних.

1 ТЕХНІЧНЕ ЗАВДАННЯ

1.1 Терміни та визначення

1.1.1 Загальні терміни

Система – вебзастосунок для візуалізації космічних даних.

Angular – це фреймворк для розробки додатків і платформа для створення ефективних і складних односторінкових додатків.

Laravel – це міжплатформовий PHP-фреймворк для створення вебдодатків.

ДВВ – Діаграма Варіантів Використання.

ДД – Діаграма Діяльності.

ДА – Діаграма Активності.

ДК – Діаграма Класів.

ДП – Діаграма Послідовності.

Користувач – людина, яка а може взаємодіяти з функціоналом системи.

1.1.2 Технічні терміни

API – це система інструментів і ресурсів в додатку, яка дозволяє розробникам створювати програмні продукти, які взаємодіють з іншими службами [8].

MVC (Model-View-Controller) – це архітектурний шаблон для розробки програмного забезпечення, який допомагає розділити програму на три основні частини [5].

Фреймворк – це набір інструментів, бібліотек та рекомендацій, призначених для розробки програмного забезпечення.

1.2 Функціональні вимоги

1.2.1 Призначення і цілі створення системи

Функціональне призначення системи – реалізувати можливість візуалізації космічних даних та інструменти для взаємодії з ними.

Експлуатаційне призначення системи: система може експлуатуватися користувачами та гостями системи.

Мета створення системи – розробити вебзастосунок для візуалізації космічних даних.

1.2.2 Загальні функціональні можливості системи

Система має надавати користувачам такі можливості:

- вибір інструменту NASA;
- перегляд APOD/ фотографій ровера/ близькоземних об’єктів/ активності сонця;
- пошук зображень та відео;
- фільтрація;
- вибір/введення дати/солу;
- перегляд інформації про близькоземний об’єкт.

1.3 Нефункціональні вимоги

Інтерфейс користувача. Система повинна коректно відображати інтерфейс користувача на будь-якому пристрої, забезпечуючи адаптивність та зручність у використанні.

Підтримка браузерів. Система повинна працювати коректно в останніх версіях наступних браузерів:

– Mozilla Firefox;
– Google Chrome;
– Safari;
– Microsoft Edge;
– Opera.

Вимоги до продуктивності. Система повинна завантажувати будь-яку сторінку не довше, ніж за 2 секунди. Генерація кожного запиту до API NASA повинна здійснюватися не довше, ніж за 1 секунду.

Вимоги до безпеки. Система повинна захищати API-ключ NASA, приховуючи його від клієнтської частини. Доступ до серверної частини повинен бути захищений від несанкціонованого використання.

1.4 Опис предметної області

Предметна область даного дослідження охоплює розробку вебзастосунку для візуалізації космічних даних та інтеграцію з API NASA. Він повинен забезпечувати зручний та інтуїтивно зрозумілий інтерфейс для користувачів, який коректно відображатиметься на будь-якому пристрої. Це включає адаптивний дизайн та юзабіліті, які дозволяють користувачам легко взаємодіяти з додатком та отримувати необхідну інформацію.

Космічні дані, що використовуються в додатку, включають астрономічні зображення, наукові дані про параметри космічних об’єктів та відомості про космічні місії NASA. Джерелом цих даних є API NASA, який надає доступ до великої кількості інформації про космічні об’єкти, місії та відкриття. Важливо забезпечити можливість пошуку та фільтрації цих даних за різними параметрами для зручного використання.

Технічна частина розробки включає використання фреймворків Angular для клієнтської частини та Laravel для серверної частини. Angular дозволяє створювати динамічний та інтерактивний інтерфейс користувача, тоді як

Laravel забезпечує надійну та ефективну роботу серверної частини додатка. Використання цих технологій забезпечує високий рівень продуктивності та безпеки системи.

Візуалізація даних у вебзастосунку здійснюється за допомогою графічних бібліотек, таких як Three.js або Hightcharts, що дозволяє створювати інтерактивні графіки, діаграми та інші візуальні елементи. Картографія також є важливою складовою, що дозволяє відображати розташування космічних об’єктів та траєкторії космічних місій на інтерактивних картах.

Система повинна відповідати певним функціональним та нефункціональним вимогам. Функціональні вимоги включають доступ до даних API NASA, можливість пошуку та фільтрації даних, а також генерацію звітів та візуалізацій. Нефункціональні вимоги включають високу продуктивність, забезпечення швидкого завантаження сторінок та обробки запитів, а також надійний захист від несанкціонованого доступу та забезпечення конфіденційності даних користувачів.

Таким чином, предметна область даного дослідження включає всі аспекти розробки вебзастосунка для візуалізації космічних даних, починаючи від вибору та обробки даних до забезпечення ефективного та безпечного функціонування системи.

1.5 Опис системи

Система для візуалізації космічних даних та взаємодії з API NASA з використанням Angular та Laravel призначена для забезпечення користувачів зручним доступом до астрономічної інформації. Вебзастосунок дозволяє переглядати, аналізувати та візуалізувати дані про космічні об’єкти, місії та інші наукові відкриття, забезпечуючи високу продуктивність та безпеку.

Інтеграція з API NASA є ключовим елементом системи, що дозволяє отримувати актуальні дані про космічні об’єкти та місії. API надає доступ до

великої кількості інформації, включаючи зображення, наукові результати та інші дані. Використання API NASA дозволяє забезпечити високу якість та актуальність інформації, представленої у вебзастосунку.

Таким чином, розроблена система забезпечує зручний та ефективний доступ до космічних даних, дозволяючи користувачам легко взаємодіяти з інформацією та отримувати необхідні результати в зручному форматі.

1.6 Огляд та порівняння аналогів

На ринку існує кілька аналогічних систем, які забезпечують візуалізацію космічних даних та взаємодію з API NASA та іншими джерелами астрономічної інформації.

Розглянемо найбільш відомі з них.

NASA’s Eyes – це інтерактивний програмний комплекс, який дозволяє користувачам досліджувати Сонячну систему, місії NASA та планету Земля в 3D. Він пропонує візуалізації в реальному часі, а також доступ до великої кількості даних і зображень [12].

SkyView – це вебдодаток від NASA, який дозволяє користувачам переглядати різні астрономічні об’єкти через віртуальний телескоп. Користувачі можуть вибирати різні діапазони електромагнітного спектра для перегляду об’єктів [13].

WorldWide Telescope, розроблений Microsoft Research, цей додаток дозволяє користувачам вивчати Всесвіт за допомогою інтерактивного інтерфейсу, який об’єднує зображення з багатьох обсерваторій і телескопів по всьому світу [16].

Програмне забезпечення Stellarium з відкритим вихідним кодом, яке функціонує як планетарій у реальному часі. Воно дозволяє користувачам вивчати зоряне небо та космічні об’єкти на їхніх комп’ютерах або мобільних пристроях [14].

Heavens-Above, вебзастосунок, який надає інформацію про супутники, комети, планети та інші астрономічні явища. Він пропонує інтерактивні карти зоряного неба та інструменти для спостереження за космічними об’єктами [9].

Вільно доступна програма Celestia, що дозволяє користувачам подорожувати у Всесвіті в реальному часі, переглядаючи планети, зорі, галактики та інші космічні об’єкти [3].

Хоча ці системи забезпечують візуалізацію космічних даних та інтерактивну взаємодію, розробка нового вебзастосунку на основі Angular та Laravel, орієнтованого на інтеграцію з API NASA, може запропонувати користувачам унікальні можливості та покращений користувацький досвід, враховуючи сучасні вебтехнології, адаптивний дизайн та спеціалізовані функції для обробки й аналізу даних.

Наведемо порівняння цих систем з нашою (рис. 1.1).

Функція	**NASA’s Eyes**	**SkyView**	**WorldWide Telescope**	**Stellarium**	**Heavens-Above**	**Celestia**	**Наша система**
Вебінтерфейс	Ні	Так	Ні	Ні	Так	Ні	Так
Інтерактивність	Висока	Середня	Висока	Висока	Середня	Висока	Висока
Джерела даних	Дані NASA	Віртуальний телескоп	Обсерваторії та телескопи	Відкриті дані	Дані супутників та планет	Відкриті дані	Дані API NASA
3D-візуалізації	Так	Ні	Так	Ні	Ні	Так	Так
Кросплатформеність	Windows, Mac, Linux	Веб	Windows, Mac	Windows, Mac, Linux	Веб	Windows, Mac, Linux	Веб, адаптивний дизайн
Використані технології	Спеціалізовані API	Вебтехнології	Спеціалізовані API	Відкритий код	Вебтехнології	Відкритий код	Angular, Laravel
Продуктивність	Висока	Висока	Висока	Висока	Середня	Висока	Висока, залежить від серверної частини
Безпека	Висока	Висока	Висока	Висока	Середня	Висока	Висока
Адаптивний дизайн	Ні	Так	Ні	Ні	Так	Ні	Так
Вартість	Безкоштовно	Безкоштовно	Безкоштовно	Безкоштовно	Безкоштовно	Безкоштовно	Безкоштовно

Рисунок 1.1 – Порівняння систем

З порівняння можемо зробити висновок, що вебзастосунок на базі Angular та Laravel має переваги завдяки своєму вебінтерфейсу, який є адаптивним і працює на будь-яких пристроях. Інші системи, такі як NASA’s Eyes та WorldWide Telescope, не мають вебінтерфейсу, що обмежує їх доступність та використання.

Вебзастосунок на базі Angular та Laravel забезпечує високу інтерактивність, продуктивність та безпеку, а також доступ до актуальних космічних даних через API NASA. Це робить його універсальним рішенням для візуалізації космічних даних у порівнянні з іншими існуючими системами.

2 ПРОЄКТУВАННЯ

2.1 Використання UML під час розробки системи

UML (Unified Modeling Language) – це стандартизована мова моделювання, яка широко використовується для аналізу, проєктування та документування програмного забезпечення. Вона дозволяє розробникам створювати структуровані моделі систем, що полегшує впровадження, тестування та підтримку цих систем. На етапі аналізу вимог UML дозволяє візуалізувати функціональні вимоги за допомогою діаграм варіантів використання, які демонструють взаємодію між користувачами (акторами) та системою [15].

Проєктування архітектури системи базується на діаграмах класів, які описують структуру системи, включаючи класи, їхні атрибути, методи та взаємозв’язки. Це створює основу для об’єктно-орієнтованого програмування. Діаграми компонентів допомагають моделювати модульність системи, визначаючи компоненти та їх взаємозв’язки, а діаграми розгортання ілюструють фізичне розміщення компонентів на вузлах [15].

Динамічні аспекти системи моделюються за допомогою діаграм послідовності, які ілюструють взаємодію об’єктів у часі, та діаграм станів, які описують стани об’єктів та переходи між ними. Діаграми діяльності використовуються для зображення робочих потоків і логіки процесів, що має вирішальне значення для моделювання бізнес-логіки системи.

Крім того, UML полегшує документування системи. Діаграми слугують візуальним представленням, яке полегшує розуміння системи розробниками, аналітиками, тестувальниками та клієнтами.

До переваг уніфікованої мови моделювання (UML) можна віднести уніфікований підхід до моделювання, підвищену зрозумілість системи для всіх учасників проєкту, а також можливість моделювання як невеликих, так і складних архітектур.

Тому UML є ефективним інструментом для розробки якісного програмного забезпечення. Її застосування дозволяє зменшити ризики проєкту, покращити комунікацію між членами команди та полегшити створення надійної архітектури системи.

2.2 Діаграма варіантів використання

Діаграма варіантів використання (Use Case Diagram) є одним з основних інструментів моделювання в UML. Діаграма варіантів використання є візуальним представленням функціональних вимог до системи і демонструє, як зовнішні користувачі (актори) взаємодіють з системою для виконання певних завдань. Основною метою цієї діаграми є аналіз вимог до системи, що дозволяє описати основні функції з точки зору користувача. Завдяки своїй простоті діаграма сприяє ефективній комунікації між розробниками, замовниками та іншими зацікавленими сторонами, а також стає частиною технічної документації [15].

Діаграма варіантів використання складається з декількох основних компонентів. У цьому контексті термін «актори» відноситься до зовнішніх користувачів або інших систем, які взаємодіють з моделлю. Варіанти використання можуть виконуватися людьми, пристроями або програмним забезпеченням. Це дії або сценарії, які виконуються системою у відповідь на запити акторів. Вони представлені у вигляді овалів з коротким описом функції, наприклад, «Фільтрувати дані». Зв’язки між елементами включають асоціації, які вказують на взаємодію, а також включають (Include) і розширюють (Extend) зв’язки, які описують залежності між варіантами використання. Всі варіанти використання розміщуються в межах прямокутника, який представляє систему.

Основними перевагами діаграм варіантів використання є їхня зрозумілість навіть для нетехнічних учасників проєкту та здатність

забезпечити візуальне представлення функціональності системи. З іншого боку, вони не позбавлені обмежень, зокрема, нездатність описати внутрішню логіку варіантів використання, що може виявитися недостатнім для моделювання складних систем. Тим не менш, на початкових етапах розробки діаграма варіантів використання є ефективним інструментом для визначення взаємодії між системою і користувачами.

На рисунку 2.1 представлена діаграма варіантів використання вебзастосунку для візуалізації космічних даних.

