НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“_” _____2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою “Інженерія програмного забезпечення комп’ютерних систем”
спеціальності 121 “Інженерія програмного забезпечення”

на тему: **«Сервіс для генерації фізичної мапи планети за допомогою шуму Перлина»**

Виконав: студент **4** курсу, групи **ІП-03**

Гуськов Данило Ігорович _____
(підпис)

Керівник: асистент, Міщенко Людмила Дмитрівна _____
(підпис)

Консультант (нормоконтроль) Волокита Артем Миколайович
(підпис)

Рецензент _____ _____
(підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2024 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальність 121 “Комп’ютерна інженерія”

(Освітньо-професійна програма

“Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”)

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“_” _____2024 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Гуськова Данила Ігоровича

1. Тема проєкту *Сервіс для генерації фізичної мапи планети за допомогою шуму Перлина*
керівник проєкту асистент Міщенко Людмила Дмитрівна.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 27 травня 2024 року № 2112-с
2. Термін здачі студентом закінченого проєкту *19 травня 2024 р.*
3. Вихідні дані до проєкту *технічна документація. теоретичні та статистичні дані, патенти на винахід*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)
Опис предметної області, дослідження методики побудови електронного

підручника на базі гіпертекстової технології, програма забезпечення гіпертекстових технологій

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) алгоритм генерації фізичної мапи планети (принципова схема), ER-діаграма (функціональна схема), архітектура сервісу (структурна схема).

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Волокита А. М		

7. Дата видачі завдання 23.11.2023

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*10.12.2024-15.12.2024*	
2.	*Вивчення та аналіз завдання*	*15.12.2024-15.03.2024*	
3.	*Розробка архітектури та загальної структури системи*	*15.03.2024-25.03.2024*	
4.	*Розробка структур окремих підсистем*	*25.03.2024-5.04.2024*	
5.	*Програмна реалізація системи*	*5.04.2024-15.04.2024*	
6.	*Оформлення пояснювальної записки*	*15.04.2024-20.05.2024*	
7.	*Захист програмного продукту*	*25.04.2024*	
8.	*Передзахист*	*15.06.2024*	
9.	*Захист*	*20.06.2024*	

Студент-дипломник _____ Данило ГУСЬКОВ
(підпис)

Керівник проєкту _____ Людмила МІЩЕНКО
(підпис)

Анотація

В бакалаврському дипломному проєкті реалізовано алгоритм генерації фізичної мапи планети, призначеної для створення відеоігор, та настільних ігор, космічних та кліматичних симуляторів. Генерація базується на алгоритмі шуму Перлина.

Програма дозволяє створити графічне зображення фізичної мапи вигаданої планети, клімат якої наближено схож на Землю. Для збільшення маштабованості та доступності, було обрано клієнт-серверну архітектуру на мові Node.JS у серверній частині та Javascript з фреймворком React у клієнтській частині та для візуалізації мапи.

Для зберігання вже створеної мапи було обрано базу даних PostgreSQL.

Ключові слова: процедурна генерація, фізична мапа, шум Перлина, Node.JS, PostgreSQL, Metarhia, Impress.

Annotation

The bachelor's diploma project implemented the algorithm for generating a physical map of the planet intended for the creation of video games and board games, space and climate simulators. The generation is based on Perlin's noise algorithm.

The program allows you to create a graphic image of a physical map of a fictional planet, the climate of which is approximately similar to Earth's. To increase scalability and availability, a client-server architecture was chosen, which was implemented on the Metarhia technology stack, which was developed by Timur Shemsendynov at the OT department, in the Node.JS language in the server part and Javascript with the React framework in the client part and for map visualization .

The PostgreSQL database was chosen to store the already created map.

Key words: procedural generation, physical map, Perlin noise, Node.JS, PostgreSQL, Metarhia, Impress.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпляра	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.003 ТЗ***	Сервіс для генерації	3		
			фізичної мапи планети за			
			допомогою шуму Перлина			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Сервіс для генерації	68		
			фізичної мапи планети за			
			допомогою шуму Перлина			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.003 Д1***	Сервіс для генерації	1		
			фізичної мапи планети за			
			допомогою шуму Перлина			
			Архітектура сервісу			
			(структурна схема)			
	А1	***ІАЛЦ.467200.003 Д2***	Сервіс для генерації	1		
			фізичної мапи планети за			
			допомогою шуму Перлина			
			ER-діаграма			
			(функціональна схема)			
	А1	***ІАЛЦ.467200.003 Д3***	Сервіс для генерації	1		
			фізичної мапи планети за			
			допомогою шуму Перлина			
			Алгоритм генерації			
			фізичної мапи планети			
			(принципова схема)			
	А1	***ІАЛЦ.467200.003 Д4***	Сервіс для генерації	8		
			фізичної мапи планети за			
			допомогою шуму Перлина			
			Програмний код сервісу			

						# *ІАЛЦ.467200.003 ОА*		
Зм	***Лист***	***№ докум.***	***Підп.***	***Дата***				
Розроб.		Гуськов Д. І..			Сервіс для генерації фізичної мапи планети за допомогою шуму Перлина Опис альбому	Літ.	Аркуш	Аркушів
Перев.		Міщенко Л. Д.					1	1
						КПІ, ім. Ігоря Сікорського, ФІОТ, ІП-03		
Н. Контр		Волокита А. М.						
Затвердив		Стіренко С. Г.						

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Сервіс для генерації фізичної мапи планети за допомогою шуму Перлина*»

Київ – 2024

Зм	Лист	№ докум.	Підп.	Дата	ІАЛЦ.467200.003 ТЗ		
Розроб.		Гуськов Д. І..			Сервіс для генерації фізичної мапи планети за допомогою шуму Перлина Технічне завдання	Лит.	Арк 1 / Аркушів 3
Перев.		Міщенко Л. Д.					
Реценз						КПІ, ім. Ігоря Сікорського, ФІОТ, ІП-03	
Н. Контр		Волокита А. М.					
Затвердив		Стіренко С. Г.					

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Технічне завдання поширюється на розробку сервісу для генерації фізичної мапи планети за допомогою шуму Перлина.

Областю застосування цієї системи є розробка ігрових світів та симуляцій. Також система може бути використана індивідуальними користувачами для особистої генерації фізичної мапи планети.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр інженерії програмного забезпечення», який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського»

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою дипломного проєкту є розробка сервісу, який забезпечить користувачам зручний та гнучкий інтерфейс для створення фізичної мапи планети.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література з розробки клієнт-серверних застосунків, алгоритмів генерації випадкових чисел, публікації в періодичних виданнях, технічна документація, публікації в мережі Інтернет.

5 ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до розробленого продукту

- Зрозумілий та гнучкий інтерфейс.
- Швидкість генерації

- Забезпечення безпеки та конфіденційності даних користувача.
- Опис системи, зокрема пояснювальна записка.

5.2 Вимоги до програмного забезпечення

- Будь яка ОС з підтримкою Docker
- Підключення до мережі Інтернет

5.3 Вимоги до апаратної частини

- 64-розрядний (x64) процесор із тактовою частотою 1 ГГц або швидший, із підтримкою віртуалізації.
- ROM не менше, ніж 2 ГБ.
- RAM не менше, ніж 2 ГБ.

6 ЕТАПИ РОЗРОБКИ

Етап	Дата
Затвердження теми проєкту	25.12.2023
Вивчення та аналіз завдання	23.01.2024
Проектування архітектури системи	20.03.2024
Програмна реалізація системи	30.05.2024
Тестування системи	01.06.2024
Оформлення пояснювальної записки	05.06.2024

Пояснювальна записка
до дипломного проєкту
на тему: «Сервіс для генерації фізичної мапи планети за допомогою шуму Перлина»

Київ – 2024

ЗМІСТ

СПИСОК СКОРОЧЕНЬ, ПОЯСНЕНЬ, ТЕРМІНІВ

ПЗ – програмне забезпечення

DDD – Domain-Driven Design, це підхід до моделювання складного об'єктно-орієнтованого програмного забезпечення

API – Application programming interface, прикладний програмний Інтерфейс

IDE – Integrated Development Environment, інтегроване середовище розробки.

ORM – Object-relational mapping, об'єктно-реляційне відображення.

JSON – JavaScript Object Notation, текстовий формат об’єкту.

Тайл (tile) або тайлова графіка – метод створення великих зображень у комп’ютерній графіці.

Сід (seed) – випадкове початкове значення, число або вектор, яке використовується для ініціалізації генератора псевдовипадкових чисел.

Псевдокод – неформальний запис алгоритму зі структурою поширених мов програмування.

Генератор – алгоритм (або його інтерфейс) для створення фізичної мапи планети.

Результат генерації – результат роботи генератора, тобто - фізична мапа планети.

Біом – кліматична зона, яка визначається висотою, температурою й вологістю території.

ВСТУП

Сучасні комп’ютери дозволяють обробляти все більші обсяги даних, що дає змогу створювати великі, захоплюючі світи для ігор, інтерактивних новел, фільмів або наукових симуляцій. В основі будь-якого віртуального світу лежить мапа, яка відображає необхідну інформацію про навколишнє середовище світу. У розробників є два варіанти створення мапи для свого віртуального світу: намалювати мапу вручну або вдатись до процедурної генерації.

Створені вручну карти зазвичай дуже деталізовані, проте їх створення може тривати багато часу і створені таким чином мапи будуть однакові у всіх користувачів. Крім того, як тільки продукт буде опублікований, єдиний спосіб змінити або розширити мапу – намалювати мапу заново і оновити продукт у всіх користувачів, що може бути довгим і складним процесом.

Саме тому наразі набирає популярність підхід з використанням процедурної генерації, оскільки вона дозволяє генерувати унікальні, нескінченні мапи віртуального світу без необхідності оновлювати програмний продукт.

У дипломній роботі розглянуто алгоритми та їх реалізації для процедурної генерації фізичної мапи світу. Реалізація алгоритму у вигляді сервісу дозволить користувачам створювати, налаштовувати і зберігати свої унікальні мапи.

РОЗДІЛ 1. ОГЛЯД СЕРВІСІВ ГЕНЕРАЦІЇ ФІЗИЧНОЇ МАПИ

1.1 Загальні положення

Мапа світу є важливою частиною для будь якої гри, книги, або віртуальної симуляції, оскільки надає необхідну інформацію про навколишнє середовище. Процедурна генерація мапи - складний процес який можна реалізувати по різному для кожної окремої ситуації. Генерація мапи потребує знання про навколишнє середовище для того аби її побудувати.

Основні напрями розробки для сервісу генерації фізичної мапи можуть включати наступні аспекти:

1. Система облікових записів користувачів: реалізація системи авторизації, аутентифікації й безпечного зберігання облікових даних користувачів.

2. Розробка інтерфейсу користувача: створення зручного та гнучкого інтерфейсу, що дозволить користувачам створювати свої унікальні фізичні мапи.

3. Функції керування: можливість експортувати, імпортувати карти та переглядати список створених карт.

4. Функції перегляду: вивід зображення мапи використовуючи проекцію, можливість приблизити або віддалити зображення мапи, дізнатись більше про її окрему ділянку, або властивості всієї мапи.

5. Алгоритм генерації: швидкий і гнучкий алгоритм для створення фізичної мапи за вказаними параметрами.

1.2 Змістовний опис та аналіз предметної області

Процедурні генератори віртуальних світів є ключовою технологією, яка знаходить застосування в багатьох сучасних комп'ютерних іграх, наукових симуляціях та інших інтерактивних застосунках. Вони дозволяють створювати великі, складні і детальні світи без потреби ручного втручання на кожному етапі розробки. Цей підхід не тільки знижує витрати на розробку, але й забезпечує високу варіативність та унікальність кожного створеного світу.

Процедурна генерація включає в себе набір алгоритмів і методів, які автоматично створюють контент на основі визначених правил і параметрів. Це може бути створення ландшафтів, екосистем, будівель, міст і навіть персонажів.

Одним з найпопулярніших методів процедурної генерації є використання генераторів шуму, такого як шум Перлина, який дозволяє створювати імітації природних ландшафтів.

Процедурні генератори дозволяють розробникам зосередитися на загальній структурі ігрового світу, задаючи основні параметри і правила, за якими алгоритм буде генерувати деталі. Це значно знижує обсяги ручної роботи і дозволяє створювати великі світи з високою деталізацією.

Крім того, процедурна генерація забезпечує унікальність кожного нового світу, що є важливим фактором для гравців, які шукають нові враження кожного разу, коли вони запускають гру.

Однак, незважаючи на всі переваги, процедурна генерація має і свої виклики. Один з головних – це забезпечення цілісності ігрового світу, оскільки автоматично згенеровані світи можуть бути менш логічними або структурованими порівняно з тими, що створені вручну. Також важливо забезпечити баланс між випадковістю і передбачуваністю, щоб створений світ був цікавим, взаємозалежним і реалістичним.

Отже, дослідження та аналіз проблематики та поточного стану розвитку IT-технологій у контексті процедурної генерації віртуальних світів показали, що такий підхід активно використовується при розробці комп'ютерних ігор з відкритим, часто нескінченним світом. Способи реалізації поділяються на дві категорії: генерація на стороні клієнта або сервера. Алгоритми генерації світу залежать від формату гри, вимог до продуктивності та бажаного рівня деталізації.

На стороні клієнта генерація дозволяє розподілити обчислювальне навантаження між сервером і клієнтськими машинами, що може значно зменшити затримки та підвищити інтерактивність. З іншого боку, генерація на стороні сервера забезпечує більш контрольоване середовище, де всі дані зберігаються централізовано, що полегшує управління та забезпечення консистентності світу для всіх гравців.

Дослідження також виявило, що для успішної реалізації процедурної генерації важливо враховувати не лише технічні аспекти, але й потреби та очікування користувачів.

Користувачі очікують, що створенні віртуальні світи будуть не тільки масштабними, але й візуально приємними, реалістичними, та насиченими.

1.3 Аналіз існуючих технологій

Для розробки сервісу для генерації фізичної мапи планети необхідно виконати аналіз уже існуючих рішень.

Аналіз існуючих рішень допоможе визначити, які інструменти та технології найкраще підходять для реалізації проєкту, а також виявити їхні сильні та слабкі сторони.

1.3.1 Аналіз допоміжних програмних засобів та засобів розробки

В основі серверної частини дипломного проєкту буде технічний стек Metarhia, використовуючи Node.Js - середовище виконання JavaScript. Використання технічного стеку Metarhia дозволить зручно взаємодіяти з іншими частинами системи та її залежностями і в результаті побудувати високоефективний програмний продукт який можна буде легко масштабувати.

Клієнтська частина сервісу буде виконана за допомогою React.Js. Це поширена бібліотека для веб-інтерфейсу інтерфейсу на JavaScript.

Інші засоби розробки, що використовуватимуться у дипломному проєкті:

1. Популярне середовище розробки Visual Studio Code для написання коду.
2. Інтерфейс для системи контролю версій GitHub Desktop, що дозволяє зручно відслідковувати зміни в коді та відновлювати попередні версії кодової бази.
3. Docker для легкого переносу і запуску проєкту на будь якій системі, яка підтримує віртуалізацію.
4. База даних PostgreSQL, яка дозволить швидко маштабувати роботу зі сховищем даних за необхідності маштабувати проєкт.

1.3.2 Аналіз існуючих технологій та IT-рішень

Уже існують проекти, які реалізують функціонал описаний у технічному завданні, проте кожен із них має свої переваги та недоліки ,також можливості для покращення розроблюваного програмного продукту.

Один з сервісів генерації віртуальних світів - Mewo [10]. Він користується популярністю серед фанатів культової настільної гри Dungeons and Dragons.

Переваги сервісу Mewo:

1. Безкоштовний доступ і відкритий код, що робить сервіс доступним і прозорим для будь якого користувача який має доступ до мережі Інтернет.
2. Повчальна складова (сервіс не тільки генерує мапу світу, а також показує поступово як це відбувається і як це працює).
3. Висока швидкість генерації мапи.

Недоліки сервісу Mewo:

1. Відсутність обліку користувачів робить неможливим зробити збереження чи завантаження результату генерації, чи переглянути минулі згенеровані карти.
2. Сервіс генерує лише частину мапи планети, але не всю планету.
3. Результат генерації світу лише у чорно-білому кольорі.
4. Відсутність вбудованого функціоналу роботи із згенерованою мапою (який би дозволяв наприклад її розмалювати, відредагувати назви територій, тощо).
5. Неочевидний інтерфейс, через покрокову генерацію яка, хоч і дозволяє узнати як саме працює сервіс, але трошки заплутує користувача який хоче одразу перейти до генерації мапи (наведений на рис. 1.1)

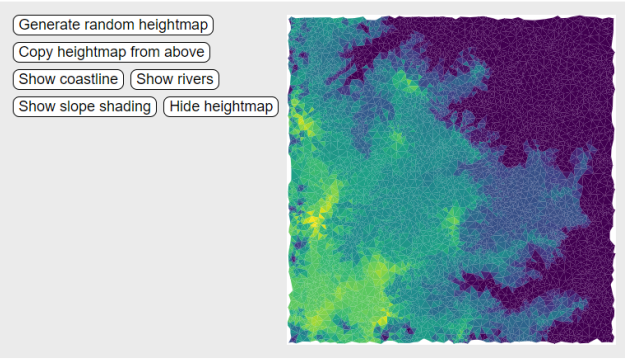


Рисунок 1.1 - Інтерфейс користувача сервісу Mewo

6. Низька деталізація мапи (відсутність кліматичних зон, озер і найкраща передача висот і низин ландшафту)

Ще один популярний сервіс в області генерації мапи віртуальних світів - Donjon [11], пропонує інший підхід до інтерфейсу користувача і генератора.

