

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

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри

Сергій СТИРЕНКО

(підпис)

“ ” 2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

на тему: Геоінформаційна система для підтримки обчислень з візуалізації
зруйнованих об’єктів

Виконав : студент 4 курсу, групи ІО-01
(шифр групи)

Гладкий Андрій Іванович

(прізвище, ім’я, по батькові)

(підпис)

Керівник асистент Гончаренко Олександр Олексійович

(посада, науковий ступінь, вчене звання, прізвище, ім’я та по батькові)

(підпис)

Консультант (нормоконтроль) професор, д.т.н., Виноградов Ю.М.

(назва розділу)

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

(підпис)

Рецензент доц., к.т.н., доцент кафедри ІСТ, Шимкович В.М

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(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”

спеціальності 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
Сергій СТИРЕНКО

(підпис)

“ ____ ” _____ 2024 р.

ЗАВДАННЯ

на бакалаврський дипломний проєкт студента

Гладкого Андрія Івановича

1. Тема проєкту Геоінформаційна система для підтримки обчислень з візуалізації зруйнованих об’єктів
керівник проєкту Гончаренко Олександр Олексійович, асистент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 11 травня 2024 року №1139-с
2. Термін здачі студентом закінченого проєкту 30 травня 2024 р.
3. Вихідні дані до проєкту технічна документація, теоретичні дані та інтернет ресурси з питань геоінформаційних систем.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)
Аналіз предметної області
Опис об’єкта проєктування
Деталі розробки системи.

Тестування системи.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) схема структурна, діаграма класів, алгоритм обробки запитів.

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

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Виноградов Ю.М.		

7. Дата видачі завдання «30» серпня 2023 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	<i>Затвердження теми проєкту</i>	<i>16.11.2023-13.12.2023</i>	
2.	<i>Вивчення та аналіз завдання</i>	<i>14.12.2023-19.01.2024</i>	
3.	<i>Розробка архітектури та загальної структури системи</i>	<i>20.01.2024-17.02.2024</i>	
4.	<i>Розробка структур окремих підсистем</i>	<i>18.02.2024-23.03.2024</i>	
5.	<i>Програмна реалізація системи</i>	<i>24.03.2024-21.05.2024</i>	
6.	<i>Оформлення пояснювальної записки</i>	<i>22.05.2024-03.06.2024</i>	
7.	<i>Передзахист</i>	<i>7.06.2024</i>	
8.	<i>Захист</i>	<i>18.06.2024</i>	

Студент-дипломник _____
(підпис)

ГЛАДКИЙ Андрій

Керівник проєкту _____
(підпис)

ГОНЧАРЕНКО Олександр

АНОТАЦІЯ

Метою дипломної роботи є розробка серверної частини геоінформаційної системи, яка спрямована на візуалізацію та аналіз зруйнованих артефактів у результаті воєнних дій на території України.

Головна мета роботи - надання точної та актуальної інформації про масштаби руйнувань, що дозволить залучити більше уваги та ресурсів від міжнародних організацій та західних партнерів для відновлення постраждалих регіонів. Система використовуватиме сучасні алгоритми обробки та представлення даних. У ході виконання роботи будуть розглянуті принципи та методи побудови серверної частини системи та проведений аналіз існуючих рішень, що дозволить обрати відповідні технології та рішення для геоінформаційної системи. Стек технологій Go, Echo, Postgresql, Kubernetes.

Ключові слова: геоінформаційна система, зруйновані об'єкти, серверна частина, Go.

ANNOTATION

The purpose of the thesis is to develop a server-side geographic information system aimed at visualizing and analyzing destroyed artifacts as a result of military operations in Ukraine.

The main goal of the work is to provide accurate and up-to-date information on the extent of the destruction, which will allow attracting more attention and resources from international organizations and Western partners to restore the affected regions. The system will use modern data processing and presentation algorithms. In the course of the work, the principles and methods of building the server part of the system will be considered and existing solutions will be analyzed, which will allow choosing the appropriate technologies and solutions for the geographic information system. Technology stack Go, Echo, Postgresql, Kubernetes.

Keywords: geographic information system, destroyed objects, server part, Go.

№ рядка	Формат	Значення	Найменування	Кіл. листів	№ екземпляра	Додаток
1			Документація загальна			
2			Знову розроблена			
3						
4	A4	ІАЛЦ.467200.002 ТЗ	Геоінформаційна система	3		
5			для підтримки обчислень з			
6			візуалізації зруйнованих			
7			об'єктів			
8			Технічне завдання			
9	A4	ІАЛЦ.467200.003 ПЗ	Геоінформаційна система	48		
10			для підтримки обчислень з			
11			візуалізації зруйнованих			
12			об'єктів			
13			Пояснювальна записка			
14	A4	ІАЛЦ.467200.004 Е1	Геоінформаційна система	1		
15			для підтримки обчислень з			
16			візуалізації зруйнованих			
17			об'єктів			
18			Схема структурна			
19	A4	ІАЛЦ.467200.005 Д2	Геоінформаційна система	1		
20			для підтримки обчислень з			
21			візуалізації зруйнованих			
22			об'єктів			
23			Діаграма класів			
24			Схема функціональна			
25	A4	ІАЛЦ.467200.006 Д3	Геоінформаційна система	1		
26			для підтримки обчислень з			
27			візуалізації зруйнованих			
28			об'єктів			
29			Алгоритм обробки запитів			
30			Схема принципова			
31	A4	ІАЛЦ.467200.007 Д4	Геоінформаційна система	31		
32			для підтримки обчислень з			
33			візуалізації зруйнованих			
34			об'єктів			
35			Текст програмного коду			
36						
37						

					ІАЛЦ.467200.001 ОА							
Зм	Лист	№ докум.	Підп	Дата								
Розроб		Гладкий А.І.			Геоінформаційна система для підтримки обчислень з візуалізації зруйнованих об'єктів Опис альбому			Літ.	Аркуш		Аркушів	
Перев		Гончаренко О.О.							1		1	
								НТУУ КПІ ім. Ігоря Сікорського ФІОТ гр. ІО-01				
Н.Контр.		Виноградов Ю.М.										

