Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ АНАЛІЗУ РУХУ ТА ВІЗУАЛІЗАЦІЇ МЕТЕОРНИХ ДАНИХ
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-21-2
Кисіль Л. О.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник асист. Кириченко І. Ю.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Кисілю Леву Олексійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка вебзастосунку для аналізу руху та візуалізації метеорних даних

затверджена наказом університету від 19 травня 2025 року № 381Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 2 червня 2025 р.
3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, програмні бібліотеки Fiber, HTMX, ESBuild, програмний інструмент Deno, інструменти мови Go, редактор коду Visual Studio Code та плагіни до нього, еталонні таблиці даних, програмні інструменти візуалізації, браузер Chrome, офіційна документація інструментів.

4. Перелік питань, що потрібно опрацювати в роботі
1. Огляд нових вебтехнологій і вебзастосунків з подальшим порівнянням.
2. Створення математичної моделі алгоритму та огляд його призначення.
3. Планування архітектури вебзастосунку та взаємодії алгоритму з користувачем.
4. Реалізація, огляд та тестування отриманого вебзастосунку.
5. Створення базових інструкцій запуску власної копії застосунку, який не потребує підключення до мережі.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) тестові зображення, зображення роботи застосунку, зображення роботи інструментів розробки.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	07.04.2025	
2	Аналіз завдання, підбір літератури	08.04.25-10.04.25	
3	Аналіз літератури з досліджуваної проблеми	11.04.25-14.04.25	
4	Аналіз технічних засобів	15.04.25-20.04.25	
5	Розробка методу	21.04.25-27.04.25	
6	Програмна реалізація	28.04.25-11.05.25	
7	Оформлення пояснювальної записки	12.05.25-20.05.25	
8	Перевірка на нормоконтроль	21.05.25-01.06.25	
9	Перевірка на плагіат	21.05.25-01.06.25	
10	Рецензування	21.05.25-01.06.25	
11	Підготовка презентації та доповіді	21.05.25-18.06.25	
12	Занесення роботи в електронний архів	02.06.25-18.06.25	
13	Попередній захист кваліфікаційної роботи	02.06.25-18.06.25	

Дата видачі завдання 7 квітня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ _____ асист. Кириченко І. Ю.___
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 75 с., 42 рис., 1 дод., 30 джерел.

ВЕБЗАСТОСУНОК, МЕТЕОРОЇДИ, DENO, GOLANG, JAVASCRIPT, НОВІ ТЕХНОЛОГІЇ, ВЕБТЕХНОЛОГІЇ, АЛГОРИТМ, АНАЛІТИКА ДАНИХ, РОЗРАХУНОК ПАРАМЕТРІВ.

Об’єктом роботи є дані руху метеороїдів у космічному просторі.

Метою роботи є розробка вебзастосунку, який здатен розраховувати та аналізувати параметри руху метеороїдів в Земній атмосфері та в Сонячній системі для використання з метеорними наземними радіолокаційними системами, які реєструють метеори імпульсно-дифракційним методом.

Був проведений аналіз існуючих вебтехнологій та існуючих вебзастосунків в астрономії. Створено інструкції користування розробленим застосунком та його розгортання, і розроблено алгоритм вторинної обробки та можливості побудови графічних представлень.

У результаті роботи здійснена програмна реалізація серверної та клієнтської частини вебзастосунку з авторською некомерційною назвою SpaceSoup для розрахунку параметрів орбіт метеороїдів з опцією візуалізації з використанням нових технологій за спеціальним алгоритмом.

WEB APPLICATION, METEOROIDS, DENO, GOLANG, JAVASCRIPT, NEW TECHNOLOGIES, WEB TECHNOLOGIES, ALGORITHM, DATA ANALYTICS, PARAMETER CALCULATION.

The subject of this work is data on the motion of meteoroids in outer space.

The aim of the work is to develop a web application capable of calculating and analyzing the motion parameters of meteoroids in the Earth's atmosphere and in the Solar System for use with ground-based meteor radar systems that register meteors using the pulse-diffraction method.

An analysis of existing web technologies and existing web applications in astronomy was conducted. User and deployment instructions for the developed application were created, as well as an algorithm for secondary data processing and capabilities for building graphical representations.

As a result of the work, the server-side and client-side software implementation of the web application was completed under the original non-commercial name SpaceSoup, designed for calculating meteoroid orbital parameters with visualization options, using new technologies and a specialized algorithm.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

IT – інформаційні технології

ХНУРЕ – Харківський національний університет радіоелектроніки

HTML – Hypertext Markup Language (мова розмітки гіпертексту)

CSS – Cascading Style Sheets (каскадні таблиці стилів)

LLVM – Low Level Virtual Machine (віртуальна машина низького рівня)

TOPCAT – Tool for Operations on Catalogues And Tables (інструмент для операцій над каталогами і таблицями)

JDK – Java Developer Kit (набір розробника Java)

МАРС – Метеорна автоматизована радіолокаційна система

SKiYMET – All-sky interferometric meteor radar (Всенебесний інтерферометричний метеорний радар)

MAARSY – the Middle Atmosphere ALOMAR Radar System (Радарна система середньої атмосфери ALOMAR)

JSON – JavaScript Object Notation (запис об’єктів JavaScript)

URL – Uniform Resource Locator (уніфікований локатор ресурсів)

VSC – Visual Studio Code

HTTP – Hypertext Transfer Protocol (протокол передачі гіпертекстових документів)

JSR – JavaScript Registry (реєстр JavaScript)

NPM – Node.js Package Manager (пакетний менеджер Node.js)

SHA-256 – Secure Hash Algorithm 256 (безпечний алгоритм хешування)

ВСТУП

На даний момент існує велика кількість технологій веброзробки у зв’язку зі збільшенням використання інтернет-послуг. Велика кількість сервісів, які є популярними, як правило є онлайн-сервісами. Це спонукає більшість компаній, які працюють у сфері IT, надавати свої сервіси, програми та інструменти у вигляді вебзастосунків.

Велика залученість у веброзробку, спонукає технології розвиватися, що створює умови для нових мов програмування, технологій та інструментів. Ці технології можуть досі не застосовуватись там, де це необхідно, або бути проігнорованими через велику конкуренцію.

Вебзастосунок – це програмний інструмент, який завжди має графічний інтерфейс, і завжди доступний у веббраузері. Вебзастосунок не є вебсайтом, проте завжди поставляється разом з ним. Вони популярні через їх доступність на будь-яких операційних системах та архітектурах, що є причиною популярності таких систем у комерції та науці. Вебзастосунки часто використовуються в астрономії, оскільки це надає доступ усім хто має веббраузер, що сприяє розвитку науки та її популяризації.

Актуальність роботи полягає у залученні нових технологій в сферу астрономічних вебзастосунків, для демонстрації переваг та недоліків таких вебтехнологій в науці. Крім цього, робота представляє астрономічний вебзастосунок, який на сьогоднішній момент не має аналогів в Україні та орієнтований на використання в метеорних радіолокаційних дослідженнях ХНУРЕ.

1 АНАЛІЗ СТАНУ ВЕБЗАСТОСУНКІВ В АСТРОНОМІЇ

1.1 Аналіз сучасних технологій

1.1.1 Популярні вебтехнології

Сьогодні існує велика кількість вебтехнологій, а мови програмування мають усі інструменти для створення вебзастосунків. Підходи, які розроблені через особливості мов програмування, можуть відрізнятися, а тому потрібно розуміти різницю між ними [1]. Знати різницю між інструментами є важливим, оскільки це може вплинути на продуктивність при розробці, та можливість підтримки у майбутньому. Вибір технологій впливає на масштабованість, безпеку та інтеграцію з іншими системами.

Для того, щоб проблем не виникло при розробці власного вебзастосунку, потрібно розглядати популярні технології, тобто ті, яким надають перевагу [2]. Це пов’язано з тим, що використання популярних технологій гарантує більшу кількість спеціалістів, які потенційно зможуть зрозуміти та підтримати створюваний застосунок. Крім того, прості інструменти є легшими для опанування дослідниками та ентузіастами, що відкриває більше можливостей для спільноти, що буде зацікавлена у ньому. Для пошуку таких технологій буде використано дані опитувань від кількох провайдерів: Stack Overflow Developer Survey [3], State of Developer Ecosystem (JetBrains) [4], GitHub Octoverse [5]. Ці опитування є досить авторитетними та мають велику кількість респондентів з різних сфер. Їх результати показують відношення людей з різним досвідом і сферою праці, представляючи вагомість конкретних технологій серед інших.

Згідно Stack Overflow Developer Survey, найпопулярнішими технологіями наприкінці 2024-го року були JavaScript, HTML/CSS, Python, TypeScript, Bash/Shell, Java, C#, C++, C, PHP, PowerShell, Go, Rust, Kotlin, Lua, Dart, Assembly, Ruby, Swift, R, Visual Basic (рис. 1.1).

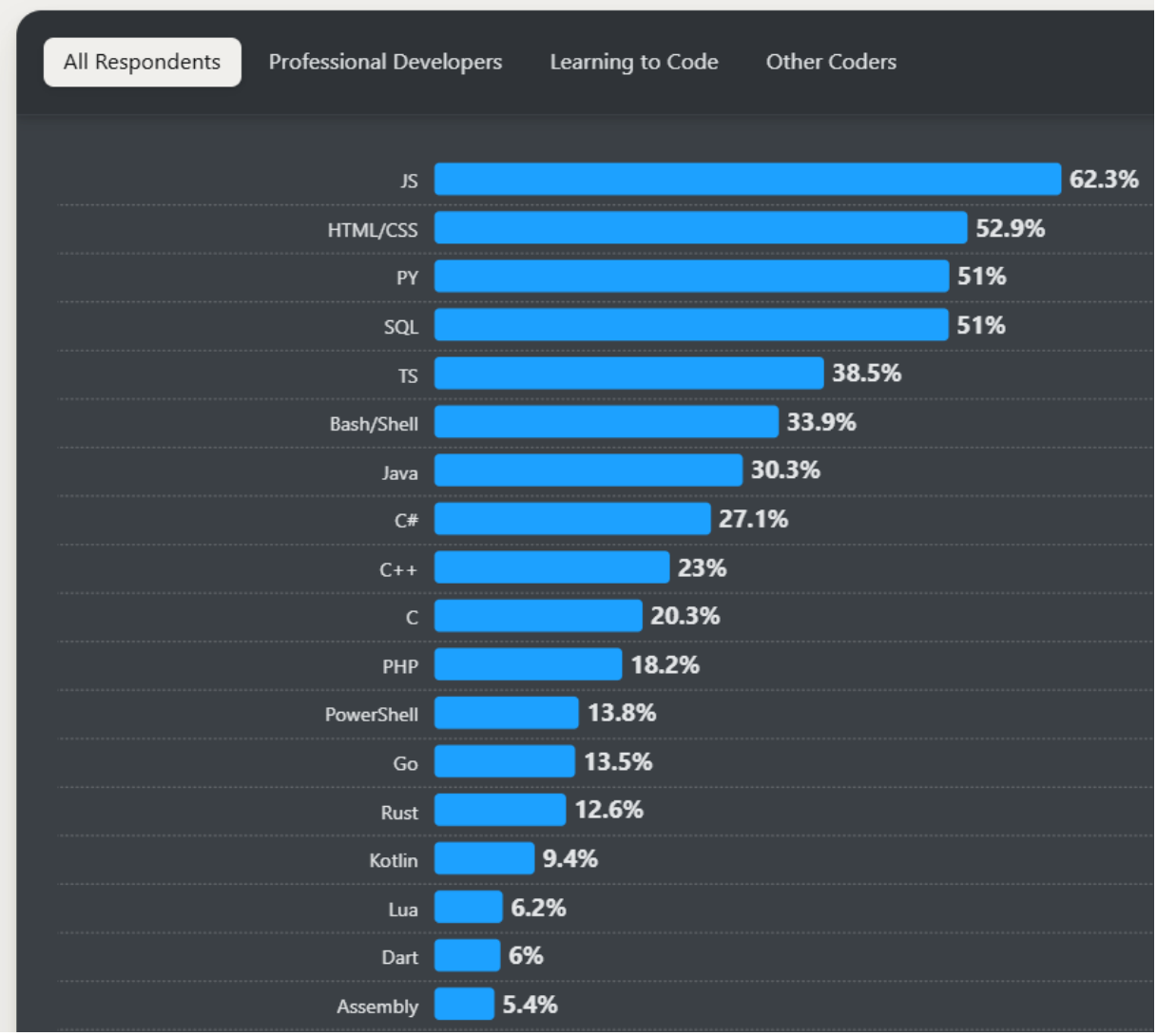

Рисунок 1.1 – Зображення відсотків використання різних технологій, для опитування Stack Overflow Developer Survey

Згідно інформації, яку надає State of Developer Ecosystem (JetBrains), у 2024-му році були популярними такі технології: JavaScript, Python, HTML/CSS, Java, TypeScript, Shell, C++, C#, C, Go, PHP, Kotlin, Rust, Dart, Swift, Lua, Ruby (рис. 1.2). Разом з цим опитуванням також надається JetBrains Language Promise Index (рис. 1.3). Як пояснюють автори дослідження, «цей індекс базується на поєднанні зростання аудиторії за останні п’ять років, стабільності цього зростання, частки людей, які висловлюють намір прийняти мову, і частки людей, які хочуть прийняти іншу мову». Згідно цього індексу, TypeScript, Rust та Python є лідерами цього індексу. На четвертому місці стоїть Go, потім Lua, C++, Kotlin, Shell, Dart, C, C# та Swift.

2017	2018	2019	2020	2021	2022	2023	2024	
65%	64%	69%	70%	69%	65%	61%	61%	JavaScript
32%	41%	49%	55%	52%	53%	54%	57%	Python
60%	55%	61%	61%	60%	54%	52%	51%	HTML / CSS
42%	47%	56%	56%	54%	49%	52%	48%	SQL
47%	51%	50%	54%	49%	48%	49%	46%	Java
12%	17%	25%	28%	29%	34%	34%	37%	TypeScript
–	29%	40%	39%	37%	34%	34%	36%	Shell
17%	18%	20%	27%	23%	25%	25%	25%	C++
20%	22%	24%	22%	21%	23%	21%	22%	C#
15%	16%	17%	23%	19%	20%	19%	18%	C
8%	12%	18%	19%	17%	19%	17%	18%	Go
30%	26%	29%	27%	32%	20%	18%	17%	PHP
2%	9%	16%	17%	14%	16%	15%	14%	Kotlin
–	2%	5%	7%	6%	9%	10%	11%	Rust
–	–	6%	9%	8%	9%	7%	8%	Dart
9%	8%	11%	9%	7%	7%	6%	6%	Swift
2%	3%	4%	3%	3%	3%	4%	5%	Lua
10%	8%	11%	8%	6%	5%	4%	4%	Ruby
7%	5%	6%	5%	3%	3%	3%	3%	Scala
7%	5%	6%	4%	3%	3%	2%	2%	Objective-C

Рисунок 1.2 – Результати опитування State of Developer Ecosystem

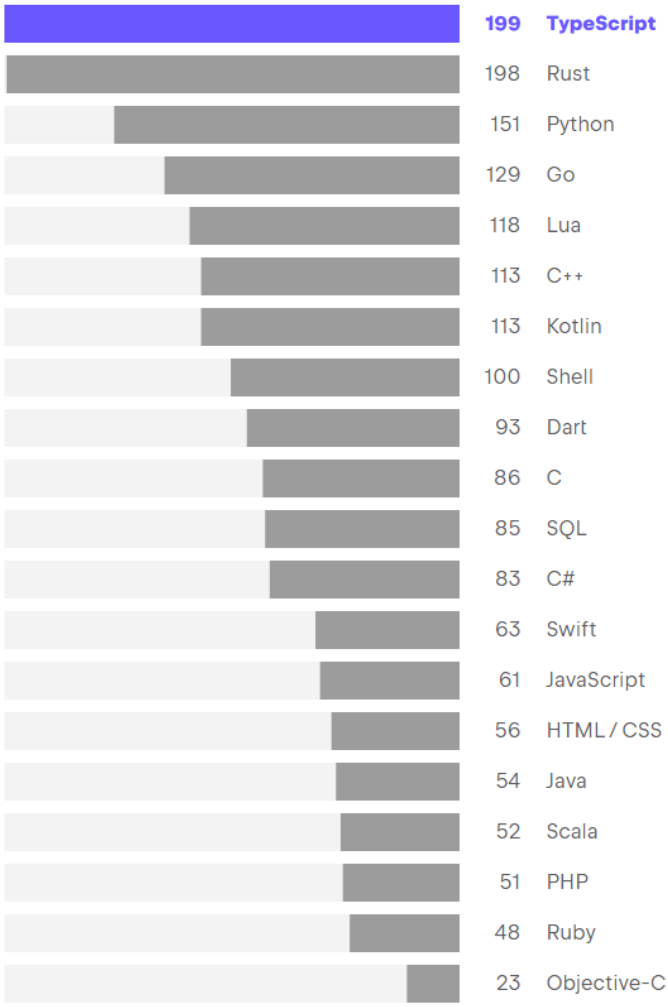

Рисунок 1.3 – Технології згідно індексу JetBrains Language Promise Index

Згідно зі статистикою, яку надає GitHub Octoverse, популярними мовами програмування на 2024-й рік є Python, JavaScript, TypeScript, Java, C#, C++, PHP, Shell, C та Go (рис. 1.4).

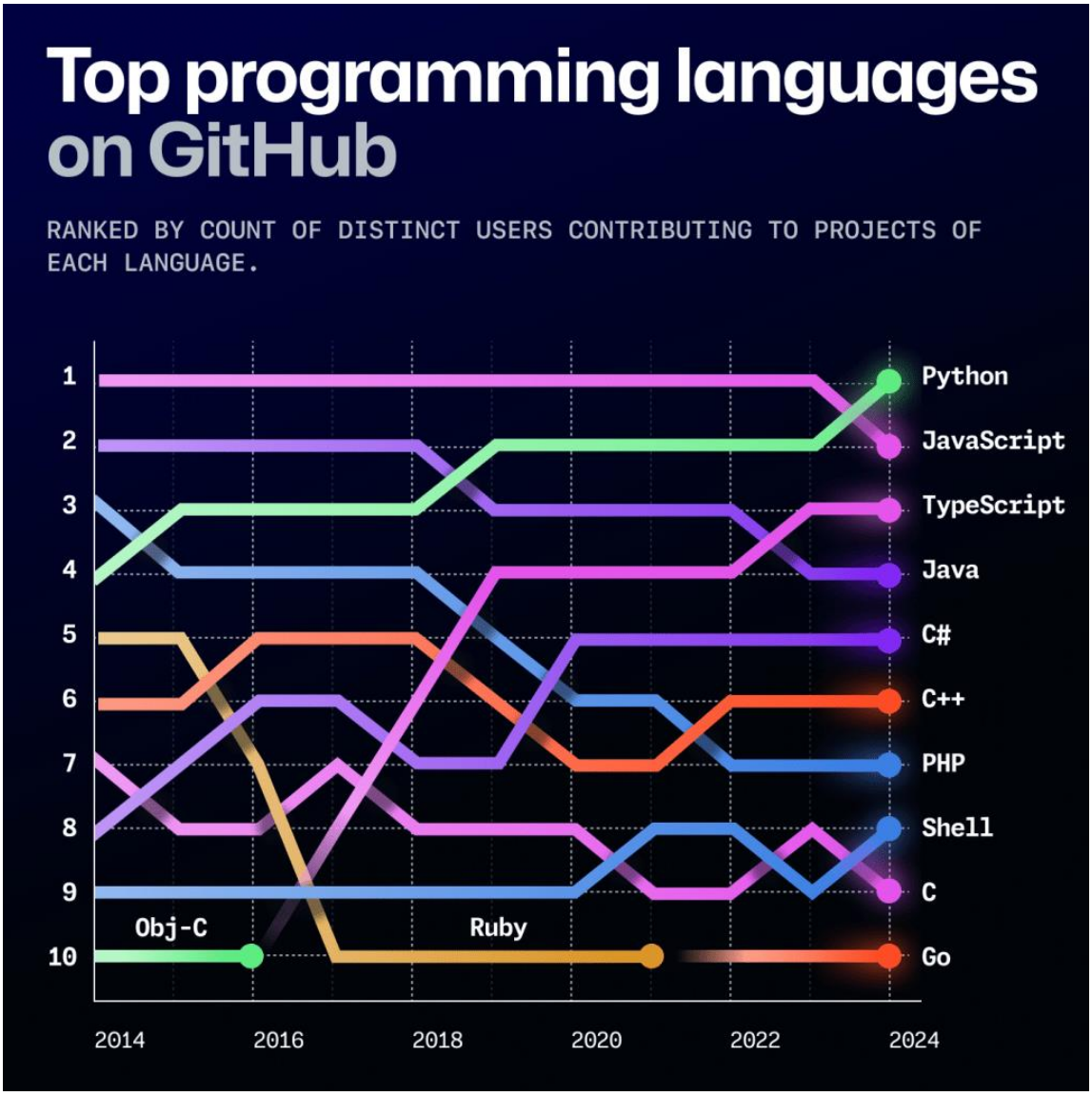

Рисунок 1.4 – Графік зміни популярності мов програмування GitHub Octoverse

У зв’язку з отриманою статистикою від кількох джерел, можна зробити висновок про те, що найкращими варіантами для розробки серверної частини вебзастосунку будуть технології, пов’язані з наступними мовами: Python [6], TypeScript, JavaScript, Java, C#, Go, Rust, Swift та PHP. Проте варто пам’ятати, що крім популярності, треба розглядати інші аспекти: зрозумілість мови для

дослідників, швидкість розробки, швидкість отриманої програми при використанні цієї мови, очікуваний час популярності мови та наявність потрібних інструментів для розробки вебзастосунків.