Переваги сервісу Donjon:

1. Безкоштовний доступ.
2. Параметризована генерація (яка не перевантажена над складними компонентами і дозволяє сервісу кастомізувати вихідну мапу під запити кожного користувача)
3. Інтуїтивно зрозумілий і зручний інтерфейс користувача (рис 1.2)

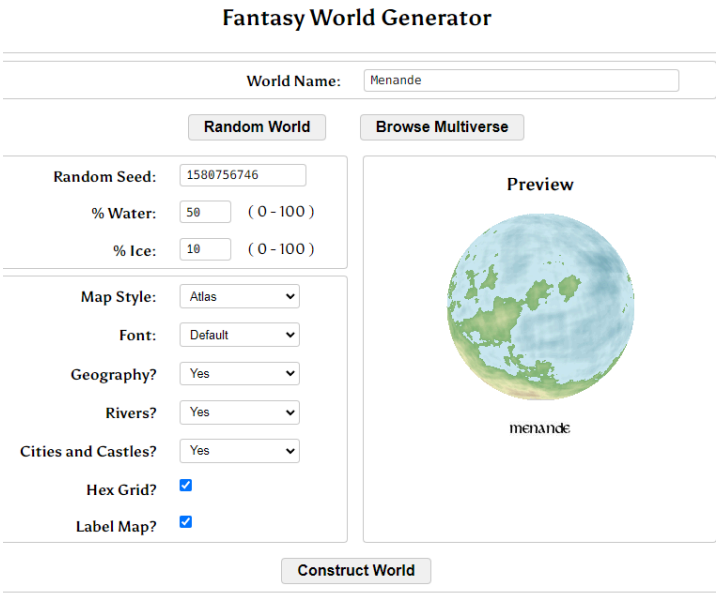


Рисунок 1.2 - Інтерфейс користувача сервісу Donjon

4. Генерація мапи цілої планети, відображена в проекції на площину зацикленої по осям координат.
5. Висока деталізація клімату і згенерованої місцевості (можна побачити дуже реалістичні кліматичні пояси, умовні позначки лісів, тощо).

6. Сервіс дозволяє генерувати дуже великі світи.
7. Дуже приємні кольори мапи і гарно підібрані шрифти для умовних позначок і текстів на мапі.

Недоліки сервісу Donjon:

1. Дуже довгий процес генерації (який збільшується відповідно до запиту збільшення розмірів мапи)
2. Відсутність інструментів для редагування згенерованої мапи.

Аналіз сучасних технологій та ІТ-рішень показав, що існуючі програмні продукти використовують різні підходи до реалізації схожого функціоналу. Наприклад, деякі проєкти зосереджуються на високій деталізації та реалістичності, використовуючи потужні алгоритми генерації, такі як шум Перлина або фрактальні методи. Інші ж обирають спрощені підходи для швидшого виконання та меншого навантаження на систему.

Таким чином, аналіз існуючих технологій та ІТ-рішень виявив переваги та недоліки існуючих програмних продуктів, що дозволить створити більш ефективний, гнучкий та зручний сервіс для користувачів.

1.4 Аналіз вимог до програмного забезпечення

Розглянуто основні функції сервісу для генерації фізичної мапи планети за допомогою шуму Перлина. Цей функціонал визначає ключові можливості, що надаються користувачам, і є важливим для розуміння вимог до сервісу. Крім того, детальний аналіз функціоналу допоможе виявити потенційні покращення та оптимізації, які можна впровадити для підвищення ефективності та зручності використання сервісу:

1. Оптимізація алгоритму генерації.
2. Оптимізація рендеру мапи в інтерфейсі користувача.

3. Можливість генерації великої мапи
4. Висока деталізація мапи
5. Параметризація генератора.
6. Інтуїтивно зрозумілий та простий інтерфейс.
7. Інструменти перегляду згенерованої мапи.
8. Інструменти редагування згенерованої мапи.

Наведений функціонал демонструє ключові можливості сервісу для генерації мапи світу.

В наступних підрозділах порівняно його з функціоналом існуючих рішень та визначено переваги та недоліки розробленого програмного продукту.

1.5 Розроблення вимог до програмного забезпечення

1.5.1 Розроблення функціональних вимог до програмного забезпечення

Для забезпечення функціонування сервісу, та надання користувачам якісних послуг з генерації фізичної мапи планети за допомогою шуму Перлина, було розроблено функціональні вимоги, які наведено у таблиці 1.1.

Таблиця 1.1 – Функціональні вимоги

Назва	Опис
Генерація мапи	Сервіс повинен мати надійний алгоритм генерації мапи
Збереження мапи	Сервіс повинен мати можливість зберегти згенеровану мапу для поточного користувача
Завантаження мапи	Сервіс повинен мати можливість завантажити збережену мапу поточного користувача
Відображення проекції мапи	Сервіс повинен відобразити згенеровану мапу в інтерфейсі користувача використовуючи проекцію мапи на площину
Деталізація	Сервіс повинен генерувати і відображати високо деталізовану мапу з кліматичними зонами, температурою, рельєфом
Маштабованість	Сервіс має бути здатним працювати зі зростаючим обсягом користувачів, паралельно генерувати мапи для декількох запитів, зберігати зростаючі обсяги даних відповідно

1.5.2 Розроблення нефункціональних вимог до програмного забезпечення

Нефункціональні вимоги встановлюють параметри продукту, які не стосуються безпосередньо його функціональності, але є надзвичайно важливими для забезпечення надійності та масштабованості. Нефункціональні вимоги сервісу для генерації фізичної мапи планети наведено у таблиці 1.2.

Таблиця 1.2 – Нефункціональні вимоги сервісу.

Назва	Опис
Оптимізація алгоритму генерації	Сервіс повинен мати швидкий алгоритм генерації мапи
Оптимізація відображення мапи	Сервіс має максимально ефективно використовувати ресурси та мережеву пропускну здатність, досягаючи швидкого відображення згенерованої мапи в інтерфейсі користувача
Безпека даних користувача	Сервіс має забезпечити надійність зберігання даних користувача
Маштабованість	Сервіс повинен мати закладений механізм розподілення навантаження на декілька серверів
Переносимість запуску	Сервіс повинен мати переносиму процедуру запуску на персональному комп’ютері, сервері чи хостингу з підтримкою розповсюджених серверних систем та архітектур
Підтримка Code of Conduct	Програмний код сервісу повинен притримуватись принципів написання зрозумілого коду

1.6 Аналіз мети дипломного проєкту

На основі аналізу існуючих рішень визначено, що метою дипломного проєкту є розробка сервісу для генерації фізичної мапи планети за допомогою шуму Перлина. Ціллю дипломного проєкту є надання користувачу можливість згенерувати фізичну мапу планети за запитом та відобразити її в інтерфейсі користувача.

Для реалізації поставленої в дипломному проєкту мети необхідно виконати наступні завдання::

1. Забезпечити масштабованість й читаємість програмного коду задля забезпечення розвитку, можливості прикладного застосування і збільшення кількості користувачів програми в майбутньому.
2. Максимально деталізувати згенеровану мапу.
3. Наблизити згенеровану мапу до реалістичної альтернативної версії мапи Землі.
4. Забезпечити безпеку даних користувача.
5. Забезпечити зручний і гнучкий інтерфейс генератора.
6. Зробити мапу інтерактивною для редагування згенерованого результату.

Поставлені завдання для розробки сервісу для генерації фізичної мапи за допомогою шуму Перлина потенційно спричиняють наступні виклики, що підлягають вирішенню при розробці програмного продукту:

1. Імплементація алгоритму генерації мапи планети.

Сервіс має використати шум Перлина за основу побудови ландшафту планети, але для більшої деталізації згенерований шум має піддаватися унікальній обробці.

Ця обробка має включати в себе розширення алгоритму генерації з додаванням накладання фільтрів, згладжування, та має використовувати параметри генерації надані користувачем через інтерфейс.

2. Збільшення реалістичності й деталізації згенерованої мапи.

Сервіс повинен генерувати температуру, вологість, річки, озера, та інші геологічні форми рельєфу та, відштовхуючись від всіх параметрів, обирати кліматичні зони кожної клітини мапи. Це збільшить реалістичність за додасть унікальності кожній мапі.

3. Розробка інтуїтивно зрозумілого та гнучкого інтерфейсу користувача.

Сервіс повинен не тільки мати простий інтерфейс для генерації мапи, а також потрібно створити наочне відображення результату генерації, всіх параметрів мапи й конкретної її частини.

4. Оптимізація відображення й генерації мапи.

Сервіс повинен швидко реагувати на інтерактивні дії користувача над мапою, максимально швидко генерувати, завантажувати, та зберігати результати.

5. Забезпечення тестування програмного продукту для переконання у його якості, надійності та відповідності всім вимогам.
6. Написання документації, опису системи, зокрема пояснювальної записки та вказівок щодо її використання.

ВИСНОВКИ ДО РОЗДІЛУ 1

У першому розділі було проведено детальне дослідження та аналіз предметної області сервісу для генерації фізичної мапи планети за допомогою шуму Перлина. Розглянуто основні аспекти, пов'язані з генерацією мапи, визначено основні потреби користувачів, що дозволяє отримати глибше розуміння проблематики та вимог до розроблюваної системи.

Аналіз наявних програмних засобів та інструментів розробки дозволив виявити існуючі рішення, які можуть бути використані під час створення сервісу. Це сприяло вибору технологій для реалізації проєкту.

На основі проведеного аналізу існуючих програмних сервісів, таких як Donjon та Mewo, розроблюваний сервіс відрізнятиметься унікальною імплементацією алгоритму генерації та матиме перевагу в додатковому функціоналі.

У аналізі вимог до програмного забезпечення охоплено як функціональні, так і нефункціональні аспекти. Сформульовано списки функцій, які повинна виконувати система, а також вимоги до продуктивності, безпеки та масштабованості.

Опрацьовано та описано конкретні завдання, які включають реалізація сервісу для генерації фізичної мапи планети за допомогою шуму Перлина. Визначено цілі розробки та сформульовані задачі, які повинні бути вирішені в результаті створення програмного забезпечення.

Таким чином, перший розділ слугує основою для подальшого проєктування та реалізації розроблюваного сервісу для генерації фізичної мапи планети за допомогою шуму Перлина.

РОЗДІЛ 2. АРХІТЕКТУРА ТА ТЕХНОЛОГІЇ РОЗРОБКИ СЕРВІСУ ГЕНЕРАЦІЇ ФІЗИЧНОЇ МАПИ ПЛАНЕТИ ЗА ДОПОМОГОЮ ШУМУ ПЕРЛИНА

2.1 Архітектура сервісу

Архітектура сервісу базується на клієнт-серверній моделі, де веб-сайт виступає в якості клієнтської частини, а сервер забезпечує генерацію фізичної мапи та функціональність веб-сайту. Архітектура складається з наступних компонентів:

1. Веб-сайт надає зручний та інтуїтивно зрозумілий інтерфейс користувача, який дозволяє побачити результат генерації та працювати з ним.
2. Вбудований балансер мережевого навантаження технологічного стеку Metarhia розподіляє навантаження між двох екземплярів серверу.
3. Два екземпляри веб-серверу виконують основну функції генерації фізичної мапи за запитом користувача, обробляють функціонал, та виступають
4. Безпека сервісу

   Задля забезпечення безпеки даних користувачів, та цілісної й безперебійної роботи сервісу кожний компонент сервісу загорнутий в ізольованому віртуальному середовищі - Docker контейнері.

Структурна схема архітектури сервісу наведена у Додатку 1.

2.2 Вибір веб-серверу

Impress [6] перший веб-сервер на Node.js, масштабований за допомогою багатопоточності та надзвичайно тонкої ізоляції робочого навантаження. Оптимізований для високоінтенсивного обміну даними, швидкого розвитку та чистої архітектури. Забезпечує все необхідне для надійного та ефективного функціонування серверної частини, мережевого зв’язку з мобільними і веб-клієнтами, протокольно-агностичного API, перевірки типу під час виконання, обробки даних у режимі реального часу та в пам’яті, а також надійних служб із збереженням стану.

Недоліки Impress: поганий вибір для публікації контенту, включаючи блоги та онлайн-магазини, рендеринг на стороні сервера, надання статичного вмісту та служби без збереження даних.

Переваги Impress: безпека та архітектура для додатків корпоративного рівня, довгострокові з’єднання через websocket для мінімізації витрат на криптографічні рукостискання, відсутність сторонніх залежностей.

Отже, Impress надає готовий веб-сервер з обробленням реєстрації та авторизації користувачів, імпорту залежностей, логування та структури файлів серверу, що надає можливість зосередитись виключно на логіці сервісу. Безпека, масштабованість, та продуктивність цілком задовольняють вимогам, а недоліки цього інструменту не завадять реалізації сервісу.

2.3 Контейнеризація компонентів сервісу

Використання Docker є необов’язковим інструментом для впровадження механізму контейнеризації компонентів сервісу, і його доцільність залежить від конкретних умов та потреб проєкту.

Застосування Docker потребує обґрунтування, тому було розглянуто доцільність використання цього інструменту в дипломному проєкті.

Docker [7] є інструментом, який дозволяє розробникам створювати, розгортати та запускати компоненти сервісу в контейнерах. Контейнери дозволяють ізолювати компоненти та їх залежності в окремих середовищах, забезпечуючи консистентність і портативність незалежно від середовища виконання. Використання Docker має кілька ключових переваг:

1. Портативність: Контейнери Docker можуть працювати на будь-якому середовищі, яке підтримує Docker, що дозволяє легко переносити додатки між різними системами (розробка, тестування, продуктивні середовища) без змін у коді.
2. Ізоляція: Кожен контейнер працює незалежно, ізолюючи додаток і його залежності від інших контейнерів. Це дозволяє уникнути конфліктів між залежностями та забезпечує стабільність роботи компонентів сервісу.
3. Швидке розгортання та масштабування: Контейнери запускаються значно швидше, ніж віртуальні машини, оскільки вони не потребують завантаження гостьової операційної системи. Це дозволяє швидко масштабувати компоненти сервісу в залежності від навантаження.
4. Консистентність середовищ: Використання Docker забезпечує однакове середовище для розробки, тестування та продуктивної експлуатації. Це допомагає уникнути проблем, пов'язаних з різницею в налаштуваннях середовищ.
5. Спрощене управління залежностями: Docker дозволяє легко визначати і управляти залежностями додатка, включаючи бібліотеки, інструменти та інші сервіси, які необхідні для його роботи.

6. Ефективне використання ресурсів: Контейнери використовують ресурси системи ефективніше, ніж віртуальні машини, оскільки вони розділяють ядро операційної системи, що знижує витрати на ресурси.
7. Інтеграція з CI/CD: Docker добре інтегрується з системами безперервної інтеграції та безперервного розгортання (CI/CD), що дозволяє автоматизувати процеси тестування та розгортання компонентів сервісу.

Недоліки Docker:

1. Перевантаження ресурсів: Хоча Docker ефективніший за віртуальні машини, використання великої кількості контейнерів може призвести до значного навантаження на ресурси системи, особливо на пам'ять і процесор.
2. Безпека: Docker контейнери працюють на одному ядрі операційної системи, що може створювати потенційні ризики безпеки. Уразливості в ядрі можуть вплинути на всі контейнери. Також Docker за замовчуванням працює з привілейованим доступом, що може створити додаткові ризики.
3. Складність відлагодження: Відлагодження компонентів сервісу у контейнерах складніше порівняно з традиційними середовищами, особливо для нових розробників і адміністраторів.

Загалом, використання Docker значно спрощує розробку, тестування, розгортання та масштабування сервісу, забезпечуючи при цьому стабільність, консистентність та ефективність, що задовольняє вимогам проєкту, а його недоліки не мають вплинути на роботу та розробку сервісу. Отже, використання контейнеризації за допомогою Docker при розробці проєкту є цілком обґрунтованим і доцільним.

2.4 База даних

Згідно з обраними функціональними вимогами, сервіс повинен зберігати дані: результати генерації та інформацію про користувачів.

2.4.1 Вибір бази даних

З огляду на акцент на масштабованість та гнучкість сервісу, а також очікування роботи з великими об’ємами даних результатів генерації, було прийнято рішення використати реляційну базу даних, а саме PostreSQL з наступних причин:

PostgreSQL [8] – це потужна, відкрита об'єктно-реляційна система управління базами даних (СУБД), яка має понад 30 років активної розробки. Вона є одним з найпопулярніших виборів для різноманітних програмних застосунків завдяки своїй надійності, масштабованості та багатому набору функцій.

Переваги PostgreSQL:

1. Відкритий вихідний код: PostgreSQL є безкоштовною та відкритою для використання, що дозволяє її використовувати без ліцензійних витрат.
2. Підтримка стандартів SQL: PostgreSQL повністю відповідає стандартам SQL, забезпечуючи сумісність з багатьма іншими базами даних.
3. Розширюваність: Можливість додавання нових типів даних, операторів, агрегатних функцій, індексів та інших компонентів без необхідності змінювати основний код.

4. Передові можливості: Підтримка транзакцій, відновлення після збоїв, багатоверсійна обробка паралельних запитів (MVCC), обмеження цілісності даних, а також підтримка складних запитів та індексів.

5. Надійність і стабільність: PostgreSQL відома своєю надійністю та стабільністю.

6. Масштабованість: Підтримка великих обсягів даних та здатність ефективно працювати як на малих, так і на великих системах.

7. Підтримка геопросторових даних: Розширення PostGIS дозволяє працювати з геопросторовими даними, що є корисним для проєктів, пов’язаних з геолокацією та картографією.

Отже, PostgreSQL - надійна, продуктивна та гнучка система управління базами даних, яка яка здатна задовольнити вимоги розроблюваного сервісу.