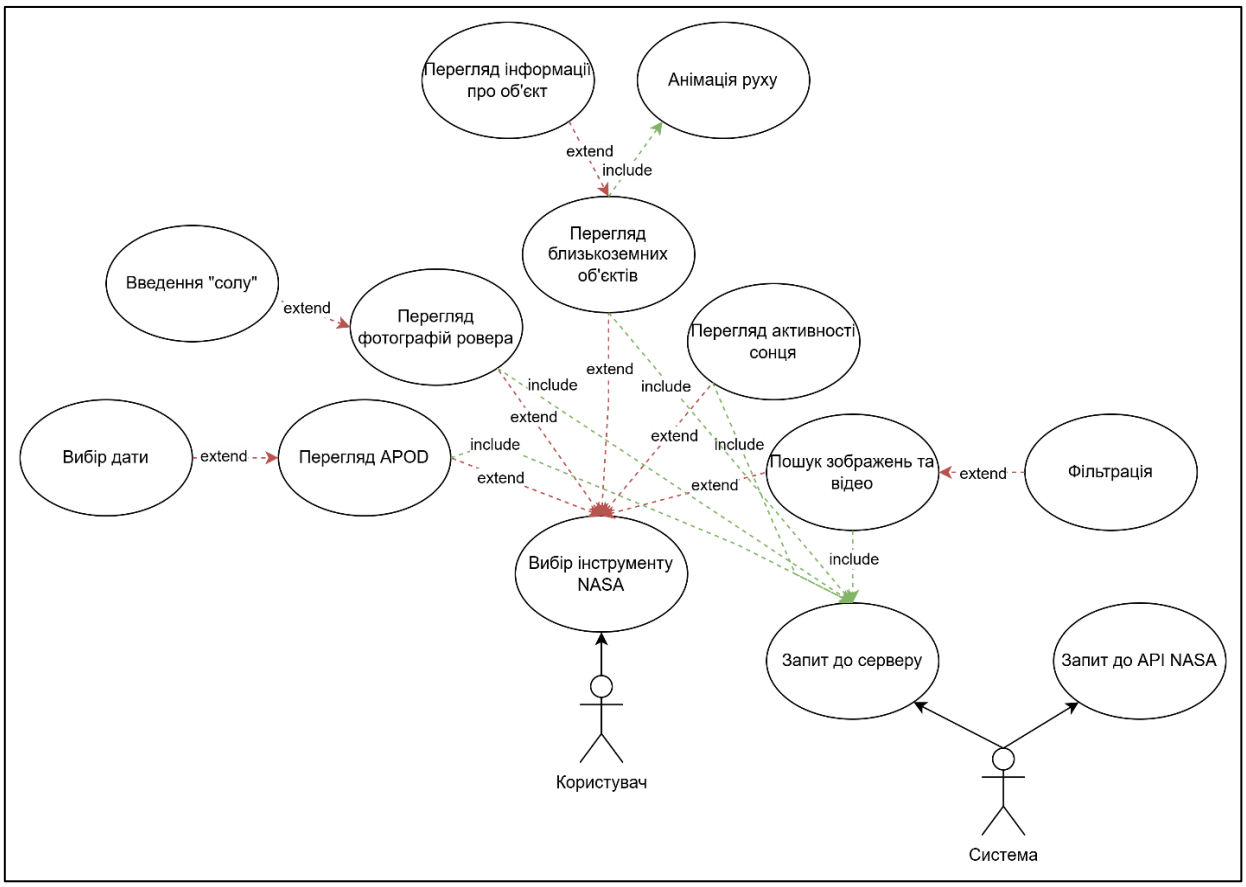


Рисунок 2.1 – Діаграма варіантів використання

На діаграмі представлено акторів «Користувач» та «Система», кожен з яких може взаємодіяти з відповідними інструментами на різних рівнях.

Виділено 1 із основних варіантів використання – «Вибір інструменту NASA». Після того, як відкриває головну сторінку системи, відображається список доступних інструментів NASA. Користувач вибирає один із

інструментів. Система перенаправляє користувача на сторінку обраного інструменту.

Варіанти використання – це ключовий метод опису того, як користувачі та інші учасники взаємодіють з програмною системою. Вони відображають конкретні ланцюжки взаємодій між актором і системою, спрямовані на досягнення певної цілі чи результату.

Головне призначення варіантів використання – це формалізація та документування того, як система має поводитись і що вона повинна вміти робити з позиції тих, хто буде нею користуватися. У кожному варіанті використання детально описується певний спосіб застосування системи.

Таким чином, варіанти використання створюють єдину мову для спілкування між усіма учасниками проєкту – розробниками, аналітиками та замовниками. Це дозволяє всім однаково розуміти, як система працюватиме на практиці та які можливості вона надаватиме своїм користувачам чи іншим зацікавленим сторонам (див. рис. 2.2 – 2.3).

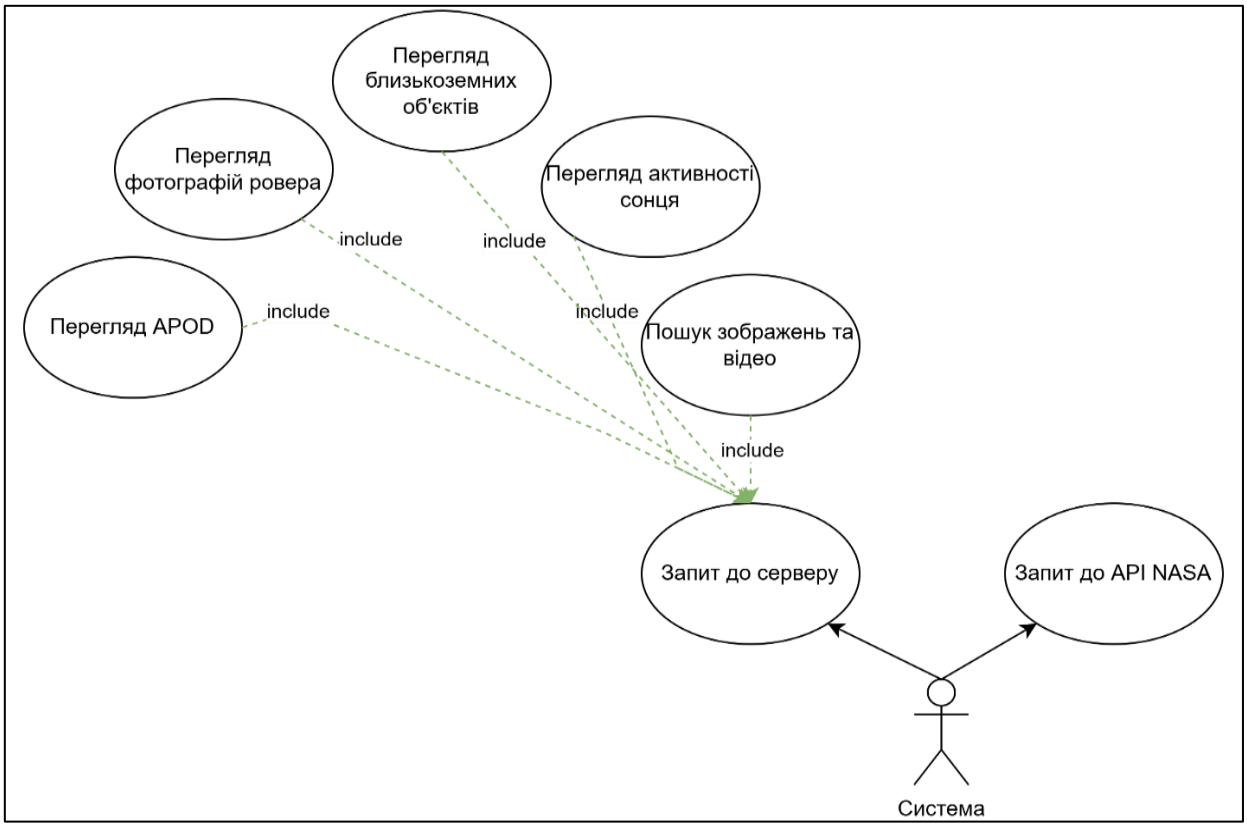


Рисунок 2.2 – ДВВ «Система»

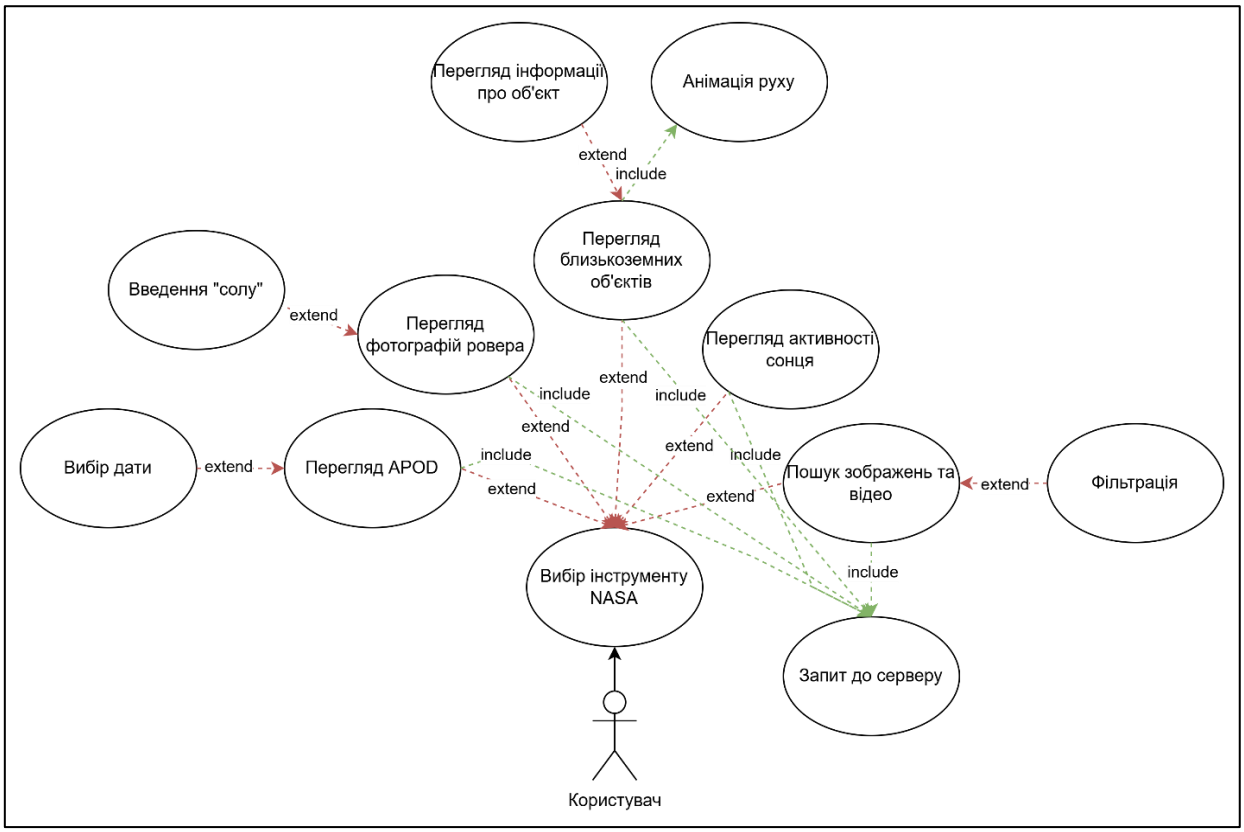


Рисунок 2.3 – ДВВ «Користувач»

2.2.1 Опис варіантів використання

Прецедент «Вибір інструменту NASA».

Призначення: даний варіант використання надає можливість користувачу обрати один із доступних інструментів NASA для перегляду даних.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач відкриває головну сторінку системи, йому відображається список доступних інструментів NASA, таких як APOD, Mars Rover Photos, NEO, CME та Image Search. Користувач обирає потрібний інструмент зі списку, після чого система автоматично перенаправляє його на сторінку вибраного інструменту для подальшої взаємодії.

Виняткова ситуація 1: відсутність інструментів на головній сторінці – система повідомляє користувача про неможливість завантаження даних.

Виняткова ситуація 2: помилка перенаправлення – система повідомляє про проблему і пропонує оновити сторінку.

Прецедент «Перегляд APOD».

Призначення: даний варіант використання надає можливість користувачу переглядати зображення дня NASA із його описом.

Основний потік подій: даний варіант використання починає виконуватися після вибору інструменту «APOD», система надсилає запит до сервера для отримання даних про поточне зображення дня. На сторінці відображається зображення, його назва та короткий опис. Якщо користувач бажає переглянути зображення за конкретну дату, він може вибрати дату через інтерактивний елемент. У відповідь на вибір дати система надсилає новий запит до сервера і оновлює відображену інформацію.

Виняткова ситуація 1: помилка завантаження зображення – система повідомляє про відсутність даних на сервері.

Виняткова ситуація 2: вибір недійсної дати – система повідомляє, що дані за вказану дату недоступні.

Прецедент «Перегляд фотографій ровера».

Призначення: даний варіант використання надає можливість користувачу переглядати фотографії, зроблені роверами NASA на Марсі.

Основний потік подій: даний варіант використання починає виконуватися після вибору інструменту «Mars Rover Photos», система за замовчуванням завантажує фотографії, зроблені ровером за стандартний «сол» (марсіанський день). На сторінці відображається список фотографій з описом, включаючи назву камери та дату. Якщо користувач хоче переглянути фотографії за інший «сол», він вводить значення у відповідне поле, після чого система надсилає новий запит до сервера і оновлює відображений список фотографій.

Виняткова ситуація 1: некоректне значення «солу» – система повідомляє про помилку формату і просить ввести коректне значення.

Виняткова ситуація 2: відсутність фотографій за вказаний «сол» – система повідомляє, що за вказаний «сол» фотографії відсутні.

Прецедент «Перегляд близькоземних об’єктів».

Призначення: даний варіант використання надає можливість користувачу переглядати близькоземні об’єкти на інтерактивній мапі.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач вибирає інструмент «NEO», після чого система надсилає запит до сервера для отримання даних про близькоземні об’єкти за вибраний період. На сторінці відображається інтерактивна мапа Сонячної системи, на якій розміщені об’єкти NEO у відповідних позиціях. Користувач може натиснути на об’єкт на мапі, щоб переглянути його деталі, які відображаються у спеціальній інформаційній панелі.