Усі проаналізовані вище мови мають свої плюси та мінуси, проте найкращим кандидатом для серверної частини є Go [7]. Ця мова має зрозумілий синтаксис, бо не має складних конструкцій та абстракцій; має стабільний шлях розвитку і великий обсяг навчальних матеріалів [8, 9]; має пакетний менеджер, завдяки якому можна легко використовувати сторонні бібліотеки, що прискорює розробку; швидко компілює програму, а отримана програма працює без затримок. Крім цього мова Go лише рухається до свого піку популярності і робить це повільно. Такий факт може свідчити про стійкість цієї мови та її здатність до тривалого існування порівняно з іншими.

Для клієнтської частини вибір лежить між наступними мовами: TypeScript, JavaScript, HTML/CSS. Мова TypeScript [10] є суперником JavaScript, і надає можливості типізації, що робить TypeScript потужним інструментом, особливо при створенні доступних бібліотек. Вибір HTML/CSS є завжди очевидним, бо ці технології були створені першими і залишилися мало не єдиними. HTML та CSS надають можливості створення каркасу та зовнішнього вигляду будь-якого вебсайту [11], включаючи вебзастосунки. Варто сказати, що більшість сучасних мов, які використовують LLVM [12], можуть бути компільовані у WebAssembly, тому для клієнтської частини можна використовувати інші мови: Go, Rust, Python. Інструменти розробки для мови Go недостатньо розвинені для застосування WebAssembly замість TypeScsript.

Зазвичай, коли мова йде про керування проєктом, використовується поняття скрипт. Скрипт, у контексті керування проєктом – це програма невеликого розміру, яка дозволяє просто запустити рутинне завдання у середовищі розробки, частіше в папці проєкту. При розробці серверу може виникнути потреба створити скрипт запуску серверу, збірки клієнтської частини, генерації документації, підтягування перекладу тексту та будь-яких інших завдань. Одним з прикладів може бути скрипт, що написаний мовою PowerShell для вебсайту VedAstro (рис. 1.5).

Рисунок 1.5 – Скрипт для запуску .NET вебсайту, створеного та опублікованого проєктом VedAstro

Крім PowerShell та Bash/Shell можуть бути використані і звичайні мови програмування, але надається перевага інтерпретованим мовам, оскільки вони не потребують компіляції. Зазвичай використовується мова Python, але у вебзастосунках також можна зустріти скрипти, написані мовою JavaScript і TypeScript.

1.1.2 Еволюція технологій

Цікава тенденція, яку можна спостерігати під час розвитку технологій – їх залежність від старих рішень. Це пов’язано з тим, що старі технології перевірені часом, а творці нових технологій добре знайомі зі старими, і знають усі недоліки. Через те що програми часто простіше переписати заново, ніж виправити, ми маємо велику кількість схожих програмних продуктів у IT.

Для демонстрації можна взяти вже згадану раніше мову Go. Ця мова була створена, оскільки на той час не існувало ні однієї мови, яка б одночасно надавала швидку компіляцію, швидкий запуск і простоту. Під простотою мається на увазі відсутність непотрібних функцій та синтаксис, який буде зрозумілий спеціалісту або досліднику без пояснень.

Середовище виконання Deno є іншим кращим прикладом еволюції веброзробки. Оскільки Node.js – інше середовище виконання з великою кількістю коду, який важко підтримувати, було створено Deno – покращену альтернативу від самого розробника Node.js, що пропонує додаткові можливості [13, 14]: вбудована підтримка TypeScript, безпека при використанні сторонніх бібліотек, готові інструменти для компіляції, форматування та тестування коду, повна підтримка вебстандарту, швидкість запуску.

Tailwind CSS [15] є альтернативою CSS. Ця вебтехнологія прискорює розробку стилів для вебсторінок завдяки використанню класів та наслідування, дозволяючи скоротити велику частину коду стилів, що прискорює навігацію і робить розробку швидшою. Tailwind CSS був створений у зв’язку з перевагою використання малих допоміжних класів у вигляді готового рішення.

У зв’язку з розвитком рішень у сфері веброзробки, почали набирати популярність так звані бандлери – інструменти, що можуть конвертувати Tailwind CSS у CSS, Deno TypeScript у JavaScript, або скорочувати код. Одними з популярних бандлерів є Webpack, проте він є чи не найповільнішим. У зв’язку з цим було створено ESBuild, який є одним з найшвидших і найновіших бандлерів, який досі знаходиться на стадії активної розробки, проте вже має більшість необхідних функцій.

Розвиток вебтехнологій не завжди рухався у бік продуктивності. У зв’язку з тим, що розробка стандарту мови HTML була недостатньою, якщо не відсутньою, спостерігався розвиток JavaScript бібліотек. Велика кількість залежностей та файлів налаштувань для популярних React.js проєктів при повільній реалізації Webpack призвела до широкого розголосу бібліотеки HTMX, яка націлена на розширення функціоналу HTML і можливостей гіпертексту, тому використовуватиметься при розробці SpaceSoup. Вона не є досконалою, а тому можна очікувати на створення кращих рішень у майбутньому.

1.2 Огляд існуючих рішень

Вебзастосунки є досить поширеними в астрономії через їхню доступність і гнучкість [16]. Вони не займають місця на комп’ютері, але для їх роботи потрібне з’єднання з мережею, що можна вважати як недоліком, так і великою перевагою. Одним з прикладів астрономічних вебзастосунків є Aladin Lite [17] (рис. 1.6) від проєкту Aladin, який надає інтерактивну мапу небесної сфери.

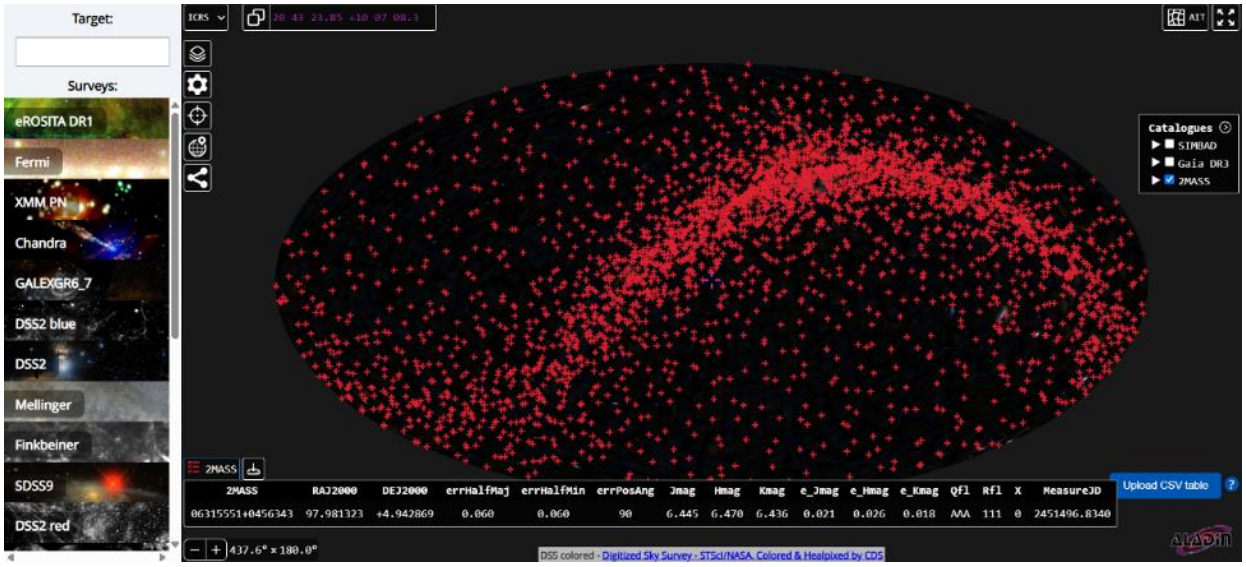

Рисунок 1.6 – Зображення роботи інтерактивного вебзастосунку Aladin Lite у браузері Chrome 135 на момент 18 квітня 2025-го року

Цей застосунок надає можливості доступу користувачів до баз даних об’єктів через взаємодію з картою. Ця карта представляє собою набір окремих зображень небесної сфери, які поєднані у єдине зображення. Таке зображення має вигляд геоїда, що явно демонструє наявність складних механізмів відображення в поєднанні з інтерактивністю. Карта взаємодіє з мишею та клавіатурою, дозволяючи керувати масштабом, широтою та довготою точки спостереження. Aladin Lite є популярним інструментом і наводиться як головний приклад відмінності вебзастосунків і звичайних статичних вебсайтів.

Наступний приклад – Horizons System [18] (рис. 1.7) – демонструє простіший приклад застосування вебтехнологій в астрономії та включає інтерактивні елементи.

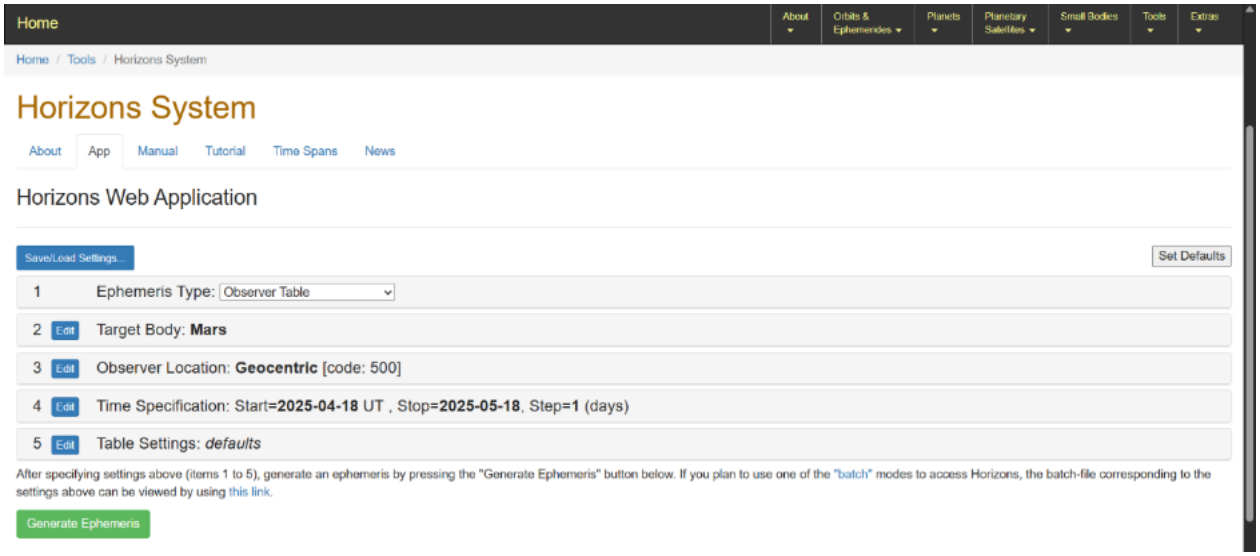

Рисунок 1.7 – Зображення головної сторінки інструменту Horizons System

Horizons System не надає інтерактивної карти або графіків руху тіл. Натомість цей інструмент має інтерфейс, який дозволяє вводити власні дані або використовувати вже наявні. Застосунок надає готові дані для планет Сонячної системи, комет, метеорів та зірок поза системою.

У результатах розрахунку можна побачити параметри руху обраного тіла на інтервалах часу згідно обраних налаштувань. Вигляд результату розрахунків (рис. 1.8) можна також налаштовувати: надається можливість зміни систем координат та центрів спостереження.

Цей вебзастосунок надає можливості переходу між системами координат та часовими проміжками замість потужних інтерактивних елементів, тому все ще може називатись вебзастосунком. Без сумнівів, Horizons System є потужним інструментом для астрономічних розрахунків, який, попри відсутність інтерактивних карт чи графіків, пропонує глибокий функціонал для аналізу руху космічних об’єктів.

Рисунок 1.8 – Часткове зображення результатів розрахунку Horizons System

Загалом існує велика кількість інших вебзастосунків, які надають можливості для перегляду унікальних даних або даних, які належать іншим відкритим джерелам [19] з додатковим аналізом. Проте, крім вебзастосунків, є також чимала кількість настільних, які можна використовувати без підключення до інтернету. Головна перевага таких програм – відсутність потреби в постійному доступі до інтернету. Їх достатньо завантажити або перенести з іншого пристрою один раз. Водночас вони займають місце на носії, що є їх недоліком порівняно з вебзастосунками.

Програма TOPCAT [20] (рис. 1.9) є прикладом програм в області астрономії, які не потребують підключення до мережі для своєї роботи. Це звичайний застосунок, який потрібно завантажувати та запускати на власному пристрої, для роботи якого може бути потрібен JDK. JDK теж є програмним інструментом, здатним запускати будь-який Java-архів на різних операційних системах та архітектурах. Наприклад, саме у такому форматі поширюється програма TOPCAT. Вона надає великі можливості візуалізації та обробки даних, наприклад, перегляду великих таблиць, візуалізації даних у вигляді діаграм, графіків, двовимірних та тривимірних зображень з кольоровими шкалами. На відміну від вебзастосунків, TOPCAT може використовуватись без доступу до мережі, але займає місце на комп’ютері, як JDK та дані, з якими ведеться робота.

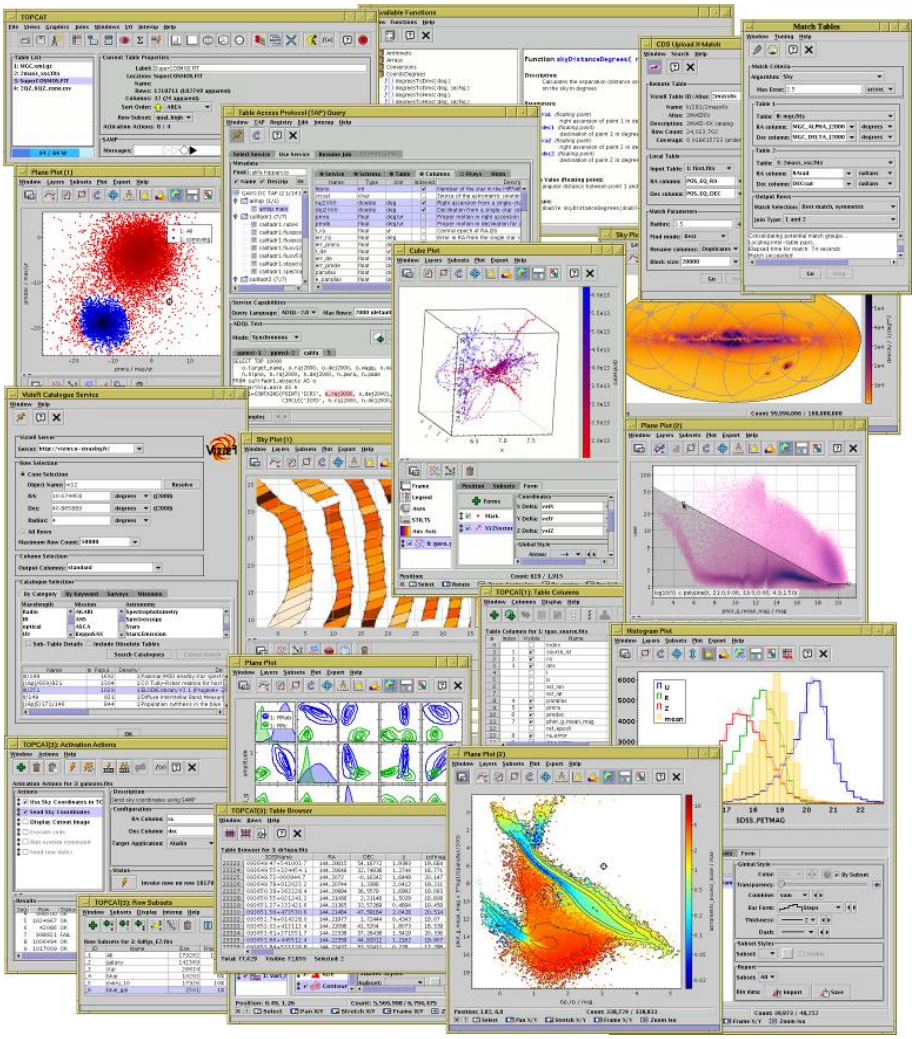

Рисунок 1.9 – Зображення роботи TOPCAT з офіційного сайту програми

1.3 Постановка задачі

У зв’язку з постійним і домінуючим розвитком вебтехнологій, а також відсутністю існуючих програмних алгоритмічних рішень у сфері астрономічних вебзастосунків в Україні та необхідність такого застосунку в метеорних радіолокаційних дослідженнях ХНУРЕ, розробка нового астрономічного вебзастосунку є актуальним завданням.

Об’єктом роботи є дані руху метеороїдів у космічному просторі.

Метою роботи є розробка вебзастосунку, який здатен розраховувати та аналізувати параметри руху метеороїдів в Земній атмосфері та в Сонячній системі для використання з метеорними наземними радіолокаційними системами, які реєструють метеори імпульсно-дифракційним методом.

Для досягнення мети необхідно реалізувати наступні завдання:

– створити математичну модель алгоритму розрахунку параметрів;

– ознайомитися з існуючими рішеннями у сфері астрономічних вебзастосунків;

– обрати інструменти, які мають потенціал для можливості підтримки програми у майбутньому;

– налаштувати інтеграцію інструментів;

– розробити програмну модель алгоритму;

– розробити серверну та клієнтську частину астрономічного застосунку;

– розробити можливості для візуалізації даних;

– протестувати роботу програми.

2 МОДЕЛЮВАННЯ ВЕБЗАСТОСУНКУ SPACESOUP

2.1 Призначення алгоритму

Алгоритми первинної та вторинної обробки даних відіграють ключову роль у метеорних і космічних дослідженнях, зокрема в радіолокаційних метеорних спостереженнях, які є основою для вивчення метеорних явищ, атмосферних процесів і космічних об’єктів. Ці алгоритми забезпечують перетворення необроблених даних, отриманих із радіолокаційних систем, у структуровану інформацію, яка може бути використана для геофізичних, астрономічних і прикладних цілей. У контексті астрономії доцільність таких алгоритмів обґрунтовується їхньою здатністю підвищувати точність результатів, ефективність і наукову цінність досліджень [21, 22].

Первинна обробка даних є першим етапом, на якому радіолокаційні сигнали, отримані від систем на кшталт харківської МАРС чи сучасних SKiYMET, очищаються від шумів і перешкод, щоб виділити релевантну інформацію про метеорні сліди. Цей процес передбачає виявлення короткочасних метеорних сигналів, які мають слабку амплітуду, і відокремлення їх від техногенних чи природних перешкод.

Вторинна обробка даних є більш складним етапом, який спрямований на глибший аналіз і синтез інформації для отримання науково значущих результатів. На цьому етапі за відомими формулами розраховуються геліоцентричні швидкості та орбітальні параметри метеороїдів, зокрема ексцентриситет, нахил орбіти, велика піввісь, аргумент перигелію, довгота висхідного вузла тощо. Вторинна обробка також дозволяє ідентифікувати метеорні потоки, визначати їх джерела, такі як комети чи астероїди, і досліджувати міжпланетний пиловий диск. Вказане входить в задачі, що вирішує метеорна астрономія по встановленню та вивченню розподілу метеорної речовини в Сонячній системі та її припливу в атмосферу Землі.