2.4.2 Опис структури бази даних

У цьому підрозділі розглянуто основні таблиці, використані при розробці сервісу для генерації фізичної мапи планети. Для повноцінної роботи системи знадобиться зберігати інформацію про користувачів, а також результати генерації карт. Також було виконано опціональну основу рольової моделі користувачів для розподілу привілеїв серед користувачів системі. Така модель допоможе виокремити роль адміністратора сервіса, та зробить імплементацію B2C (Business-to-consumer) моделі в сервісі простішою в майбутньому.

Структура таблиці користувачів показана у таблиці 2.1, відображаючи її поля, характеристики та опис.

Таблиця 2.1 – Структура таблиці користувачів

Назва стовпця	Тип даних	Опис
id	bigint	Унікальний ідентифікатор користувача в сервісі
login	varchar(64)	Унікальне ім’я користувача в рамках сервісу
password	varchar	Хеш пароля користувача в сервісі
isBlocked	boolean	Зарезервований флаг для блокування користувача в сервісі

Таблиця ролей буде співставляти користувача з його роллю в сервісі (її деталі наведено в таблиці 2.2).

Таблиця 2.2 – Структура таблиці ролей користувачів

Назва стовпця	Тип даних	Опис
id	bigint	Унікальний ідентифікатор ролі в сервісі
name	varchar	Унікальна назва ролі в рамках сервісу

Таблиця сессій сервісу зберігатиме інформацію щодо поточних підключень користувачів до сервісу (наведена в таблиці 2.3).

Таблиця 2.3 – Структура таблиці поточних сесій користувачів.

Назва стовпця	Тип даних	Опис
id	bigint	Унікальний ідентифікатор сесії
accountId	bigint	Ідентифікатор користувача поточної сесії
token	varchar	Токен сесії
ip	inet	IP адреса користувача сесії
data	jsonb	Опціональні дані сесії

Інформація щодо результатів генерації буде зберігатися в наступному форматі наведеному в таблиці 2.4.

Таблиця 2.4 – Структура таблиці збережених результатів генерації фізичних мап.

Назва стовпця	Тип даних	Опис
id	bigint	Унікальний ідентифікатор результату генерації
name	varchar	Назва мапи
height	integer	Висота сгенерованої мапи в тайлах (tile)
width	integer	Ширина сгенерованої мапи в тайлах (tile)

tiles	jsonb	Дані за якими можна відтворити збережену мапу
generatorConfig	jsonb	Параметри за якими було згенеровано мапу
generationTime	integer	Час за який було згенеровано мапу в мілісекундах
createdAt	timestamp	Час (в форматі UTC) коли було створено запис в таблиці

Така структура бази даних дозволяє виконати всі наявні вимоги до сервісу. Крім того, вона забезпечує ефективну обробку даних, високу продуктивність, надійність та можливість масштабування в майбутньому.

2.5 Застосування об'єктно-реляційного відображення

Об'єктно-реляційне відображення (ORM) – це технологія, що дозволяє взаємодіяти з базами даних за допомогою об'єктно-орієнтованого підходу. Використання ORM є необов’язковим компонентом сервісу, і його доцільність залежить від конкретних умов та потреб проєкту. Застосування ORM може бути виправдане для спрощення роботи з базою даних, прискорення розробки та зменшення кількості помилок у коді, проте потребує ретельного обґрунтування, зважаючи на можливі обмеження та вплив на продуктивність.

2.5.1 Визначення доцільності застосування об'єктно-реляційного відображення

У цьому підрозділі розглянуто аргументи, які впливають на рішення щодо використання ORM:

1. Полегшення роботи з базою даних та оптимізація запитів: ORM сприяє спрощенню взаємодії з базою даних та забезпечує оптимізацію запитів і доступу до даних. Інструменти ORM автоматично генерують SQL-запити, оптимізовані для конкретної бази даних, що може підвищити продуктивність сервісу та забезпечити швидкий доступ до даних.
2. Зручність: ORM надає зручний інтерфейс для роботи з базою даних, дозволяючи розробникам використовувати об'єктно-орієнтований підхід замість написання складних SQL-запитів. Це спрощує процес розробки, скорочує час на розробку та підтримку системи.
3. Підтримка різних баз даних та масштабування: ORM-інструменти часто підтримують різні бази даних і надають можливості масштабування. Вони дозволяють змінювати базу даних без необхідності значних змін у вихідному коді системи, що може бути корисним при потребі змінити базу даних у майбутньому або масштабувати систему для високих навантажень.
4. Знайомий підхід для розробників: ORM дозволяє розробникам працювати з базою даних, використовуючи знайомий об'єктно-орієнтований підхід. Це полегшує процес розробки, розуміння коду та підтримку системи.

Незважаючи на можливі недоліки використання ORM, такі як додатковий шар абстракції, можливість втрати продуктивності при складних запитах або несумісність з деякими особливостями конкретної бази даних, ці недоліки не становлять загрози для імплементації вимог до програмного забезпечення. Таким чином, використання ORM при розробці системи є цілком обґрунтованим і доцільним.

2.5.2 Вибір інструменту об'єктно-реляційного відображення

Технологічний стек Metarhia має вбудовану ORM - Metasql, яка оптимізована під використання разом з веб-сервером Impress та базою даних PostgreSQL обраних для реалізації програмного продукту, що майже нівелює можливі недоліки використання ORM, хоча створює додатковий шар абстракції [9]. Metasql дозволяє зручно та ефективно виконувати операції з базою даних, такі як створення таблиць, вставка, оновлення та видалення даних.

Нижче наведені основні аргументи на користь використання Metasql:

1. Підтримка PostgreSQL.

Metasql має вбудовану підтримку бази даних PostgreSQL та веб-серверу Impress, що гарантує продуктивність опрацювання запитів до бази даних і уніфікованість розробки.

2. Простота використання

Metasql надає звичний та інтуїтивно зрозумілий API для взаємодії з базою даних. Він дозволяє зручно створювати SQL запити та асинхронно виконувати їх за допомогою механіки мови програмування Node.Js - промісу (Promise). Metasql має вже створені функції для CRUD (акронім Create, Read, Update, Delete) у вигляді роботи з об’єктами та методами які представляють базу даних.

3. Легкість конфігурації

Оскільки Metasql має вбудовану підтримку інструментів розробки використовуємих в проєкті, його конфігурація під вимоги проєкту не потребує багато часу, а при необхідності може бути легко змінена без необхідності переписувати код.

4. Безпека

Metasql має механізм захисту від SQL-ін’єкцій, який додає безпеки при комунікації серверу з базою даних.

Отже, з огляду на цілі та вимоги дипломного проєкту, Metasql є потужним інструментом для роботи з базою даних. Він забезпечує зручну підтримку, написання SQL запитів та їх ефективне виконання.

ВИСНОВКИ ДО РОЗДІЛУ 2

У другому розділі дипломного проєкту було проведено детальний аналіз архітектури та технологій, які необхідні для розробки сервісу генерації фізичної мапи за допомогою шуму Перлина. Обрано веб-сервер, інструмент для контейнеризації компонентів сервісу, базу даних та ORM, і наведено основні переваги та особливості цих технологій з урахуванням вимог до програмного забезпечення. Було описано загальну структуру бази даних відповідно до вимог та функціональності системи.

Визначено оптимальні архітектурні рішення та вибрати технології, які найкраще відповідають потребам сервісу генерації фізичної мапи планети за допомогою шуму Перлина. Це дозволило ретельно проаналізувати потреби проєкту та обрати найефективніші варіанти для досягнення мети. На основі теоретичного аналізу потреб користувачів обрано архітектуру бази даних та використання ORM.

Теоретично обґрунтовано, що обрані архітектурні рішення та технології відповідають вимогам сервісу генерації фізичної мапи планети за допомогою шуму Перлина і дозволяють реалізувати необхідний функціонал, задовольняючи потреби користувачів.

РОЗДІЛ 3. РОЗРОБКА СЕРВІСУ ДЛЯ ГЕНЕРАЦІЇ ФІЗИЧНОЇ МАПИ ПЛАНЕТИ ЗА ДОПОМОГОЮ ШУМУ ПЕРЛИНА

У третьомуцьому підрозділі розглянуто розробку інтерфейсу користувача (веб-сайту), основною метою якого є забезпечення користувача сервісу гнучким та зручним інтерфейсом для надання можливості генерації, перегляду та редагування фізичної мапи планети. Для досягнення мети було використано фреймворк Javascript - React.JS [12].

3.1 Розробка інтерфейсу користувача

Для того, щоб користувач міг використати функціонал сервісу, йому спершу потрібно авторизуватися використовуючи свій логін та пароль. Інтерфейс сторінки авторизації наведений на рисунку 3.1.

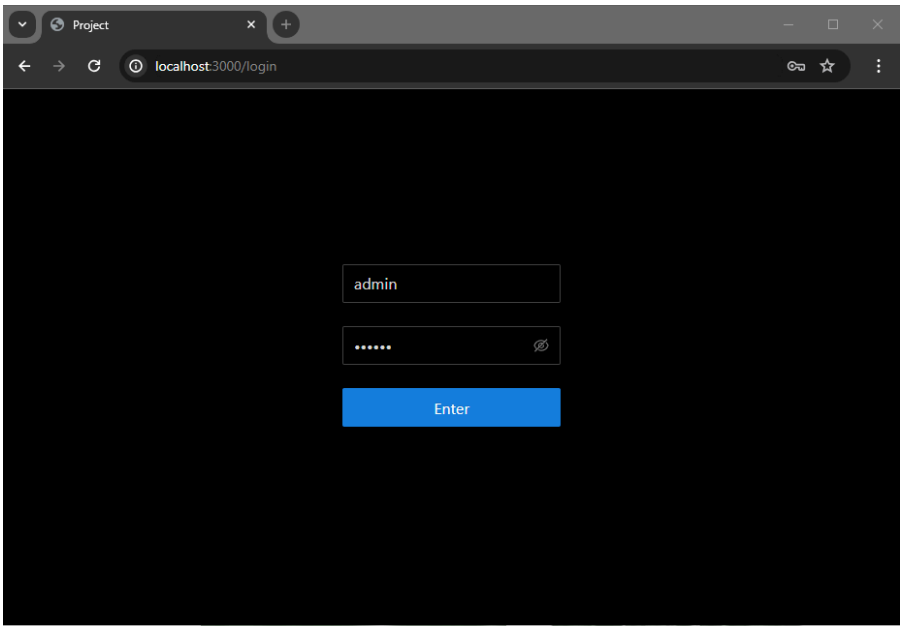


Рисунок 3.1 - Сторінка авторизації сервісу

В лівій частині інтерфейсу буде знаходитись панель з навігацією по сервісу (наведено на рис 3.2)

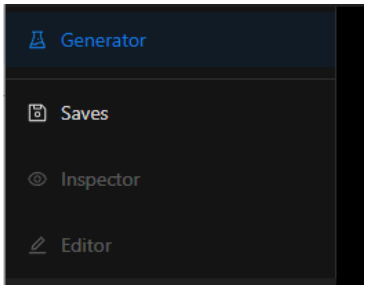


Рисунок 3.2 - Панель з навігацією по сервісу

3.1.1 Модуль генерації фізичної мапи планети

Для розробки інтерфейсу для створення фізичної мапи, необхідно, щоб розроблювальний модуль містив поля для введення всіх необхідних параметрів для генератора, при цьому не перевантажуючи нового користувача завеликою кількістю елементів керування, але надаючи велику гнучкість, корисну для досвідченого користувача. Згенеровані мапи повинні автоматично зберігтись на сервері.

Зважаючі на вимоги цей модуль було розділено на частину з основними параметрами та поглибленими. Вигляд модулю наведено на рисунку 3.3, а на рисунку 3.4 зображено розгорнуті параметри генератора.

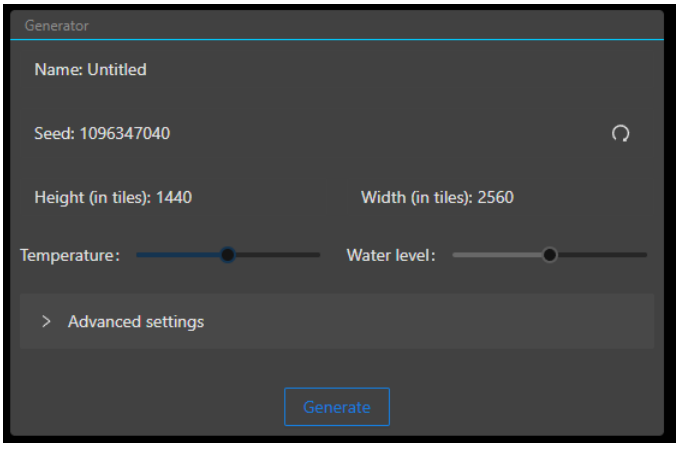


Рисунок 3.3 - Інтерфейс для генератора

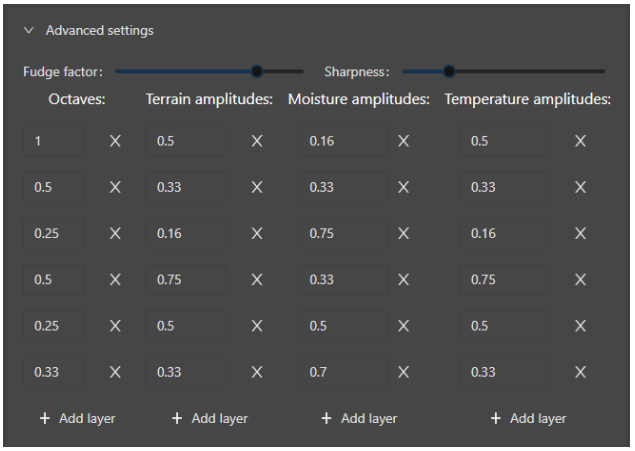


Рисунок 3.4 - Розгорнутий інтерфейс додаткових параметрів генератора

Опис інтерфейсу:

1. Name - поле вводу ім’я для мапи.
2. Seed - випадкова символьно-числова послідовність, яка використовується алгоритмом генерації. Має кнопку справа, для обрання нового, випадкового зерна.
3. Height та Width - відповідно поля для вводу висоти та ширини мапи вимірюванні в тайлах.
4. Temperature - регулює середню температуру на планеті, мапу якої буде створено.
5. Water level - регулює рівень води на мапі яку буде створено.
6. Advanced settings - набір параметрів (згорнутих за замовчуванням) для гнучкого налаштування генератора. Більш детально ці параметри описані в підрозділах 3.3, 3.4 та 3.5 дипломного проєкту.

Важливо зазначити, що для всіх параметрів є значення за замовчуванням, отже користувачу не обов’язково підбирати ці значення самому, він може використати всі значення за замовчуванням.

Отже, модуль надає користувачам гнучкий та зрозумілий інтерфейс для створення фізичної мапи планети, що задовольняє вимогам сервісу.

3.1.2 Модуль збережених результатів генерації

Завдання цього модулю - надати користувачу функціонал роботи з попередніми результатами генерації. Для цього модуль відображає таблицю з останніми згенерованими мапами, мінімальний їх опис, ім’я та кнопки для керування (наведено на рис 3.5). Також модуль надає можливість завантажити мапу збережену локально, у форматі JSON.

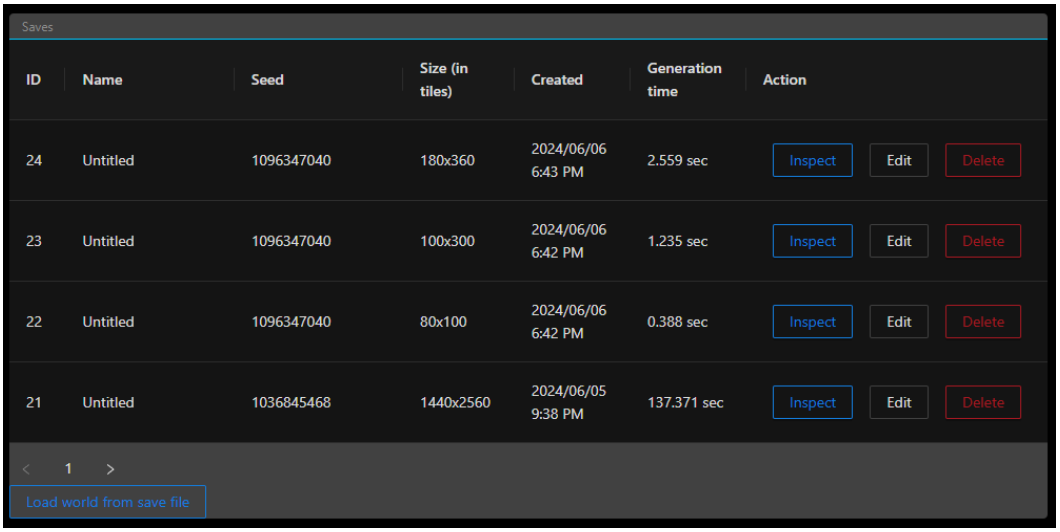


Рисунок 3.5 - Інтерфейс модулю збережених мап

Опис модулю:

Інтерфейс виводить таблицю попередніх результатів генерації. Кожна строка містить: ім’я, сід, розміри, дату створення, час який зайняла генерація, та кнопки керування.

Кнопки керування виконують наступні дії:

1. Inspect - відкриває обрану мапу в модулі для перегляду мапи.
2. Edit - відкриває обрану мапу в модулі для редагування мапи.
3. Delete - видаляє обрану мапу.

Знизу таблиці знаходиться елемент керування сторінками таблиці. На сторінці таблиці відображається максимум 20 рядків.

Отже, модуль задовольняє вимогам сервісу з надання можливості користувачам збереження, перегляду і керування результатами генерації.