Виняткова ситуація 1: неможливість завантажити мапу – система пропонує оновити сторінку.

Виняткова ситуація 2: відсутність даних про об’єкти – система повідомляє про проблему з отриманням даних із NASA API.

Прецедент «Перегляд активності сонця».

Призначення: даний варіант використання надає можливість користувачу переглядати графік або таймлайн сонячної активності.

Основний потік подій: даний варіант використання починає виконуватися після вибору інструменту «CME», система надсилає запит до сервера для отримання інформації про корональні масові викиди за заданий період. На сторінці відображається інтерактивний графік або таймлайн, який демонструє активність сонця. Користувач може переглянути детальну інформацію про кожен викид, наприклад, час початку та координати.

Виняткова ситуація 1: помилка отримання даних – система повідомляє про недоступність даних і пропонує спробувати пізніше.

Прецедент «Пошук зображень та відео».

Призначення: даний варіант використання надає можливість користувачу шукати зображення та відео в бібліотеці NASA.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач вибирає інструмент «Image Search» і вводить ключове слово для пошуку у відповідне поле. Система надсилає запит до

сервера та отримує результати пошуку, які відображаються у вигляді галереї. Якщо необхідно, користувач може застосувати фільтри, наприклад, вибрати тип медіа (зображення чи відео) або задати діапазон років. Система оновлює результати відповідно до заданих фільтрів.

Виняткова ситуація 1: відсутність результатів пошуку – система повідомляє про те, що дані за запитом не знайдено.

Виняткова ситуація 2: помилка завантаження результатів – система повідомляє про проблему і пропонує оновити сторінку.

Прецедент «Фільтрація».

Призначення: даний варіант використання надає можливість користувачу застосовувати фільтри для уточнення результатів пошуку.

Основний потік подій: даний варіант використання починає виконуватися після виконання пошуку, користувач може застосувати додаткові фільтри, такі як тип медіа або діапазон років. Система обробляє задані параметри та оновлює результати відповідно до вибраних критеріїв, дозволяючи користувачу швидше знайти потрібну інформацію.

Виняткова ситуація 1: відсутні результати після фільтрації – система повідомляє про відсутність даних за обраними критеріями.

Прецедент «Вибір дати».

Призначення: даний варіант використання надає можливість користувачу вибирати дату для перегляду зображення APOD.

Основний потік подій: даний варіант використання починає виконуватися, коли на сторінці інструменту «APOD» користувач вводить дату вручну або користується інтерактивним календарем для вибору потрібної дати. Після підтвердження вибору система формує запит до сервера з зазначеною датою. У відповідь система отримує зображення APOD за вказаний день і оновлює сторінку, відображаючи нові дані.

Виняткова ситуація 1: недійсний формат дати – система повідомляє користувача про некоректний формат дати та пропонує ввести її повторно.

Виняткова ситуація 2: відсутність зображення за обрану дату – система інформує користувача, що дані за вказану дату недоступні.

Прецедент «Введення “солу”».

Призначення: даний варіант використання надає можливість користувачу вводити значення «солу» (марсіанського дня) для фільтрації фотографій ровера.

Основний потік подій: даний варіант використання починає виконуватися, коли на сторінці інструменту «Mars Rover Photos» користувач вводить значення «солу» у спеціальне поле. Після підтвердження введення система формує запит до сервера, зазначаючи вибраний «сол». У відповідь система отримує фотографії, зроблені ровером у вказаний день, та оновлює сторінку з новими даними.

Виняткова ситуація 1: некоректне значення «солу» – система повідомляє про помилку введення та пропонує ввести значення повторно.

Виняткова ситуація 2: відсутність фотографій за вказаний «сол» – система інформує користувача, що фотографії за вибраний «сол» недоступні.

Прецедент «Перегляд інформації про об’єкт».

Призначення: даний варіант використання надає можливість користувачу переглядати детальну інформацію про обраний об’єкт на інтерактивній мапі.

Основний потік подій: даний варіант використання починає виконуватися, коли на сторінці інструменту «NEO» користувач натискає на один із об’єктів, відображених на інтерактивній мапі Сонячної системи. Система отримує інформацію про вибраний об’єкт із раніше завантажених даних та виводить її у вигляді детальної інформаційної панелі. Користувач може переглядати такі деталі, як назва об’єкта, дата наближення до Землі, швидкість та відстань.

Виняткова ситуація 1: об’єкт не містить детальних даних – система інформує користувача про відсутність інформації для вибраного об’єкта.

Виняткова ситуація 2: помилка взаємодії з мапою – система повідомляє користувача про тимчасову недоступність функції і пропонує оновити сторінку.

Прецедент «Анімація руху».

Призначення: даний варіант використання надає можливість системі забезпечувати анімацію руху планет та близькоземних об’єктів на інтерактивній мапі.

Основний потік подій: даний варіант використання починає виконуватися після завантаження інтерактивної мапи Сонячної системи, система ініціює анімацію руху об’єктів. Планети рухаються по своїх орбітах навколо Сонця, а об’єкти NEO демонструють наближення до Землі згідно з отриманими даними. Анімація продовжується в реальному часі або за визначеним симуляційним інтервалом. Користувач може спостерігати рух об’єктів та взаємодіяти з мапою (змінювати масштаб, позицію).

Виняткова ситуація 1: анімація працює повільно – система повідомляє про недостатню продуктивність і пропонує зменшити якість візуалізації.

Виняткова ситуація 2: помилка завантаження даних для анімації – система інформує користувача про проблему з даними і пропонує оновити сторінку.

Прецедент «Запит до API NASA».

Призначення: даний варіант використання надає можливість серверу надсилати запити до NASA API для отримання необхідних даних про зображення, об’єкти або фотографії.

Основний потік подій: даний варіант використання починає виконуватися, коли сервер отримує запит від клієнтської частини, він формує відповідний запит до NASA API, включаючи параметри, необхідні для конкретного інструменту (наприклад, дату для APOD або «сол» для фотографій ровера). Сервер надсилає запит до NASA API, отримує відповіді з необхідними даними, після чого обробляє їх та повертає клієнтській частині для відображення користувачу.

Виняткова ситуація 1: помилка автентифікації (недійсний API-ключ) – система повідомляє адміністратора про необхідність оновлення API-ключа.

Виняткова ситуація 2: недоступність NASA API – система інформує користувача про тимчасову недоступність сервісу і рекомендує спробувати пізніше.

Виняткова ситуація 3: некоректні дані у відповіді NASA API – система повідомляє користувача про помилку отримання даних.

Прецедент «Запит до сервера».

Призначення: даний варіант використання надає можливість системі забезпечувати передачу запиту від клієнтської частини до серверної частини.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач ініціює дію в інтерфейсі (наприклад, вибір інструменту, перегляд фотографій або пошук зображень), клієнтська частина надсилає відповідний запит до серверної частини. Сервер обробляє запит, додаючи параметри для звернення до NASA API, і надсилає результати назад до клієнтської частини для відображення.

Виняткова ситуація 1: недоступність серверної частини – система повідомляє користувача про проблему з сервером і пропонує спробувати пізніше.

Виняткова ситуація 2: некоректний формат запиту – сервер повертає повідомлення про помилку. Система відображає помилку.

2.3 Діаграма класів

Діаграма класів – це ключовий елемент мови UML, призначений для візуалізації структурної організації системи. На ній відображаються класи з їх характеристиками (атрибутами і методами) та взаємозв’язками. Це допомагає розробникам спроєктувати архітектуру, визначити головні складові системи та їх взаємодію, що створює фундамент для подальшого об’єктно-орієнтованого програмування.

На рисунку 2.4 представлена діаграма класів клієнтської частини вебзастосунку для візуалізації космічних даних.

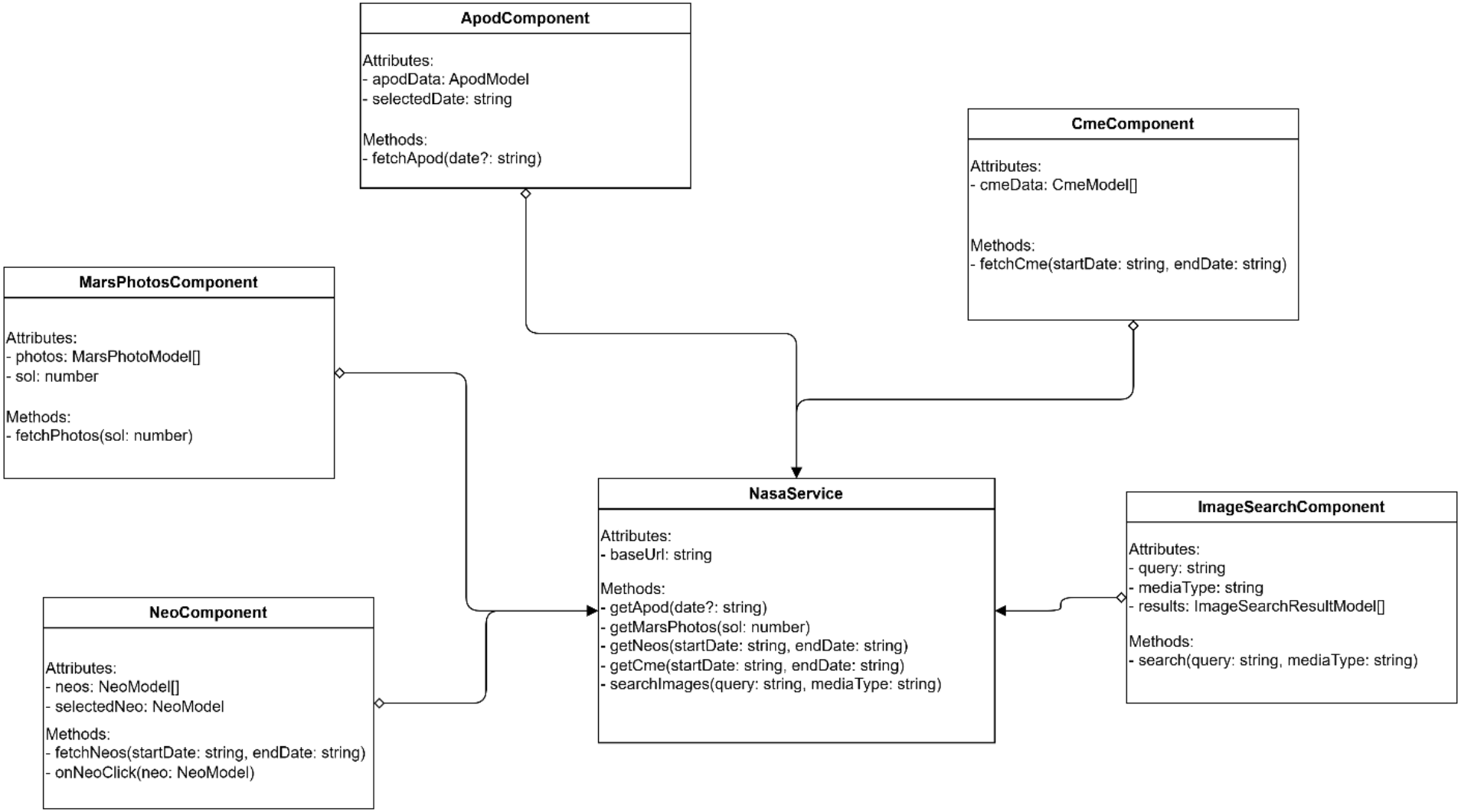


Рисунок 2.4 – Діаграма класів клієнтської частини

Кожен клас на діаграмі зображується як прямокутник з трьома розділами: назва, властивості (атрибути) та дії (методи). Між класами можуть існувати різні типи зв’язків. Асоціація показує просту взаємодію між класами. Агрегація відображає відношення «містить», де складові можуть існувати окремо від цілого. Композиція – це тісніший зв’язок, де частини невід’ємні від цілого. Успадкування дозволяє створювати ієрархію класів, де нащадки отримують властивості та поведінку від батьківського класу.

Таким чином, діаграма класів – це потужний інструмент для проєктування програмного забезпечення, який допомагає створити чітку архітектуру та спрощує подальшу розробку, тестування та підтримку системи.

2.4 Діаграма послідовності

Діаграма послідовності – це динамічний інструмент UML, який використовується для моделювання взаємодії між об’єктами в часі. Вона відображає послідовність, в якій відбувається обмін повідомленнями між об’єктами для виконання певного сценарію або функціональності. Основна мета цієї діаграми – проілюструвати послідовність подій у системі, що сприяє більш повному розумінню динамічної поведінки програми [15].

Діаграма послідовності складається з декількох основних елементів. Об’єкти, що беруть участь у взаємодії, зображуються у вигляді прямокутників, розташованих у вершині діаграми, із зазначенням їхніх імен і типів. Вертикальні лінії, що відходять від кожного об’єкта, називаються лініями життя і ілюструють тривалість існування об’єкта. Горизонтальні стрілки між об’єктами вказують на обмін повідомленнями, які супроводжуються підписами, що пояснюють зміст повідомлення або виклику методу. Послідовність стрілок вказує на хронологічний порядок виконання операцій.

Ця діаграма особливо корисна для моделювання складних сценаріїв, наприклад, пов’язаних з аутентифікацією користувачів, обробкою замовлень або виконанням транзакцій.

Однією з головних переваг діаграми послідовності є її здатність візуально відображати взаємодію між компонентами системи в часі, що сприяє більш повному розумінню логіки, яка лежить в її основі. Вона дозволяє виявити потенційні недоліки в дизайні системи, такі як надлишкові або відсутні взаємодії. Однак створення таких діаграм може бути тривалим процесом для великих систем з великою кількістю об’єктів. Тому доцільно використовувати їх для аналізу основних сценаріїв.