Одним з методів такого вивчення є побудова та вивчення розподілу по небесній сфері метеорних радіантів в різних системах координат (геоцентричній та геліоцентричній).

Обробка інформації про метеори передбачає систематизацію даних у вигляді каталогів і баз даних, які стають доступними для міжнародного наукового співтовариства через віртуальні обсерваторії тощо. Важливим моментом є автоматизація процесу обробки даних у реальному часі. Автоматизація обробки метеорних радіолокаційних даних у реальному часі була започаткована в Україні в Харкові в ХНУРЕ на початку 1970-х років на метеорній системі МАРС в епоху перших електронно-обчислювальних машин і широко впроваджена в сучасних метеорних системах із використанням інтернет-технологій.

У XXI столітті алгоритми стають дедалі складнішими завдяки штучному інтелекту та машинному навчанню, що застосовуються, наприклад, у MAARSY [21] для автоматичного виявлення сигналів і класифікації потоків. Для України, з її унікальним науковим спадком, оновлення алгоритмів і модернізація систем на кшталт МАРС є важливим завданням для інтеграції в міжнародні мережі, як FRIPON, і збереження лідерства в метеорній науці.

2.2 Математична модель алгоритму

Алгоритм вторинної обробки швидкостей метеорів і параметрів метеорних орбіт є послідовністю кроків, спрямованих на розрахунок орбітального руху метеороїдів та їхніх характеристик після первинного збору даних. Первинними є наступні дані: дата спостереження, часові затримки зареєстрованих радіосигналів між головним та рознесеними пунктами прийому та визначені швидкості, що отримуються від приладів у кожному

пункті. Результатом розрахунків є геліоцентрична швидкість та параметри орбіти метеороїдів у Сонячній системі [23].

Для розрахунку азимуту радіанту A використовується формула головного значення $A_{\text{гл}}$:

$$\operatorname{tg} A_{\text{гл}} = \frac{\cos \varphi_1 - K \cdot \cos \varphi_2}{K \cdot \sin \varphi_2 - \sin \varphi_1}, \tag{2.1}$$

де $\varphi_1 = 34°10'16''$;

$\varphi_2 = 110°16'22''$ – азимути баз, що відраховуються від півночі до сходу;

K – коефіцієнт, необхідний для розрахунку головного значення азимуту.

Для розрахунку значення K використовується формула:

$$K = 1{,}94787 \frac{\tau_1}{\tau_2} + 10^{-5}, \tag{2.2}$$

де τ_1, τ_2 – часові затримки, отримані з приладів.

Значення азимуту A можна знайти за наступним алгоритмом дій:

Крок 1. Якщо $\tau_1 > 0$, $\tau_2 \geq 0$, $A_{\text{гл}} > 0$, тоді $A = A_{\text{гл}} + \pi$.

Крок 2. Якщо $\tau_1 > 0$, $\tau_2 \geq 0$, $A_{\text{гл}} < 0$, тоді $A = A_{\text{гл}} + 2\pi$.

Крок 3. Повинна виконуватись умова $200°16'22'' < A \leq 304°10'16''$.

Крок 4. Якщо $\tau_1 \leq 0$, $\tau_2 < 0$, $A_{\text{гл}} > 0$, тоді $A = A_{\text{гл}}$.

Крок 5. Якщо $\tau_1 \leq 0$, $\tau_2 < 0$, $A_{\text{гл}} < 0$, тоді $A = A_{\text{гл}} + \pi$.

Крок 6. Повинна виконуватись умова $20°16'22'' < A \leq 124°10'16''$.

Крок 7. Якщо $\tau_1 < 0$, $\tau_2 \geq 0$, $0 < A_{\text{гл}} \leq 20°16'22''$, тоді $A = A_{\text{гл}}$.

Крок 8. Якщо $\tau_1 < 0$, $\tau_2 \geq 0$, $A_{\text{гл}} < 0$, тоді $A = A_{\text{гл}} + \pi$.

Крок 9. Повинна виконуватись умова $0 < A_{\text{гл}} \leq 20°16'22''$.

Крок 10. Якщо $\tau_1 > 0$, $\tau_2 \leq 0$, тоді $A = A_{\text{гл}} + \pi$.

Крок 11. Повинна виконуватись умова $124°10'16'' < A \leq 200°16'22''$.

Для розрахунку зенітного кута радіанту $z_{\text{ср}}$ використовується наступна формула, яка враховує ваги для двох власних головних значень:

$$z_{\text{ср}} = \frac{W_1 \cdot \sin z_1 + W_2 \cdot \sin z_2}{W_1 + W_2}, \tag{2.3}$$

де W_1, W_2 – коефіцієнти ваги;

z_1, z_2 – є основними значеннями для розрахунку зенітного куту радіанту, що повинні бути більше нуля. Для них дано наступні формули:

$$W_1 = \cos(A - \varphi_1), \qquad W_2 = \cos(A - \varphi_2), \tag{2.4}$$

$$\sin z_1 = \frac{0{,}9252 \cdot 10^{-3} \cdot v_{\text{ср}} \cdot \tau_1}{\cos A - \varphi_1}, \qquad \sin z_2 = \frac{0{,}4749 \cdot 10^{-3} \cdot v_{\text{ср}} \cdot \tau_2}{\cos A - \varphi_2}, \tag{2.5}$$

де $v_{\text{ср}}$ – середнє значення швидкості, що є середнім значенням для швидкостей, отриманих з приладів.

Для отримання значення зенітного кута радіанту, виправленого за тяжіння використовуються наступні формули:

$$z_{\text{випр}} = z_{\text{ср}} + \Delta z, \tag{2.6}$$

$$\operatorname{tg}\frac{\Delta z}{2} = \left|\frac{v' - v_0}{v' + v_0}\right| \cdot \operatorname{tg}\frac{z_{\text{ср}}}{2}, \tag{2.7}$$

$$v_0 = 1{,}0398 \cdot v_{\text{ср}} + 0{,}65, \tag{2.8}$$

$$v' = \sqrt{v_0^2 - 123{,}2}. \tag{2.9}$$

Для розрахунку схиляння радіанту використовується наступна формула, за умови $-\frac{\pi}{2} \leq \delta \leq \frac{\pi}{2}$, а широта місця спостереження $\varphi = 49°24'50''$:

$$\sin\delta = \sin\varphi\cos z_{\text{випр}} - \cos\varphi \cdot \sin z_{\text{випр}} \cdot \cos A. \quad (2.10)$$

Годинний кут радіанта t, розраховується з урахуванням умови $0 \le t \le 2\pi$:

$$\sin t = \frac{\sin z_{\text{випр}} \cdot \sin A}{\cos\delta}, \quad (2.11)$$

$$\cos t = \frac{\cos\varphi \cdot \cos z_{\text{випр}} + \sin\varphi \cdot \sin z_{\text{випр}} \cdot \cos A}{t}. \quad (2.12)$$

Для отримання зіркового часу S використовується формула:

$$S = C_2 + 0{,}98565 \cdot d + 15{,}0411 \cdot h + 0{,}25068 \cdot m, \quad (2.13)$$

де C_2 – кутове значення корони Землі для конкретної дати;

d – кількість днів з початку року починаючи з 0;

h – години;

m – хвилини.

Пряме сходження радіанта α розраховується за умови $0 < \alpha < 2\pi$ за формулою:

$$\alpha = S - t. \quad (2.14)$$

Поправки за добову аберацію $\Delta\alpha$ та $\Delta\delta$:

$$\Delta\alpha = -\frac{26{,}948^\circ}{v'} \cdot \frac{\cos t}{\cos\delta} \cdot \cos\varphi, \quad (2.15)$$

$$\Delta\delta = \frac{26{,}948}{v'} \cdot \sin t \cdot \sin\delta \cdot \cos\varphi. \quad (2.16)$$

Виправлені екваторіальні координати радіанта $\delta_{\text{випр}}$ та $\alpha_{\text{випр}}$:

$$\alpha_{\text{випр}} = \alpha + \Delta\,\alpha, \qquad \delta_{\text{випр}} = \delta + \Delta\,\delta. \tag{2.17}$$

Елонгація ψ_{E} розраховується за формулою:

$$\cos\psi_{\text{E}} = -\sin t * \cos\delta_{\text{випр}}. \tag{2.18}$$

Геліоцентрична швидкість розраховується при $0 \leq \psi_{\text{E}} \leq \pi$:

$$v_g = \frac{v'\sin(\psi_E - \Delta S)}{\sin\psi_E}, \tag{2.19}$$

$$\Delta\,S = \sqrt{\left(\Delta\,\alpha \cdot \cos\delta_{\text{випр}}\right)^2 + (\Delta\,\delta)^2}. \tag{2.20}$$

Позаатмосферна швидкість розраховується за формулою:

$$v_\infty = \sqrt{v_g^2 + 123{,}2}. \tag{2.21}$$

Широта радіанту β, при $-\frac{\pi}{2} \leq \beta \leq \frac{\pi}{2}$ і куті нахилу площини екліптики до площини екватора $\varepsilon = 23°26'40''$:

$$\sin\beta = -\sin\varepsilon \cdot \sin\alpha_{\text{випр}} \cdot \cos\delta_{\text{випр}} + \cos\varepsilon \cdot \sin\delta_{\text{випр}}. \tag{2.22}$$

Довгота радіанта λ при $0 < \lambda < 2\pi$:

$$\cos\lambda = \frac{\cos\delta_{\text{випр}} \cdot \cos\alpha_{\text{випр}}}{\cos\beta}. \tag{2.23}$$

Її можна також розрахувати за формулою 2.24.

$$\sin\lambda = \frac{\cos\delta_{\text{випр}}\cdot\sin\alpha_{\text{випр}}\cdot\cos\varepsilon+\sin\delta_{\text{випр}}\cdot\sin\varepsilon}{\cos\beta}, \quad (2.24)$$

де ε – кут місця.

Довгота сонця $\lambda_{\odot}$ можу бути розрахована за наступною формулою:

$$\lambda_{\odot} = 0{,}98565°d + 1{,}973°\sin[0{,}98565(d-2)] - C_3. \quad (2.25)$$

Довгота радіанту відносно апекса $\lambda - \lambda_a$, де $\pi_o = 102°21'14''$, при умові $0 < \lambda - \lambda_a < 2\pi$:

$$\lambda_a = \lambda_{\odot} + \Delta\odot - 90°, \quad (2.26)$$

$$\Delta\odot = 0{,}958°\sin(\lambda_{\odot} - \pi_o). \quad (2.27)$$

Кут елонгації видимого радіанта від апекса E_α при $0 \le E_\alpha \le \pi$:

$$\cos E_\alpha = \cos\beta\cdot\cos(\lambda - \lambda_a). \quad (2.28)$$

Радіус – вектор орбіти Землі R – може бути розрахований за такою формулою за умови $e_0 = 0{,}01675$.

$$R = \frac{1-e_0^2}{1-e_0\cos(\lambda_\theta - \pi_0)}. \quad (2.29)$$

Орбітальна швидкість Землі v_t для цього дня:

$$v_t = 29{,}76\sqrt{\frac{2}{R} - 1}. \quad (2.30)$$

Орбітальна швидкість Землі для цього дня λ', при $0 \le \lambda' \le 2\pi$:

$$\lambda' = \lambda_\theta + \Delta\,\theta - (\lambda_\theta + \Delta\,\theta - \lambda'), \tag{2.31}$$

$$\operatorname{tg}\lambda_\theta + \Delta\,\theta - \lambda' = \frac{\sin(\lambda_\theta + \Delta\theta - \lambda) - \frac{v_t}{v_g \cos\beta}}{\cos(\lambda_\theta + \Delta\theta - \lambda)}. \tag{2.32}$$

Геліоцентрична швидкість v_h:

$$v_h = \sqrt{v_g^2 + v_t^2 - 2 v_g v_t \cos E_a}. \tag{2.33}$$

Широта істинного радіанту β', при $-\frac{\pi}{2} \le \beta' \le \frac{\pi}{2}$:

$$\beta' = \arcsin\left(\frac{v_g}{v_h'}\sin\beta\right). \tag{2.34}$$

Нахил орбіти частинки до площини екліптики i, при $0 \le i \le \pi$:

$$i = \operatorname{arctg}\left(-\frac{|\operatorname{tg}\beta'|}{\sin(\lambda_\theta - \lambda')}\right). \tag{2.35}$$

Велика піввісь a розраховується за формулою 2.36.

$$a = \frac{R}{2 - R\left(\frac{v_h}{29{,}76}\right)^2}. \tag{2.36}$$

Кут E, утворений радіус-вектором метеорного тіла з вектором його швидкості розраховується при умові $0 \le E \le \pi$:

$$E = \arccos[-\cos\beta' \cos(\lambda_\Theta - \lambda')]. \tag{2.37}$$

Тоді елонгація радіанта від Сонця E'_Θ, при $\pi \equiv 180°$:

$$E'_{\Theta} = \pi - E. \tag{2.38}$$

Інші елементи орбіти можна знайти за наступними формулами:

$$p = R^2 \left(\frac{V_n}{29{,}76}\right)^2 \sin^2 E'_{\Theta}, \quad \sin E'_{\Theta} = 1 - \cos^2\beta' \cos^2(\lambda' - \lambda_{\Theta}), \tag{2.39}$$

$$b = \sqrt{p|a|}, \qquad p = a(1 - e^2), \tag{2.40}$$

$$c = \sqrt{a^2 - b^2} \text{ при } a > 0, \quad c = \sqrt{a^2 + b^2} \text{ при } a < 0, \tag{2.41}$$

$$e^2 = 1 - \frac{p}{a}, \qquad a = \frac{r}{2-Q}, \tag{2.42}$$

$$q = \text{a} - c \text{ при } a > 0, \qquad q = c - |a| \text{ при } a < 0, \tag{2.43}$$

$$q' = \frac{r}{2-Q} - a(e - 1). \tag{2.44}$$

Довгота висхідного кута Ω, при умові $0 \leq \Omega \leq 2\pi$, розраховується за наступною формулою:

$$\Omega = \lambda_{\Theta} \text{ при } \beta' > 0, \qquad \Omega = \lambda_{\Theta} + \pi \text{ при } \beta' < 0. \tag{2.45}$$

Істинна аномалія ν за умови $0 \leq \nu \leq 2\pi$:

$$\cos \nu = \frac{p-R}{R \cdot e}. \tag{2.46}$$

Аргумент перигелію ω за умови $0 \leq \omega \leq 2\pi$:

$$\omega = \pi - nu. \tag{2.47}$$

У результаті розрахунків за наведеним вище алгоритмом можна отримати дані, що спираються на дві системи відліку. Геоцентрична система відліку є сферичною системою координат відносно Землі, а геліоцентрична система є системою координат відносно Сонця. Завдяки наявності обох систем координат, можна не лише встановити місцезнаходження координат метеорного радіанта метеороїда на небесній сфері, але й дослідити його рух у Сонячній системі, що є необхідним для вивчення взаємодії метеороїдів з іншими космічними тілами.

Знання положення відносно Сонця також є корисним, оскільки це може допомогти дізнатися походження метеороїдів. Крім того, використання поправок за аберацію, тяжіння та орбітальних рухів Землі забезпечує більшу точність отриманих результатів, що є стандартною процедурою, залежних від прийнятих констант в алгоритмі, затверджених Міжнародним астрономічним союзом та пов’язаними структурами щодо референтних систем координат. Це дасть змогу залучати індивідуальні дані дослідників після обробки за допомогою SpaceSoup до міжнародних баз даних метеорних орбіт. Проте варто звертати увагу на правильність реалізації алгоритму, на відповідність заявленої епохи його вибраних констант обчислень та бездоганність початкових даних для констатації високої точності результатів, через що необхідно реалізувати тестування застосованого алгоритму.

Тестування реалізації алгоритму має включати як перевірку окремих етапів обчислень, так і фінальних результатів з використанням незалежних джерел або еталонних розрахунків. Такий підхід дозволить переконатися в коректності реалізації і визначити межі застосування методу у разі проходження перевірок.

Окрім тестування алгоритму, важливим аспектом є його інтеграція з новітніми технологіями обробки даних. Наприклад, використання хмарних обчислень або штучного інтелекту для аналізу великих обсягів даних про метеорні орбіти може значно підвищити ефективність обробки та точність прогнозування траєкторій метеороїдів.

2.3 Структура застосунку

2.3.1 Вибір технологій

У зв’язку з проведеним аналізом популярних вебтехнологій стало очевидним, що мова Go є доцільною для розробки застосунку SpaceSoup. Однією з ключових причин цього вибору є висока продуктивність, яка необхідна для будь-якого вебсерверу. Крім того, ця мова часто використовується вебсервісами та хмарними сховищами, що каже про високий рівень стабільності, а стабільність дуже критична у ситуаціях підтримки багатьох користувачів. Стабільність є очевидним параметром, оскільки Go використовується багатьма всесвітньо відомими технічними компаніями: Microsoft, Google, Meta, Netflix, Twitch, Dropbox та іншими. Це робить мову Go бажаним інструментом не тільки у зв’язку з надійністю, але й простотою. Синтаксис мови Go не містить прихованої поведінки або абстракції, що робить мову не тільки функціональною, але й зрозумілою для дослідників. Завдяки цьому код вебзастосунку допоможе зробити SpaceSoup технічно доступним при необхідності його покращення.

Для роботи клієнтської частини буде використовуватись мова TypeScript, яка є дуже популярною на даний момент. У зв’язку з її використанням, вебзастосунок SpaceSoup отримує можливості, які будуть корисні для потенційних вкладників у проєкт. Однією з ключових переваг цієї мови є статична типізація, яка дозволяє виявляти помилки до виконання коду, завдяки чому можна уникнути багатьох непередбачуваних ситуацій, бо типізація сприяє написанню більш передбачуваного, зрозумілого та надійного коду при розробці у команді. Ця мова є сталою у сфері розробки як клієнтської частини, так і серверної.

Іншим необхідним інструментом при розробці є Deno, який надає можливості для керування інфраструктурою проєкту. SpaceSoup буде

використовувати його для запуску скриптів, що написані мовою TypeScript, і для клієнтської частини для керування необхідними бібліотеками, що також будуть використовуватись на стороні клієнта. Deno є простим у завантаженні і легким у використанні, що робить його чудовим інструментом для SpaceSoup.

2.3.2 Загальна структура проєкту

Структура застосунку відповідає типовим проєктам, розробленим на Deno та Go, і забезпечує зручність у підтримці та масштабуванні. У корені папки проєкту знаходяться головні файли налаштувань: deno.json, який призначений для керування інструментами, які надає Deno, включаючи усі необхідні залежності для збірки коду клієнта; tailwind.config.ts, який призначений для налаштування компіляції Tailwind CSS коду стилів клієнта; файли lite.go та main.go, які використовуються для запуску серверу в залежності від умов компіляції; .gitignore, який описує ігнорування файлів при публікації на зовнішній репозиторій Git, наприклад, GitHub; папки .github та .vscode для керування налаштуваннями GitHub репозиторієм і програмним середовищем Visual Studio Code; папка client, що містить усі потрібні компоненти для клієнтської частини, включаючи статичні зображення, вебсторінки, та код логіки; папка server, яка містить лише файли Go, які використовуються тільки для логіки серверу; папка scripts, яка має Deno TypeScript файли, які можуть запускатися незалежно і використовуються для спрощення керування проєктом; папка node_modules, яка є необхідною для компіляції коду клієнта і може бути проігнорована при публікації на зовнішній репозиторій через свій великий розмір, бо містить сторонні JavaScript бібліотеки, які автоматично завантажуються у разі необхідності; папка dist, що може містити бінарний файл Go серверу.

Папки node_modules та dist можуть бути відсутніми, бо генеруються автоматично при запусках конкретних скриптів. Проєкт також може містити файл .env, який потрібно створити вручну. Цей файл описує спосіб запуску серверу і SpaceSoup використовує цей файл для вибору порту та налаштування часу життя кешу зображень.

У папці server знаходиться модуль controller, що має підмодуль controller/controller_http, який описує оболонку обробника запитів, та модуль controller/model_http, який містить набір обробників запитів з полями. Зазначені запити можуть представляти більшість відомих форматів разом в одній Go структурі: запити від форм, заголовки, Cookies, JSON, URL та Query-параметри. Приклад такої структури представлено у лістингу 2.1, що також має метод для застосування алгоритму вторинної обробки до даних, які зберігає ця структура; цей метод представлено у лістингу 2.2.