3.1.3 Модуль перегляду мапи

Цей модуль надає функціонал з перегляду мапи, а саме: змінення масштабу, зміну позиції перегляду мапи, також цей модуль надає всю наявну інформацію про мапу, та окремі її тайли. Модуль поділений на частини задля полегшення користувацького досвіду шляхом візуального відокремлення функціоналу різного типу.

Основною частиною модулю є вікно із зображенням мапи у проекції на двохвимірну площину (наведено на рис 3.6).

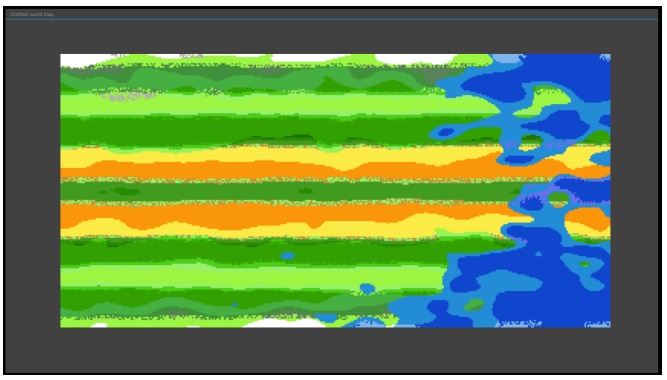


Рисунок 3.6 - Зображення мапи в інтерфейсі

Масштаб мапи може бути змінений використовуючи колесо миші, а за допомогою drag-and-drop може бути змінено положення з якого користувач дивиться на мапу. Важливо відмітити, що мапа зображується у тороїдальній проекції, тобто зациклений по осям XY [18].

У нижній-лівій частині модуля знаходяться всі параметри мапи (рис 3.7): ім’я, розміри, сід генерації, кількість тайлів та всі значення параметрів генератора за якими було створено мапу. Завдяки цій інформації можна відтворити таку саму мапу.

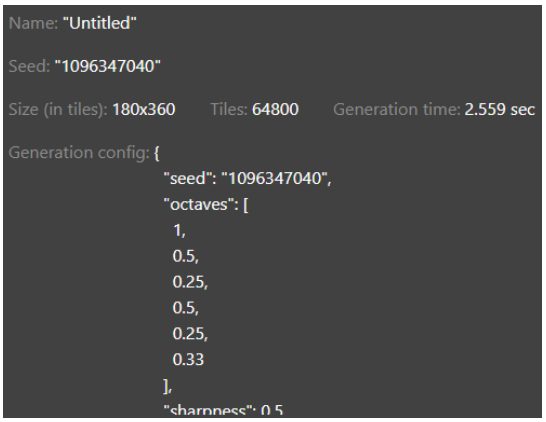


Рисунок 3.7 - Параметри мапи в інтерфейсі перегляду мапи

При натисканні лівою кнопкою миші по мапі, в нижній частині модуля, з’явиться детальна інформація про тайл мапи на який натиснув користувач (рис 3.8).

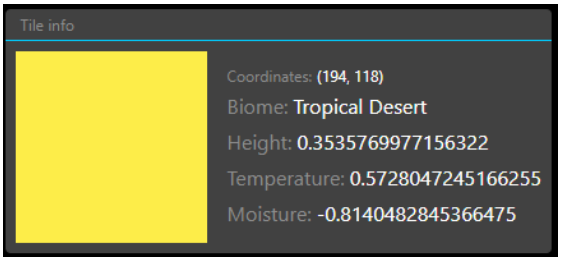


Рисунок 3.8 - Детальна інформація про обраний тайл на мапі

Опис інформації про тайл:

1. Зображення цього тайлу зліва
2. Координати тайлу
3. Кліматична зона в який знаходиться тайл (біом)
4. Висота тайлу
5. Температура
6. Вологість

У правій частині модуля знаходяться елементи керування, та просмотру мапи (наведені на рис 3.9)

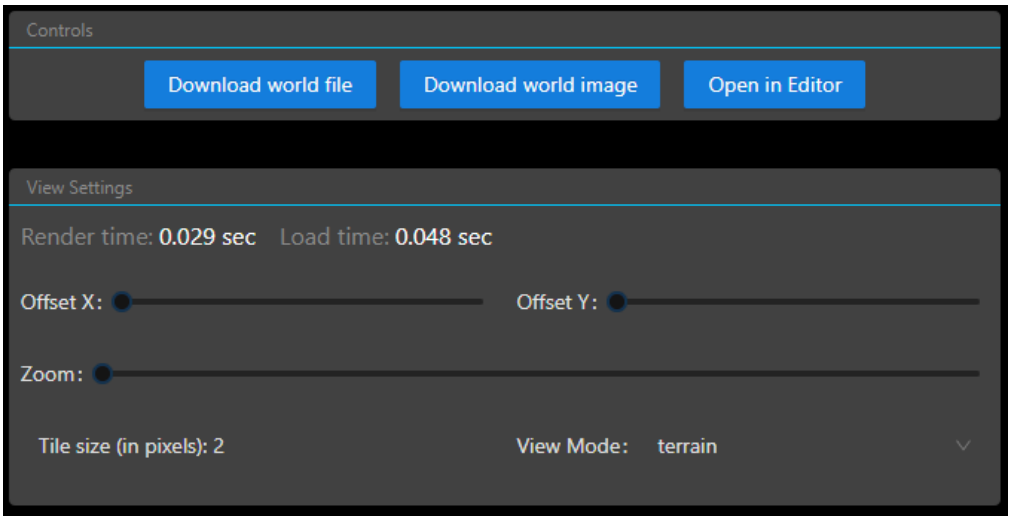


Рисунок 3.9 - Елементи керування модулю перегляду мапи

Опис елементів керування:

1. Завантаження згенерованої мапи в файл (в форматі JSON)
2. Завантаження зображення мапи
3. Відкриття мапи в редакторі
4. Вивід часу за який мапа була завантажена з серверу (в секундах)
5. Вивід часу за який мапа відображається в інтерфейсі користувача (в секундах)
6. Зміна положення погляду на мапу по осям XY
7. Приближення мапи
8. Зміна розміру тайлів мапи (в пікселях)
9. Зміна режима відображення мапи:
    1) terrain - відображення рельєфу
    2) height - відображення висот
    3) moisture - відображення вологості на мапі
    4) temperature - відображення температури на планети

Різні режими відображення однієї мапи наведені на рис 3.10

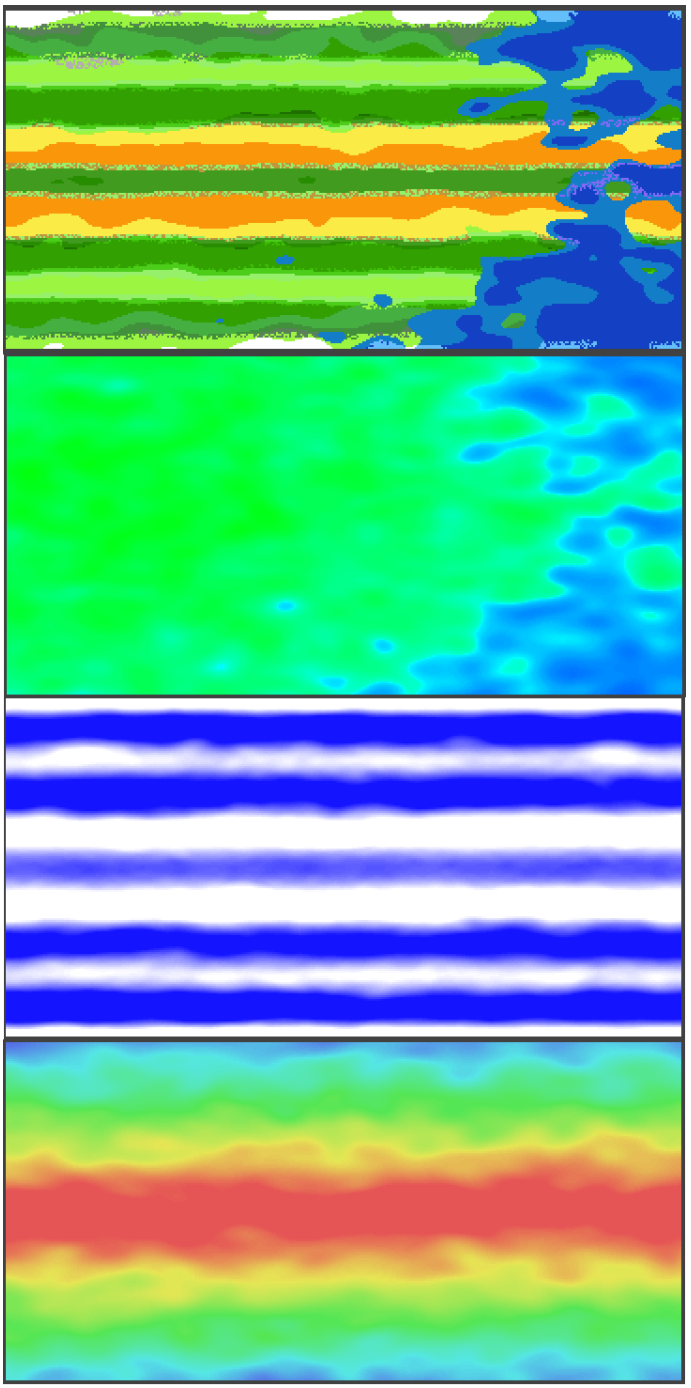


Рисунок 3.10 - Режими відображення мапи, зверху вниз відповідно: terrain, height, moisture, temperature

В цьому підрозділі було розглянуто розроблений модуль перегляду результату генерації. У висновку можна зазначити, що модуль надає дуже гнучкий та розширений інтерфейс для перегляду мапи, її властивостей та властивостей окремих її тайлів, що задовольняє вимогам сервісу.

3.1.4 Модуль редагування мапи

Функціоналом цього модуля є надання користувачу інтерфейсу з можливістю редагування створеної фізичної мапи. Для досягнення цієї мети було запозичено інтерфейс модуля перегляду, та доповнено його вікном для редагування біомів мапи (рис 3.10).

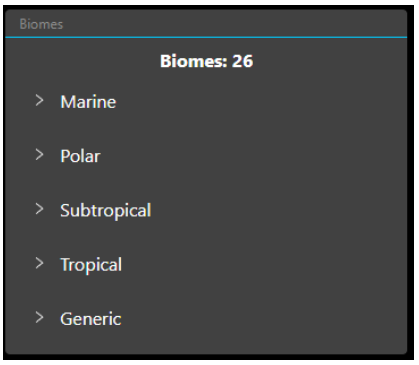


Рисунок 3.10 - Вікно редагування біомів мапи (згорнуте)

Для редагування біомів користувачу потрібно розгорнути відповідну кліматичну зону й перейти до редагування. Інтерфейс надає можливість редагувати колір біома, а також параметри при яких він може бути присутнім на тайлі мапи (наведено на рис 3.1.11). Задля покращення досвіду користувача, збереження змін виконано динамічно (збереження виконується автоматично після 500мс після останньої зміни).

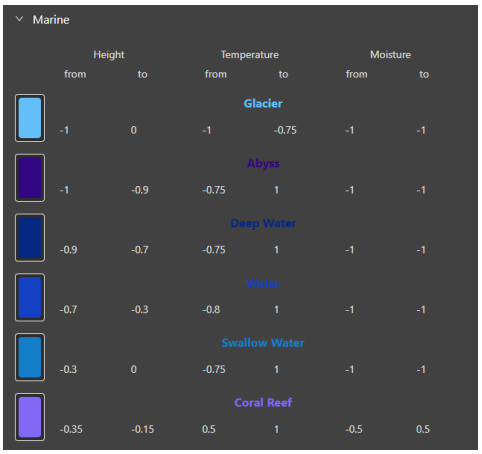


Рисунок 3.11 - Редактор біомів мапи (розгорнутий)

Також варто зазначити графік в нижній частині інтерфейсу (рис 3.12), який наочно зображає поширення біомів на мапі. За допомогою цього графіку користувач може зрозуміти як саме біоми будуть розподілені на мапі у відповідних кліматичних зонах.

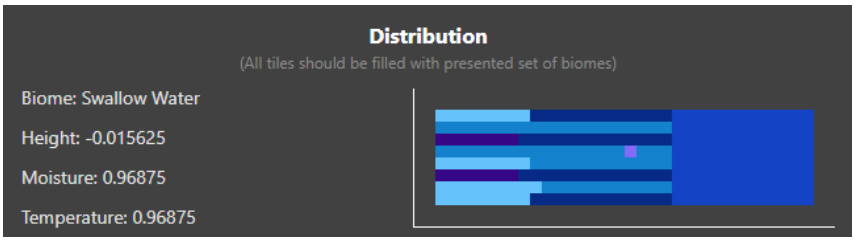


Рисунок 3.12 - Графічне зображення поширення біомів на мапі (враховуючи температуру, вологість та висоту кожного тайлу)

Отже, модуль надає необхідний мінімум інтерфейсу для редагування мапи, що задовольняє вимоги сервісу.

3.2 Розробка серверної частини сервісу

Серверна частина сервісу забезпечує функціональність клієнтської частини, інтеграцію з базою даних на головну функцію сервісу - генерація фізичної мапи планети. Для досягнення мети буде використано веб-сервер Impress та ORM Metasql технологічного стеку Metarhia, а також сутність бібліотеки pg для інтегрування бази даних PostgreSQL у сервіс.

Задля забезпечення функціональності клієнтської частини, було реалізуване відповідне API на базі веб-сервера.

Структура файлів серверної частини поділяється на декілька підпапок: domain, lib, api, та інші внутрішні системні папки, які строго завдані веб-сервером, та надають скелет структури для DDD (Domain Driven Design) [13] архітектури програмної частини.

- Папка domain містить бізнес-логіку сервіса та запити до бази даних.
- Папка lib містить необхідні прикладні функції для роботи системи.
- Папка api містить відповідно API сервісу, яке поєднує клієнтську та серверну частини.

Виглядає структура файлів наступним чином:

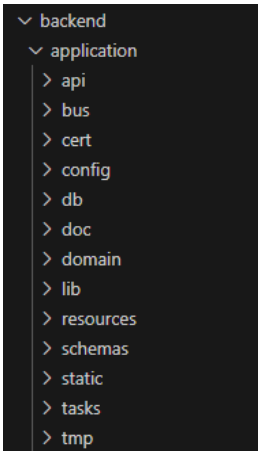


Рисунок 3.13 - Файлова структура серверної частини

Нижче наведено реалізоване API сервісу.

api/world/delete.js відповідає за видалення збереженої мапи планети:

```
method: lib.utils.wrapErrorForApi(async ({ id }) => {
        const data = await domain.world.delete(id);
        return { data };
}),
```

api/world/generate.js відповідає за ініціювання процесу генерації мапи

```
method: lib.utils.wrapErrorForApi(async ({ name, width, height, generatorConfig }) => {
        const time1 = new Date().getTime();
        const terrain = await domain.world.terrain.generate(width, height, generatorConfig);
        const time2 = new Date().getTime();
        const id = await domain.world.save({
                ...terrain,
                accountId: context?.session?.state?.accountId,
                generationTime: time2 - time1,
                name,
        });
        return { data: { id } };
}),
```

Варто зауважити, що вище наведений метод відслідковує час генерації.

api/world/get.js відповідає за завантаження мапи планети до клієнтської частини

```
method: lib.utils.wrapErrorForApi(async ({ id }) => {
        const data = await domain.world.get(id);
        return { data };
}),
```

api/world/getBiomes.js повертає до клієнтської частини список біомів які можуть використовуватись на мапі

```
method: lib.utils.wrapErrorForApi(async () => {
    const data = domain.world.getBiomes();
    const indexed = data.reduce((acc, { biome, color }) => ({ ...acc, [biome]: color }), {});
    return { data, indexed };
}),
```

api/world/getLastSave.js відповідає на запит останньої відкритої мапи поточним користувачем

```
method: lib.utils.wrapErrorForApi(async () => {
        const { data } = await domain.world.getLastSave(
                context?.session?.state?.accountId
        );
        const time = new Date().getTime();
        return { data, time };
}),
```

api/world/getList.js відповідає повертає список збережених результатів генерації, з використанням пагінації, та фільтрів

```
method: lib.utils.wrapErrorForApi(async ({ filters, ...condition }) => {
        const { data, total } = await domain.world.getList(
                context?.session?.state?.accountId,
                filters,
                condition
        );
        return { data, total };
}),
```

api/world/setBiomes.js оновлює біоми мапи

```
async ({ biomes }) => {
        domain.world.setBiomes(biomes);
        return { data: 'ok' };
}
```

Також варто зазначити, що завдяки використаній файловій структурі, можна легко змінити реалізації API, не змінюючи бізнес-логіку сервісу. Це може бути корисно для версіонування API.

api/user/create.js додає нового користувача до сервісу

```
method: lib.utils.wrapErrorForApi(async (user) => {
        await lib.permission.assert(context?.session?.state?.accountId, ['admin']);
        const { password, ...account } = user;
        const data = await domain.query.Account.create({
                password: await metarhia.metautil.hashPassword(password),
                ...account,
        });
        return { data };
}),
```

api/user/delete.js видаляє користувача із сервісу

```
method: lib.utils.wrapErrorForApi(async ({ id }) => {
        await lib.permission.assert(context?.session?.state?.accountId, ['admin']);
        const data = await lib.crud.delete('Account', { id });
        return { data };
}),
```

api/user/getList.js повертає список користувачів сервісу з використанням пагінації та (опціональних) фільтрів

```
method: lib.utils.wrapErrorForApi(async ({ id }) => {
        await lib.permission.assert(context?.session?.state?.accountId, ['admin']);
        const data = await domain.query.Account.get({ id });
        return { data };
}),
```

api/user/update.js оновлює дані користувача за наданим ідентифікаційним номером

```
method: lib.utils.wrapErrorForApi(async ({ id, delta }) => {
        await lib.permission.assert(context?.session?.state?.accountId, ['admin']);
        const data = await domain.query.Account.update({
                id,
                delta,
        });
        return { data };
}),
```

На цьому рівні виконується опрацювання сесії, прав доступу, та станів користувача.