Отже, діаграма послідовності є ефективним інструментом для проєктування, документування та аналізу динамічних характеристик системи. Вона забезпечує краще розуміння поведінки системних об’єктів, тим самим полегшуючи розробку високоякісного і функціонального програмного забезпечення.

На рисунку 2.5 описана діаграма послідовності прецеденту «Перегляд APOD».

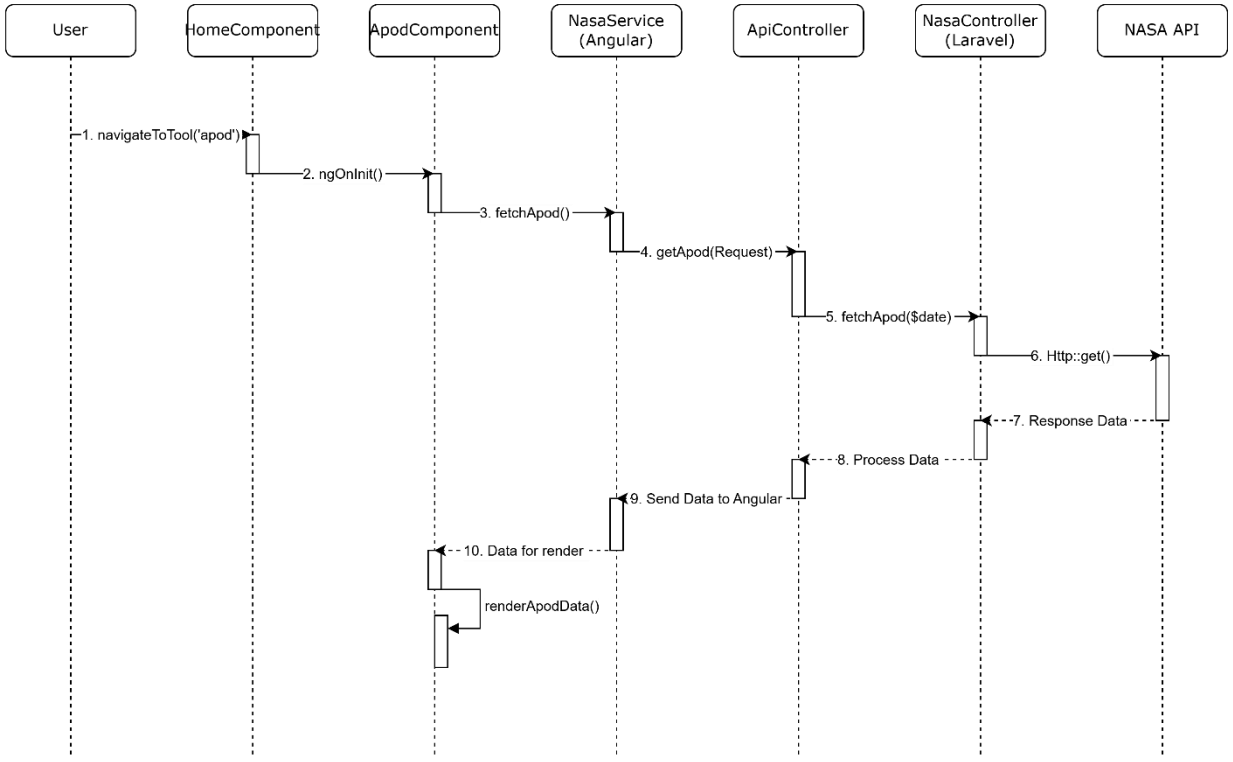


Рисунок 2.5 – Діаграма послідовності прецеденту «Перегляд APOD»

2.5 Діаграма діяльності

Діаграма діяльності – це один з динамічних інструментів UML, який використовується для моделювання процесів, логіки дій і потоків робіт у системі. Вона дозволяє візуалізувати послідовність дій та умови, за яких ці дії виконуються, що є перевагою для аналізу бізнес-процесів, алгоритмів та робочих процесів [15].

Фундаментальними компонентами діаграми діяльності є дії, які зображуються у вигляді прямокутників із закругленими кутами. Вони представляють окремі операції або кроки процесу. Потоки управління між діями ілюструються стрілками, які вказують на послідовність виконання. Початковий стан процесу зображується чорною крапкою, а кінцевий стан – концентричним колом. Крім того, діаграма може використовувати умовні оператори (розгалуження) для представлення рішень, а також вузли злиття для об’єднання декількох потоків.

Діаграма діяльності є цінним інструментом для моделювання як програмних процесів, так і бізнес-процесів.

Однією з ключових переваг діаграми діяльності є її універсальність. Її можна використовувати для моделювання різних процесів, від високорівневих бізнес-процесів до детальних алгоритмів. Діаграма сприяє виявленню логічних помилок у процесах, оптимізації робочих процесів та чіткій візуалізації послідовності дій. Її простий формат робить її цінним інструментом як для технічних, так і для нетехнічних користувачів [15].

Однак для великих або складних процесів з великою кількістю опцій діаграма діяльності може стати громіздкою. Тому дуже важливо обмежити її застосування конкретними сценаріями або розбити процес на окремі діаграми для полегшення розуміння.

Таким чином, діаграма діяльності є ефективним інструментом для вивчення та розвитку динамічних елементів системи. Вона полегшує розуміння логіки процесів, що, в свою чергу, підвищує їх ефективність і спрощує розробку програмного забезпечення.

На рисунках 2.6 – 2.7 наведено діаграму діяльності прецеденту «Перегляд APOD».

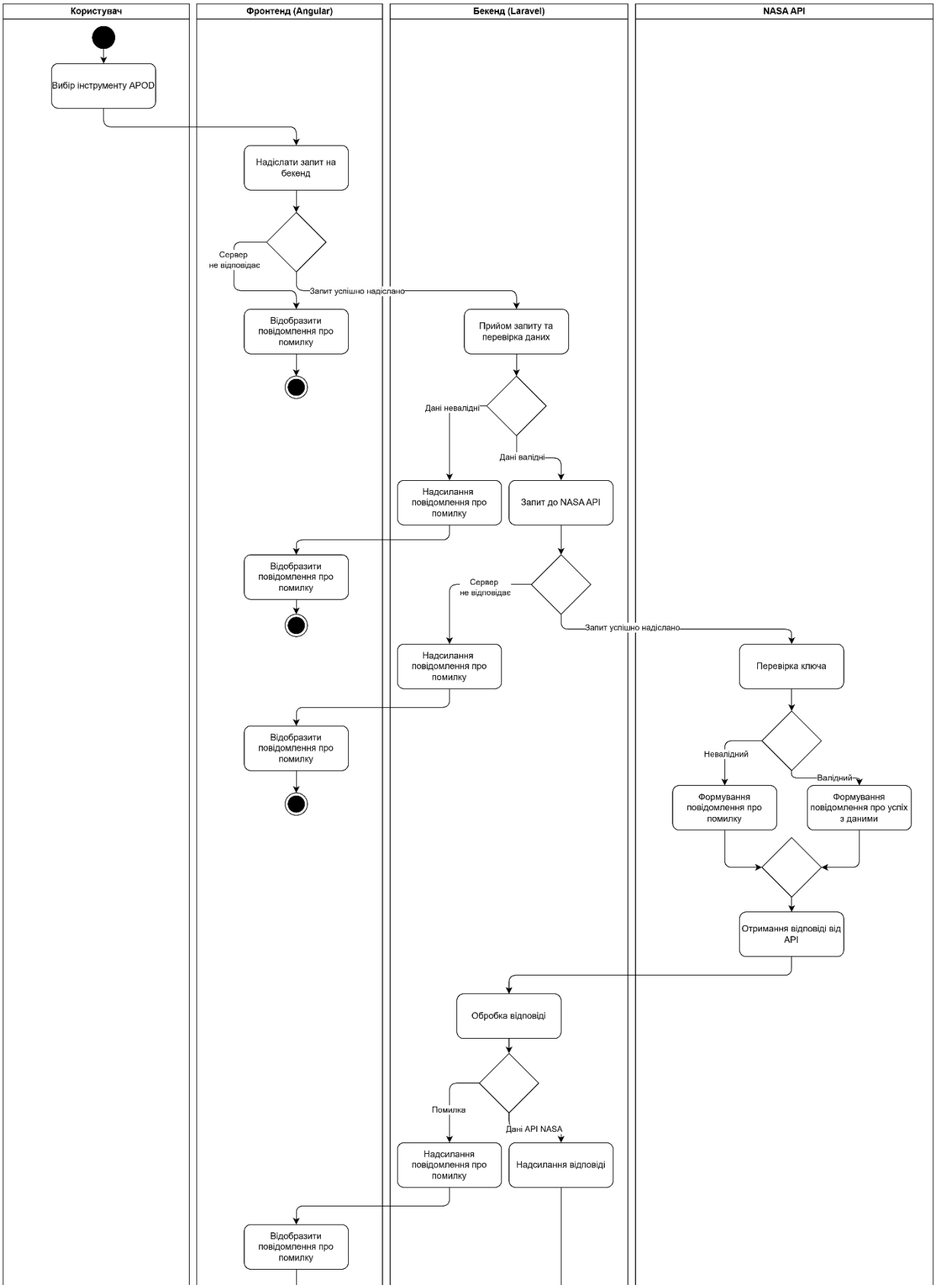


Рисунок 2.6 – Діаграма діяльності (1 частина)

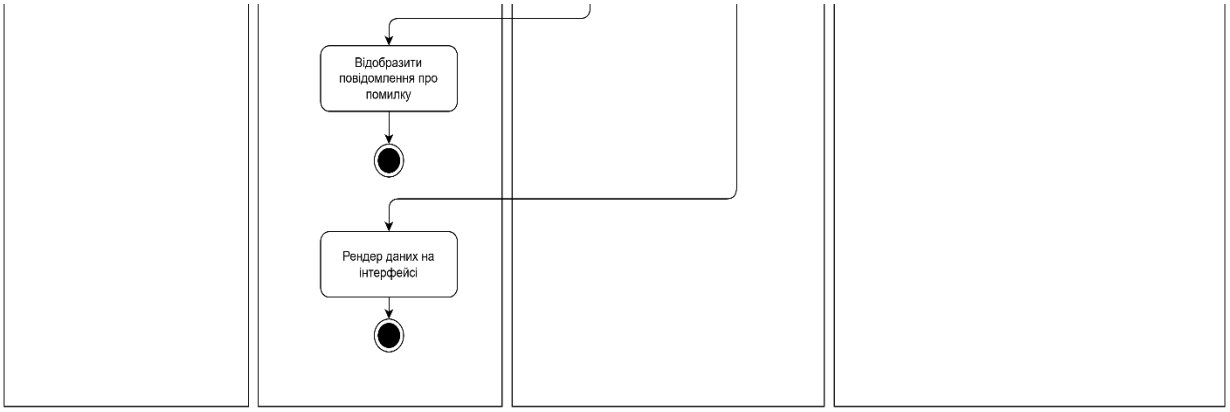


Рисунок 2.7 – Діаграма діяльності (2 частина)

2.6 Діаграма розгортання

Діаграма розгортання – це статичний інструмент UML, який використовується для моделювання фізичної архітектури системи. Вона відображає спосіб розгортання програмних компонентів на апаратних вузлах та ілюструє зв’язки між ними. Основна мета цієї діаграми – забезпечити візуальне представлення інфраструктури системи та її розподілу в реальному середовищі. Це полегшує планування розгортання та забезпечує його оптимальну продуктивність [15].

Основними компонентами діаграми розгортання є вузли, які представляють апаратні пристрої, сервери або інші елементи фізичної інфраструктури. Такі об’єкти зображуються у вигляді тривимірних прямокутників, що супроводжуються позначенням, наприклад, «Сервер бази даних» або «Клієнтський комп’ютер». Програмні компоненти, розгорнуті на цих вузлах, позначаються або всередині самих вузлів, або у вигляді вкладених прямокутників. Взаємозв’язки між вузлами представлені лініями, які позначають мережеве з’єднання або іншу форму взаємодії.

Діаграма розгортання є корисним та наочним інструментом для проєктування розподілених систем, включаючи вебдодатки, системи управління базами даних і мережеві сервіси. Наприклад, в контексті

багаторівневої вебархітектури діаграма може продемонструвати, що клієнтська програма знаходиться на пристроях користувачів, вебсервер обробляє запити користувачів, а сервер бази даних зберігає дані. Взаємозв'язки між вузлами пояснюють спосіб передачі даних між клієнтськими та серверними компонентами.

Ця діаграма полегшує розуміння фізичної архітектури системи серед розрізнених команд, включаючи розробників, мережевих адміністраторів та інженерів DevOps.

Однак діаграма розгортання може бути надто детальною або громіздкою для невеликих систем або проєктів з простою інфраструктурою, і в цьому випадку її корисність може бути обмеженою. Крім того, створення точної діаграми вимагає точної інформації про інфраструктуру та технологічні вимоги.

На рисунку 2.8 наведено діаграму розгортання системи.