Лістинг 2.1 Код структури, що використовується для обробки запиту розрахунку, який надіслано з форми:

```
type OrbitInput struct {
    Tau1 float64 `form:"tau1"`
    Tau2 float64 `form:"tau2"`
    V1   float64 `form:"v1"`
    V2   float64 `form:"v2"`
    V3   float64 `form:"v3"`
    Date string  `form:"date"`
}
```

Лістинг 2.2 Код методу, що використовується для застосування алгоритму до даних запиту:

```
func (p *OrbitInput) Input() (soup.Input, error) {
    date, err := soup.ParseDateJSON(p.Date)
```

```
    if err != nil {
        return soup.Input{}, err
    }
    speedList := []float64{}
    for _, v := range []float64{p.V1, p.V2, p.V3} {
        if v > 999. {
            continue
        }
        speedList = append(speedList, v)
    }
    return soup.Input{
        Id:    1,
        Tau1:  p.Tau1,
        Tau2:  p.Tau2,
        V_avg: soup.Average(speedList),
        Date:  date,
    },
    nil
}
```

Папка server також має модуль environment, що надає можливості для налаштування серверу, включно з реалізацією завантаження файлу .env, та змінної BuildModeValue, що задається через умовну компіляцію на одне зі значень: test, dev, prod. Ці значення використовуються надання серверу інформації про його програмне середовище. Наприклад, dev буде шукати перший незайнятий порт, а prod буде видавати помилку одразу, якщо налаштований порт є зайнятим.

Папка server також має модуль soup, який має програмну реалізацію алгоритму вторинної обробки та методи для створення результатів тестування цього алгоритму. У цьому модулі також реалізована візуалізація даних,

допоміжні методи та модульне тестування власних функцій, що представлено в лістингу 2.3.

Лістинг 2.3 Приклад тестування функції Float:

```
func TestFloat64(t *testing.T) {
    assert := assert.New(t)
    assert.Equal(Float64("12"), 12.)
    assert.Equal(Float64("12.64"), 12.64)
    assert.Equal(Float64("12,64"), 12.64)
}
```

Також при розробці застосовувано інструмент Git, який надає можливості для контролю версій застосунку та відкату до старих версій. Git знадобиться при з'єднанні кількох версій при розробці двох нових функцій одночасно, що корисно при наявності кількох розробників. Крім цього, проєкт повинен бути доступним і публічним, у зв'язку з чим SpaceSoup повинен бути розташований на GitHub. Ця платформа представляє можливості для безкоштовного необмеженого зберігання проєкту та спільної розробки. Схему роботи Git зображено на рисунку 2.1.

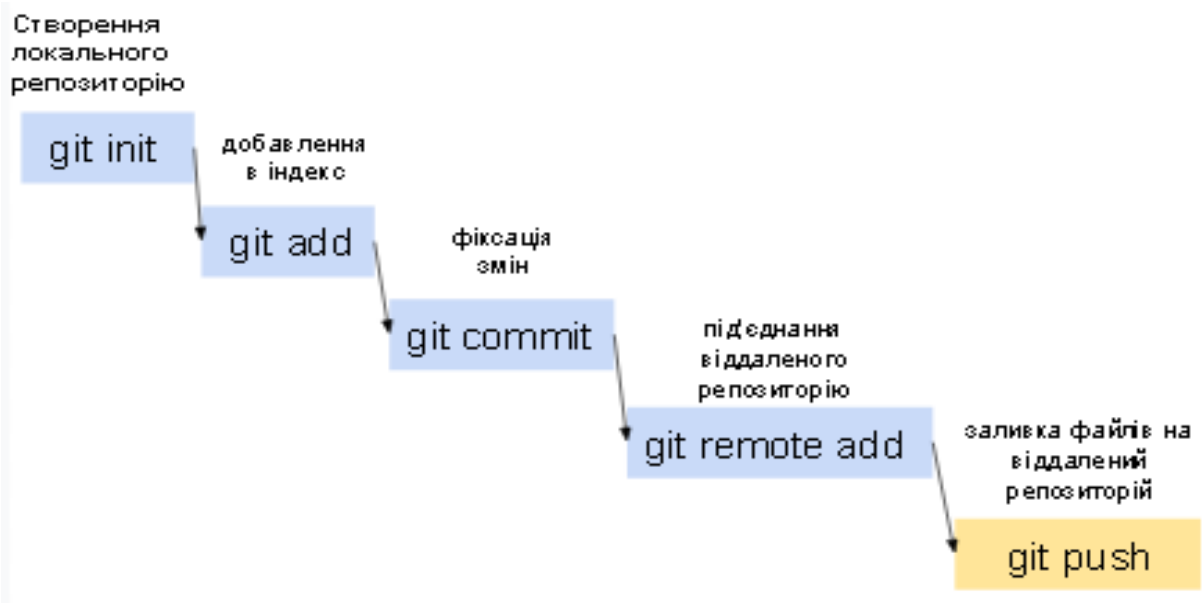

Рисунок 2.1 – Схема роботи Git репозиторію. Автор схеми – HyperHost.ua

2.3.3 Інтеграція компонентів

Для реалізації вебзастосунку, потрібно звернути увагу на інтеграцію компонентів для злагодженої роботи компонентів для полегшення роботи при розробці [24]. Deno – основний компонент, який використовується для керування проєктом і інтеграції компонентів. Головним середовищем розробки усіх компонентів та використання інструментів є VSC.

Для роботи над проєктом використовується редактор коду VSC, який надає можливості використання сторонніх розширень для оптимізації роботи. Deno надає власне офіційне розширення, що допомагає при розробці, особливо при редагуванні TypeScript файлів. Разом з Deno існують розширення для мови Go, та Tailwind CSS (рис. 2.2), які також використовуються для полегшення процесу розробки вебзастосунку. Ці розширення надають можливості автоматичної заміни, створення та заміни конструкцій, підсвічування синтаксису, заставлення дужок, перемикання коментарів, автоматичного форматування, перегляду інформації при наведенні, автоматичного завершення, переходу до визначення, перевірки помилок та інших. Усі ці розширення є офіційними і розробляються паралельно зі змінами самих інструментів, що є великою перевагою над неофіційними розширеннями.

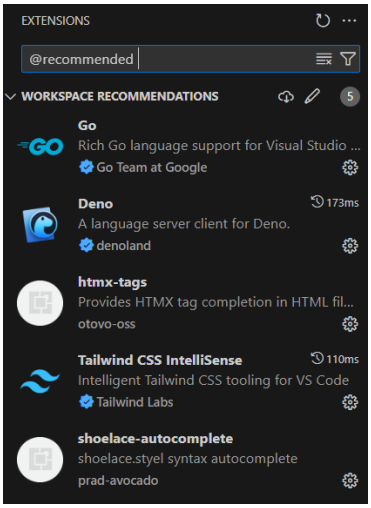

Рисунок 2.2 – Приклад рекомендацій розширень VSC для проєкту SpaceSoup, що описуються вручну в файлі .vscode/extensions.json

Deno використовується для управління проєктом, що базується на використанні основного файлу конфігурації deno.json у корені проєкту. Цей файл містить основні скрипти поміж інших: init, serve, compile:client, compile:server. Ці скрипт можуть бути запущені через командний рядок або при використанні VSC завдань (рис. 2.3), які описані у файлі .vscode/tasks.json.

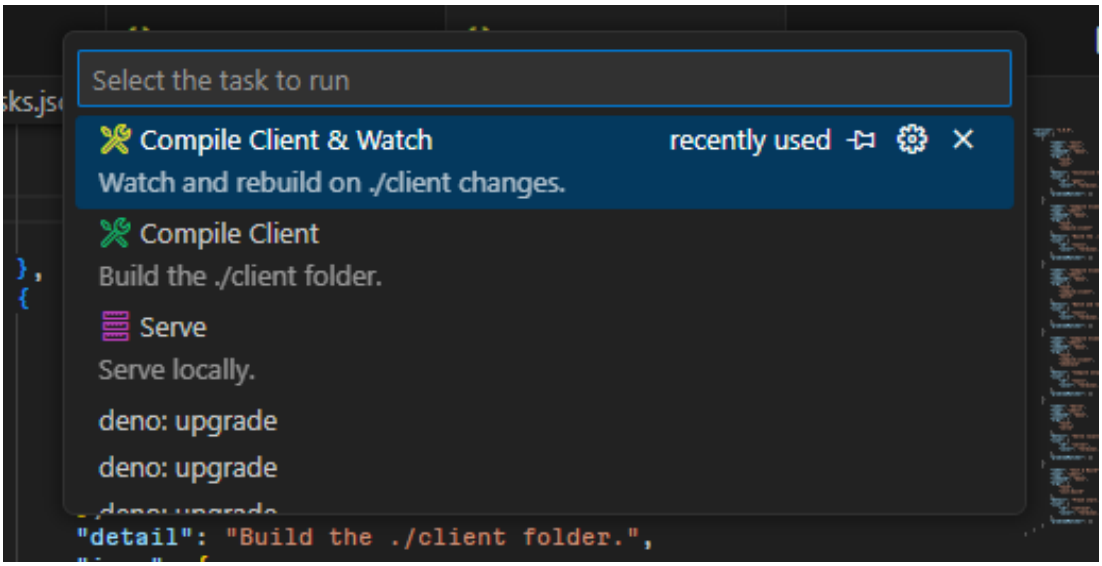

Рисунок 2.3 – Приклад спливаючого вікна у середовищі VSC, яке надає можливості запуску завдань

Папка serve є основним прикладом інтеграції TypeScript скриптів з сервером Go, але більша кількість інтеграцій відбувається у клієнтській частині завдяки іншому скрипту – scripts/build.ts. Цей скрипт описує роботу пакувальника клієнтського коду ESBuild, який збирає вміст папки client/src з нуля в папку client/static менше ніж за 6 секунд, включно з приблизно 2000 іконок, кодом TypeScript та Tailwind CSS.

Через мову TypeScript реалізується залучення та інтеграція сторонніх бібліотек HTMX та Shoelace завдяки сторонньому розширенню для ESBuild, @luca/esbuild-deno-loader [25]. HTMX є бібліотекою, що розширює фунціонал HTML через JavaScript, і надає можливості відправляти прості запити та зміни частин вебсторінки [26]. HTMX використовує hx-атрибути для налаштування цих запитів, проте запити потребують інтеграції вебкомпонентів з формами.

Shoelace – це бібліотека, яка надає 49 вебкомпонентів: кнопки, модальні вікна, поля введення, іконки, повзунки, картки, роздільники, палітри кольорів, прапорці, хлібні крихти, меню, перемикачі, підказки та інші. Усі вони мають префікс sl, і можуть використовуватись у звичайному HTML. Крім того, такі компоненти мають власні атрибути і часто схожі на стандартні. Наприклад, атрибути type, form, step, value є стандартними для звичайного елементу input, але sl-input лише симулює їх згідно стандартів і не завжди можна стверджувати, що інші стандартні атрибути працюватимуть. Приклад HTMX форми з використанням Shoelace зображено у лістингу 2.4.

Лістинг 2.4 Приклад коду форми, яка використовує HTTP метод post. Зверніть увагу на використання hx-атрибутів:

```
<form id="form-manually"
hx-post="/alg" hx-target="#form-manually-output">
        <sl-input  form="form-manually" label="Observation date"
          name="date" type="datetime-local"
          value="1972-01-25T05:23" required></sl-input>
        <sl-input form="form-manually"
          label="Time delay (&#x03C4;&#x2081;)"
          name="tau1" step="any" type="number"
          value="" required></sl-input>
        ...
        <sl-input form="form-manually" label="Speed (v&#x2081;)"
          name="v1" step="any" type="number" value="" required>
        </sl-input>
        ...
        <div class="dialog-footer"><sl-button form="form-manually"
            type="submit" variant="primary">
            <sl-icon slot="prefix" name="calculator"></sl-icon>
            Calculate</sl-button></div></form>
```

Зважаючи на те, що на етапі відправки даних форм HTMX ігнорує значення компонентів Shoelace, для виправлення цього недоліку було розроблено просте розширення для HTMX. Зазначене безіменне розширення додає значення Shoelace компонентів у запити форм, які використовують HTMX. Це розширення є суто програмним і не має відношення до VSC розширень. Воно може бути опубліковано незалежно в майбутньому у вигляді окремого проєкту, якщо не з’являться альтернативи. На теперішній момент публічних альтернатив не було знайдено у відомих реєстрах JSR та NPM. Проста структура HTMX розширення представлена в лістингу 2.5. Його застосування відбувається з використанням hx-ext, приклад якого можна знайти в наступному лістингу 2.6.

Лістинг 2.5 Структура розробленого HTMX розширення:

```
HTMX.defineExtension("shoelace", {onEvent(name, event) {...}})
```

Лістинг 2.6 Застосування розробленого HTMX розширення:

```
<body hx-ext="shoelace">
```

Код стилів клієнта Tailwind CSS теж обробляється пакувальником ESBuild, перетворюючись на код CSS, який підтримується браузерами. Для цього використовується стороннє розширення для ESBuild – esbuild-plugin-tailwindcss. Це розширення є простішим рішенням.

У зв’язку з тим, що Tailwind CSS має офіційний плагін Post CSS, необхідно також розглянути альтернативу використання Post CSS у майбутньому, оскільки esbuild-plugin-tailwindcss підтримує лише конкретну версію Tailwind CSS і є нестандартним та небажаним рішенням.

3 РЕАЛІЗАЦІЯ SPACESOUP

3.1 Програмна реалізація вебзастосунку

Вебзастосунок SpaceSoup розташовано публічно на платформі для розробників – GitHub – з дотриманням підходу відкритого коду. Це означає, що застосунок доступний публічно для кожного і він є доступним для розширення з боку третіх сторін, а також гарантує безпеку, бо кожен може впевнитися у поведінці застосунку. Публічні програмні проєкти можуть бути дуже вразливими до атак, оскільки усі користувачі можуть побачити вразливі місця серверу. Безпека користувачів застосунку є одним із основних пріоритетів, а тому сервер повинен забезпечувати безпечну обробку запитів. Під час розробки вебзастосунку SpaceSoup на меті є внесок в українську науку, який не можна ставити під загрозу наявністю програмних вразливостей. Під час розробки SpaceSoup вони були зведені до мінімуму для забезпечення стабільності роботи серверу у разі його публічного запуску і публічному використанні третіми сторонами.

Структура розробленого вебзастосунку має вигляд модульної архітектури, що пов’язано з використанням мови Go, яка підтримує лише такий підхід. Цього підходу також повністю дотримано під час створення інших частин проєкту, які не використовують мову Go зовсім. Модульна архітектура дозволяє розробляти частини вебзастосунку окремо, включаючи тестування, скрипти і конфігурації.

У зв’язку з тим, що застосунок було опубліковано на GitHub, використано сервіс GitHub Actions і реалізовано робочий процес для автоматичного форматування коду (рис. 3.1). Форматування коду може бути запущено вручну і в запитах на витягування, що робить код чистим і приємним для читання без залежності від автора, який вносить зміни у проєкт.

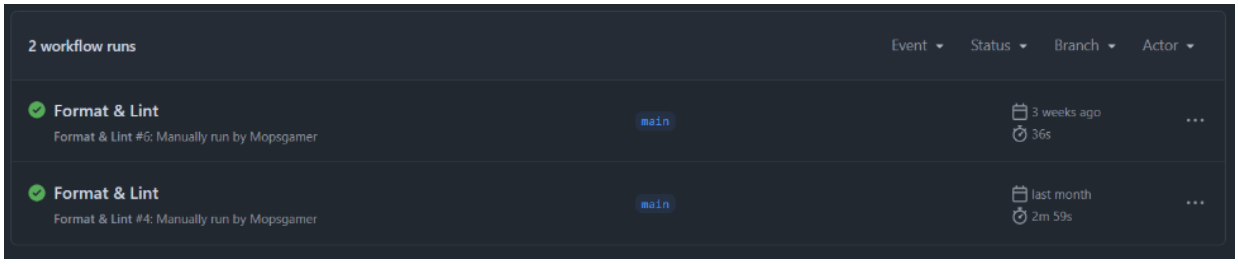

Рисунок 3.1 – Приклад запусків форматування коду на платформі GitHub

Для написання коду серверу використовується мова Go. У зв’язку з цим використовуються Go бібліотека Fiber, яка використовує підхід JavaScript бібліотеки Express.js – функціональне програмування кінцевих точок серверу. У зв’язку з тим, що наразі мова JavaScript є популярною, вибір бібліотеки Fiber є оптимальним, оскільки більше розробників зрозуміють роботу застосунку. У наступних лістингах є приклади підходів згаданих бібліотек.

Лістинг 3.1 Приклад функціонального підходу бібліотеки Express.js:

```
app.get(‘/’, (req, res) => {
  res.send(‘Hello World!’)
})
```

Лістинг 3.2 Приклад функціонального підходу бібліотеки Fiber:

```
app.Get("/", func (c *fiber.Ctx) error {
      return c.SendString("Hello, World!")
   })
```

Використовуючи бібліотеку Fiber, сервер SpaceSoup реалізує наступні вхідні точки: /static, /partials, /manually, /parse, /tests, /acknowledgements, /alg. Деякі з них опціонально можуть мати дочірні точки, наприклад, /alg/parse є дочірньою точкою /alg і не є кінцевою. Точка /alg/parse/image є дочірньою для alg/parse і кінцевою, оскільки не має дочірніх. Це сталий підхід будь-якого

вебсерверу. Кожен веббраузер використовує запит з HTTP методом get; щоб відобразити відповідь на запит у вигляді вебсторінки потрібно використовувати метод Fiber під назвою app.Get. Виклик цього методу створить обробник, який викликатиме іншу функцію, що реалізує відображення сторінки. У випадку з точкою /alg, клієнт буде отримувати сторінку з калькулятором.

Сторінка з калькулятором є окремим файлом, яка написана з використанням HTML з модифікацією. Такий файл проходить обробку сервером згідно реалізації Go HTML Template. Це дозволяє змінювати логіку відображення графічного інтерфейсу до його відображення. Для зміни графічних елементів після відображення необхідне інше рішення – JavaScript бібліотека, яку зможе запустити браузер. Такою бібліотекою є HTMX, яка може управляти графічними елементами очікуючи та створюючи зміни та події HTML. Це звільняє від написання супроводжуючого JavaScript коду, роблячи HTML майже самодостатнім.

Для використання вебзастосунку користувач повинен мати на своєму пристрої інтернет-браузер. Для цього можна використовувати браузер Chrome, Firefox, Edge, Safari, або будь-який інший браузер, що підтримує основні стандарти вебтехнологій: HTML5/CSS3, ECMAScript 2024, HTTP 2. Вебзастосунок підтримує не лише персональні комп’ютери, але й планшети та смартфони; для його роботи обов’язково потрібне інтернет-підключення, що може бути недоліком для деяких користувачів. У такому разі рекомендовано запустити свій власний сервер у зв’язку з наявністю інструкцій дій та публічності SpaceSoup.

Для початку роботи необхідно перейти на сторінку калькулятора; після цього користувач отримає доступ до полів введення даних. Для цього користувач має обов’язково заповнити поля «Observation date», «Time delay» та «Speed» (рис. 3.2) для правильного розрахунку параметрів. Після цього користувач зможе побачити результати розрахунків (рис. 3.3), натиснувши кнопку «Calculate».