Нижче розглянуто бізнес-логіку сервісу.

domain/world/get.js робить запит до бази даних, повертаючи збережений на сервері JSON об’єкт з даними мапи

```
async (id) => {
    const data = await lib.crud.get('World', parseInt(id));
    const path = node.path.join(config.world.savesPath, `${id}.json`);
    const tiles = node.fs.readFileSync(path);
    return { ...data, tiles: JSON.parse(tiles) };
}
```

domain/world/delete.js виконує видалення даних мапи, за ідентифікаційним номером, з файлової системи сервісу, та бази даних

```
async (id) => {
        await lib.crud.delete('World', id);
        const path = node.path.join(config.world.savesPath, `${id}.json`);
        if (node.fs.existsSync(path)) {
                node.fs.unlinkSync(path);
        }
        return { data: 'ok' };
}
```

domain/world/getList.js повертає список збережених результатів генерації з використанням пагінації, та (опціональних) фільтрів

```
async (accountId, filters = [], condition) => {
        const data = await lib.crud.getList(
                'World',
                [...filters, { accountId }],
                condition,
                ['id', 'name', 'generatorConfig', 'height', 'width', 'generationTime', 'createdAt']
        );
        const total = await lib.crud.count('World', [{ accountId }]);
        return { data, total };
}
```

domain/world/save.js зберігає створену мапу до файлової системи серверу та її мета-дані до бази даних

```
async ({ tiles, ...worldData }) => {
    const id = await lib.crud.create('World', worldData);
    if (!node.fs.existsSync(config.world.savesPath)) {
        node.fs.mkdirSync(config.world.savesPath, { recursive: true });
    }
    const path = node.path.join(config.world.savesPath, `${id}.json`);
    node.fs.writeFileSync(path, JSON.stringify(tiles), { flag: 'w+' });
    return id;
};
```

На цьому рівні відбувається опрацювання станів сервісу, виклик генератора та виклики до бази даних за допомогою ORM. Таким чином, серверна частина сервісу забезпечує роботу та надає функціонал клієнтській частині, зв’язуючі її з базою даних та бізнес-логікою.

Веб-сервер потребує налагодження для початку роботи, вміст налаштувань веб-серверу наведено на рис 3.14.

```
({
        host: '0.0.0.0',
        protocol: process.env.HTTPS,
        balancer: 8000,
        ports: [8001, 8002],
        nagle: false,
        timeouts: {
                bind: 2000,
                start: 30000,
                stop: 5000,
                request: 300000,
                watch: 1000,
        },
        scheduler: {
                concurrency: 1000,
                size: 2000,
                timeout: 3000,
        },
        queue: {
                concurrency: 1000,
                size: 2000,
                timeout: 3000,
        },
        workers: {
                pool: 4,
                wait: 2000,
                timeout: 5000,
        },
        cors: {
                origin: '*',
        },
});
```

Рисунок 3.14 - config/server.js - файл конфігурації веб-сервісу

Також потребує налагодження Docker [14], конфігураційні файли якого наведені на рис 3.15, 3.16, та 3.17.

```
FROM node:16-alpine
WORKDIR /app
COPY package*.json ./
RUN npm ci
WORKDIR /app/frontend
RUN mkdir -p /app/frontend/node_modules/.cache && chmod -R 777 /app/frontend/node_modules/.cache
RUN mkdir -p /app/node_modules/.cache && chmod -R 777 /app/node_modules/.cache
CMD npm run start
```

Рисунок 3.15 - Dockerfile.frontend, файл конфігурації клієнтської частини Docker

```
FROM node:16-alpine as build
WORKDIR /app/backend
COPY . .
RUN apk add --no-cache python3 \
        make \
        g++
RUN npm ci
# RUN npm ci --only=production
# RUN npm ci --omit=dev
# FROM node:16-alpine
# COPY --from=build /app /app
WORKDIR /app/backend
CMD node server.js
```

Рисунок 3.16 - Dockerfile.backend, файл конфігурації серверної частини Docker

```
({
        host: '0.0.0.0',
        protocol: process.env.HTTPS,
        balancer: 8000,
        ports: [8001, 8002],
        nagle: false,
        timeouts: {
                bind: 2000,
                start: 30000,
                stop: 5000,
                request: 300000,
                watch: 1000,
        },
        scheduler: {
                concurrency: 1000,
                size: 2000,
                timeout: 3000,
        },
        queue: {
                concurrency: 1000,
                size: 2000,
                timeout: 3000,
        },
        workers: {
                pool: 4,
                wait: 2000,
                timeout: 5000,
        },
        cors: {
                origin: '*',
        },
});
```

Рисунок 3.17 - config/server.js - файл конфігурації веб-сервісу

Також потребує налагодження Docker, конфігураційні файли якого наведені на рис 3.18, 3.19, та 3.20.

```
FROM node:16-alpine
WORKDIR /app
COPY package*.json ./
RUN npm ci
WORKDIR /app/frontend
RUN mkdir -p /app/frontend/node_modules/.cache && chmod -R 777 /app/frontend/node_modules/.cache
RUN mkdir -p /app/node_modules/.cache && chmod -R 777 /app/node_modules/.cache
CMD npm run start
```

Рисунок 3.18 - Dockerfile.frontend, файл конфігурації клієнтської частини Docker

```
FROM node:16-alpine as build
WORKDIR /app/backend
COPY . .
RUN apk add --no-cache python3 \
        make \
        g++
RUN npm ci
# RUN npm ci --only=production
# RUN npm ci --omit=dev
# FROM node:16-alpine
# COPY --from=build /app /app
WORKDIR /app/backend
CMD node server.js
```

Рисунок 3.19 - Dockerfile.backend, файл конфігурації серверної частини Docker

```
version: "3.3"
services:
  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile.${MODE}
      args:
        - REACT_APP_METACOM_PORT=8000
    container_name: world-frontend
    stdin_open: true
    volumes:
      - ./frontend:/app/frontend
      - /app/frontend/node_modules
    ports:
      - "${HTTP_PORT}:${HTTP_PORT}"
      - "${HTTPS_PORT}:${HTTPS_PORT}"
    environment:
      - REACT_APP_METACOM_PORT=8000
      - HTTPS=${HTTPS}
      - SSL_CRT_FILE=${SSL_CRT_FILE}
      - SSL_KEY_FILE=${SSL_KEY_FILE}
  db:
    image: "postgres:11"
    container_name: world-db
    ports:
      - "5432:5432"
    volumes:
      - ./backend/application/schemas/database.sql:/docker-entrypoint-initdb.d/1_database.sql:Z
      - ./backend/application/db/required_data.sql:/docker-entrypoint-initdb.d/3_initial.sql:Z
      - ./backend/application/db/data.sql:/docker-entrypoint-initdb.d/4_data.sql:Z
      - pgdata:/var/lib/postgresql/data:Z
    environment:
      - POSTGRES_DB=${POSTGRES_DB}
      - POSTGRES_USER=${POSTGRES_USER}
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile.${MODE}
    container_name: world-backend
    volumes:
      - ./backend:/app/backend
      - /app/backend/node_modules
    ports:
      - "8000:8000"
    depends_on:
      - db
    environment:
      - POSTGRES_DB=${POSTGRES_DB}
      - POSTGRES_USER=${POSTGRES_USER}
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
      - HTTPS=${HTTPS}
      - DOMAIN=${DOMAIN}
volumes:
  pgdata:
```

Рисунок 3.20 - docker-compose.yml, файл конфігурації Docker

Вищенаведені конфігурації Docker дозволяют розгорнути і запустити проєкт на будь якому комп’ютері з підтримкою віртуалізації, та мінімальними вимогами для запуску інструменту Docker в операційній системі. Функціональна схема сервісу наведена у Додатку 2.

3.3 Розробка алгоритму генерації рельєфу на мапі планети

У цьому підрозділі розглянуто розробку алгоритму генерації рельєфу місцевості на мапі, який стане основою фізичної мапи планети. Принципова схема алгоритму генерації фізичної мапи планети наведена в Додатку 3.

3.3.1 Огляд результату генерації шуму Перлина

Поширеним способом генерації 2D-карт є використання функції шуму з обмеженою смугою пропускання, наприклад шуму Simplex або Perlin [15], як будівельного блоку. Для розробки алгоритму застосуємо бібліотечну реалізацію шуму Перлина яка буде модифікована під потреби сервісу.

Шум Перлина генерує нескінченний список випадкових чисел від -1 до 1. Якщо прийняти 1 за білий колір й -1 - за чорний, то візуалізація результату роботи алгоритму Перлина схожа на зображення зі старого телевізора (наведено на рис 3.21)

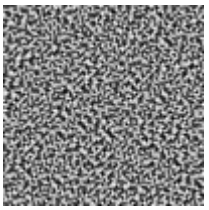


Рисунок 3.21 - Візуалізоване зображення шуму Перлина

Отже результат генерації шуму Перлина - це лише набір випадкових чисел від -1 до 1, який потрібно дуже сильно перетворити аби отримати схожість з рельєфом [5].

3.3.2 Інтерпретація шуму Перлина

Згенерований набір випадкових чисел шуму Перлина інтерпретуємо як набір значень висоти, присвоївши кожному тайлу мапи цифру яка буде визначати висоту в цьому тайлі (наведено на рис 3.22).

```
const nx = x / width - 1;
const ny = y / height - 1;
elevation[y][x] = noise(nx, ny);
```

Рисунок 3.22 – Псевдокод присвоєння тайлу мапи висоти

Це згенерує багато гострих вершин з різкими перепадами висоти.

Додаємо амплітуду до алгоритму, аби врегулювати перепади висоти (наведено на рис 3.23).

```
elevation[y][x] = noise(amplitude * nx, amplitude * ny);
```

Рисунок 3.23 – Псевдокод алгоритму з доданою амплітудою

В залежності від амплітуди результат генерації тепер коливається від хвилястих фракталів до гострих вершин.

Також алгоритм має використати ключову фразу (seed) надану користувачем як один з параметрів алгоритму, для того аби з однаковими параметрами (але різним сідом) можно було генерувати різні мапи. Варто зазначити, що при однакових параметрах та однаковому значенні зерна - результат генерації алгоритму має бути однаковим.

3.3.3 Підвищення деталізації генерації висоти

Аби внести більшого різноманіття й деталізації в результат генерації, можна згенерувати три висоти з різними параметрами для кожної клітки мапи, та інтерферувати (додавши) їх між собою (рис 3.24).

Назвемо ці параметри октавами.

```
elevation[y][x] = octave1 * noise(amplitude1 * nx, amplitude1 * ny);
                + octave2 * noise(amplitude2 * nx, amplitude2 * ny);
                + octave3 * noise(amplitude3 * nx, amplitude3 * ny);
```

Рисунок 3.24 – Псевдокод отримання висоти для тайлу мапи із застосуванням інтерференції трьох шумів

Якщо згенерувати три шуми з різними октавами й інтерферувати їх між собою (скласти відповідні значення), то отриманий шум буде виглядати більш деталізованим фракталом (рис 3.25)

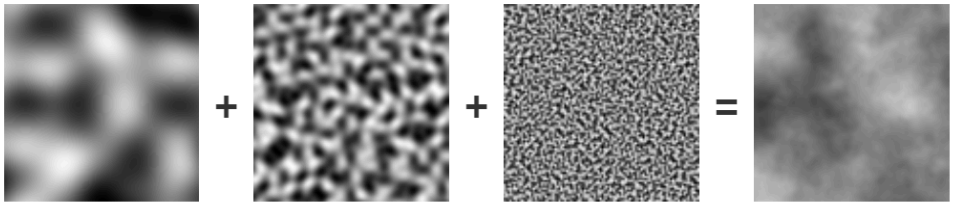


Рисунок 3.25 – Інтерференція різних шумів породжує новий шум

Такий фрактал схожий на рельєф з малими і великими пагорбами, але все ще не буде вистачати плоских долин, високих гір та деталізації. Крім того при додаванні шумів може виникнути проблема через те, що значення висоти буде більше (або менше) за граничні [-1, 1]. Для вирішення цієї проблеми потрібно розділити результат на суму амплітуд [16]. Ще однією проблемою може бути корельованість (схожість) значень шуму при nx та ny близьких до 0. Для отримання найкращих результатів, потрібно, щоб октави були незалежними. Вирішенням цієї проблеми може бути використання окремих зерен (seed) для кожної з октав. Ці зерна потрібно зробити на основі зерна мапи аби зберегти властивість генерації однакових мап при одинаковому зерні [17]. На рисунку 3.26 наведено код алгоритму з виправленнями.

```
noise = PerlinNoise(seed);
seedHash = createMD5Hash(seed);
octaveSeed1 = parseInt(seedHash.substring(0, 8));
octaveSeed2 = parseInt(seedHash.substring(8, 16));
e    =    octave1 * noise(amplitude1 * nx, amplitude1 * ny);
     +    octave2 * noise(
              amplitude2 * nx + (octaveSeed2 / octaveSeed1),
              amplitude2 * ny + (octaveSeed2 / octaveSeed1) / 2
          );
     +    octave3 * noise(
              amplitude3 * nx + (octaveSeed2 / octaveSeed1) * 2,
              amplitude3 * ny + (octaveSeed2 / octaveSeed1)
          );
elevation[y][x] = e / (amplitude1 + amplitude2 + amplitude3);
```

Рисунок 3.26 – Псевдокод генерації висоти для тайлу мапи

3.3.4 Реалізація генератора висот рельєфу

Останнім кроком алгоритма генерації буде перерозподіл висоти за допомогою математичної функції піднесення в степінь (наведено на рис 3.27). Для цього потрібно ввести в алгоритм ще два параметри: sharpness, який буде степенем, та fudgeFactor який буде помножувати висоту на деяке число трішки більше за одиницю, для запобігання піднесення в степінь дуже малих значень висоти, що могло б призвести до неякісного результату генерації.

```
noise = PerlinNoise(seed);
seedHash = createMD5Hash(seed);
octaveSeed1 = parseInt(seedHash.substring(0, 8));
octaveSeed2 = parseInt(seedHash.substring(8, 16));
e   =   octave1 * noise(amplitude1 * nx, amplitude1 * ny);
    +   octave2 * noise(
            amplitude2 * nx + (octaveSeed2 / octaveSeed1),
            amplitude2 * ny + (octaveSeed2 / octaveSeed1) / 2
        );
    +   octave3 * noise(
            amplitude3 * nx + (octaveSeed2 / octaveSeed1) * 2,
            amplitude3 * ny + (octaveSeed2 / octaveSeed1)
        );
e = e / (amplitude1 + amplitude2 + amplitude3);
e = Math.pow(Math.abs(e) * fudgeFactor, sharpness);
elevation[y][x] = e;
```

Рисунок 3.27 – Остаточна версія генератору висоти для тайлу мапи описаний псевдокодом

Такий метод перерозподілу висоти має на меті зменшити різкість перепадів, та доповнити рельєф мапи плоскими долинами і високими горами.

Використовуючи обрану мову програмування - Node.JS [19], було реалізовано алгоритм генерації мапи рельєфу (рис 3.28)

Варто зазначити, що в алгоритмі було закладено можливість використання довільної кількості октав, задля забезпечення гнучкості алгоритму.

```
const tiles = [];
const nx = x / width - 1;
const ny = y / height - 1;
for (let y = 0; y < height; y++) {
    for (let x = 0; x < width; x++) {
        // generate terrain height in coordinate (x, y)
        let h = terrainAmplitudes.length === 0
            ? wrapNoiseXY(genTerrain, nx, ny)
            : terrainAmplitudes.reduce((acc, amplitude, i) =>
                acc + octaves[i] * wrapNoiseXY(
                    genTerrain,
                    amplitude * nx + (octaveSeed2 / octaveSeed1) * i,
                    amplitude * ny + (octaveSeed2 / octaveSeed1) * i / 2,
                    i
                )
                , 0
            ) / terrainAmplitudes.reduce((acc, amplitude) => acc + amplitude, 0);

        // redistribution
        h = Math.pow(Math.abs(h) * fudgeFactor, sharpness);
        // save generated tile
        tiles.push({ x, y, height: h });
    }
}
```

Рисунок 3.28 – Реалізація алгоритму генераторації рельєфу мапи

3.4 Розробка алгоритму генерації температури на мапі планети

У цьому підрозділі розглянуто розробку алгоритму генерації температури на мапі планети. Температура - важливий елемент навколишнього середовища, без якого неможливо побудувати якісну фізичну мапу планети.