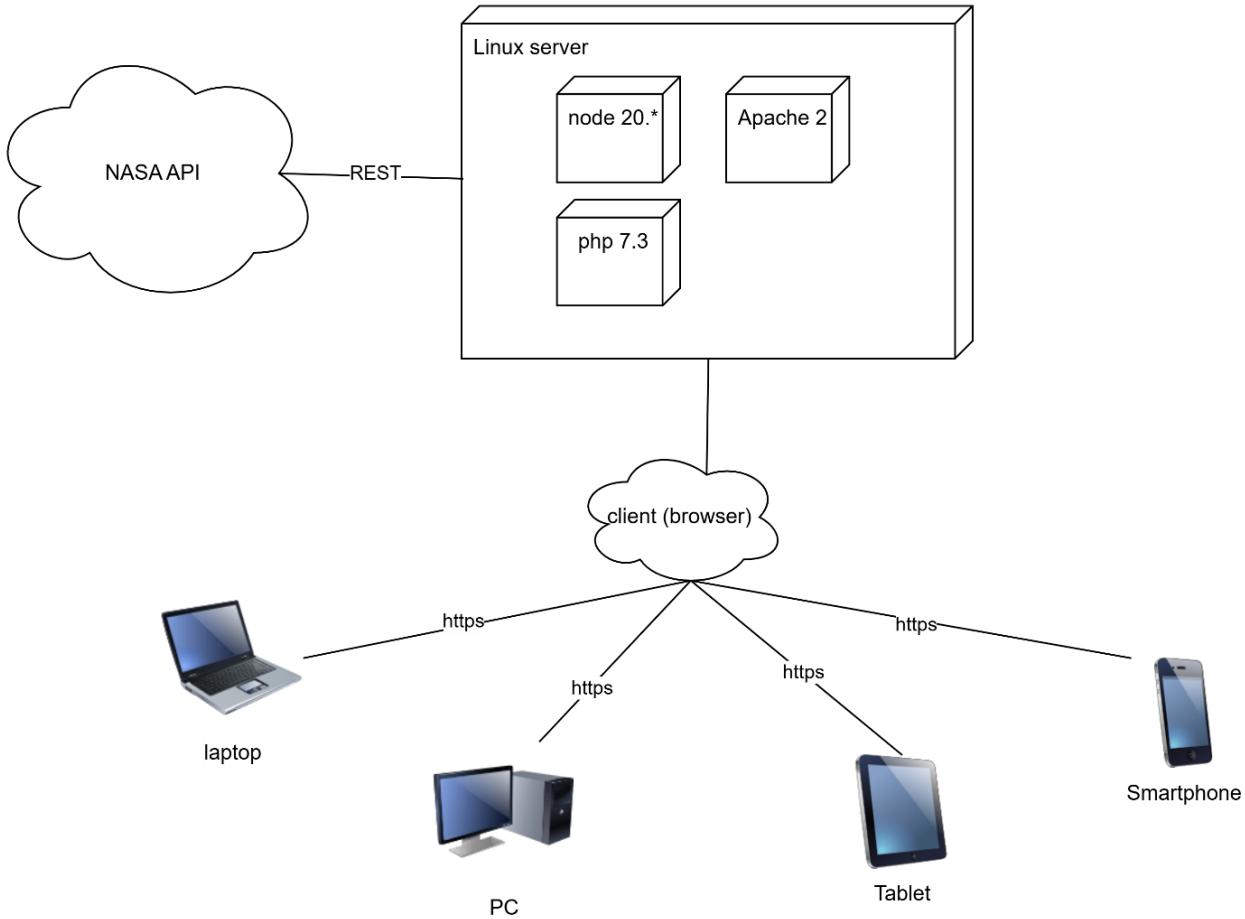


Рисунок 2.8 – Діаграма розгортання

Отже, діаграма розгортання є важливим інструментом для моделювання фізичної архітектури складних систем. Вона полегшує перевірку правильності розгортання компонентів у реальному середовищі, тим самим сприяючи стабільності, безпеці та ефективності системи.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Опис інструментів розробки

Для розробки вебзастосунку для візуалізації космічних даних засобами API NASA використовувалися інструменти, що забезпечують високу продуктивність та масштабованість.

Зі сторони фронтенду було обрано Angular завдяки його компонентній архітектурі, яка дозволяє легко створювати динамічні інтерфейси користувача. Він забезпечує інтеграцію з різними бібліотеками та має власний механізм обробки запитів через HTTP-клієнт [1, 2, 7].

Для бекенд-розробки було використано Laravel, один із найпопулярніших PHP-фреймворків. Він забезпечив готову архітектуру MVC, що сприяло кращій структурованості коду. Його інструменти для маршрутизації, взаємодії з API та обробки запитів дозволили створити гнучку серверну частину [5, 10].

Додатково використовувалися бібліотеки:

- Three.js для 3D-візуалізації Сонячної системи;
- Highcharts для створення інтерактивних графіків.

Інтеграція з API NASA забезпечила доступ до даних космічних місій, фотографій та інших ресурсів.

3.2 Основні класи системи

Фронтенд побудовано на основі компонентного підходу, що передбачає розбиття інтерфейсу на окремі модулі.

Основні класи, що використовуються в системі:

– NasaService – сервіс для обробки HTTP-запитів до API NASA.

Включає методи для завантаження даних для кожного з інструментів, таких як APOD, Mars Photos, тощо;

– SolarSystemMapComponent – клас, що відповідає за візуалізацію Сонячної системи з використанням Three.js.: він містить методи для додавання планет, орбіт та візуалізації об’єктів NEO.

Також до основних можна віднести класи, які візуалізують конкретні дані отримані від API NASA. Кожен з цих класів має відповідні HTML-шаблони та CSS-файли для відображення даних та стилізації.

Нижче приведено код компонента ImagesSearch (рис. 3.1).

```
import { Component } from '@angular/core';
import {NasaService} from "../../services/nasa.service";

@Component({  Show usages
  selector: 'app-images-search',
  templateUrl: './images-search.component.html',
  styleUrls: ['./images-search.component.scss']
})
export class ImagesSearchComponent {
  query :string  = 'moon';
  mediaType :string  = 'image';
  images: any[] = [];

  constructor(private nasaService: NasaService) {}  no usages

  searchImages(): void {  Show usages
    this.nasaService.searchImages(this.query, this.mediaType).subscribe((data: any) => {
      this.images = data.collection.items;
    }, error :any  => {
      console.error('Error searching images:', error);
    });
  }
}
```

Рисунок 3.1 – Компонент ImagesSearch

З боку серверу основними класами є ті, що забезпечують обробку запитів, взаємодію з API NASA, маршрутизацію та обробку бізнес-логіки.

Ось ключові класи, які використовуються в проєкті:

– NasaController – основний контролер для обробки запитів до API NASA (використовується для маршрутів, що обслуговують клієнтські запити Angular);
– NasaApiService – клас для роботи з API NASA (він викликається контролером та реалізує бізнес-логіку для підготовки запитів і обробки відповідей).

Нижче приведено одну із функцій NasaController (рис. 3.2).

```php
public function searchImages(Request $request)
{
    $response = Http::get('https://images-api.nasa.gov/search', [
        'q' => $request->input('query', ''),           // Пошуковий запит
        'media_type' => $request->input('media_type', 'image'), // Тип медіа (image, video, audio)
        'year_start' => $request->input('year_start'), // Початковий рік
        'year_end' => $request->input('year_end'),     // Кінцевий рік
    ]);

    if ($response->successful()) {
        return response()->json($response->json());
    }

    return response()->json(['error' => 'Unable to fetch images'], 500);
}
```

Рисунок 3.2 – Функція «searchImages» NasaController

Детально ознайомитися з прикладами коду клієнтської та серверної частин можна в додатках А, Б, В.

3.3 Отримання API-ключа NASA

Для того щоб взаємодіяти з API NASA, необхідно насамперед згенерувати ключ. Для цього потрібно перейти на сайт https://api.nasa.gov/ і заповнити форму заявки для генерації API-ключа (див. рис. 3.3).

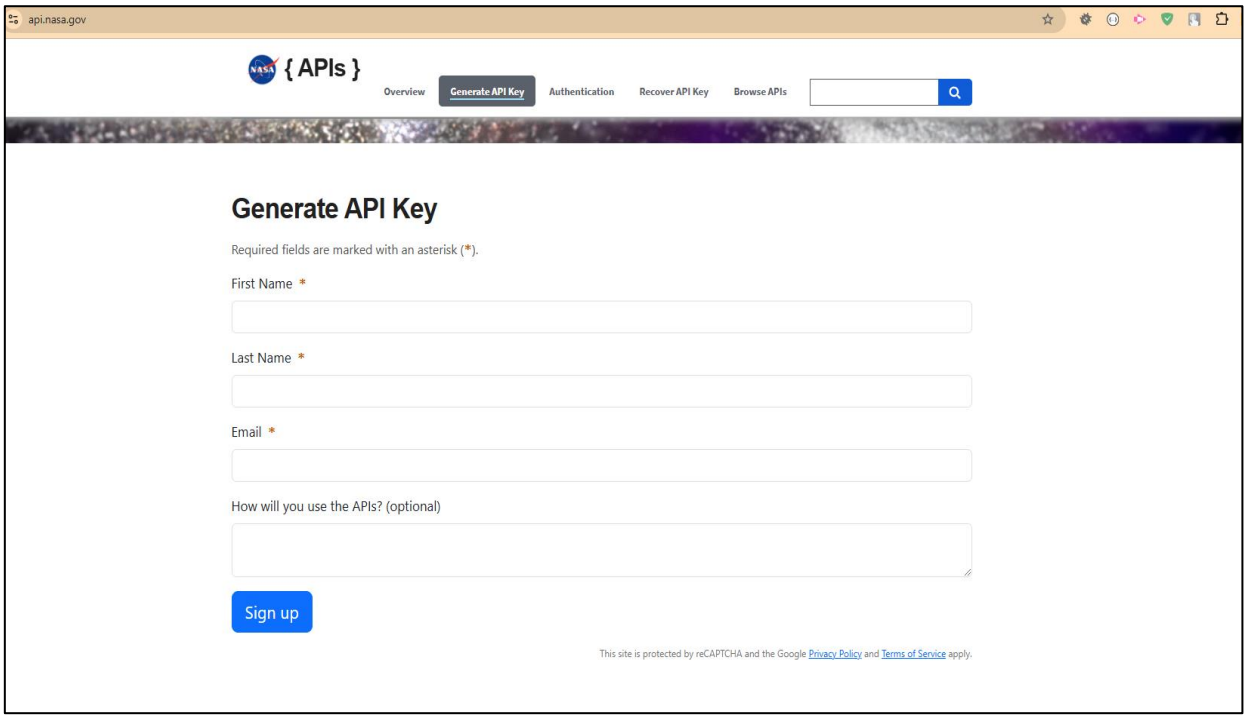


Рисунок 3.3 – Форма запиту для генерації API-ключа

Після заповнення форми, необхідно натиснути на кнопку «Sign up». Після підтвердження запиту, на вказану поштову адресу надійде повідомлення з API-ключем (див. рис. 3.4).

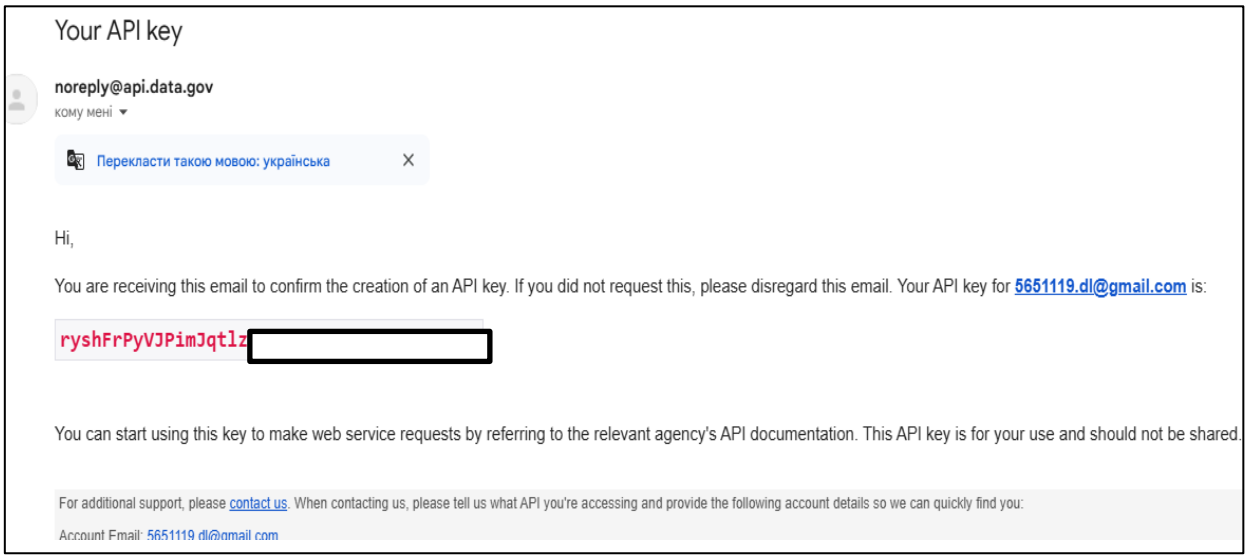


Рисунок 3.4 – API-ключ

За допомогою наданого ключа можна робити 1000 запитів до NASA API в годину.

3.4 Тестування проєкту

Тестування окремих компонентів (Unit-тестування) є важливим елементом розробки Angular-додатків. Фреймворк включає потужний набір інструментів для тестування: Jasmine для написання тестових сценаріїв та Karma як платформу виконання тестів [2, 4].

При створенні нового компонента, сервісу чи класу, Angular автоматично генерує відповідний тестовий файл з розширенням *.spec.ts* в тій же директорії. Основні налаштування тестового середовища зберігаються в файлах *karma.conf.js* та *test.ts*.

Для спрощення процесу тестування Angular пропонує спеціалізовані інструменти:

- TestBed – для тестування компонентів;
- HttpClientTestingModule – для перевірки HTTP-взаємодій;
- RouterTestingModule – для тестування маршрутизації.

При роботі з компонентами використовуються методи TestBed.configureTestingModule та TestBed.createComponent, які дозволяють налаштувати ізольоване тестове оточення. Сервіси можна тестувати напряму через створення їх екземплярів [6].

Jasmine забезпечує можливості створення mock-об’єктів та spy-функцій для імітації залежностей та відстеження їх використання. Запуск тестів здійснюється через команду «*ng test*» або безпосередньо з IDE (наприклад, VSCode чи WebStorm).