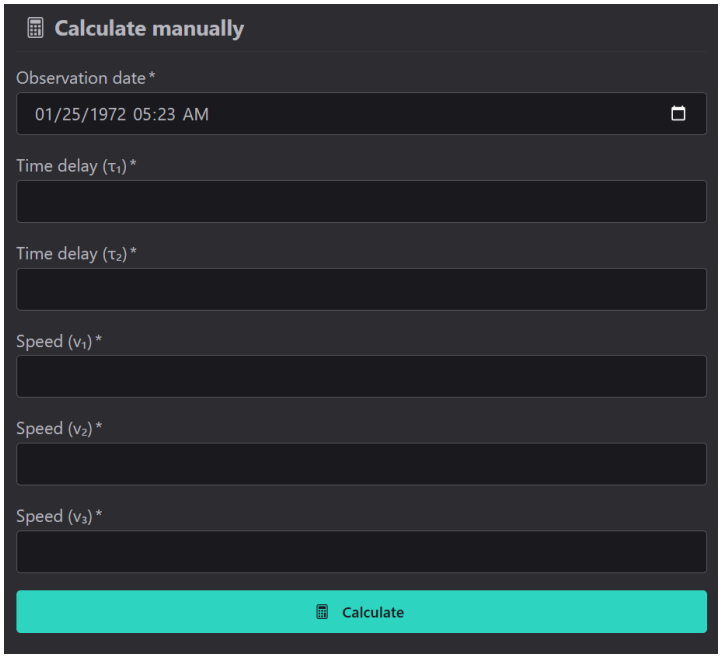

Рисунок 3.2 – Початковий інтерфейс калькулятора орбіти

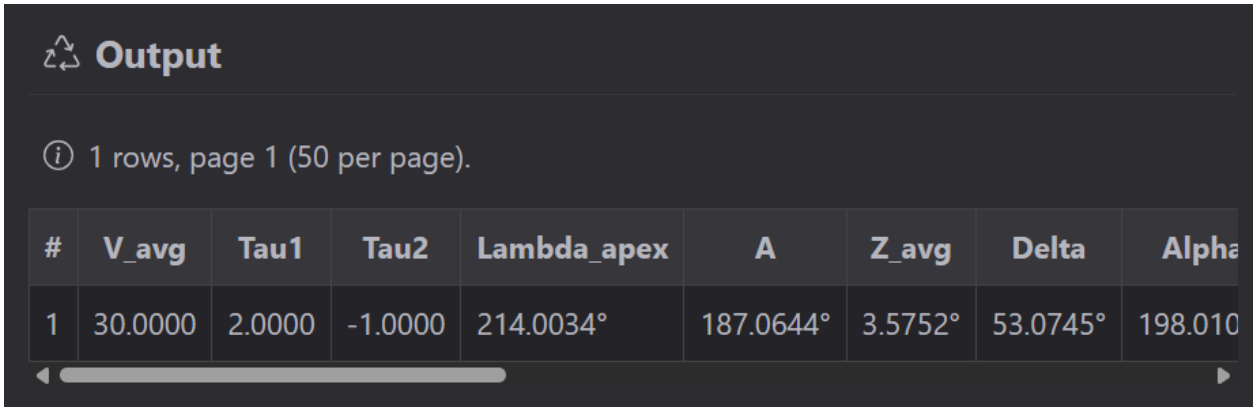

Рисунок 3.3 – Інтерфейс калькулятора орбіти після введення даних

Після натискання цієї кнопки бібліотека HTMX відправляє HTTP запит з використанням методу post, згідно значення hx-post, до точки /alg. Необхідно звернути увагу на те, що при різних методах точка /alg буде поводити себе зовсім інакше. При отриманні запиту post на точку /alg, замість повернення сторінки з калькулятором, сервер поверне частковий HTML документ з таблицею. Ця таблиця буде програмно розміщена у розділ «Output» замість інформаційного повідомлення завдяки можливостям, які надає використання бібліотеки HTMX. Результати розрахунків, вхідні дані та будь-які дані про користувача не будуть збережені на сервері.

Ця таблиця призначена для відображення вхідних даних та додатково розрахованих параметрів: азимут A, зенітний кут радіанта Z_avg, схиляння радіанта Delta, пряме сходження радіанта Alpha, екліптична широта радіанта Beta, екліптична довгота радіанта Lambda, довгота істинного радіанта Lambda_deriv, широта істинного радіанта Beta_deriv, нахил орбіти частинки до площини екліптики Inc, довгота висхідного вузла Wmega, аргумент перигелію Omega, геоцентрична швидкість V_g, геліоцентрична швидкість V_h, велика піввісь Axis, ексцентриситет Exc, істинна аномалія Nu. Швидкості 1, 2 та 3 використовуються для розрахунку середньої швидкості V_avg як середнє арифметичне значення трьох значень швидкості, вважаючи швидкість постійною.

У зв'язку з використанням стандартів HTML, перевірка значень форми введення стає простішим завданням, оскільки не вимагає написання перевірок на мові JavaScript. Форма не дасть відправити дані які не можуть бути використані для обчислення нечислових значень (рис. 3.4). Усі перевірки відбуваються на стороні сервера і сповіщають про будь-які порушення, навіть якщо користувач обходить перевірки браузера (рис. 3.5), що захищає сервер від несподіваних запитів.

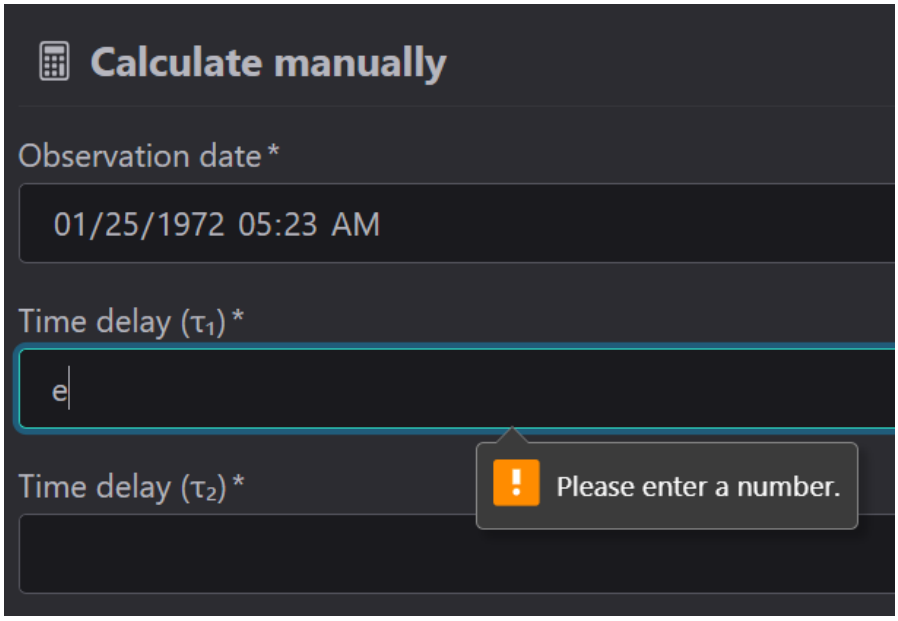

Рисунок 3.4 – Приклад помилки при введенні даних

```
OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    PROBLEMS

PS C:\Users\user\Documents\GitHub\space-soup> curl -X POST -F
"tau1=hello" http://localhost:3000/alg
<sl-alert open id="err-request"  variant="danger">
    <sl-icon slot="icon" name="exclamation-octagon"></sl-icon>
    bind: schema: error converting value for &#34;tau1&#34;
</sl-alert>
```

Рисунок 3.5 – Відповідь сервера на неправильні дані з обходом перевірки

Крім цього, розроблений застосунок надає можливості розрахунку за алгоритмом для довільної кількості орбіт і використання довільної кількості швидкостей для кожної орбіти на відміну від ручного введення. Для цього існує окрема вебсторінка, яка реалізує розрахунок параметрів через завантаження одного файлу (рис. 3.6). Вебзастосунок підтримує два наступні формати файлів: .csv, .tsv. Ці файли є стандартними і можуть бути легко відкриті сучасними інструментами редагування таблиць.

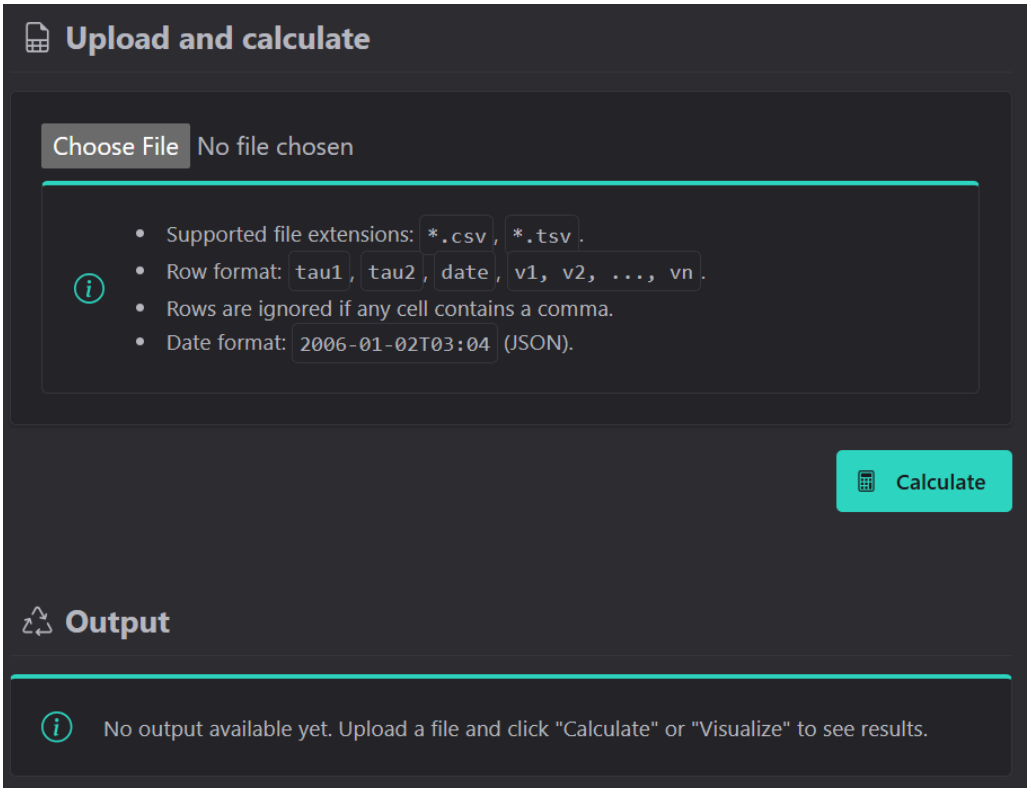

Рисунок 3.6 – Інтерфейс вебсторінки з можливістю завантаження файлу

Для початку користувач повинен натиснути кнопку «Обрати файл», після чого відкриється вікно вибору файлу (рис. 3.7) яке може бути різним для різних операційних систем.

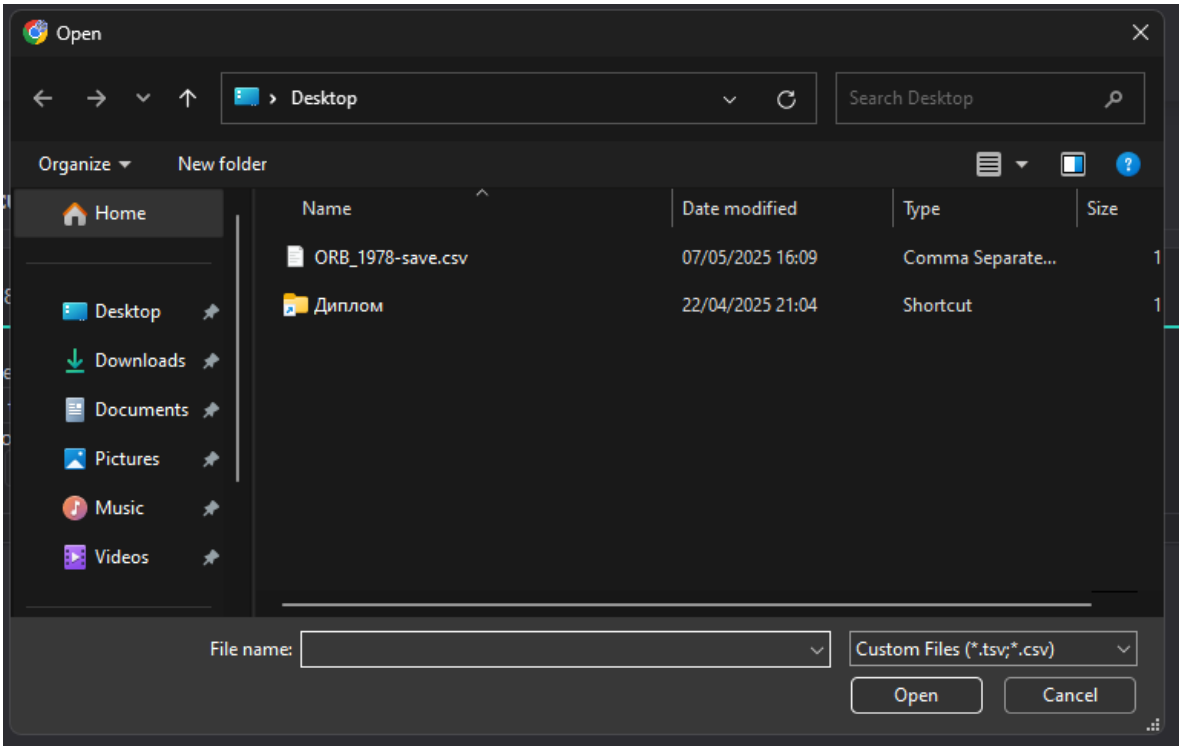

Рисунок 3.7 – Вікно вибору файлу

Користувач повинен обрати файл та підтвердити вибір. У випадку відміни вибору, користувач повинен знову побачити вебзастосунок в останньому положенні (рис. 3.6). Після обрання файлу користувач може розрахувати параметри, натиснувши кнопку «Calculate». Тоді у секції «Output» з’явиться таблиця (рис. 3.8) у якій кожен рядок буде відповідати кожному рядку у файлі. Деякі рядки будуть проігноровані у тому разі, якщо комірка не є числом або датою; у такому разі результати у згенерованій таблиці можуть бути зміщені згідно наявності таких комірок. Крім таблиці, користувач також зможе побачити кнопки для завантаження результатів у вигляді .xlsx або зображення .png (рис. 3.8).

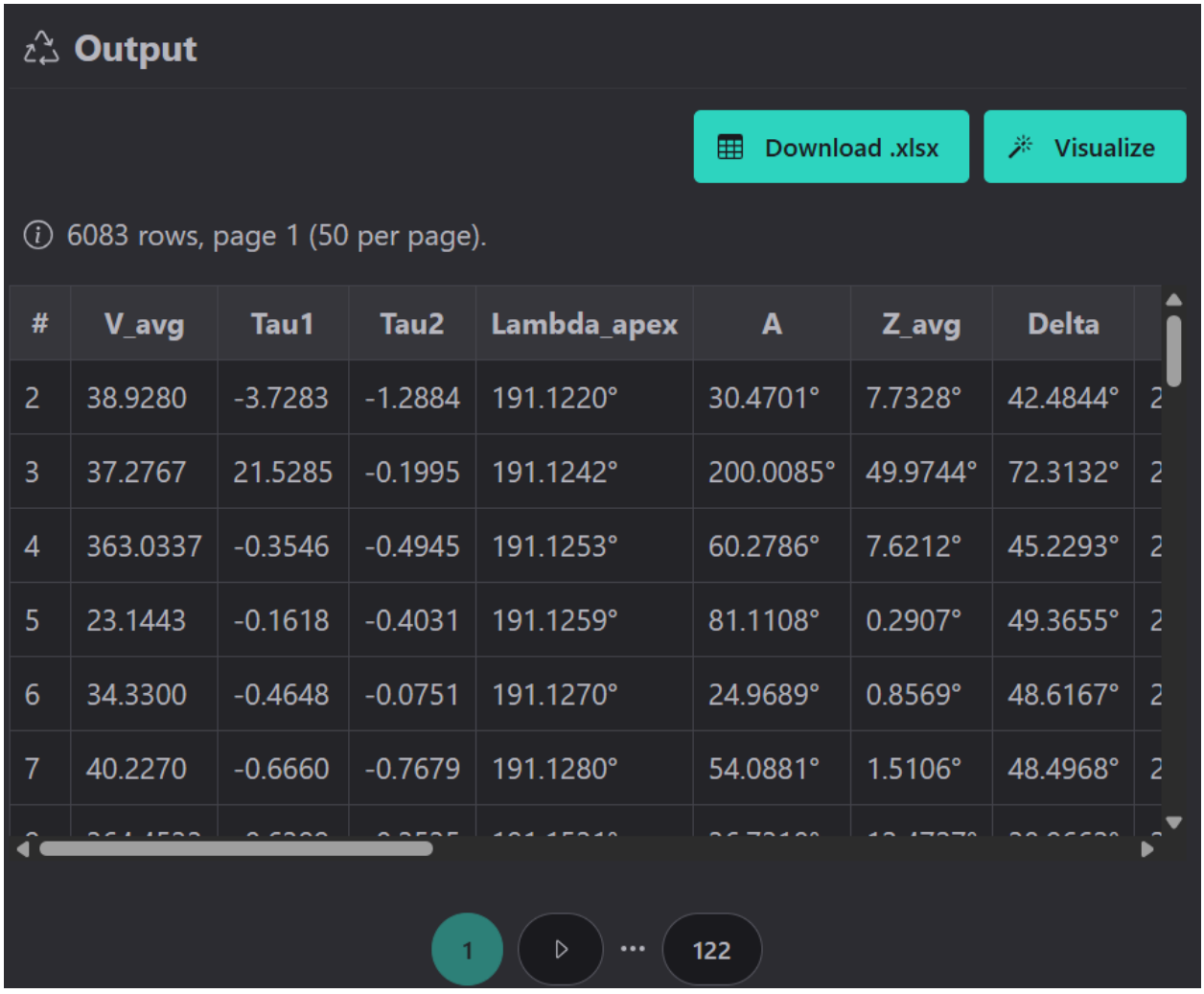

Рисунок 3.8 – Розрахунок таблиці з файлу

При використанні файлу з вхідними даними, до його формату застосовуються наступні правила, які не можуть бути проігноровані.

– файл повинен мати розширення, згідно свого формату;

– кожен рядок повинен включати дані у конкретному порядку;

– нецілі числа можуть бути записані з використанням коми або точки;

– формат дати повинен бути JSON рядком, 2006-01-02T03:04.

У разі порушення цих правил та натисканні кнопки «Calculate», користувач отримає інформаційне повідомлення з усією інформацією щодо помилок (рис. 3.9). Теж саме відбудеться при натисканні кнопки «Calculate» без попередньо завантаженого файлу (рис. 3.10).

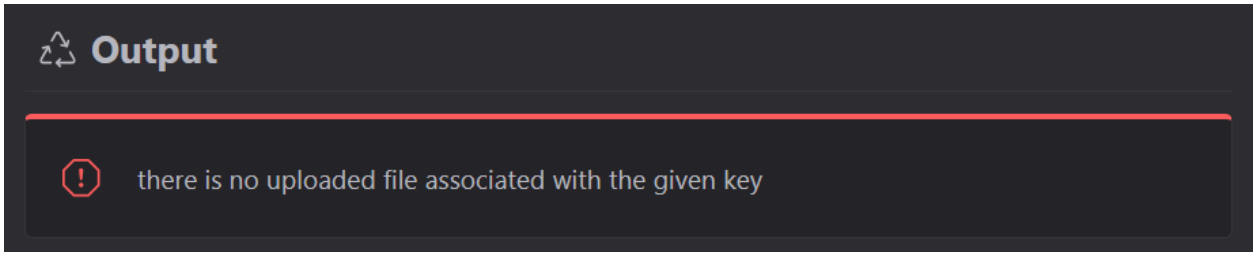

Рисунок 3.9 – Помилка під час розрахунку при відсутньому файлі, що зображає одночасно дві помилки для двох рядків у файлі

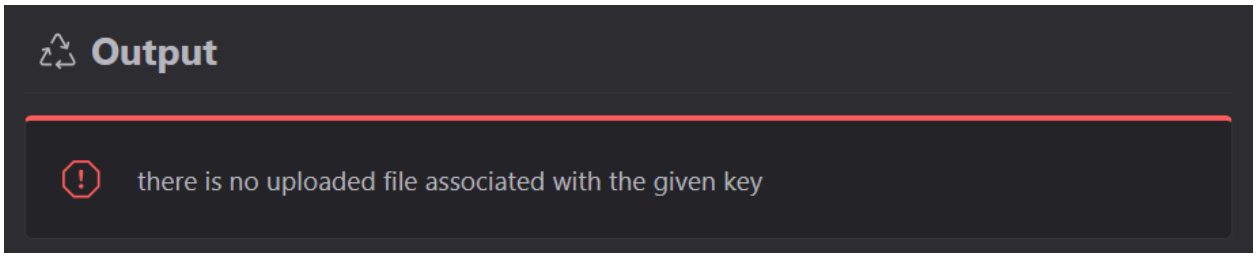

Рисунок 3.10 – Помилка під час розрахунку при відсутньому файлі «не існує завантаженого файлу пов’язаного з даним ключем»

Усі ці правила розташовані поряд з полем завантаження файлу, тому користувачу буде легко зрозуміти який формат він повинен мати. Для такого інформаційного повідомлення використовується елемент sl-alert з атрибутами variant та open, як показано у лістингу 3.3. В ньому знаходиться компонент sl-icon, що використовує позиціонування slot, яке працює не лише для sl-icon, а для будь-яких елементів. При цьому, елемент повинен бути нащадком для Shoelace компоненту, який підтримує надане значення для slot. За замовчанням sl-alert прихований, а тому він повинен мати атрибут open, який робить повідомлення видимим. Елемент sl-icon має атрибут name, який потрібен для позначення необхідної для відображення іконки.