Для того щоб згенерувати реалістичну температуру для кожного тайлу мапи, потрібно прийняти до уваги декілька критеріїв від яких вона повинна залежати [20]:

1. Чим більше висота тайлу - тим нижча температура.
2. На полюсах планети температура найнижча
3. На екваторі планети температура найвища

Прийнявши до уваги вижчезазначені критерії, було реалізовано алгоритм генерації температури для окремого тайлу фізичної мапи (наведено на рис 3.29)

```
// generate common temperature value in coordinate (x, y)
const t1 = Math.abs((height / 2 - y) / height * 2) * -2 + 1;

// generate temperature noise in coordinate (x, y)
const t2 = temperatureAmplitudes.length === 0
    ? wrapNoiseXY(genTemperature, nx, ny)
    : temperatureAmplitudes.reduce((acc, amplitude, i) =>
        // acc + amplitude * wrapNoiseXY(genTemperature, Math.pow(2, i) * nx, Math.pow(2, i) * ny), 0
        acc + octaves[i] * wrapNoiseXY(
            genTemperature,
            amplitude * nx + (octaveSeed2 / octaveSeed1) * i,
            amplitude * ny + (octaveSeed2 / octaveSeed1) * i / 2,
            i
        )
    ) / temperatureAmplitudes.reduce((acc, amplitude) => acc + amplitude, 0);

let t = t1 + t2 + temperature;
if (Number.isNaN(t)) {
    throw new Error('temperature is NaN', {
        t1, t2, temperature, temperatureAmplitudes
    });
}
if (t > 1) t = 1;
if (t < -1) t = -1;
```

Рисунок 3.29 – Реалізація алгоритму генерації температури тайла мапи

Отже, в цьому підрозділі було реалізовано алгоритм генерації температури для фізичної мапи планети за допомогою шуму Перлина. Варто зазаначити, що розподілення температури по поверхні планети буде відбуватись рівномірно від екватора до полюсів: на екваторі температура буде найвища, на полюсах - найнижча. Також значення температури буде трішки рандомізовано завдяки інтерференції з шумом Перлина, завдяки чому було досягнуто реалістичного зображення покриття температурою фізичної мапи планети.

3.5 Розробка алгоритму генерації вологості на мапі планети

Для розробки алгоритму генерації вологості на мапі планети необхідно врахувати поняття вологість, що є важливим елементом навколишнього середовища, без якого неможливо побудувати якісну фізичну мапу планети.

Для того щоб згенерувати реалістичне значення вологості для кожного тайлу мапи, потрібно прийняти до уваги декілька критеріїв від яких вона повинна залежати:

1. Чим більше температура - тим більша вологість.
2. Вологість навкруги водойм вижче.
3. Вологість зменшується хвилями в результаті опадів

Прийнявши до уваги вижчезазначені критерії, було вирішено, що найкраще для реалізації алгоритму генерації вологості на мапі планети підходить тригонометрична функція косинуса (наведено на рис 3.30).

```
// generate common moisture value in coordinate (x, y)
const mBaseY = Math.abs((height / 2 - y) / height * 2) * -1;
const m1 = Math.pow(Math.E, -mBaseY) * Math.cos(15 * mBaseY);
const m3 = Math.cos(4 * t);

// generate moisture value noise in coordinate (x, y)
const m2 = moistureAmplitudes.length === 0
    ? wrapNoiseXY(genMoisture, nx, ny)
    : moistureAmplitudes.reduce((acc, amplitude, i) =>
        // acc + amplitude * wrapNoiseXY(genMoisture, Math.pow(2, i) * nx, Math.pow(2, i) * ny), 0
        acc + octaves[i] * wrapNoiseXY(
            genMoisture,
            amplitude * nx + (octaveSeed2 / octaveSeed1) * i,
            amplitude * ny + (octaveSeed2 / octaveSeed1) * i / 2,
            i
        )
    ) / moistureAmplitudes.reduce((acc, amplitude) => acc + amplitude, 0);

let m = m1 + m2 + m3;
if (Number.isNaN(m)) {
    throw new Error('moisture is NaN', {
        m, m1, m2, moistureAmplitudes
    });
}
if (m > 1) m = 1;
if (m < -1) m = -1;
```

Рисунок 3.30 – Реалізація алгоритму генерації вологості тайла мапи

В цьому підрозділі було реалізовано алгоритм генерації вологості для фізичної мапи планети за допомогою шуму Перлина. Варто зазначити, що розподілення вологості по поверхні планети буде відбуватись хвилями від екватора до полюсів, що було досягнуто завдяки використанню тригонометричних функцій. Також значення вологості буде рандомізовано завдяки інтерференції з шумом Перлина, завдяки чому було досягнуто реалістичного зображення покриття вологості фізичної мапи планети.

ВИСНОВКИ ДО РОЗДІЛУ 3

У третьому розділі детально розглянуто розробку модулей клієнтської та серверної частин сервісу для генерації фізичної мапи планети за допомогою шуму Перлина.

Інтерфейс користувача, поділений на модуль генерації, зберігання, перегляду та редагування фізичних мап, надає можливість згенерувати фізичну мапу планети з гнучким налаштуванням, за бажанням користувача.

Визначено, що модуль генерації надає користувачу можливість згенерувати фізичну мапу планети, з гнучким налаштуванням параметрів генерації та без перевантаження інтерфейсу великою кількістю елементів керування, що робить досвід користувача гладким і приємним.

Модуль редагування надає користувачу можливість змінювати мапу планети, що дозволяє редагувати створену мапу не виходячи з сервісу.

Модуль перегляду надає великий набір інструментів для детального аналізу та перегляду мапи, та інформації щодо окремих її тайлів.

Модуль збереження результатів генерації надає користувачу можливість зберігати мапи для подальшого редагування та перегляду на сервісі, що виділяє сервіс серед існуючих аналогів.

Проаналізовано, що серверна частина сервісу забезпечує функціональність інтерфейсу користувача та виконує основну роботу щодо генерації фізичної мапи планети за запитом користувача.

В результаті сервіс для генерації фізичної мапи планети забезпечує користувача широким функціоналом з генерації та перегляду фізичної мапи планети.

РОЗДІЛ 4. ДОСЛІДЖЕННЯ ТА АНАЛІЗ РОЗРОБЛЕНОГО СЕРВІСУ ДЛЯ ГЕНЕРАЦІЇ ФІЗИЧНОЇ МАПИ ПЛАНЕТИ ЗА ДОПОМОГОЮ ШУМУ ПЕРЛИНА

В четвертому розділі проведено функціональне та мануальне тестування розробленого сервісу для генерації фізичної мапи планети за допомогою шуму Перлина.

4.1 Тестування сервісу

Для початку роботи з розробленим програмним продуктом, необхідно авторизуватись в сервісі за наданими адміністратором логіном та паролем.

Для виконання генерації фізичної мапи планети, користувачу необхідно перейти на головну сторінку веб-сайту, яка містить модуль генератора, після чого достатньо натиснути кнопку Generate для відправки запиту на створення мапи і почекати її створення. Головна сторінка веб-сайту наведена на рис. 4.1.

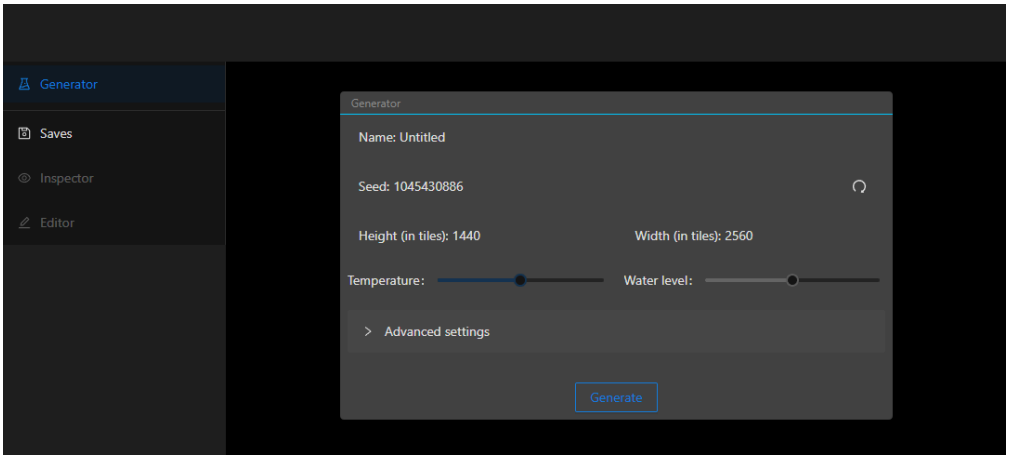


Рисунок 4.1 – Головна сторінка веб-сайту

Створену мапу можна знайти на сторінці збережених результатів генерації, оскільки створенні мапи автоматично зберігаються у сервісі. При натисканні кнопки Inspect - відкривається сторінка для перегляду мапи.

Кнопка Edit - відкриває редактор мапи, відповідно. Сторінка зі збереженими мапами наведена на рис. 4.2.

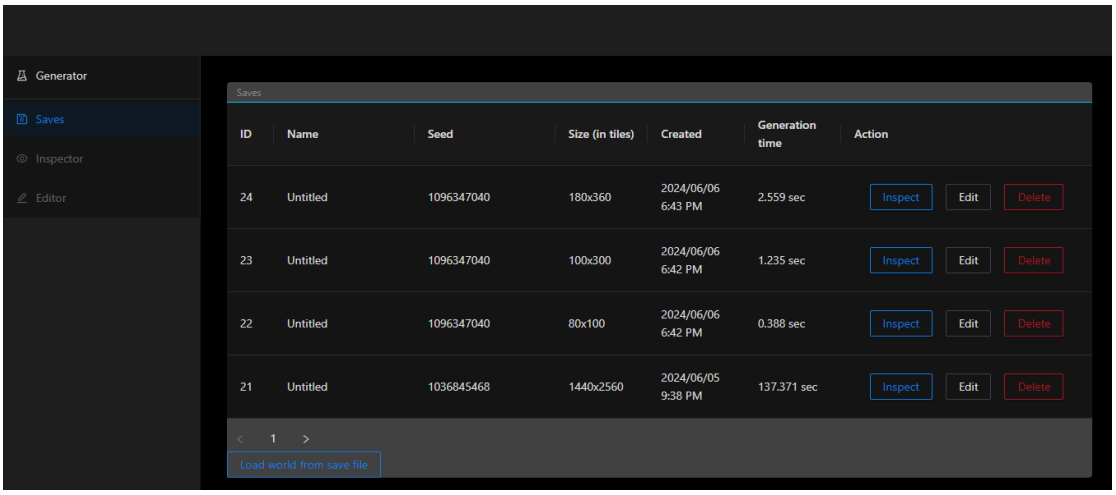


Рисунок 4.2 – Сторінка зі збереженими мапами

На сторінці перегляду знаходяться елементи керування переглядом мапи (наведено на рис. 4.3), а на сторінці редактору дублюються деякі елементи керуванням переглядом, та додаються елементи редагування мапи.

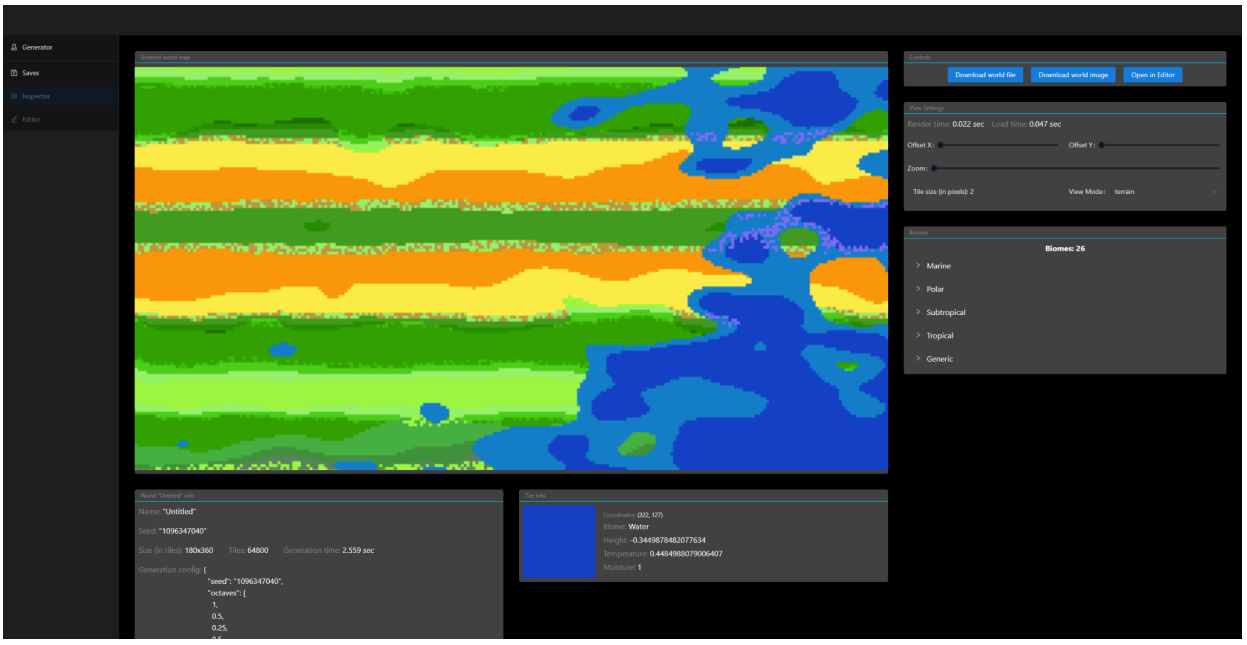


Рисунок 4.3 – Сторінка перегляду мапи

Функціональне випробування сервісу було виконано шляхом генерування 34 мап з різноманітними параметрами, що дало впевненість у правильній роботі головного функціоналу сервісу з надання користувачу можливості генерації фізичної мапи планети за допомогою шуму Перлина. Результат тестування наведено на рис. 4.4.

ID	Name	Seed	Size (in tiles)	Created	Generation time
34	Untitled	1004612099	100x100	2024/06/09 7:01 PM	0.397 sec
33	Untitled	1004612099	100x100	2024/06/09 7:01 PM	0.376 sec
32	Untitled	1004612099	100x100	2024/06/09 7:01 PM	0.397 sec
31	Untitled	1004612099	100x100	2024/06/09 7:01 PM	0.392 sec
30	Untitled	1004612099	100x100	2024/06/09 7:00 PM	0.416 sec
29	Untitled	1004612099	100x100	2024/06/09 7:00 PM	0.41 sec
28	Untitled	1037405526	100x100	2024/06/09 7:00 PM	0.402 sec
27	Untitled	1037405526	1080x1925	2024/06/09 7:00 PM	84.026 sec
26	Untitled	1037405526	50x75	2024/06/09 6:58 PM	0.15 sec
25	Untitled	1037405526	200x500	2024/06/09 6:57 PM	3.886 sec
24	Untitled	1096347040	180x360	2024/06/06 6:43 PM	2.559 sec
23	Untitled	1096347040	100x300	2024/06/06 6:42 PM	1.235 sec
22	Untitled	1096347040	80x100	2024/06/06 6:42 PM	0.388 sec
21	Untitled	1036845468	1440x2560	2024/06/05 9:38 PM	137.371 sec

Рисунок 4.4 – Результат випробування функції сервісу з генерації фізичної мапи планети

В результаті випробування було отримано 34 мапи з різними параметрами, як і очікувалось.

Крім випробувального тестування, було проведено мануальне тестування сервісу, яке мало на меті перевірити функціонування веб-сервісу в реальних умовах, взаємодію з користувачем, та відповідність вимогам. В результаті мануального тестування було опрацьовано наступні тест-кейси:

1. Модуль генератора мапи:

- Створення мапи з параметрами за замовчуванням.
- Створення мапи зі зміненими параметрами
- Створення дуже малих мап (розмірами 50x75 тайлів)
- Створення дуже великих мап (розмірами 1440x2650 тайлів)
- Створення нового зерна генерації
- Створення мап з різними іменами
- Послідовне створення 10 мап з однаковими параметрами
- Послідовне створення 10 мап з різними параметрами

2. Модуль збережених результатів генерації:

- Відкриття результату генерації у сторінці редагування
- Відкриття результату генерації у сторінці перегляду
- Видалення результату генерації
- Завантаження збереженої мапи з файлу формату JSON
- Перехід між сторінками таблиці
- Перевірка максимальної кількості рядків таблиці

3. Модуль перегляду мапи:

- Завантаження зображення мапи
- Завантаження мапи у форматі JSON
- Відкриття мапи у редакторі з вікна перегляду
- Наближення маштабу мапи
- Зміна точки перегляду мапи перетягуванням
- Зміна центру перегляду мапи по осі X
- Зміна центру перегляду мапи по осі Y
- Зміна центру перегляду мапи по осям XY послідовно
- Зміна розміру тайла мапи
- Зміна режиму перегляду мапи на moisture
- Зміна режиму перегляду мапи на temperature
- Зміна режиму перегляду мапи на height

- Повернення до режиму перегляду за замовчуванням (terrain)
- Видалення мапи зі сторінки перегляду
- Натискання лівою кнопкою по мапі задля виділення тайлу мапи

4. Модуль редагування мапи

- Розгортання списка біомів по черзі та згортання
- Розгортання списка біомів у випадковому порядку та згортання
- Зміна кольору біома
- Зміна параметрів при яких біом повинен з’являтися на мапі

Всі тест-кейси були перевірені згідно функціональних вимог до сервісу. Мануальне тестування дозволило виявити помилки, які можуть виникнути при роботі зі сервісом.

Таким чином, комплексне тестування сервісу для генерації фізичної мапи планети за допомогою шуму Перлина показало її працездатність. Розроблений продукт успішно пройшов функціональне випробування та мануальне тестування, що підтверджує його готовність до запуску в реальному середовищі.

4.2 Аналіз результату

Після завершення розробки та тестування сервісу для генерації фізичної мапи планети за допомогою шуму Перлина був проведений аналіз результатів, який дозволяє оцінити якість, працездатність, швидкість, функціональність та коректність роботи системи. Метою аналізу було виявлення потенційних проблем та недоліків, а також оцінка відповідності сервісу вимогам та очікуванням користувачів.

Під час аналізу були враховані наступні аспекти системи:

1. Функціональність та відповідність вимогам

Перевірка наявності та правильної реалізації ключової функції сервісу – генерації фізичної мапи планети за допомогою шуму Перлина. Оцінка відповідності сервісу вимогам, визначеним у перших розділах проєкту.