Unit-тестування сприяє стабільності додатку, полегшує його підтримку та покращує якість коду. Це дозволяє виявляти помилки на ранніх етапах та спрощує подальший рефакторинг.

На рисунку 3.5 продемонстровано приклад unit-тестів для компонента ImagesSearch.

```
it('should create', () => {
  expect(component).toBeTruthy();
});

it('should call searchImages when search button is clicked', () => {
  const mockResponse = {
    collection: { items: [{ links: [{ href: 'image_url' }], data: [{ title: 'Test Image' }] }] },
  };
  nasaServiceSpy.searchImages.and.returnValue(of(mockResponse));
  component.query = 'moon';
  component.mediaType = 'image';

  component.searchImages();

  expect(nasaServiceSpy.searchImages).toHaveBeenCalledWith('moon', 'image');
  expect(component.images.length).toBe(1);
  expect(component.images[0].links[0].href).toBe('image_url');
});
```

Рисунок 3.5 – Unit-тести компонента ImagesSearch

Для запуску тестів використовується команда «*ng test*», після чого результати відображаються у браузері [1, 7]. Це дозволяє виявити можливі проблеми в системі (рис. 3.6).

```
16 specs, 9 failures, randomized with seed 43204
Spec List | Failures

NeoComponent
  × should create
NasaService
  × should handle errors gracefully
  × should fetch images based on query and media type
  • should be created
ApodComponent
  × should create
MarsPhotosComponent
  × should create
ImagesSearchComponent
  • should create
  • should call searchImages when search button is clicked
  • should handle empty response gracefully
AppComponent
  • should have as title 'nasa-visualization-frontend'
  × should render title
  • should create the app
HomeComponent
  • should create
HelioviewerComponent
  × should create
DonkiCmeComponent
  × should create
SolarSystemMapComponent
  × should create
```

Рисунок 3.6 – Результат тестування ImagesSearch

Детально ознайомитися з повним кодом тесту можна в додатку Г.

3.5 Керівництво користувача

Вебзастосунок для візуалізації космічних даних засобами API NASA призначений для перегляду та дослідження даних NASA через інтерактивний інтерфейс. Його функціональність включає доступ до космічних зображень, даних про Марс, інформації про об’єкти, що наближаються до Землі, а також перегляд сонячної активності. У цьому керівництві наведено покрокові інструкції щодо використання основних інструментів.

3.5.1 Головна сторінка

Після переходу на сайт користувач потрапляє на головну сторінку, яка містить загальну інформацію про доступні інструменти. Головна сторінка виконана в космічному стилі з фоновим зображенням космосу, що створює атмосферу занурення. У центрі екрану розташовані інтерактивні картки з назвами інструментів. Кожна картка містить назву інструменту, короткий опис та інтерактивну кнопку для переходу до відповідної сторінки.

На рисунку 3.7 продемострована головна сторінка вебзастосунку.

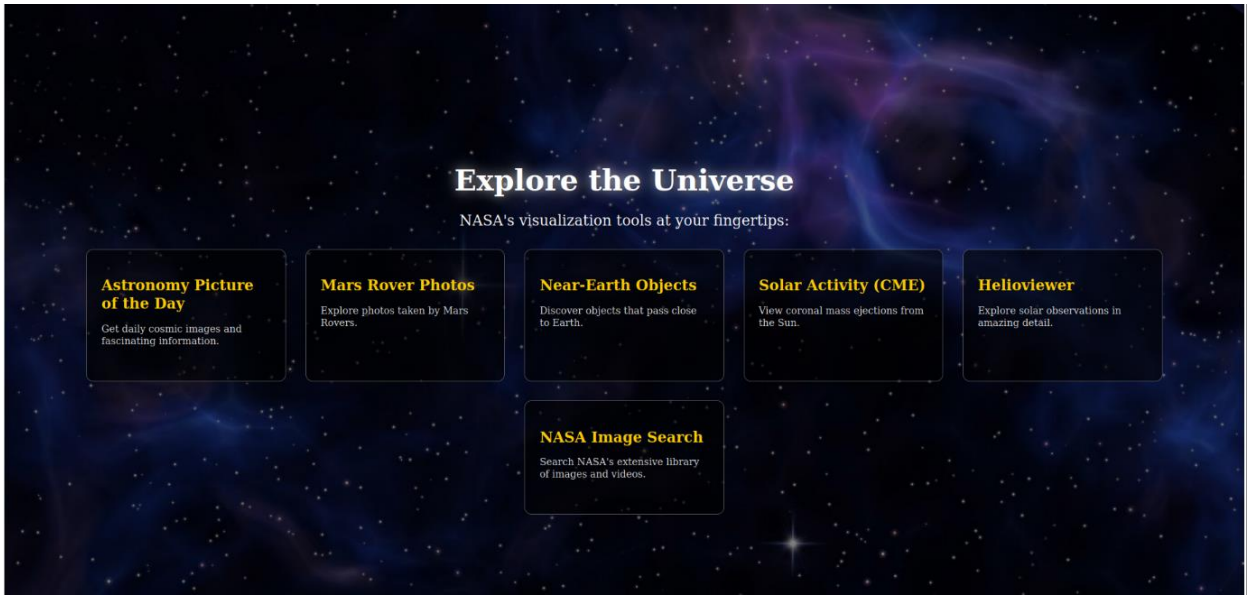


Рисунок 3.7 – Головна сторінка

3.5.2 Інструмент APOD

Щоб скористатися інструментом «Astronomy Picture of the Day (APOD)», користувачу потрібно вибрати відповідну картку на головній сторінці. На сторінці APOD автоматично завантажується поточне зображення дня із описом. Під зображенням розміщено текстовий опис, що розповідає про особливості цього знімка.

На рисунку 3.8 продемонстровано інтерфейс сторінки інструменту APOD.



Рисунок 3.8 – Інтерфейс сторінки APOD

3.5.3 Інструмент Mars Rover Photos

Для перегляду фотографій з Марсу, зроблених марсоходами, користувачу необхідно вибрати інструмент «Mars Rover Photos». На сторінці розташоване текстове поле для введення номера марсіанського дня (солу). У полі пошуку за замовчуванням вказано значення «1000» (див. рис. 3.9), за

необхідності користувач може його змінити на інше. Для цього потрібно ввести номер солу та натиснути на кнопку «Search» (див. рис. 3.10), і зображення, зроблені марсоходами, відобразяться в галереї під полем пошуку. Галерея адаптивна, зображення показуються у вигляді карток із зазначенням дати та назви камери.

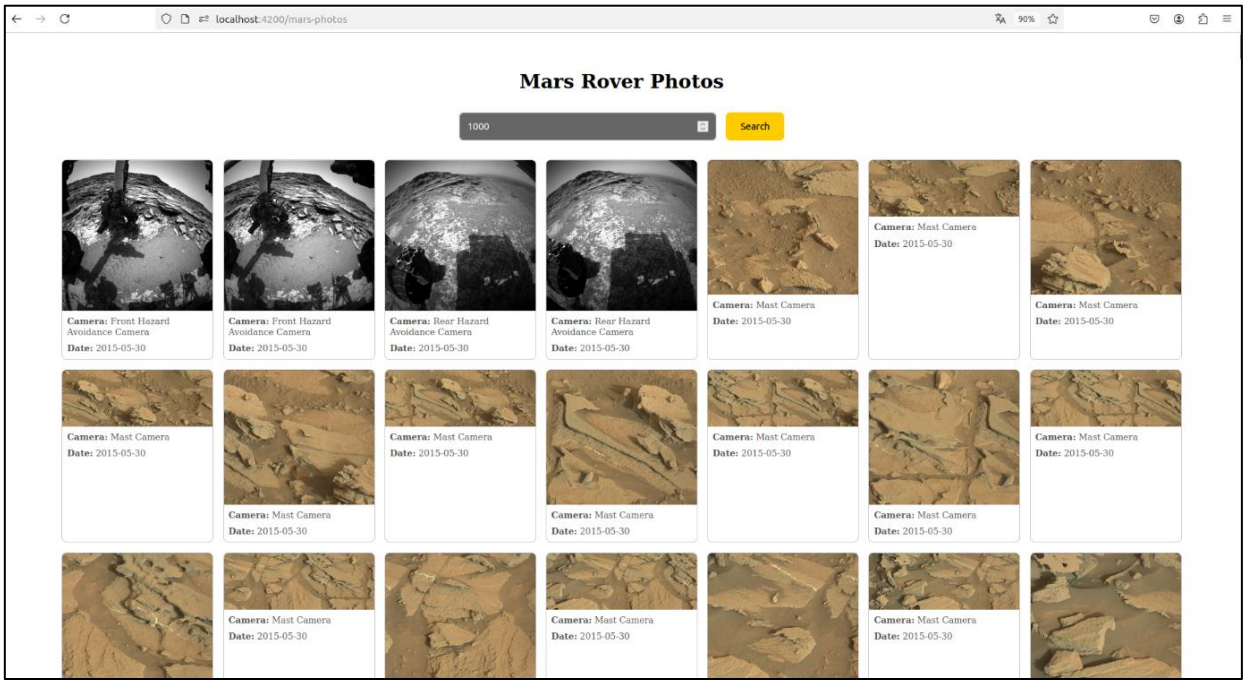


Рисунок 3.9 – Інтерфейс сторінки Mars Rover Photos

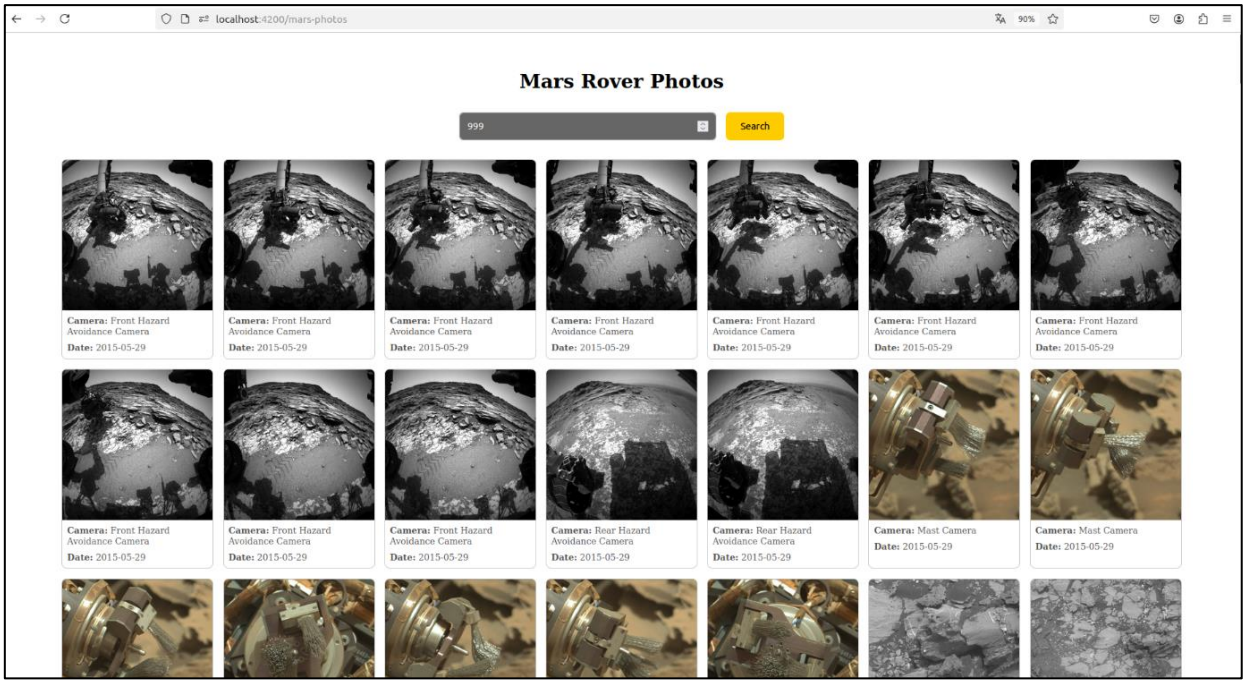


Рисунок 3.10 – Зміна «солу»

3.5.4 Інструмент NASA Image Search

Сторінка «NASA Image Search» надає можливість пошуку зображень або відео з бібліотеки NASA. Користувач вводить ключове слово в текстове поле, обирає тип медіа за допомогою випадаючого списку та натискає кнопку «Search».

Результати пошуку відображаються у вигляді галереї (див. рис. 3.11 – 3.12).