Лістинг 3.3 HTML код інформаційного повідомлення:

```
<sl-alert variant="primary" open>
<sl-icon slot="icon" name="info-circle"></sl-icon>
...</sl-alert>
```

Результати розрахунків, які зображені на рисунку 3.8, можливо завантажити на комп’ютер у форматі Microsoft Excel Sheets (.xlsx). Цей формат є не лише дуже популярним, але й комфортним для перегляду. На відміну від .csv та .tsv форматів, використання даного формату надає можливості для вирівнювання тексту вертикально і горизонтально, зміни його розміру, жирності, кольору, повороту та багатьох інших параметрів. Також він надає можливості для зміни розмірів комірок за текстом, їх кольору, меж та багатьох інших параметрів стилізації.

Для створення та завантаження файлу .xlsx на свій пристрій необхідно натиснути кнопку «Download .xlsx» після чого файл буде одразу збережено (рис. 3.11). Якщо браузер налаштований інакше, користувачу буде необхідно обрати як саме він хоче зберегти файл.

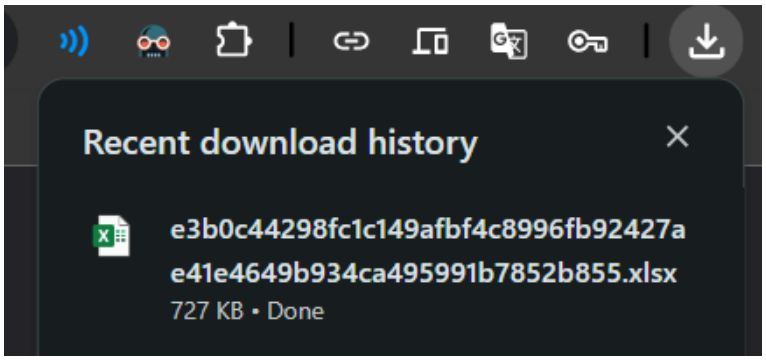

Рисунок 3.11 – Вигляд завантаження файлу у Chrome

Для генерації файлу використовується бібліотека github.com/xuri/excelize/v2. У процесі створення файлу, використовується масив результатів, який конвертується у .xlsx файл, а саме масив байтів. Байти згенерованого файлу з таблицею, байти файлу зображення та масив результатів зберігаються у кеші на основі значення ALG_CACHE_DURATION, яке можна змінити у файлі .env. За замовчанням час життя кешу триває десять хвилин і очищення відбувається кожен раз під час звертання до кешу. Цей підхід дозволяє уникнути проблем у разі використання асинхронних викликів функції очищення на основі часових

інтервалів. Видалення кешу необхідно для звільнення пам’яті від непотрібних результатів розрахунків.

Кеш зберігається у вигляді словника в оперативні пам’яті без використання файлової системи, що забезпечує додаткову швидкість. Водночас його буде знищено при зупинці серверу. Кожен запис має ключ, що розраховується з використанням алгоритму SHA-256 на основі змісту файлу вхідних даних. Кешування не застосовується при розрахунку однієї орбіти; воно виконується лише при завантаженні файлів з даними. Це робить функцію завантаження файлів кращою за функцію ручного введення. Наступний лістинг представляє Go код словника з кешем та функцію його очищення.

Лістинг 3.4 Кеш у вигляді словника записів розрахунків:

```
type SoupCache struct {
    ExpiresAt time.Time

    TestList []MovementTest

    ExcelBytes     []byte
    PlotImageBytes []byte
}

type FileHashCacheMap map[string]*SoupCache

func (m FileHashCacheMap) Free() {
    for hash, cache := range m {
        if cache.ExpiresAt.After(time.Now()) {
            continue
        }
        delete(m, hash)
    }}
```

Наявність кешу дозволяє не робити ідентичні розрахунки кожен раз для однакових файлів та різних типів результатів. У зв'язку з цим, час, який необхідний для перегляду зображень або завантаження файлу, стає мізерним після першого розрахунку. Це критично важливо не тільки при великій кількості користувачів для працездатності серверу, але й для швидкої роботи інтерфейсу. За відсутності високої швидкості, користувач буде відчувати певну незручність при роботі з розробленим застосунком; наявність кешування робить SpaceSoup привабливим для користування.

3.2 Тестування програмної реалізації

Для тестування програмної реалізації було реалізовано окрему вебсторінку на вкладці /tests для перегляду результатів перевірки еталонних значень (рис. 3.12, рис. 3.13). Ця сторінка має результати розрахунку 6743 орбіт (табл. А.1). Результати можна завантажити у вигляді таблиці .xlsx натисканням «Download .xlsx» (рис. 3.14) або відкрити зображення з графіком параметрів прямого сходження α та схиляння радіанта δ натисканням кнопки «Visualize».

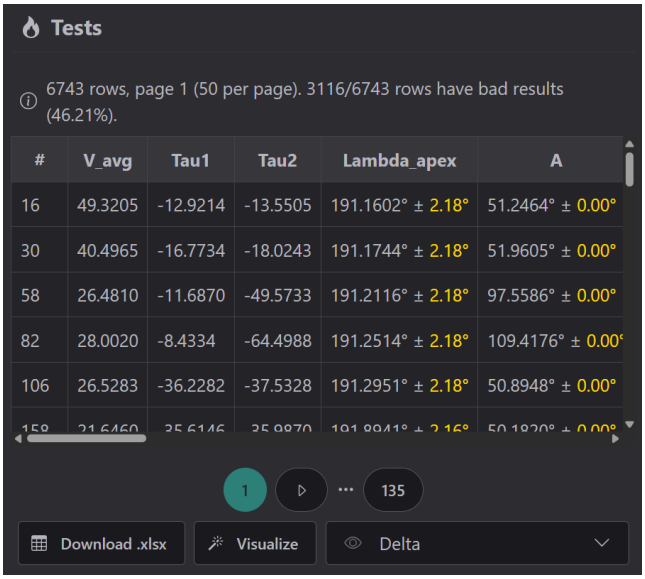

Рисунок 3.12 – Сторінка тестування алгоритму з переглядом похибки

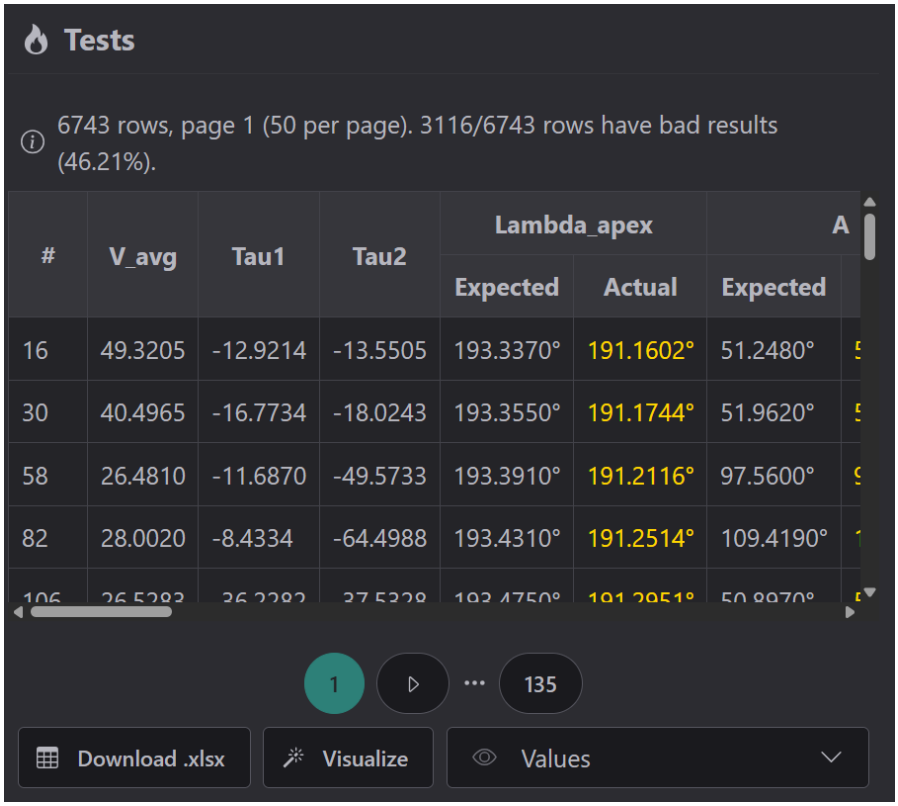

Рисунок 3.13 – Сторінка тестування алгоритму з переглядом еталону

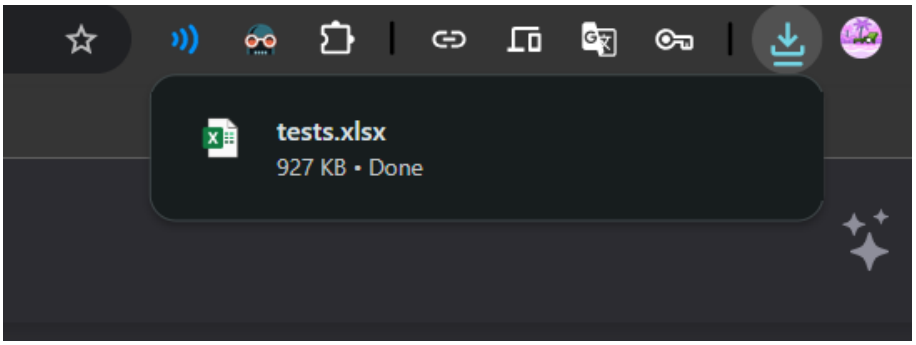

Рисунок 3.14 – Завантаження таблиці з тестовими даними

Ці результати є неповними, оскільки початкові вхідні дані містять 18716 елементів, що містять дату, часові затримки та швидкості, а еталонні значення складають лише 6743 орбіт. Згідно з розрахунками, за реалізованим алгоритмом можна зробити висновок про високу точність розрахунку азимуту, екліптичної довготи радіанту, великої піввісі та ексцентриситету.

Візуалізація зображення генерує двовимірний графік з двома осями та кривою екліптики. Згенероване зображення (рис. 3.15) буде містити точки, що мають координати радіанта: схиляння δ та пряме сходження α. Частина

вебзастосунку, яка відповідає за налаштування візуалізації, не реалізована в повному обсязі і використовується для тестування на еталонних розрахунках. Для перевірки отриманого зображення було побудовано відповідне зображення з використанням Aladin (рис. 3.16).

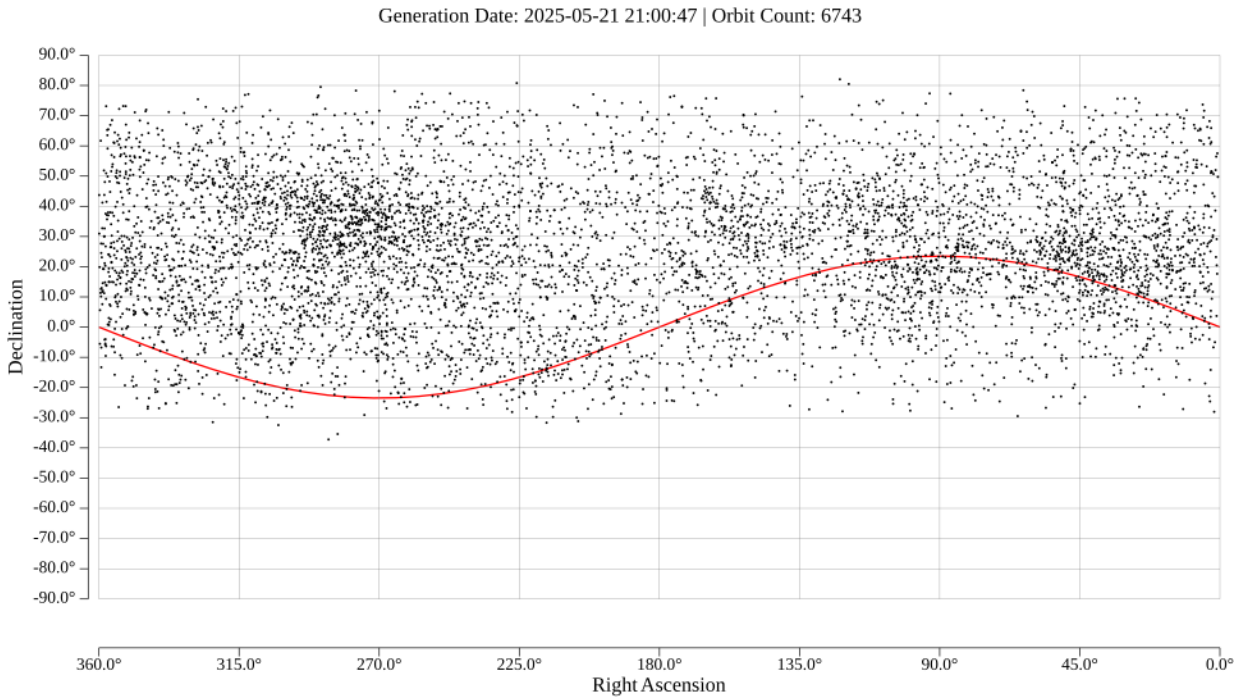

Рисунок 3.15 – Візуалізація даних тестування 6743 орбіт в SpaceSoup з використанням сторонньої бібліотеки gonum

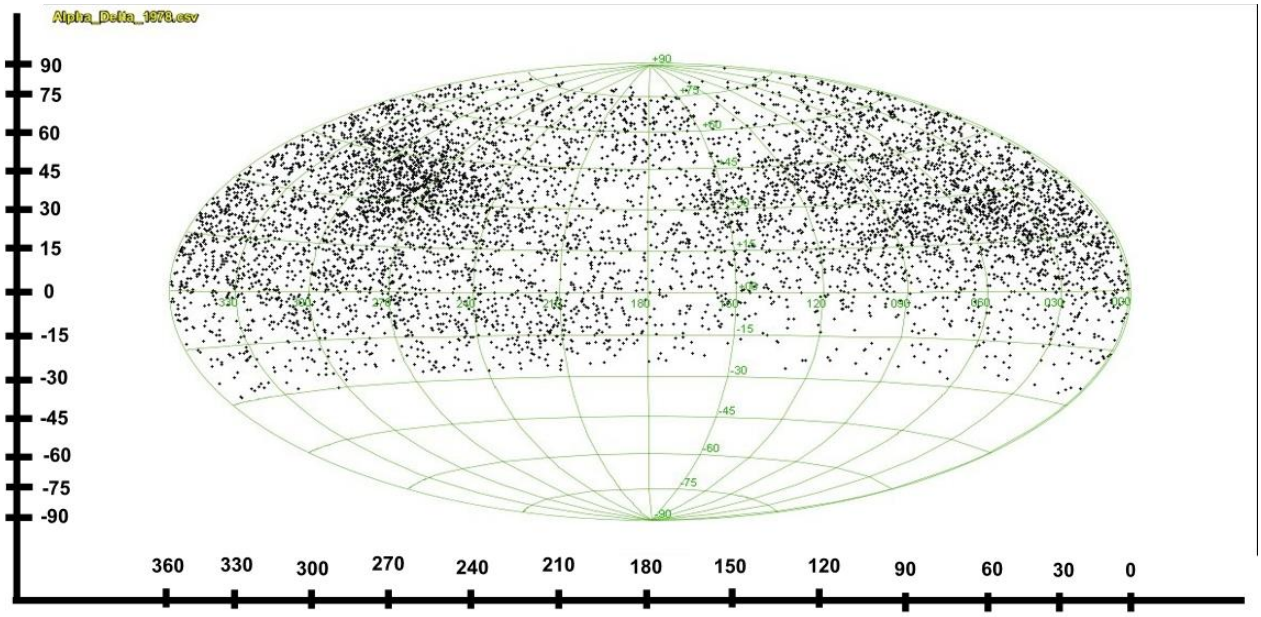

Рисунок 3.16 – Візуалізація даних тестування в Aladin

Для додаткової перевірки була використана програма Statistica (сильною стороною якої є графіка і побудова гістограм). Statistica було використано для побудови гістограми на основі тестових даних. Гістограми, які зображені на рисунках 3.17 та 3.18, показують скупчення схиляння δ і прямого сходження радіанта α у практично однакових діапазонах цих координат радіантів. Згідно з цією інформацією, можна сказати, що три застосунки мають за візуальним аналізом дуже схожі результати, які свідчать про правильно реалізовану функцію генерації.

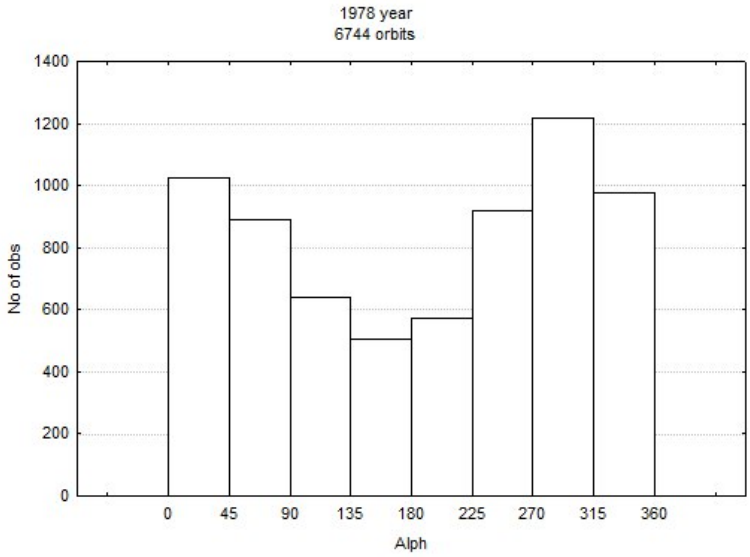

Рисунок 3.17 – Розподіл даних в Statistica для прямого сходження радіанта α

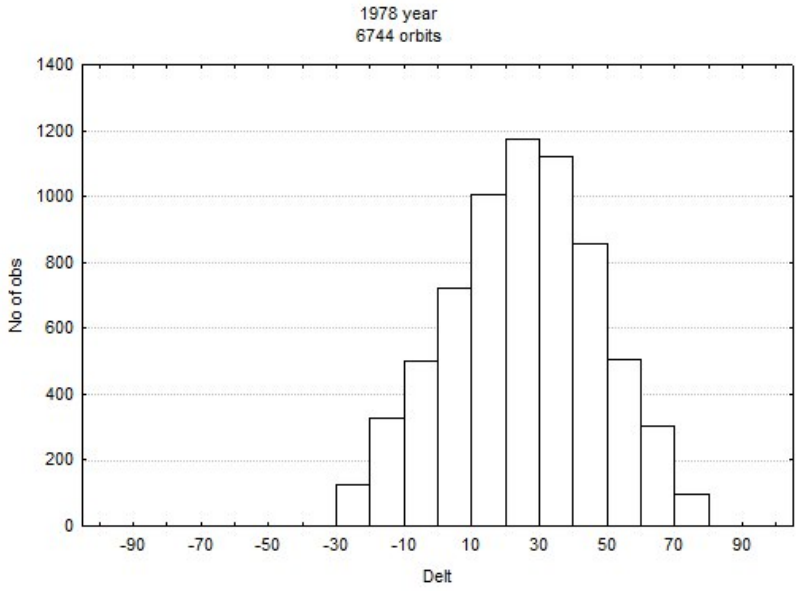

Рисунок 3.18 – Розподілу даних в Statistica для схиляння радіанта δ

Проаналізувавши рисунки отримані за допомогою SpaceSoup, Aladin та Statistica можна зробити висновок, що найбільша кількість метеорних орбіт спостерігається в діапазоні значень схиляння радіанта δ від 20 до 40, з піком поблизу 30 градусів, де кількість зареєстрованих орбіт перевищує 1200. Для значень δ менше 0 та понад 60 градусів спостерігається суттєве зниження – кількість орбіт у цих діапазонах не перевищує 300. Такий характер розподілу координат радіантів відповідає попереднім результатам аналізу, що проводилися іншими дослідниками у минулому та не суперечить даним, наявним в еталонній базі даних, наявними в базі даних.