2. Надійність та стабільність

Аналіз обробки помилок та винятків, забезпечення коректної роботи сервісу навіть у непередбачуваних ситуаціях та при некоректних діях користувачів. Перевірка на відсутність критичних помилок та винятків, які можуть призвести до несподіваної поведінки сервісу.

3. Інтерфейс та зручність використання

Оцінка зручності та інтуїтивності інтерфейсу сервісу для користувачів. Перевірка відповідності сервісу загальноприйнятим практикам та очікуванням користувачів генераторів мап.

4. Безпека та захист даних

Аналіз заходів безпеки, таких як шифрування даних та захист від несанкціонованого доступу. Перевірка відповідності сервісу вимогам щодо захисту особистої інформації користувачів.

Результати аналізу розробленого сервісу показали, що він відповідає встановленим вимогам та забезпечує зручну і ефективну взаємодію з користувачами. Функціональність сервісу була успішно протестована, а мануальні тести підтвердили його працездатність та надійність. Були виявлені та виправлені недоліки, що сприяло покращенню якості сервісу.

Загалом, розроблений сервіс виконує поставлені перед ним завдання, забезпечуючи користувачів зручним та надійним інструментом для створення, перегляду та редагування фізичної мапи планети. Можливість зберегти мапу для подальшого редагування виділяє сервіс серед аналогів.

4.3 Перспективи подальшого розвитку системи

Розроблена сервіс для генерації фізичної мапи планети за допомогою шуму Перлина вже демонструє надійність та високу функціональність, проте його можливості можуть бути розширені шляхом подальшого розвитку. Нижче розглянуті перспективи, які можуть бути реалізовані для поліпшення та розширення сервісу:

1. Розширення функціоналу редактору мапи

Однією з перспективних напрямів розвитку сервісу є розширення функціоналу редактору мапи. Наприклад, можна додати до редактору можливість міняти температуру, висоту, та вологість на окремих тайлах, та на великій території, просто розмальовуючи мапи обраним інструментом редагування, як в редакторі зображень. Це дозволить розширити користувацький досвід у сервісі, й надасть додаткові переваги серед аналогів.

2. Удосконалення алгоритму генерації

Напевно одним з найперспективніших напрямів розвитку сервісу буде удосконалення алгоритму генерації, метою якого буде підвищення реалістичності та деталізації мапи. Наприклад додавання нових форм рельєфу: озер, річок, тощо. А також додавання до алгоритму генерації вітрів та течій, що сильно збільшить деталізації та реалістичність створених мап. Таким чином користувач може отримати більш якісне зображення мапи.

3. Автоматизоване тестування

Оскільки подальша розробка проєкту може передбачати сильне втручання в програмний код, має сенс розробити автоматизовані тести основного функціоналу, для зменшення часу відлагодження в майбутньому

та забезпечення безперебійної роботи сервісу.

4. Оптимізація алгоритму генерації

Пришвидшення алгоритму генерації позитивно вплине на досвід роботи користувача із сервісом, а зменшення використання ресурсів операційної системи веб-сервером надасть можливість надалі вдосконалювати алгоритм без ризику перевикористати виділені серверу ресурси.

5. Оптимізація відображення мапи

Занадто великі мапи використовують багато ресурсів і часу для відображення, тому є ще один шлях розвитку - оптимізувати процес відображення мапи.

6. Покращення інтерфейсу користувача

Подальша розробка інтерфейсу сервісу буде залежати від потреб користувачів, що потребує детальнішого дослідження та збору зворотного зв’язку від користувачів про їх досвід роботи з сервісом.

7. Локалізація інтерфейсу

Ще одним шляхом розвитку сервісу може бути локалізація інтерфейсу для надання функцій сервісу більшій аудиторії, що може забезпечити сервіс більшою кількістю користувачів.

Реалізація цих перспектив розвитку допоможе сервісу для генерації фізичної мапи планети за допомогою шуму Перлина стати ще більш корисним інструментом для користувачів. Завдяки постійному вдосконаленню та відповіді на потреби користувачів, сервіс може стати потужним інструментом для генерації фізичної мапи планети.

ВИСНОВКИ ДО РОЗДІЛУ 4

У четвертому розділі дипломного проєкту описано випробування та мануальне тестування сервісу для генерації фізичної мапи планети за допомогою шуму Перлина. Розроблено мануальні тест-кейси, які покривають основні функції сервісу, а мануальне тестування дозволило перевірити роботу сервісу в реальних умовах.

Проаналізовано, що завдяки мануальному тестуванню опрацьовано різноманітні тест-кейси, включаючи створення, перегляд та редагування результатів генерації. Комплексне тестування підтвердило працездатність сервісу та перевірило його функціональність. Виявлені під час тестування помилки були виправлені, що сприяло покращенню якості сервісу.

Аналіз результатів роботи сервісу для генерації фізичної мапи планети за допомогою шуму Перлина підтвердив, що його відповідність вимогам та очікуванням користувачів. Сервіс виявився надійним, стабільним та зручним у використанні. Було враховано всі аспекти функціональності, надійності, інтерфейсу та безпеки.

Окрім цього, розглянуто перспективи подальшого розвитку дипломного проєкту. Вони включають оптимізацію та удосконалення алгоритму генерації, локалізацію інтерфейсу, розширення функціоналу сервісу та розробку автоматизованих тестів.

Таким чином, розроблений сервіс для генерації фізичної мапи планети готовий до запуску в реальному середовищі. Він забезпечує ефективний, надійний та гнучкий спосіб створення фізичної мапи планети, задовольняючи всі вимоги та очікування користувачів.

ВИСНОВКИ

У межах дипломного проєкту розроблено сервіс для генерації фізичної мапи планети за допомогою шуму Перлина. Створений сервіс надає можливість користувачу створити, переглядати та редагувати фізичну мапу планети.

У першому розділі проведено огляд інструментів та програмного забезпечення для створення та управління розкладом, здійснено аналіз предметної області й існуючих технологій. Виявлено, що існуючі рішення не відповідають потребам сучасного ринку програмного забезпечення.

У другому розділі розглянуто архітектура сервісу, вибір бази даних, веб-сервер, та проведено аналіз переваг і недоліків використаних технологій та інструментів, завдяки якому їх було обрано та обґрунтовано їх доцільність застосування.

У третьому розділі описано розроблений функціонал сервісу, який включає модулі збереження, створення, перегляду та редагування фізичної мапи планети, а також докладно описано розробку алгоритму генерації фізичної мапи планети за допомогою шуму Перлина.

У четвертому розділі проведено випробувальне та мануальне тестування, а також проведено аналіз розробленого сервісу. В результаті тестування були виявлені та виправлені деякі помилки, що в цілому покращило якість програмного продукту. Сервіс для генерації фізичної мапи планети за допомогою шуму Перлина показав високу гнучкість, ефективність та надійність у роботі, що підтверджує її відповідність поставленим вимогам.

Отже, результатом дипломного проєкту є сервіс для генерації фізичної мапи планети за допомогою шуму Перлина, що надає користувачам гнучкий функціонал зі створення фізичної мапи планети, редагування та перегляду отриманого результату генерації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Wikipedia Map Projection [Електронний ресурс]. Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Map_projection
2. Wikipedia Perlin Noise [Електронний ресурс]. Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Perlin_noise
3. Wikipedia User (Computer Programing) [Електронний ресурс]. Режим доступу до ресурсу: https://en.wikipedia.org/wiki/User_(computing)
4. Wikipedia Parameter (Computer Programing) [Електронний ресурс]. Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Parameter_(computer_programming)
5. Making maps with noise functions from Red Blob Games [Електронний ресурс]. Режим доступу до ресурсу: https://www.redblobgames.com/maps/terrain-from-noise
6. Metarhia Impress. [Електронний ресурс]. Режим доступу до ресурсу: https://github.com/metarhia/impress
7. Docker Build Cloud | Docker Docs [Електронний ресурс]. Режим доступу до ресурсу: https://docs.docker.com/build/cloud/
8. PostgreSQL: About [Електронний ресурс]. Режим доступу до ресурсу: https://www.postgresql.org/about/
9. Metarhia Metasql Docs [Електронний ресурс]. Режим доступу до ресурсу: https://github.com/metarhia/Docs/blob/main/content/en/DATA.md#metasql-query-builder
10. Mewo [Електронний ресурс]. Режим доступу до ресурсу: https://mewo2.com/notes/terrain/
11. Donjon [Електронний ресурс]. Режим доступу до ресурсу: https://donjon.bin.sh/fantasy/world/

12. React.JS Docs [Електронний ресурс]. Режим доступу до ресурсу: https://react.dev/reference/react
13. Wikipedia Domain-driven Design [Електронний ресурс]. Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Domain-driven_design
14. Containerize a Node.js application [Електронний ресурс]. Режим доступу до ресурсу: https://docs.docker.com/language/nodejs/containerize/
15. Youtube “Генерація ігрових світів. Частина 1 - теорія шумів (Perlin, Value, Random noise)” [Електронний ресурс]. Режим доступу до ресурсу: https://www.youtube.com/watch?v=oOy3IkOLbWs
16. Youtube “Генерація ігрових світів. Частина 2 - одновимірний шум” [Електронний ресурс]. Режим доступу до ресурсу: https://www.youtube.com/watch?v=oOy3IkOLbWs
17. Youtube “Генерація ігрових світів. Частина 3 - Perlin Noise (двовимірний шум Перлина)” [Електронний ресурс]. Режим доступу до ресурсу: https://www.youtube.com/watch?v=oOy3IkOLbWs
18. Wikipedia Toroidal Planet [Електронний ресурс]. Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Toroidal_planet
19. Node.JS 18 LTS Docs [Електронний ресурс]. Режим доступу до ресурсу: https://devdocs.io/node~18_lts/
20. Generating terrain in Cuberite [Електронний ресурс]. Режим доступу до ресурсу:http://cuberite.xoft.cz/docs/Generator.html

ДОДАТОК 1

Сервіс для генерації фізичної мапи планети за допомогою шуму Перлина

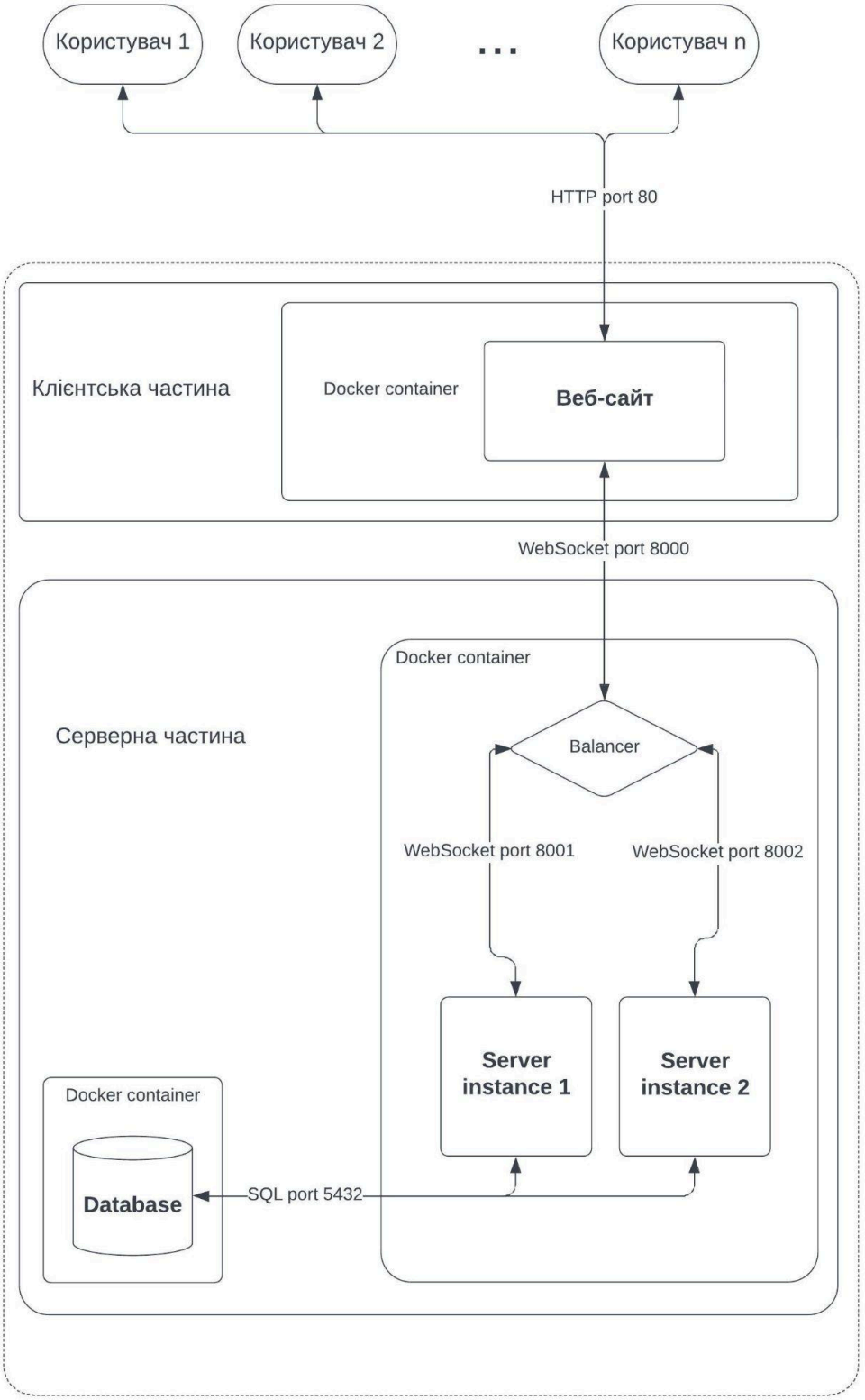
Архітектура сервісу

(структурна схема)

ІАЛЦ.467200.003 Д1

Аркушів 1

Київ – 2024



					ІАЛЦ.467200.003 Д1		
Зм	Лист	№ докум.	Підп	Дата			
Розроб.		Гуськов Д. І..			Сервіс для генерації фізичної мапи планети за допомогою шуму Перлина	*Літ.*	*Арк.* 1 / *Аркушів* 1
Перев.		Міщенко Л. Д.					
Реценз						КПІ ім. Ігоря Сікорьского, ФІОТ, ІП-03	
Н. Контр		Волокита А. М.			Архітектура сервісу (структурна схема)		
Затвердив		Стіренко С. Г.					

ДОДАТОК 2

Сервіс для генерації фізичної мапи планети за допомогою шуму Перлина

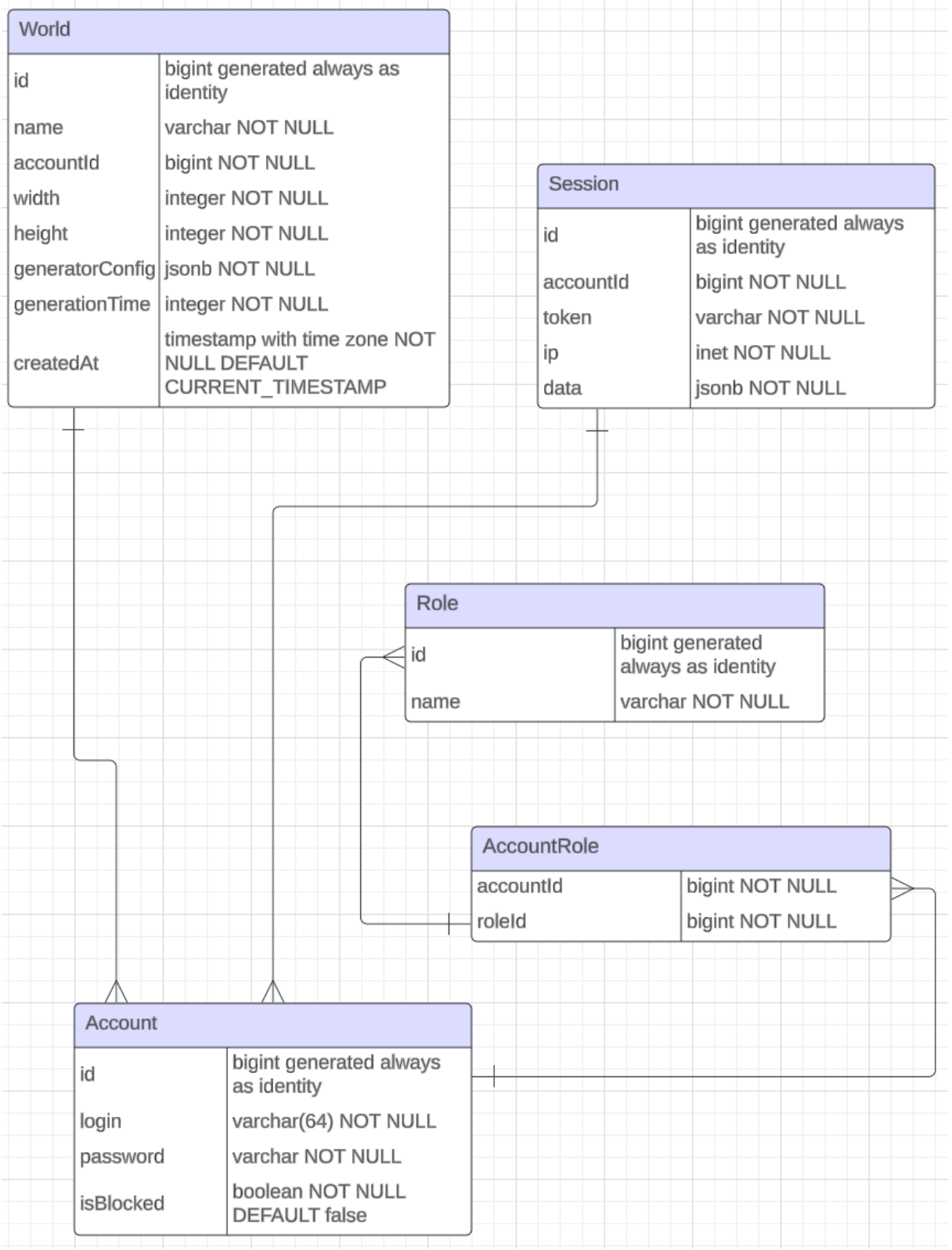
ER-діаграма

(функціональна схема)

ІАЛЦ.467200.003 Д2

Аркушів 1

Київ – 2024



Зм	Лист	№ докум.	Підп.	Дата	ІАЛЦ.467200.003 Д2		
Розроб.		Гуськов Д. І..			Сервіс для генерації фізичної мапи планети за допомогою шуму Перлина ER-діаграма (функціональна схема)	*Літ.*	*Арк.* 1 / *Аркушів* 1
Перев.		Міщенко Л. Д.					
Реценз						КПІ ім. Ігоря Сікорьского, ФІОТ, ІП-03	
Н. Контр		Волокита А. М.					
Затвердив		Стіренко С. Г.					