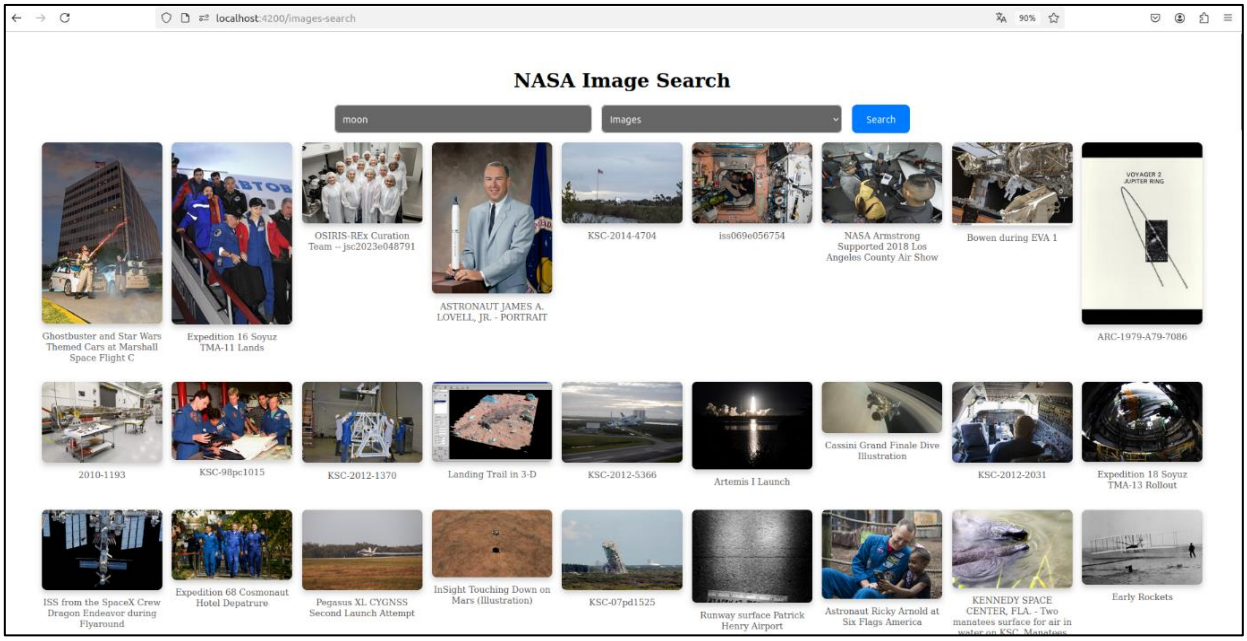


Рисунок 3.11 – Результати пошуку зображень

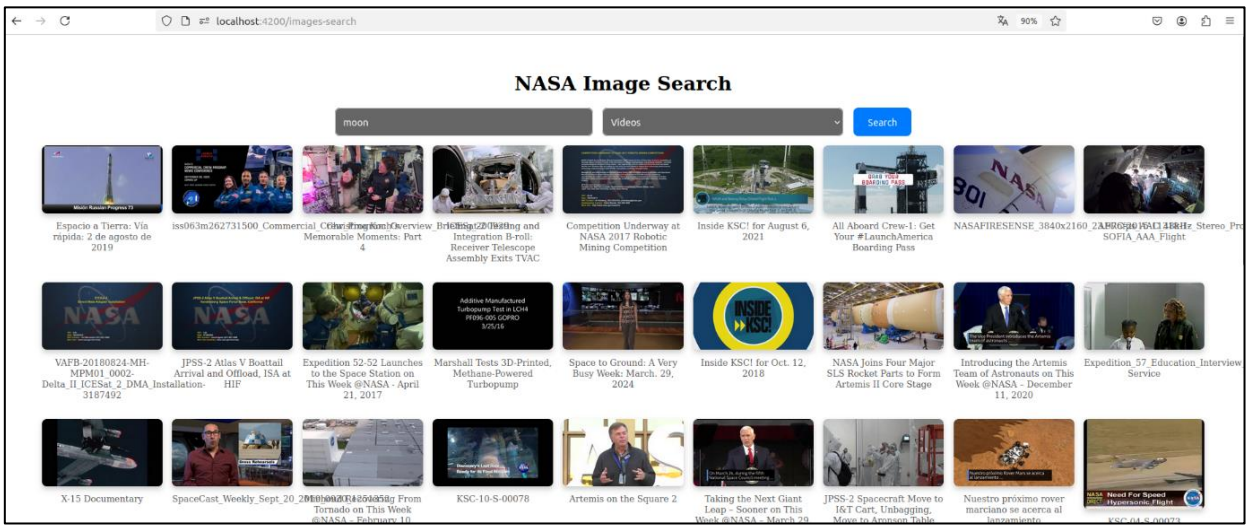


Рисунок 3.12 – Результати пошуку відео

3.5.5 Інструмент Near-Earth Objects (NEO)

Сторінка «Near-Earth Objects (NEO)» дозволяє користувачу переглядати інтерактивну карту Сонячної системи, де візуалізовано об’єкти, що наближаються до Землі. На карті зображено всі планети Сонячної системи, а також об’єкти NEO у вигляді зелених точок із позначенням їхніх траєкторій (див. рис. 3.13).

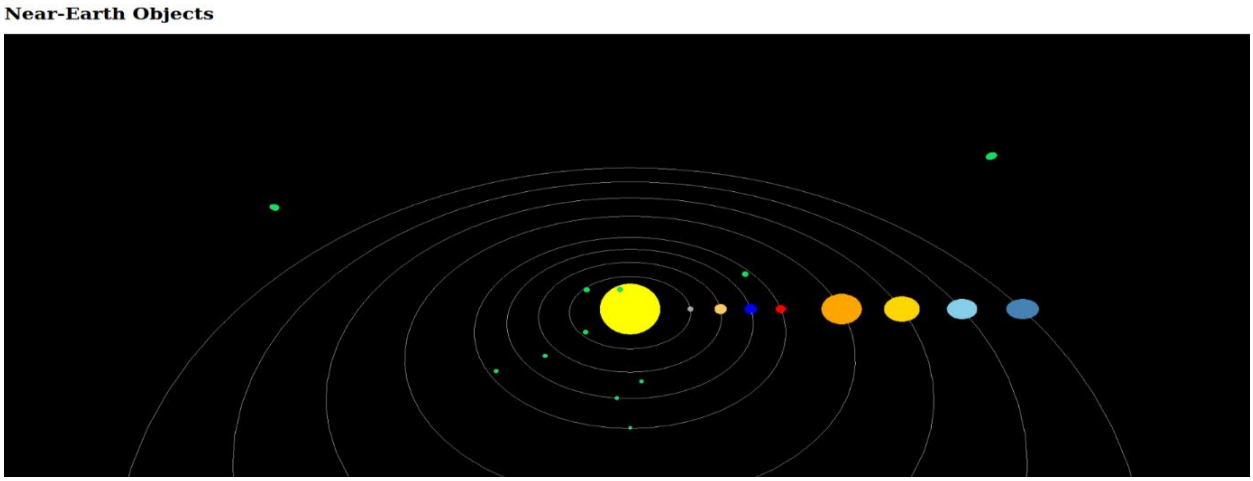


Рисунок 3.13 – Інтерфейс сторінки Near-Earth Objects (NEO)

Після натискання на об’єкт відкривається інформаційна панель із детальною інформацією, такою як дата найближчого зближення, відстань до Землі та швидкість (див. рис. 3.14 – 3.15).

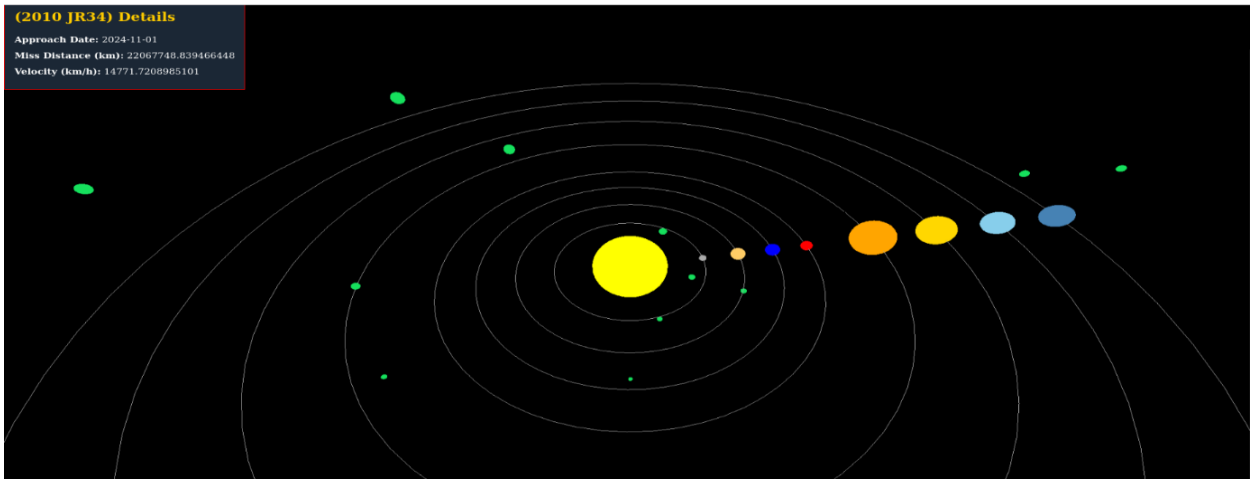


Рисунок 3.14 – Інформація про об’єкт NEO

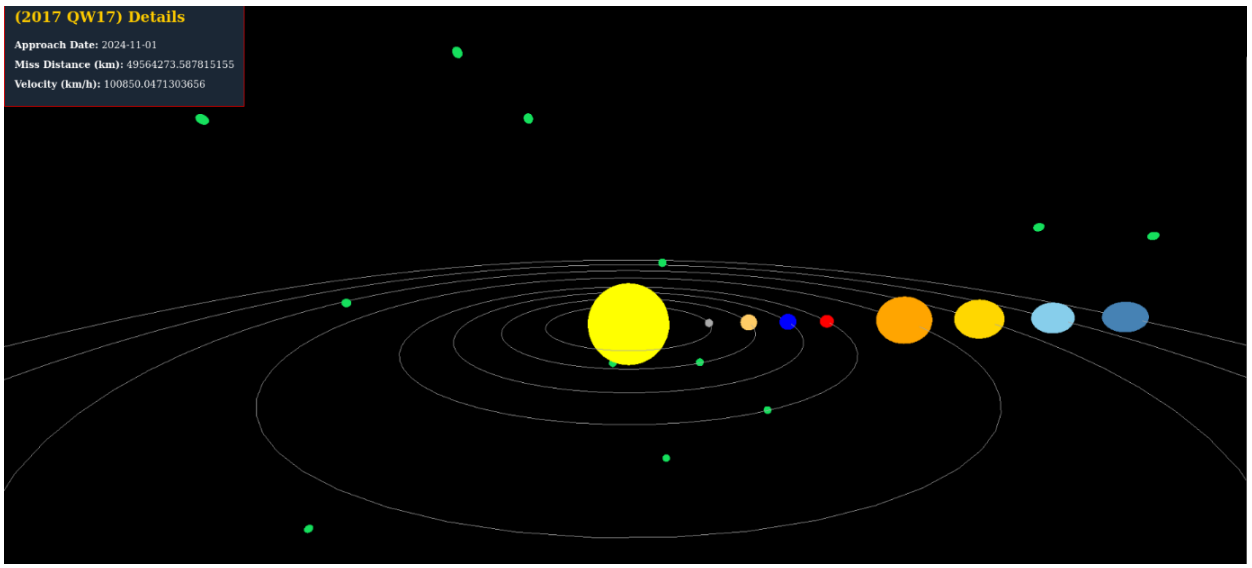


Рисунок 3.15 – Інформація про об’єкт NEO

3.5.6 Інструмент Solar Activity (CME)

Інструмент «Solar Activity (CME)» дозволяє переглядати графіки корональних викидів мас із Сонця. На сторінці автоматично завантажуються дані за останній тиждень. Графік побудований у вигляді таймлайнів, які візуалізують час та силу викидів. Інтерфейс дозволяє швидко аналізувати ці дані.

На рисунку 3.16 показано приклад графіка, створеного на основі даних NASA.

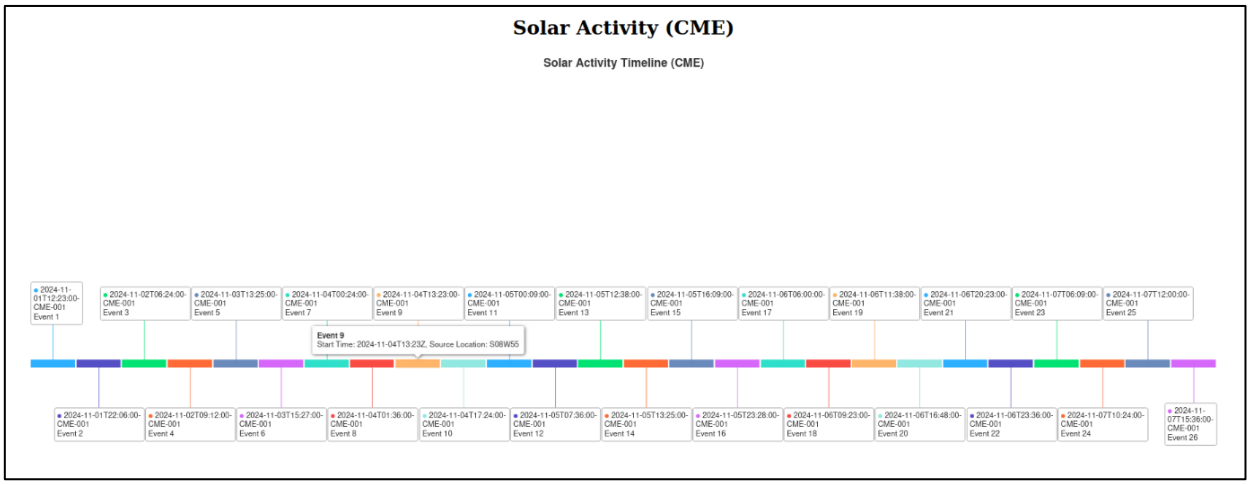


Рисунок 3.16 – Графік Solar Activity (CME)

Кожна сторінка додатка адаптивна і відображається коректно на різних пристроях, включаючи мобільні телефони та планшети. Використання текстових полів та кнопок стилізовано так, щоб забезпечити максимальний комфорт під час роботи з додатком. Функціональність забезпечується через інтеграцію з API NASA, що гарантує актуальність і точність даних.

ВИСНОВКИ

У ході виконаної роботи було створено та протестовано вебзастосунок для візуалізації космічних даних. Моделювання за допомогою UML як структурних компонентів, так і процесів взаємодії, дозволило досягти надійності та якості в процесі розробки системи. Система успішно виконує функції візуалізації даних отриманих від API NASA. Процес модульного тестування продемонстрував, що основні компоненти системи функціонують належним чином.