Суттєвих розбіжностей для координат радіантів на рисунках 3.15 та 3.16 не виявлено. Серед орбітальних параметрів розрахованої та еталонної бази виявлені малі, значні або дуже значні відхилення (рис. 3.19, рис. 3.20). Щодо проведеного дослідження: виявлені окремі незначні відхилення обчисленої бази даних за допомогою застосунку SpaceSoup від еталонної бази можуть бути зумовлені використанням запозичених формул. На рисунку 3.19 червоним кольором зображені радіанти орбіт, що мають один або декілька параметрів з істотною похибкою.

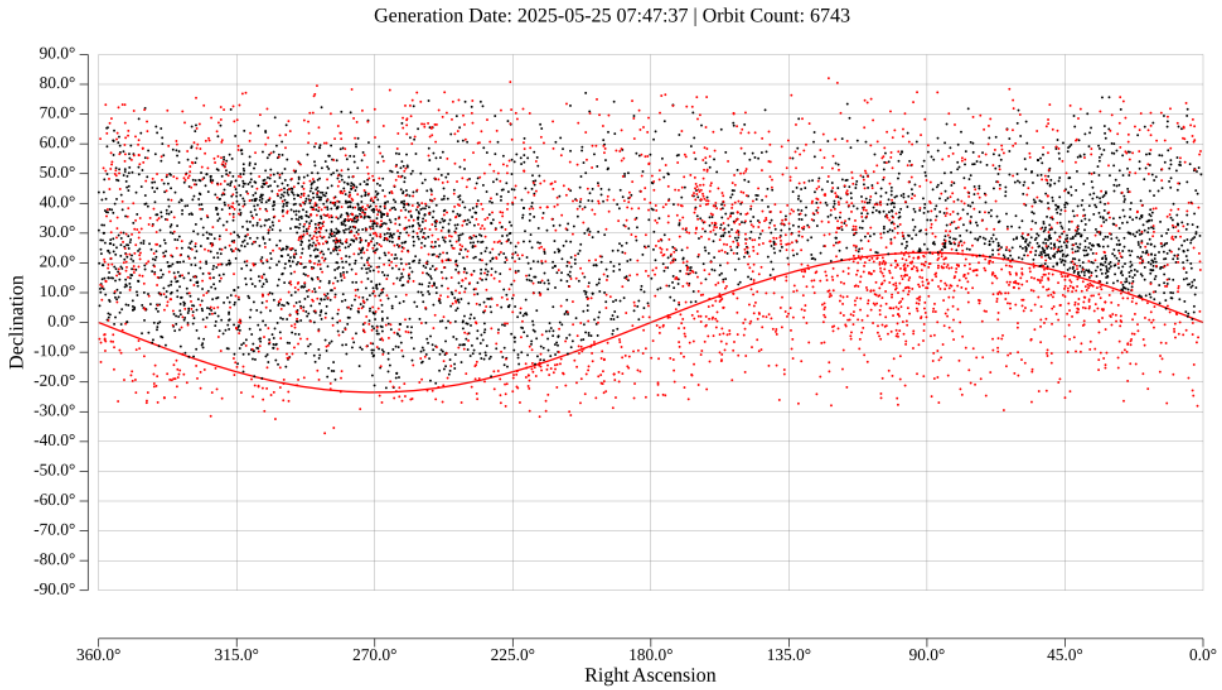

Рисунок 3.19 – Візуалізація радіантів орбіт з правильно визначеними орбітами та такими, що потребують переобчислення

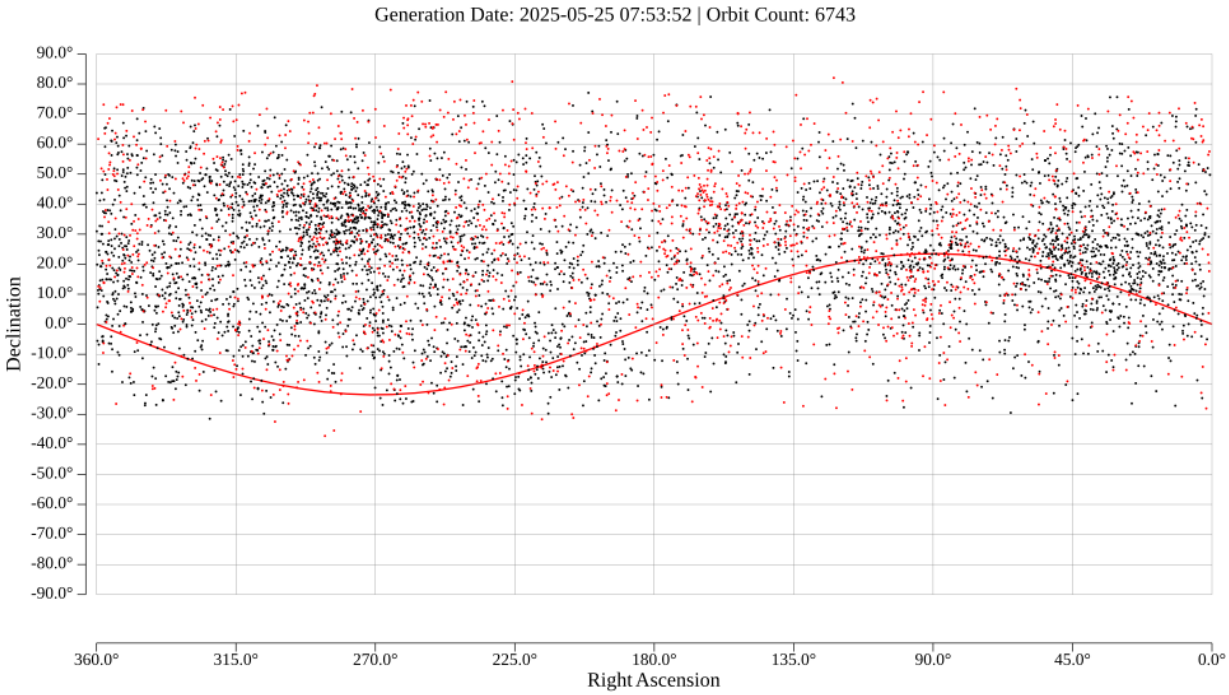

Рисунок 3.20 – Візуалізація з виключенням орбіт з аномальними значеннями невизначеності аргументу перигелію

Тестування показало 46,21% червоних точок: 3116 орбіти з 6743 (база 1978 року, дані MAPC II). Ці відхилення не є проблемою роботи застосунку SpaceSoup і надалі ця проблема може бути легко усунута в подальшому самим користувачем, коли будуть узяті правильні значення констант у результаті їх перевірки або моделювання.

Виявлені окремі значні відхилення обчисленої бази даних у кутових параметрах (рис. 3.19, рис. 3.20) за допомогою застосунку SpaceSoup від еталонної бази можуть бути пов’язані з невірним визначенням четверті кутових параметрів [19], проте це не є характеристикою роботи самого застосунку SpaceSoup, а свідчить про недосконаліть застосованих формул, що надалі можуть бути виправлені як описки. На рисунку 3.20 червоними будуть 31,23% (2106 з 6743), якщо ігнорувати аномальне значення аргументу перигелію.

Через явну можливість допущення помилок та описок під час реалізації алгоритму та вебзастосунку необхідно програмно розробити модульні тести [27]. На такі випадки Go надає вбудовані можливості для тестування

програмного забезпечення, яке використовується для тестування алгоритму і допоміжних методів. Функція тестування TestOrbit800016 відображена у наступному лістингу в якості прикладу.

Лістинг 3.5 Функція, що призначена для модульного тестування:

```
func TestOrbit800016(t *testing.T) {
   assert := assert.New(t)
   date, _ := ParseDateJSON("1978-01-02T10:28")
   CheckOrbit(assert,
      Input{
         Id:    800016, Date:  date,  Tau1:  -12.9214, Tau2:  -13.5505,
         V_avg: Average([]float64{49.037, 49.604}),
      },Movement{
         A:          RadiansFromDegrees(51.248),
         Lambda_apex:  RadiansFromDegrees(193.337),
         Z_avg:       RadiansFromDegrees(38.540),
         Delta:       RadiansFromDegrees(19.833),
         Alpha:       RadiansFromDegrees(219.011),
         Beta:        RadiansFromDegrees(33.152),
         Lambda:      RadiansFromDegrees(209.186),
         Lambda_deriv: RadiansFromDegrees(240.753),
         Beta_deriv:   RadiansFromDegrees(60.410),
         Inc:         RadiansFromDegrees(111.022),
         Wmega:       RadiansFromDegrees(87.743),
         Omega:       RadiansFromDegrees(283.346),
         V_g:         50.882,  V_h:         31.998,
         Axis:        1.1365,  Exc:         0.3846,
         Nu:          RadiansFromDegrees(92.257),
      },)}
```

Ця функція автоматично буде викликана інструментом go test, оскільки її назва починається з Test. Усі інші функції будуть проігноровані інструментом, проте можуть бути використані у функціях тестування. Функція CheckOrbit виконує роль спрощення написання коду тестування в лістингу 3.5. Також вона використовується при генерації таблиці тестування (рис. 3.12, рис. 3.13). Це є прикладом того, як SpaceSoup використовує сучасний підхід до написання повторно використовуваного коду. CheckOrbit використовується під час генерації графічних таблиць та при модульному тестуванні одночасно.

Тест може бути запущено з використанням графічного інтерфейсу (рис. 3.21) або консольної команди go test (рис. 3.22). Варто зазначити, що наявність графічного інтерфейсу пов’язана з наявністю розширення Go для VSC, тому без цього розширення такий інтерфейс буде відсутнім. Це розширення базується на go test, тому воно показує ідентичні результати з цим інструментом. Результати тестування мають успіх – негативні тестові випадки відсутні. Це означає, що функції алгоритму працюють як очікується. На рисунках 3.21, 3.22 зображені результати тестування алгоритму та допоміжних функцій; усі результати кажуть про те, що методи працюють як очікується.

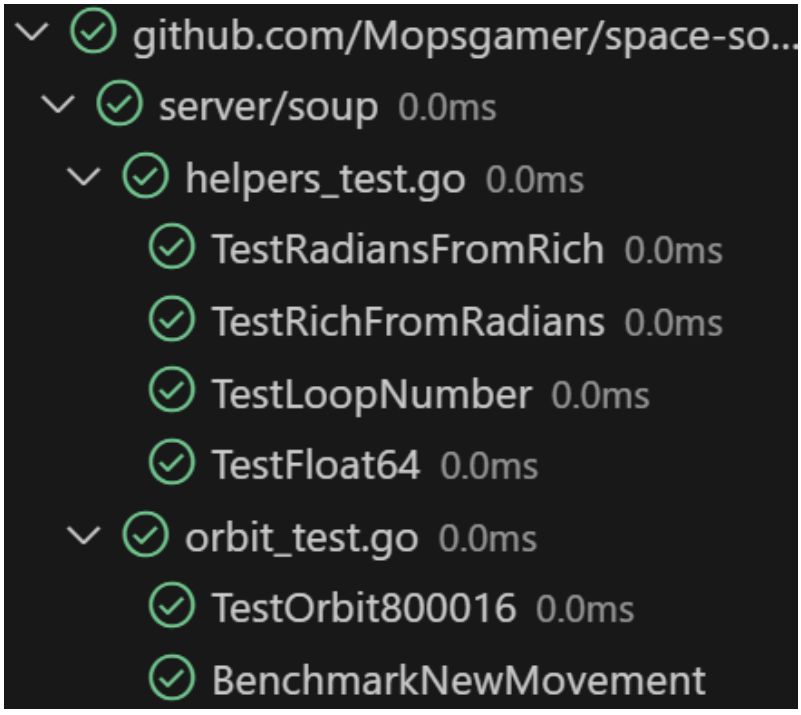

Рисунок 3.21 – Графічний інтерфейс тестування

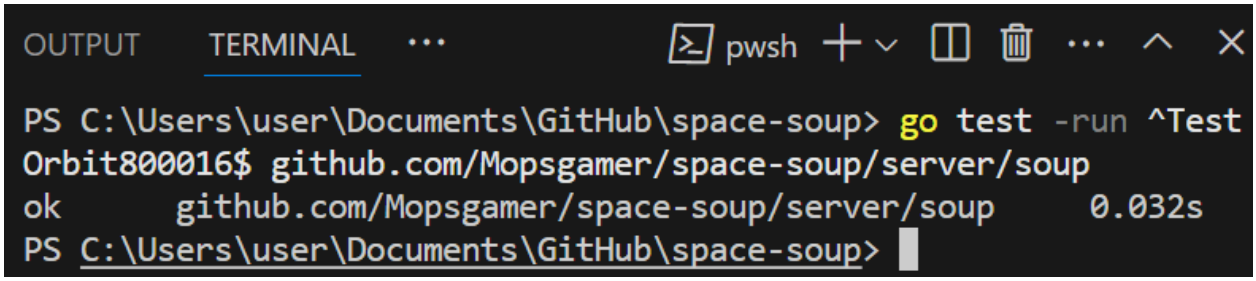

Рисунок 3.22 – Результат запуску команди тестування TestOrbit800016

У зв’язку з цим можна зробити висновок: алгоритм надає результати, які знаходяться в межах еталонних значень. Проте, результати цих тестувань не можна порівнювати з результатами таблиці на рисунку 3.12, оскільки зазначені тестування стосується лише конкретних орбіт. Відсутність модульного тестування для усіх орбіт пов’язана з їх великою кількістю; у разі великої кількості негативних результатів тестів, їх не можна вільно переглядати. Для боротьби з цією проблемою необхідно додати конкретні тестові випадки, але такі випадки важко відслідкувати і для цього потрібен об’ємний аналіз.

Крім модульного тестування, також було реалізовано простий тест продуктивності – теж з використанням інструменту go test, але з прапором bench. Згідно цього тесту можна сказати, що реалізація алгоритму не виділяє пам’яті під час розрахунків і буде працювати швидко. Функція такого тестування починається з Benchmark замість Test і представлена у лістингу 3.6.

Лістинг 3.6 Код функції тестування продуктивності:

```
func BenchmarkNewMovement(b *testing.B) {
    var date, err = ParseDateJSON("1972-01-25T06:07")
    if err != nil {
        panic(err)
    }
    b.ResetTimer()
    for range b.N {
```

```
        NewMovement(Input{
            Tau1:  -12.7572,
            Tau2:  -17.5536,
            V_avg: Average([]float64{33.858, 33.832, 33.965}),
            Date:  date,
        })
    }}
```

Результати показують 659115 ітерацій, де кожна ітерація займає в середньому 1781 наносекунд, або 1,17 секунд для усіх ітерацій. Якщо тест продуктивності працює вірно, для розрахування одного мільйона орбіт необхідно менше трьох секунд. Кожна ітерація є операцією розрахунку параметрів за реалізованим алгоритмом вторинної обробки для однієї орбіти. Також необхідно провести тестування вилучення початкових даних, оскільки це теж може впливати на швидкість отримання результатів розрахунків попри швидку роботу алгоритму. Для цього необхідно створити інший тест продуктивності з тестуванням швидкості зчитування початкових значень для орбіт з файлу; це стане сильнішим підтвердженням стресостійкості алгоритму та його реалізації.

3.3 Перспективи розвитку

Вебзастосунок SpaceSoup надає готову інфраструктуру для розробки, створюючи прості умови для створення та поліпшення головних функцій. Для розробки цих функцій потрібна вагома кількість часу, тому вони не були реалізовані під час виконання цієї роботи. У зв’язку з цим потрібно представити перспективи розвитку застосунку.

У зв’язку з тим, що вебзастосунок націлений на обслуговування великої кількості користувачів, він має працювати швидко. Одним з недоліків може

стати повільна швидкість під час розрахунку великої кількості орбіт. При його виправленні варто звернути увагу на перевагу використання мови Go, яка вже є частиною вебзастосунку SpaceSoup – можливість запуску асинхронних операцій розрахунку. Також для оптимізації роботи серверу потрібно оптимізувати кількість операцій виділення пам’яті. В нагоді стануть інструменти профілювання методів та тести продуктивності. Одним із багатьох підходів оптимізації роботи вебзастосунку є використання масивів відомої довжини, розумного використання покажчиків, кращої логіки кешування та циклів.

Покращенням може стати можливість завантаження застосунку на комп’ютер, смартфон та інші пристрої. Це може стати складним завданням, оскільки вебзастосунки мають мало спільного зі звичайними застосунками. Крім того, настільні та мобільні застосунки також відрізняються, що додає труднощі з підтримкою усіх пристроїв. Паралельно із цим потрібно налаштувати сервер, щоб він надавав можливість завантаження через посилання. При розробці цих функцій необхідно володіти великою кількістю навичок роботи з багатьма інструментами; необхідно буде використовувати інструменти мови Go, бо на цій мові написано серверну частину. Наразі Go підтримує наступні цілі для компіляції: aix/ppc64, android/386, android/amd64, android/arm, android/arm64, darwin/amd64, darwin/arm64, dragonfly/amd64, freebsd/386, freebsd/amd64, freebsd/arm, freebsd/arm64, freebsd/riscv64, illumos/amd64, ios/amd64, ios/arm64, js/wasm, linux/386, linux/amd64, linux/arm, linux/arm64, linux/loong64, linux/mips, linux/mips64, linux/mips64le, linux/mipsle, linux/ppc64, linux/ppc64le, linux/riscv64, linux/s390x, netbsd/386, netbsd/amd64, netbsd/arm, netbsd/arm64, openbsd/386, openbsd/amd64, openbsd/arm, openbsd/arm64, openbsd/ppc64, openbsd/riscv64, plan9/386, plan9/amd64, plan9/arm, solaris/amd64, wasip1/wasm, windows/386, windows/amd64, windows/arm64. Згідно цього списку, створення такої функції є теоретично можливим, проте представлення реального алгоритму дій не є можливим – Go надто молода мова.

Алгоритм має власні обмеження і недоліки – одним з таких недоліків є незмінні константи, які не підходять для точних розрахунків, що пов’язано зі зміною положення Землі залежно від дати або недостатньою точністю самих констант.

Інтерфейс вебзастосунку SpaceSoup також має обмеження для перегляду великих таблиць, тому необхідно розробити інструменти пошуку та фільтрації. Дані можливості вже частково розроблені, проте потребують калібрування на серверній частині та реалізації на стороні клієнта. Ця можливість дозволить легше переглядати конкретні значення, використовуючи номери рядків. Вже створена функція Range для застосування проміжку на масиві, яка повертає часткові дані з цього масиву в залежності від рядка rang, міститься у наступному лістингу. Також можна представити ще один недолік – недостатня гнучкість введення швидкостей. Необхідно реалізувати можливість створення та видалення форм введення даних, бо зараз ця можливість обмежена обов’язковим введенням трьох швидкостей. Це завдання має високий рівень складності, бо вимагає створення власного вебкомпоненту мовою TypeScript.

Лістинг 3.7 Попередньо створений метод Range:

```
func Range(tests []MovementTest, bounds string) (testsRanged []MovementTest, err error) {
    testsRanged = []MovementTest{}
    err = nil
    if bounds == "" {
        testsRanged = tests
        return
    }
    rangeList, err := expandrange.Parse(bounds)
    if err != nil {
        return
```

```
    }
    for _, i := range rangeList {
        testsRanged = append(testsRanged, tests[i])
    }
    return
}
```

3.4 Інструкції для розробників

Для спрощення роботи із застосунком ентузіастам, дослідникам та розробникам доцільно мати інструкцію користування, яка полегшить запуск власних версій при створенні нових функцій або виправленні наявних. Початок роботи з власною версією вебзастосунку потребує базових знань про роботу файлової системи та терміналу. У результаті після виконання інструкцій буде запущено сервер, який надасть можливості для використання вебзастосунку локально. Наявність серверу на власному комп’ютері надає можливості для використання застосунку без інтернету і тестування власних розроблених або виправлених функцій.

Для усунення складнощів, які можуть бути пов’язані з використанням Git, доцільно використовувати вбудовані можливості у редакторі VSC. Редактор надає візуальний інтерфейс для роботи з Git та GitHub, який допоможе правильно налаштувати власну версію проєкту.

Її створення починається з клонування оригіналу, що відбувається інструментом Git, причина вибору якого вже було розглянуто в першому розділі. Щоб встановити Git можна використати офіційний встановлювач на офіційному сайті Git або скористатись командою winget add git.git у терміналі (рис. 3.23). Для його видалення ви можете використати winget remove git.git.