ДОДАТОК 3

Сервіс для генерації фізичної мапи планети за допомогою шуму Перлина

Алгоритм генерації фізичної

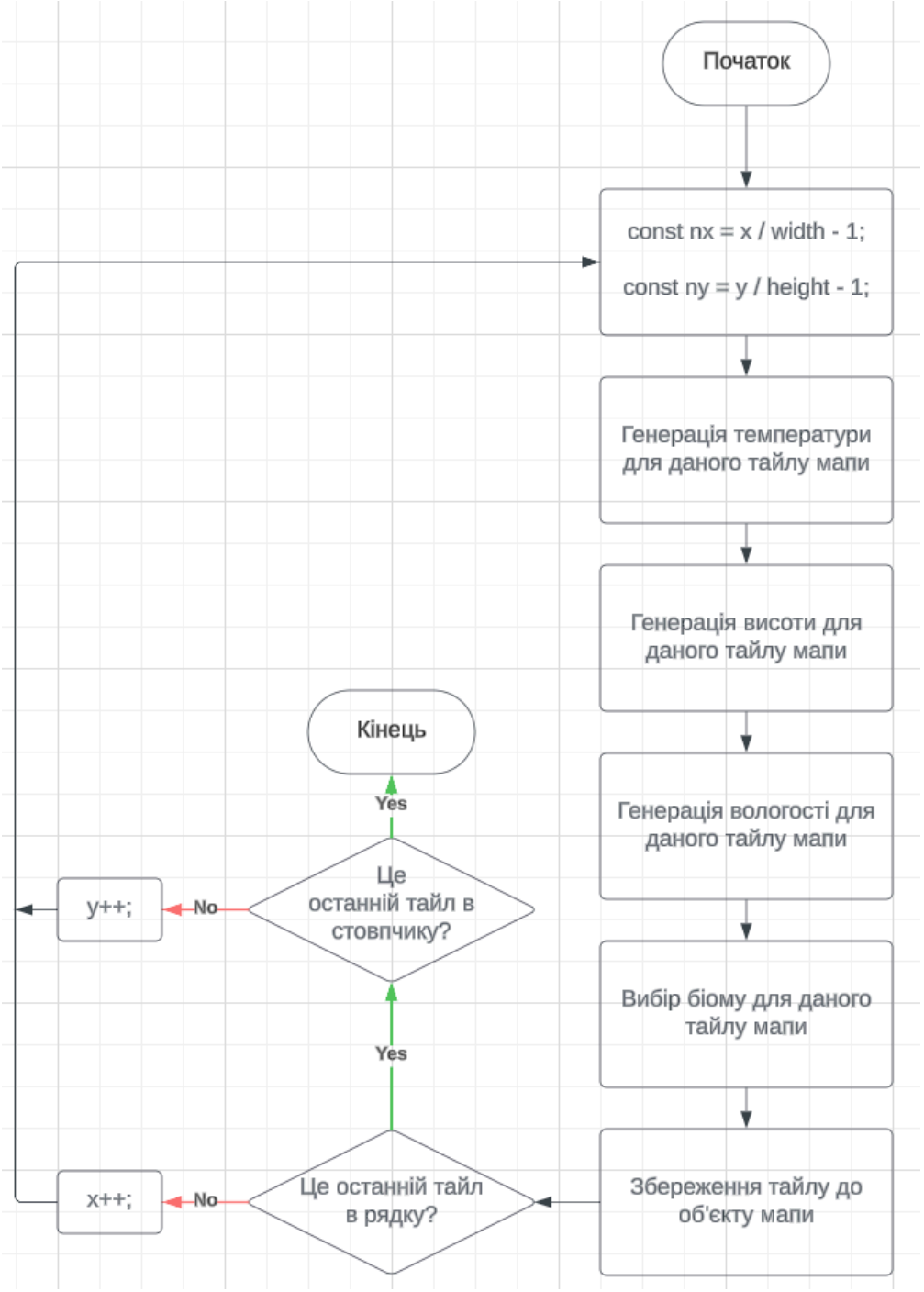
мапи планети

(принципова схема)

ІАЛЦ.467200.003 Д3

Аркушів 1

Київ – 2024



Зм	Лист	№ докум.	Підп	Дата	ІАЛЦ.467200.003 Д3		
Розроб.		Гуськов Д. І..			Сервіс для генерації фізичної мапи планети за допомогою шуму Перлина	*Літ.*	*Арк.* 1 / *Аркушів* 1
Перев.		Міщенко Л. Д.					
Реценз							
Н. Контр		Волокита А. М.					
Затвердив		Стіренко С. Г.			Алгоритм генерації фізичної мапи планети (принципова схема)	КПІ ім. Ігоря Сікорьского, ФІОТ, ІП-03	

ДОДАТОК 4

Сервіс для генерації фізичної мапи планети за допомогою шуму Перлина

Програмний код сервісу

ІАЛЦ.467200.003 Д4

Аркушів 8

Київ – 2024

```
({
    getTile(height, moisture, temperature) {
        // Find biomes that suits for current tile
        const defs = domain.world.biomes.defs
            .filter(({ height: h, moisture: m, temperature: t }) =>
                // filter by terrain height
                (!h ? true : (height >= h[0] && height <= h[1]))
                // filter by moisture value
                && (!m ? true : (moisture >= m[0] && moisture <= m[1]))
                // filter by temperature value
                && (!t ? true : (temperature >= t[0] && temperature <= t[1]))
            );
        const index = Math.floor(Math.random() * defs.length);
        if (!defs[index]) {
            throw new Error('no biome found', {
                height, moisture, temperature
            });
        }
        return {
            biome: defs[index].biome,
            height,
            moisture,
            temperature,
        };
    },

    // width, height - in tiles size
    // return struct of generated world data
    // see more https://www.redblobgames.com/maps/terrain-from-noise/
    generate(width, height, generatorConfig) {
        const wrapNoiseXY = lib.noise.wrapNoiseXY;
        const {
            seed,
            sharpness,
            fudgeFactor,
            temperature,
            waterLevel,
            octaves = [],
            terrainAmplitudes = [],
            moistureAmplitudes = [],
            temperatureAmplitudes = [],
        } = generatorConfig;
        // const tiles = new Array(height).fill(new Array(width).fill({}));
```

```
    const tiles = [];
    const seedHash = node.crypto.createHash('md5').update(seed).digest('hex');
    const octaveSeed1 = parseInt(seedHash.substring(0, 8), 16);
    const octaveSeed2 = parseInt(seedHash.substring(8, 16), 16);
    const { genTerrain, genTemperature, genMoisture } = lib.noise.generator(seed);
    const generationTimeStart = new Date().getTime();

    for (let y = 0; y < height; y++) {
      for (let x = 0; x < width; x++) {
        const nx = x / width - 1;
        const ny = y / height - 1;

        // generate common temperature value in coordinate (x, y)
        const t1 = Math.abs((height / 2 - y) / height * 2) * -2 + 1;

        // generate temperature noise in coordinate (x, y)
        const t2 = temperatureAmplitudes.length === 0
          ? wrapNoiseXY(genTemperature, nx, ny)
          : temperatureAmplitudes.reduce((acc, amplitude, i) =>
            // acc + amplitude * wrapNoiseXY(genTemperature, Math.pow(2, i) * nx,
Math.pow(2, i) * ny), 0
            acc + octaves[i] * wrapNoiseXY(
              genTemperature,
              amplitude * nx + (octaveSeed2 / octaveSeed1) * i,
              amplitude * ny + (octaveSeed2 / octaveSeed1) * i / 2,
              i
            )
          ) / temperatureAmplitudes.reduce((acc, amplitude) => acc + amplitude, 0);

        let t = t1 + t2 + temperature;
        if (Number.isNaN(t)) {
          throw new Error('temperature is NaN', {
            t1, t2, temperature, temperatureAmplitudes
          });
        }
        if (t > 1) t = 1;
        if (t < -1) t = -1;
        // generate terrain height in coordinate (x, y)
        let h = terrainAmplitudes.length === 0
          ? wrapNoiseXY(genTerrain, nx, ny)
          : terrainAmplitudes.reduce((acc, amplitude, i) =>
            // acc + amplitude * wrapNoiseXY(genTerrain, Math.pow(2, i) * nx,
Math.pow(2, i) * ny), 0
            acc + octaves[i] * wrapNoiseXY(
              genTerrain,
              amplitude * nx + (octaveSeed2 / octaveSeed1) * i,
              amplitude * ny + (octaveSeed2 / octaveSeed1) * i / 2,
              i
            )
            , 0
```

```
    ) / terrainAmplitudes.reduce((acc, amplitude) => acc + amplitude, 0);

  // redistribution
  const hCopy = h;
  h = Math.pow(Math.abs(h) * fudgeFactor, sharpness);
  if (Number.isNaN(h)) {
    throw new Error('height is NaN', {
      hCopy, h, terrainAmplitudes
    });
  }
  if (hCopy < 0) h *= -1;
  if (h > 1) h = 1;
  if (h < -1) h = -1;

  // generate common moisture value in coordinate (x, y)
  const mBaseY = Math.abs((height / 2 - y) / height * 2) * -1;
  const m1 = Math.pow(Math.E, -mBaseY) * Math.cos(15 * mBaseY);
  // const m1 = y > 0
  //   ? Math.pow(Math.E, -Math.abs(mBaseY)) * Math.cos(14 * mBaseY)
  //   : Math.pow(Math.E, Math.abs(mBaseY)) * Math.cos(14 * mBaseY);
  const m3 = Math.cos(4 * t);

  // generate moisture value noise in coordinate (x, y)
  const m2 = moistureAmplitudes.length === 0
    ? wrapNoiseXY(genMoisture, nx, ny)
    : moistureAmplitudes.reduce((acc, amplitude, i) =>
      // acc + amplitude * wrapNoiseXY(genMoisture, Math.pow(2, i) * nx,
Math.pow(2, i) * ny), 0
      acc + octaves[i] * wrapNoiseXY(
        genMoisture,
        amplitude * nx + (octaveSeed2 / octaveSeed1) * i,
        amplitude * ny + (octaveSeed2 / octaveSeed1) * i / 2,
        i
      )
    ) / moistureAmplitudes.reduce((acc, amplitude) => acc + amplitude, 0);

  let m = m1 + m2 + m3;
  if (Number.isNaN(m)) {
    throw new Error('moisture is NaN', {
      m, m1, m2, moistureAmplitudes
    });
  }
  if (m > 1) m = 1;
  if (m < -1) m = -1;

  // tiles[x][y] = domain.world.terrain.getTile(h, m, t);
  tiles.push({
    x, y,
    ...domain.world.terrain.getTile(h, m, t)
  });
```

```
            }
        }
        const generationTimeEnd = new Date().getTime();
        return {
            tiles,
            width,
            height,
            generatorConfig,
            generationTime: generationTimeEnd - generationTimeStart,
        };
    },

    // worldData - result from generate method above
    // tileSize - size of tile in pixel (tile is a square)
    // return png of world in buffer
     async draw(worldData, tileSize) {
        const {
            tiles,
            width,
            height,
        } = worldData;

        const widthInPixel = width * tileSize;
        const heightInPixel = height * tileSize;
        const bitmap2d = new Array(widthInPixel).fill(new Array(heightInPixel).fill(0));

        for (let i = 0; i < tiles.length; i++) {
            const tile = tiles[i];
            const x = tile.x * tileSize;
            const y = tile.y * tileSize;
            for (let i = x; i < x + tileSize; i++) {
                for (let j = y; j < y + tileSize; y++) {
                    bitmap2d[i][j] = tile.color;
                }
            }
        }

        const image = new npm.jimp(widthInPixel, heightInPixel, (err, image) => {
            if (err) throw err;
            bitmap2d.forEach((row, y) => {
                row.forEach((color, x) => {
                    image.setPixelColor(color, x, y);
                });
            });
        });

        return image.getBufferAsync(npm.jimp.MIME_PNG);
    }
})
```

```
({
    defs: [
        //////////// Marine ////////////
        {
            biome: 'Glacier',
            height: [-1, 0],
            temperature: [-1, -0.75],
            color: '#65C2FA',
        },
        {
            biome: 'Abyss',
            height: [-1, -0.9],
            temperature: [-0.75, 1],
            color: '#350786',
        },
        {
            biome: 'Deep Water',
            height: [-0.9, -0.7],
            temperature: [-0.75, 1],
            color: '#072A86',
        },
        {
            biome: 'Water',
            height: [-0.7, -0.3],
            temperature: [-0.8, 1],
            color: '#1444C4',
        },
        {
            biome: 'Swallow Water',
            height: [-0.3, 0],
            temperature: [-0.75, 1],
            color: '#1481CC',
        },
        {
            biome: 'Coral Reef',
            height: [-0.35, -0.15],
            temperature: [0.5, 1],
            moisture: [-0.5, 0.5],
            color: '#826BF6',
        },
        //////////// Polar ////////////
        {
            biome: 'Polar desert',
            height: [0, 1],
            temperature: [-1, -0.75],
            color: '#FFFFFF',
        },
        {
            biome: 'Ice peaks',
            height: [0.7, 1],
            temperature: [-1, -0.3],
```

```
    moisture: [-1, -0.5],
    color: '#B4B3B2',
  },
  {
    biome: 'Mountain Tundra',
    height: [0.7, 1],
    temperature: [-0.75, -0.6],
    moisture: [-0.5, 1],
    color: '#4A6E4A',
  },
  {
    biome: 'Alpine Meadow',
    height: [0.7, 0.8],
    temperature: [-0.5, -0.3],
    moisture: [-0.5, 0.5],
    color: '#5C835B',
  },
  {
    biome: 'Polar Plains',
    height: [0, 1],
    temperature: [-0.75, -0.3],
    moisture: [-1, 0.5],
    color: '#9DF841',
  },
  {
    biome: 'Tundra',
    height: [0, 1],
    temperature: [-0.75, -0.6],
    moisture: [-0.5, 1],
    color: '#5C835B',
  },
  {
    biome: 'Taiga',
    height: [0, 1],
    temperature: [-0.6, -0.5],
    moisture: [-0.5, 1],
    color: '#44943D',
  },
  {
    biome: 'Boreal Forest',
    height: [0, 1],
    temperature: [-0.5, -0.3],
    moisture: [-0.5, 1],
    color: '#4AB141',
  },
  ////////////// Subtropical //////////////
  {
    biome: 'Grassland',
    height: [0, 1],
    temperature: [-0.3, 0.5],
    moisture: [-1, -0.5],
```

```
    {
      biome: 'Shrubland',
      height: [0, 1],
      temperature: [-0.3, 1],
      moisture: [-0.5, 0],
      color: '#99F16D',
    },
    {
      biome: 'Woodland',
      height: [0, 1],
      temperature: [-0.3, 0.5],
      moisture: [0, 0.5],
      color: '#4ECE19',
    },
    {
      biome: 'Subtropical Forest',
      height: [0, 1],
      temperature: [-0.3, 0.5],
      moisture: [0.5, 0.8],
      color: '#3CC006',
    },
    {
      biome: 'Subtropical Rainforest',
      height: [0, 1],
      temperature: [-0.3, 0.5],
      moisture: [0.5, 0.9],
      color: '#32A304',
    },
    {
      biome: 'Swamp',
      height: [0, 1],
      temperature: [-0.3, 0.5],
      moisture: [0.5, 1],
      color: '#32A304',
    },
    ////////////// Tropical //////////////
    {
      biome: 'Extreme Desert',
      height: [0, 1],
      temperature: [0.8, 1],
      moisture: [-1, -0.5],
      color: '#FA980A',
    },
    {
      biome: 'Tropical Desert',
      height: [0, 1],
      temperature: [0.5, 0.8],
      moisture: [-1, -0.5],
      color: '#FDED49',
    },
```

```
        {
          biome: 'Savanna',
          height: [0, 1],
          temperature: [0.5, 1],
          moisture: [-0.5, 0],
          color: '#BE9A3C',
        },
        {
          biome: 'Tropical Forest',
          height: [0, 1],
          temperature: [0.5, 1],
          moisture: [0, 0.5],
          color: '#449E1F',
        },
        {
          biome: 'Tropical Rainforest',
          height: [0, 1],
          temperature: [0.5, 1],
          moisture: [0.5, 0.9],
          color: '#2B9002',
        },
        {
          biome: 'Tropical Swamp',
          height: [0, 1],
          temperature: [0.5, 1],
          moisture: [0.9, 1],
          color: '#226E03',
        },
        // {
        //   biome: 'Beach',
        //   height: [0, 0.15],
        //   temperature: [0.5, 1],
        //   color: '#CCCF70',
        // },
        /////////// Generic //////////
    ],
})
```