У відповідності з метою кваліфікаційної роботи було розроблено вебзастосунок для візуалізації космічних даних з використаннями наступних технологій:

– Angular для реалізації frontend;

– Laravel для реалізації backend;

– Three.js та Highcharts для інтерактивної візуалізації.

У відповідності з поставленими задачами були виконані наступні етапи створення вебзастосунку:

− сформовані функціональні/нефункціональні вимоги до системи, розглянуто та наведено порівняння аналогів;

− спроєктована та побудована структура системи (побудовані UML-діаграми: прецедентів, класів, послідовності, діяльності та розгортання; також надано опис кожного з варіантів використання діаграми прецедентів);

− реалізовано вебзастосунок для візуалізації космічних даних (наведено інструкцію по отриманню ключа API NASA; розроблено керівництво користувача);

− протестовано роботу системи.

ПЕРЕЛІК ПОСИЛАНЬ

1. Angular Developer Documentation. URL: https://angular.io/docs/ (дата звернення: 18.09.2024).
2. Bampakos A., Deeleman P. Learning Angular: A no-nonsense guide to building web applications with Angular 15. Birmingham : Packt Publishing, 2023. 446 p.
3. Celestia. URL: https://celestiaproject.space/ (дата звернення: 30.08.2024).
4. Chebbi L. Reactive Patterns with RxJS for Angular: A practical guide to managing your Angular application’s data reactively and efficiently using RxJS 7. Birmingham : Packt Publishing, 2022. 224 p.
5. Dwivedi R. Laravel Essentials: A Step-by-Step Guide for Web Developers: «Laravel by Example: Building Real-World Applications». Michigan : Independently published, 2023. 269 p.
6. Fain Y., Moiseev A. Angular Development with TypeScript. New York : Manning Publications, 2019. 560 p.
7. Freeman A. Pro Angular: Build Powerful and Dynamic Web Apps. London : Apress, 2022. 905 p.
8. Gaitatzis T. Learn REST APIs: Your guide to how to find, learn, and connect to the REST APIs that powers the Internet of Things revolution. BackupBrain Press, 2019. 109 p.
9. Heavens-Above. URL: https://www.heavens-above.com/ (дата звернення: 29.08.2024).
10. Laravel 8 Developer Documentation. URL: https://laravel.com/docs/8.x/ (дата звернення: 15.09.2024).
11. NASA API. URL: https://api.nasa.gov/ (дата звернення: 20.08.2024).
12. NASA’s Eyes. URL: https://science.nasa.gov/eyes/ (дата звернення: 28.08.2024).
13. SkyView. URL: https://skyview.gsfc.nasa.gov/current/cgi/titlepage.pl (дата

звернення: 28.08.2024).

14. Stellarium. URL: https://stellarium.org/uk/ (дата звернення: 29.08.2024).
15. Unhelkar B. Software Engineering with UML. Boca Raton : Auerbach Publications, 2020. 426 p.
16. WorldWide Telescope. URL: https://worldwidetelescope.org/home/ (дата звернення: 28.08.2024).

ДОДАТОК А

Angular-компонент

```
import { Component, AfterViewInit, ElementRef, ViewChild } from '@angular/core';
import * as THREE from 'three';
import { NasaService } from 'src/app/services/nasa.service';
import { OrbitControls } from 'three/examples/jsm/controls/OrbitControls';

@Component({
  selector: 'app-solar-system-map',
  templateUrl: './solar-system-map.component.html',
  styleUrls: ['./solar-system-map.component.scss']
})
export class SolarSystemMapComponent implements AfterViewInit {
  @ViewChild('rendererContainer', { static: false }) rendererContainer!: ElementRef;

  private scene!: THREE.Scene;
  private camera!: THREE.PerspectiveCamera;
  private renderer!: THREE.WebGLRenderer;

  selectedNeo: any = null;

  constructor(private nasaService: NasaService) {}

  ngAfterViewInit(): void {
    this.initScene();
    this.addPlanets();
    this.fetchNeoData();
    this.animate();
  }

  private initScene(): void {
    this.scene = new THREE.Scene();
    this.scene.background = new THREE.Color(0x000000);

    this.camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 0.1, 1000);
    this.camera.position.set(0, 50, 50); // Камера зверху

    this.renderer = new THREE.WebGLRenderer();
```

```
  this.renderer.setSize(window.innerWidth, window.innerHeight);
  this.rendererContainer.nativeElement.appendChild(this.renderer.domElement);

  // Додаємо OrbitControls
  const controls = new OrbitControls(this.camera, this.renderer.domElement);
  controls.enableDamping = true; // Плавність руху камери
  controls.dampingFactor = 0.1;

  this.renderer.domElement.addEventListener('click', this.onObjectClick.bind(this), false);
}

private addPlanets(): void {
  const planets = [
    { name: 'Mercury', distance: 10, size: 0.5, color: 0xaaaaaa },
    { name: 'Venus', distance: 15, size: 1, color: 0xffcc66 },
    { name: 'Earth', distance: 20, size: 1, color: 0x0000ff },
    { name: 'Mars', distance: 25, size: 0.8, color: 0xff0000 },
    { name: 'Jupiter', distance: 35, size: 3, color: 0xffa500 },
    { name: 'Saturn', distance: 45, size: 2.5, color: 0xffd700 },
    { name: 'Uranus', distance: 55, size: 2, color: 0x87ceeb },
    { name: 'Neptune', distance: 65, size: 2, color: 0x4682b4 }
  ];

  const sunGeometry = new THREE.SphereGeometry(5, 32, 32);
  const sunMaterial = new THREE.MeshBasicMaterial({ color: 0xffff00 });
  const sun = new THREE.Mesh(sunGeometry, sunMaterial);
  sun.userData = { name: 'Sun' };
  this.scene.add(sun);

  planets.forEach(planet => {
    this.addPlanet(planet.distance, planet.size, planet.color, planet.name);
    this.addOrbit(planet.distance);
  });
}

private addPlanet(distance: number, size: number, color: number, name: string): void {
  const planetGeometry = new THREE.SphereGeometry(size, 32, 32);
  const planetMaterial = new THREE.MeshBasicMaterial({ color: color });
  const planetMesh = new THREE.Mesh(planetGeometry, planetMaterial);
  planetMesh.position.set(distance, 0, 0);
  planetMesh.userData = { name: name };
  this.scene.add(planetMesh);
```

```
}

private addOrbit(radius: number): void {
  const orbitCurve = new THREE.EllipseCurve(
    0, 0,    // центр
    radius, radius // радіус
  );

  const points = orbitCurve.getPoints(64);
  const orbitGeometry = new THREE.BufferGeometry().setFromPoints(points);

  const orbitMaterial = new THREE.LineBasicMaterial({ color: 0xaaaaaa });
  const orbit = new THREE.Line(orbitGeometry, orbitMaterial);

  orbit.rotation.x = Math.PI / 2; // Розміщення в площині
  this.scene.add(orbit);
}



private fetchNeoData(): void {
  const startDate = '2024-11-01';
  const endDate = '2024-11-07';
  this.nasaService.getNeos(startDate, endDate).subscribe((data: any) => {
    const objects = data.near_earth_objects[startDate] || [];
    this.addNeoObjects(objects);
  });
}

private addNeoObjects(neos: any[]): void {
  neos.forEach(neo => {
    const neoGeometry = new THREE.SphereGeometry(0.5, 16, 16);
    const neoMaterial = new THREE.MeshBasicMaterial({ color: 0x16e05
```

ДОДАТОК Б

Angular-сервіс

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})

export class NasaService {
  private baseUrl = 'http://127.0.0.1:8000/api/nasa';

  constructor(private http: HttpClient) {}

  // APOD (Astronomy Picture of the Day)
  getApod(): Observable<any> {
    return this.http.get(${this.baseUrl}/apod);
  }

  // Mars Rover Photos
  getMarsPhotos(sol: number = 1000): Observable<any> {
    return this.http.get(${this.baseUrl}/mars-photos, { params: { sol: sol.toString() } });
  }

  // Near-Earth Objects
  getNeos(startDate: string, endDate: string): Observable<any> {
    return this.http.get(${this.baseUrl}/neo, { params: { start_date: startDate, end_date: endDate } });
  }

  // Solar Activity (CME)
  getDonkiCme(startDate: string, endDate: string): Observable<any> {
    return this.http.get(${this.baseUrl}/donki-cme, { params: { start_date: startDate, end_date: endDate } });
  }

  // NASA Image Search
  searchImages(query: string, mediaType: string = 'image', yearStart?: string, yearEnd?: string): Observable<any> {
    const params: any = { q: query, media_type: mediaType };
```

```
    if (yearStart) params.year_start = yearStart;
    if (yearEnd) params.year_end = yearEnd;
    return this.http.get(${this.baseUrl}/images-search, { params });
  }

  getHelioviewerData(date: string): Observable<any> {
    return this.http.get(${this.baseUrl}/helioviewer, { params: { date } });
  }
}
```

ДОДАТОК В

Laravel-контролер

```
<?php

namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Http;
use Illuminate\Support\Facades\Config;

class NasaController extends Controller
{
    private $nasaApiKey;

    public function __construct()
    {
      $this->nasaApiKey = env('NASA_API_KEY');
    }

   public function getAstronomyPicture(Request $request)
   {
      $response = Http::get('https://api.nasa.gov/planetary/apod', [
         'api_key' => $this->nasaApiKey,
         'date' => $request->input('date', date('Y-m-d')), // Опціонально: конкретна дата
         'hd' => $request->input('hd', true),              // Отримати HD-зображення
      ]);

      if ($response->successful()) {
         return response()->json($response->json());
      }

      return response()->json(['error' => 'Unable to fetch APOD data'], 500);
   }

   public function getMarsPhotos(Request $request)
   {
      $response = Http::get('https://api.nasa.gov/mars-photos/api/v1/rovers/curiosity/photos', [
```

```
        'api_key' => $this->nasaApiKey,
        'sol' => $request->input('sol', 1000),
        'camera' => $request->input('camera'),
    ]);

    if ($response->successful()) {
        return response()->json($response->json());
    }

    return response()->json(['error' => 'Unable to fetch Mars Rover photos'], 500);
}

public function getDonkiCme(Request $request)
{
    $response = Http::get('https://api.nasa.gov/DONKI/CME', [
        'api_key' => $this->nasaApiKey,
        'startDate' => $request->input('start_date', date('Y-m-d')),
        'endDate' => $request->input('end_date', date('Y-m-d', strtotime('+7 days'))),
    ]);

    if ($response->successful()) {
        return response()->json($response->json());
    }

    return response()->json(['error' => 'Unable to fetch CME data'], 500);
}


public function getHelioviewerData(Request $request)
{
    $response = Http::get('https://api.nasa.gov/Helioviewer', [
        'api_key' => $this->nasaApiKey,
        'date' => $request->input('date', date('Y-m-d')),
        'resolution' => $request->input('resolution', 1024),
    ]);

    if ($response->successful()) {
        return response()->json($response->json());
    }

    return response()->json(['error' => 'Unable to fetch Helioviewer data'], 500);
}
```

```
    public function searchImages(Request $request)
    {
        $response = Http::get('https://images-api.nasa.gov/search', [
            'q' => $request->input('query', ''),            // Пошуковий запит
            'media_type' => $request->input('media_type', 'image'), // Тип медіа (image, video, audio)
            'year_start' => $request->input('year_start'), // Початковий рік
            'year_end' => $request->input('year_end'),     // Кінцевий рік
        ]);

        if ($response->successful()) {
            return response()->json($response->json());
        }

        return response()->json(['error' => 'Unable to fetch images'], 500);
    }

    public function getNearEarthObjects(Request $request)
    {
        $response = Http::get('https://api.nasa.gov/neo/rest/v1/feed', [
            'api_key' => $this->nasaApiKey,
            'start_date' => $request->input('start_date', date('Y-m-d')),
            'end_date' => $request->input('end_date', date('Y-m-d', strtotime('+7 days'))),
        ]);

        if ($response->successful()) {
            return response()->json($response->json());
        }

        return response()->json(['error' => 'Unable to fetch NEO data'], 500);
    }
}
```

ДОДАТОК Г

Тестування NasaService

```
import { TestBed } from '@angular/core/testing';
import { HttpClientTestingModule, HttpTestingController } from '@angular/common/http/testing';
import { NasaService } from './nasa.service';

describe('NasaService', () => {
  let service: NasaService;
  let httpMock: HttpTestingController;

  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [HttpClientTestingModule],
      providers: [NasaService],
    });
    service = TestBed.inject(NasaService);
    httpMock = TestBed.inject(HttpTestingController);
  });

  afterEach(() => {
    httpMock.verify();
  });

  it('should be created', () => {
    expect(service).toBeTruthy();
  });

  it('should fetch images based on query and media type', () => {
    const mockResponse = {
      collection: { items: [{ links: [{ href: 'image_url' }], data: [{ title: 'Test Image' }] }] },
    };

    service.searchImages('moon', 'image').subscribe((data) => {
      expect(data.collection.items.length).toBe(1);
      expect(data.collection.items[0].links[0].href).toBe('image_url');
    });

    const req = httpMock.expectOne((request) =>
```

```
      request.url.includes('/search') && request.params.get('q') === 'moon' && request.params.get('media_type') ===
'image'
    );
    expect(req.request.method).toBe('GET');
    req.flush(mockResponse);
  });

  it('should handle errors gracefully', () => {
    service.searchImages('moon', 'image').subscribe(
      () => fail('Should have failed with the network error'),
      (error) => {
        expect(error.status).toBe(500);
      }
    );

    const req = httpMock.expectOne((request) =>
      request.url.includes('/search') && request.params.get('q') === 'moon' && request.params.get('media_type') ===
'image'
    );
    req.flush('Error fetching data', { status: 500, statusText: 'Internal Server Error' });
  });
});
```