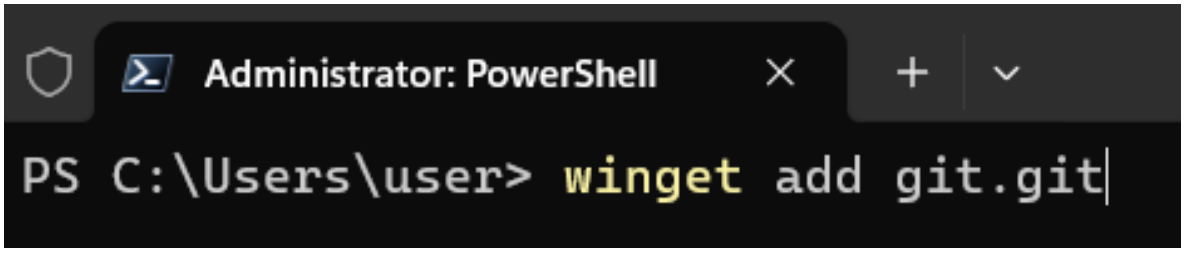

Рисунок 3.23 – Завантаження Git через термінал PowerShell 7

Після завантаження Git ви можете клонувати GitHub репозиторій. SpaceSoup зберігає вихідний код на GitHub, тому команда клонування має вигляд git clone https://github.com/Mopsgamer/space-soup.git (рис. 3.24).

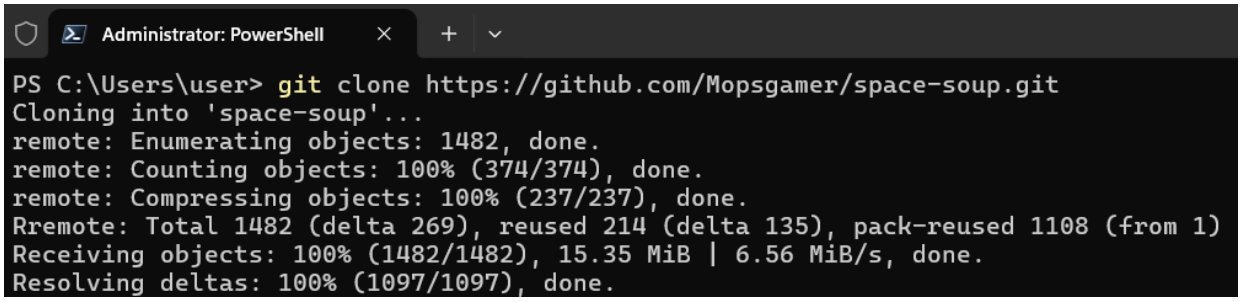

Рисунок 3.24 – Клонування SpaceSoup

Ця команда клонує останню версію проєкту з мережі для чого необхідне підключення до інтернету. На рисунку 2.24, команда була виконана у папці C:\Users\user, а папка проєкту знаходиться у C:\Users\user\space-soup. Після клонування проєкту необхідно перейти до нього командою cd space-soup. Перед запуском серверу необхідно скомпілювати клієнтський код; для цього на пристрої повинні бути встановлені Deno та Go. Команда winget add golang.go – швидкий спосіб встановити Go, але ви також можете скористатись інструкціями з офіційного сайту. Для завантаження Deno рекомендовано використовувати офіційний скрипт встановлення, який можна запустити командою irm https://deno.land/install.ps1 | iex. Також можна застосувати winget add denoland.deno у разі виникнення проблем з використанням офіційної. Після завантаження обох інструментів, ви можете перевірити їх версії (рис. 3.25) командами go version та deno -v.

```
PS C:\Users\user> go version
go version go1.24.3 windows/amd64
PS C:\Users\user> deno -v
deno 2.3.3
```

Рисунок 3.25 – Перевірка версій Go та Deno

Усі наступні команди виконуються у контексті папки C:\Users\user\space-soup. На рисунку 3.26 зображено команду для ініціалізації – deno task init:build. Вона підготує усе що потрібно для правильної роботи клієнтської частини.

```
PS C:\Users\user\space-soup> deno task init:build
Task init:build deno task init && deno task compile:client
Task init deno run -A scripts/init.ts && deno install --allow-scripts
- [init-files] Initializing '.env'...done
Task compile:client deno run -A scripts/build.ts
ⓘ [client-compilation] Starting bundling "./client" ...
- [client-compilation] Bundling 1/4: ./client/static/shoelace/assets...done
- [client-compilation] Bundling 2/4: ./client/static/assets...done
- [client-compilation] Bundling 3/4: ./client/static/js...done
- [client-compilation] Bundling 4/4: ./client/static/css...done
✔[client-compilation] Bundled successfully
```

Рисунок 3.26 – Компіляція клієнтського коду

Ця команда збудує клієнтську частину та встановить необхідні бібліотеки. Після цього ви можете запустити сервер, використовуючи команду deno task serve. Після отримання сповіщення у терміналі, яке зображено на рисунку 3.27, можна почати використовувати вебзастосунок як звичайний користувач, відкривши вебсторінку у браузері за вебадресою http://127.0.0.1:3000 (рис. 3.28).

```
 ┌───────────────────────────────────────────────────┐
 │                   Fiber v3.0.0-beta.4             │
 │               http://127.0.0.1:3000               │
 │       (bound on host 0.0.0.0 and port 3000)       │
 │                                                   │
 │ Handlers ............ 20  Processes ........... 1 │
 │ Prefork ....... Disabled  PID ............. 7196 │
 └───────────────────────────────────────────────────┘
```

Рисунок 3.27 – Запущений сервер SpaceSoup

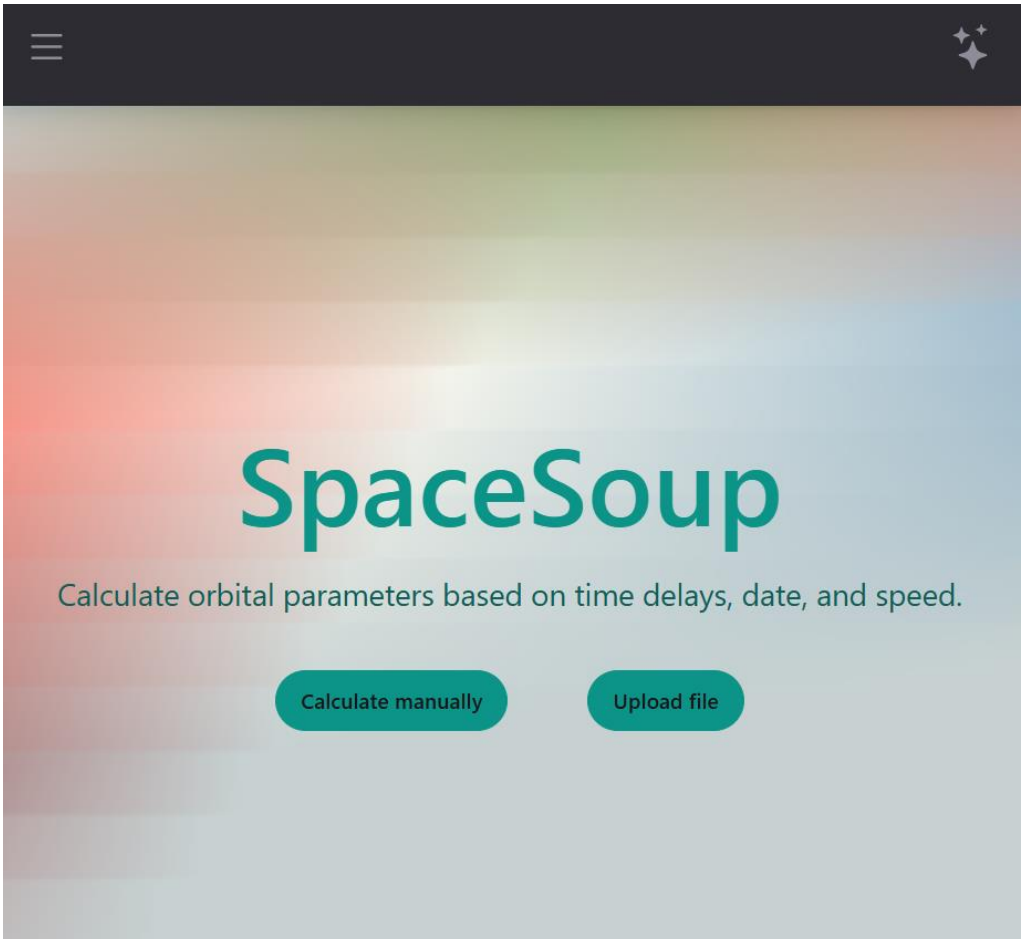

Рисунок 3.28 – Вигляд вебсторінки SpaceSoup, localhost:3000

Під час роботи сервера у терміналі будуть з’являтися сповіщення про запити, які обробляє вебзастосунок, включно з результатами відповідей та помилками (рис. 3.29). Зазвичай помилки виникають при зміні функцій на стороні сервера, наприклад, коли код клієнту або серверу був змінений.

```
23:33:23 | 200 |      6.6016ms |       127.0.0.1 | GET      | /
23:33:23 | 200 |      80.469ms |       127.0.0.1 | GET      | /static/js/homepage.js
23:33:23 | 200 |     79.1171ms |       127.0.0.1 | GET      | /static/css/homepage.css
23:33:23 | 200 |      1.9977ms |       127.0.0.1 | GET      | /static/shoelace/assets/icons/calculator.svg
23:33:23 | 200 |         999µs |       127.0.0.1 | GET      | /static/shoelace/assets/icons/file-earmark-spreadsheet.svg
23:33:23 | 200 |       505.2µs |       127.0.0.1 | GET      | /static/shoelace/assets/icons/table.svg
23:33:23 | 200 |           0s |       127.0.0.1 | GET      | /static/shoelace/assets/icons/sun.svg
23:33:23 | 200 |       1.513ms |       127.0.0.1 | GET      | /static/shoelace/assets/icons/display.svg
23:33:23 | 200 |       1.619ms |       127.0.0.1 | GET      | /static/shoelace/assets/icons/moon.svg
23:33:23 | 200 |           0s |       127.0.0.1 | GET      | /static/shoelace/assets/icons/lightning-charge.svg
23:33:23 | 200 |       1.003ms |       127.0.0.1 | GET      | /static/shoelace/assets/icons/three-dots.svg
23:33:23 | 200 |           0s |       127.0.0.1 | GET      | /static/shoelace/assets/icons/github.svg
23:33:23 | 200 |       507.4µs |       127.0.0.1 | GET      | /static/shoelace/assets/icons/mailbox-flag.svg
23:33:23 | 200 |       507.7µs |       127.0.0.1 | GET      | /static/shoelace/assets/icons/ticket-perforated.svg
23:33:23 | 200 |       569.9µs |       127.0.0.1 | GET      | /static/shoelace/assets/icons/hearts.svg
23:33:23 | 200 |      2.2256ms |       127.0.0.1 | GET      | /static/shoelace/assets/icons/bar-chart-line.svg
23:33:23 | 200 |       560.4µs |       127.0.0.1 | GET      | /static/shoelace/assets/icons/shield.svg
23:33:23 | 200 |       505.8µs |       127.0.0.1 | GET      | /static/shoelace/assets/icons/lock.svg
23:33:23 | 200 |      2.3747ms |       127.0.0.1 | GET      | /static/assets/image-mesh-gradient.png
23:33:23 | 200 |           0s |       127.0.0.1 | GET      | /static/shoelace/assets/icons/list.svg
23:33:23 | 200 |       631.5µs |       127.0.0.1 | GET      | /static/shoelace/assets/icons/stars.svg
23:33:24 | 200 |      4.1526ms |       127.0.0.1 | GET      | /favicon.ico
```

Рисунок 3.29 – Вигляд роботи серверу

Для зупинки серверу потрібно натиснути звичайну для усіх операційних систем та терміналів комбінацію клавіш *CTRL + C*. Після цього сервер буде зупинено з відповідним сповіщенням «Served!», яке зображено на рисунку 3.30 в якості прикладу.

Рисунок 3.30 – Припинення роботи серверу

Якщо змін не вносилося, клонований застосунок можна оновити. Для цього стануть у нагоді команди git fetch та git pull. Команда git fetch звернеться до GitHub і отримає останні зміни, які вносились у SpaceSoup. Після цього необхідно запустити команду git pull, яка завершить копіювання. Після цього необхідно заново скомпілювати код клієнта, використовуючи команду deno task compile:client.

Якщо були внесені зміни у вашій локальній версії проєкту SpaceSoup, ви можете поділитися своїми змінами з іншими розробниками, надіславши їх до віддаленого репозиторію на GitHub; для цього використовується команда git push. Перед запуском цієї команди переконайтеся, що всі зміни збережено в коміті, а гілка, в якій ведеться робота, є актуальною. Також переконайтеся у правильному налаштуванні Git щодо наявності віддаленої гілки на GitHub, яка належить вам та міститься у конфігурації Git. Редактор VSC може сильно полегшити виконання цих дій, проте вам необхідно мати Github аккаунт.

ВИСНОВКИ

У рамках кваліфікаційної роботи був розроблений вебзастосунок, який здатен розраховувати параметри руху метеороїдів в Земній атмосфері та Сонячній системі для використання з метеорними радіолокаційними системами, які реєструють метеори імпульсно-дифракційним методом [28]. Програма SpaceSoup виявилася придатною для обробки метеорних даних, отриманих у Харкові в ХНУРЕ радіолокаційним методом [29].

Під час розробки цього вебзастосунку було проаналізовано велику кількість вебтехнологій для обрання найдоцільніших, що стало основою для вибору таких мови Go та середовища Deno.

Також було проаналізовано поточний стан вебзастосунків у сфері астрономії та виявлено, що розроблюваний застосунок є актуальним і корисним для української науки. Розроблений застосунок враховує сучасні вимоги та використовує актуальні технології, що сприяє його тривалій експлуатації та зручності використання.

Після програмної реалізації також було висунуто пропозиції для його покращення, що може стати у нагоді при продовженні розробки або досліджень. Крім того, було описано функції програми для полегшення користування.

Результати роботи апробовано у вигляді тез доповідей під час Міжнародного молодіжного форуму «РАДІОЕЛЕКТРОНІКА І МОЛОДЬ У XXI СТОЛІТТІ» [30].

Роботу виконано в рамках пілотного освітньо-наукового проєкту кафедри інформатики та науково-дослідної лабораторії радіоастрономії імені Б.Л. Кащеєва (НДЛ РА) ХНУРЕ. Консультації з астрономії та розділів 2 та 3, що пов’язані з даними та результатами метеорних радіолокаційних досліджень, провів старший дослідник, к.ф.-м.н., науковий керівник НДЛ РА, науковий співробітник Астрономічного інституту Чеської Академії наук Коломієць С.В.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Пікуль, І. С. (2024). Аналіз функціоналів сучасних програмних засобів для розроблення вебзастосунків.

2. Танянський, О., & Руденко, Д. (2018). Порівняльний аналіз популярних JavaScript-фреймворків та бібліотек для front-end розробки, *Інформаційні системи та технології: матеріали статей 7-ї Міжнародної науково-технічної конференції*, 7, 347– 349.

3. Stack Overflow Developer Survey. URL: https://survey.stackoverflow.co/2024/technology (дата звернення 15.05.2025).

4. State of Developer Ecosystem (JetBrains). URL: https://www.jetbrains.com/lp/devecosystem-2024/ (дата звернення 15.05.2025).

5. GitHub Octoverse. URL: https://github.blog/news-insights/octoverse/octoverse-2024/ (дата звернення 15.05.2025).

6. Python витіснив JavaScript і став найпопулярнішою мовою програмування на GitHub. URL: https://ain.ua/2024/10/31/python-javascript-github/ (дата звернення 10.04.2025).

7. Мова програмування Golang – що це? URL: https://lemon.school/blog/mova-programuvannya-golang-shho-tse (дата звернення 10.05.2025).

8. Введення в мову Go (Golang). URL: https://w3schoolsua.github.io/hyperskill/golang-intro.html (дата звернення 10.5.2025).

9. Як і для чого вивчати Golang. Переваги і недоліки мови. URL: https://dou.ua/forums/topic/41933/ (дата звернення 08.04.2025).

10. Що таке Typescript і навіщо він потрібен. URL: https://foxminded.ua/typescript/ (дата звернення 10.5.2025).

11. Руденко Д. О. Артьомов Д. О. (2024) ОГЛЯД ПІДХОДІВ ДО СТВОРЕННЯ LANDING PAGE, *The 4th International scientific and practical conference «Scientific research: modern challenges and future prospects»*, 4, 135-138.

12. Lattner, C. (2008, May). LLVM and Clang: Next generation compiler technology. In *The BSD conference*, 5, pp. 1-20.

13. Відбувся реліз Deno 2.0 RC. Чи замінить він Node.js? URL: https://dou.ua/forums/topic/50480/ (дата звернення 10.5.2025).

14. Хостинг: Deno Deploy. URL: https://grammy.dev/uk/hosting/deno-deploy (дата звернення 10.5.2025).

15. Tailwind CSS v4.1. URL: https://tailwindcss.com/blog/tailwindcss-v4-1 (дата звернення 10.5.2025).

16. Бобейко К. С. і Кобилін О. А. (2024) ДОСЛІДЖЕННЯ МЕТОДІВ ОПТИМІЗАЦІЇ ПРОЦЕСУ РОЗРОБКИ МУЗИЧНОГО ЗАСТОСУНКУ ЗА ДОПОМОГОЮ АНАЛІЗУ ДАНИХ, *The 20th International scientific and practical conference «Trends in the development of quality training of future specialists»*, 20, 351.

17. Aladin Lite. URL: https://aladin.cds.unistra.fr/ (дата звернення 24.04.2025).

18. Horizons System. URL: https://ssd.jpl.nasa.gov/horizons/app.html#/ (дата звернення 20.04.2025).

19. Kolomiyets S. V. (2015) Uncertainties in MARS Meteor Orbit Radar Data. *Journal Atmospheric and Solar-Terrestrial Physics*, 124, 21–29.

20. TOPCAT. URL: https://www.star.bris.ac.uk/mbt/topcat/ (дата звернення 18.04.2025).

21. Kolomiyets, S., & Kundyukov, S. (2023). On the question of constructing the distribution of the flux density of meteoroids over the celestial sphere in ground-based single-position radar measurements of meteor activity and velocity: The experience of past years. *Advances in Space Research*, 72(2), 623-637.

22. Kolomiyets, S. V., Kolomiyets, K. A., Kyrychenko, I. Y., & Pryimachov, Y. D. (2020). Centenary of the birth of the famous Ukrainian researcher of radiometeors prof. BL Kashcheyev (1920-2004), his heritage and trajectory measurements today. *Одеські астрономічні публікації*, 33, 109-114.

23. Волощук Ю.І., Кащеєв Б.Л., Кручиненко В.Г. (1989) Метеори і метеорна речовина. *Наукова думка, Київ*, 292.

24. Vechirska A., Shyrokorad K. and Vechirska I. (2022) VISUAL MODELING OF PURCHASE PROCESS MANAGEMENT IN THE ELECTRONICS ONLINE STORE, *Матеріали конференцій МНЛ, (3 червня 2022 р., м. Львів)*, 189–192.

25. Tvoroshenko Iryna and Andrieieva Anastasiia (2021) DEVELOPMENT OF WEB APPLICATIONS FOR REMOTE LEARNING OF ENGLISH, *The XXVII International Science Conference «Multidisciplinary academic research and innovation»*. 27, 646-651.

26. R. Mark Volkmann. (2024). Server-Driven Web Apps with htmx. Pragmatic Bookshelf.

27. Tvoroshenko Iryna and Maksimenko Heorhii (2021) TO THE QUESTION OF ANALYSIS OF EXISTING MECHANISMS OF WEB APPLICATION TESTING, *Problems of modern science and practice*, 1, 403.

28. Kolomiyets S.V. and Kyrychenko I.Yu. (2021) On radiometeor information processing ACTUAL QUESTIONS OF GROUND-BASED OBSERVATIONAL ASTRONOMY. *International Conference. Abstract book*, 48, 26.

29. Kolomiyets S., Kyrychenko I., Mitrokhina V. (2022) What can the orbital elements and radiants of precipitated meteorites and some incoming meteoroids tell us? *44th COSPAR Scientific Assembly. Held 16-24 July, 2022. Online at https://www.cosparathens2022.org/. Abstract B1.3-0015-22*, 44.

30. Кисіль Л. (2025) SPACESOUP: ВИКОРИСТАННЯ СУЧАСНИХ ВЕБ-ТЕХНОЛОГІЙ ДЛЯ РОЗРАХУНКУ ПАРАМЕТРІВ РУХУ МЕТЕОРОЇДІВ. *Радіоелектроніка та молодь у XXI столітті. Конференція «Комп'ютерна інженерія: Сучасні технології розробки та програмування комп'ютерний систем та мереж»*, 5, 13-15.