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Геоінформаційна система для підтримки обчислень з візуалізації»

Київ – 2024

ЗМІСТ

НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ	4
ПІДСТАВИ ДЛЯ РОЗРОБКИ	4
МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ.....	4
ДЖЕРЕЛА РОЗРОБКИ.....	4
ТЕХНІЧНІ ВИМОГИ.....	5
Вимоги до розробленого продукту	5
Вимоги до програмного забезпечення	5
Вимоги до апаратної частини	5
ЕТАПИ РОЗРОБКИ	5

					ІАЛЦ.467200.006 ТЗ			
		№ докум.	Підпис	Дата				
Розробив	Гладкий А.І.				Геоінформаційна система для підтримки обчислень з візуалізації зруйнованих об'єктів Технічне завдання	Літ.	Аркуш	Аркушів
Перевірив	Гончаренко О.О.						1	4
						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, гр. ІО-01		
Н.Контр.	Виноградов Ю.М.							
Затвердив								

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку геоінформаційної системи для підтримки обчислень з візуалізації зруйнованих об'єктів, в тому числі на її подальше покращення.

Областю застосування цієї системи є моніторинг руйнувань, аналіз даних, планування відновлення, підтримка гуманітарних операцій, навчання та підвищення кваліфікації та залучення громадськості

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання на 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 та призначенням даної роботи є розробка геоінформаційної системи для підтримки обчислень з візуалізації зруйнованих об'єктів.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проєкту є офіційні документації, публікації та статті в мережі Інтернет на дану тему, науково-технічна література.

					ІАЛЦ.467200.003 ПЗ	Арк.
						4
Зм.	Арк.	№ докум.	Підпис	Дата		

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Надати можливість користувачам передавати вхідні дані для пошуку отримання та модифікації зруйнованих об'єктів та зображень
- Надати можливість користувачам власноруч фільтрувати отримувані дані та їх редагувати
- Надати можливість користувачам власноруч завантажувати зображення
- Надати вичерпну та зрозумілу документацію для розробленого продукту.

5.2. Вимоги до програмного забезпечення

- ОС MS Windows 10, Mac OS 10 чи Linux Ubuntu 16 версії або вище.

5.3. Вимоги до апаратної частини

- ЦП не менше ніж Intel® Core (TM) i3-2100T.
- Зовнішня пам'ять не менше ніж 16 ГБ.
- Оперативна пам'ять не менше ніж 8 ГБ.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	16.11.2023-13.12.2023
Вивчення та аналіз завдання	14.12.2023-19.01.2024
Розробка архітектури та загальної структури системи	20.01.2024-17.02.2024
Розробка структур окремих частин системи	18.02.2024-23.03.2024
Програмна реалізація системи	24.03.2024-09.05.2024
Виправлення помилок	10.05.2024-21.05.2024
Оформлення пояснювальної записки	22.05.2024-03.06.2024

					ІАЛЦ.467200.003 ПЗ	Арк.
						5
Зм.	Арк.	№ докум.	Підпис	Дата		

**ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ**

на тему: «Геоінформаційна система для підтримки обчислень з візуалізації»

Київ – 2024

ЗМІСТ

ЗМІСТ	1
ПЕРЕЛІК СКОРОЧЕНЬ.....	1
ВСТУП	2
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	4
1.1 Актуальність та доцільність розроблення геоінформаційних систем з візуалізації зруйнованих об'єктів.....	4
1.2 Роль серверної частини в геоінформаційній системі	5
1.3 Огляд існуючих геоінформаційних систем зруйнованих об'єктів	6
ВИСНОВОК ДО РОЗДІЛУ 1	13
РОЗДІЛ 2. ОПИС ОБ'ЄКТА ПРОЄКТУВАННЯ	15
2.1 Огляд серверних технологій	15
2.2 Вибір бази даних	18
2.3 Контейнеризація серверу	19
2.4 Додаткові технології.....	22
ВИСНОВОК ДО РОЗДІЛУ 2	23
РОЗДІЛ 3. ДЕТАЛІ РОЗРОБКИ СИСТЕМИ.....	24
3.1 Архітектура серверної частини	24
3.1.1. REST API та Router з Middleware.....	24
3.1.2 Handler.....	24

					ІАЛЦ.467200.003 ПЗ			
		№ докум.	Підпис	Дата	Геоінформаційна система для підтримки обчислень з візуалізації зруйнованих об'єктів	Літ.	Аркуш	Аркушів
Розробив	Гладкий А.І.							
Перевірів	Гончаренко О.О.						1	48
Н.Контр.	Виноградов Ю.М.					НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, гр. ІО-01		
Затвердив								
					Пояснювальна записка			

3.1.3. Service.....	24
3.1.4 Repository	25
3.1.5 Model	25
3.2 Проєктування моделей	25
3.2.1 Зруйнований об'єкт	25
3.2.2 Фото зруйнованого об'єкта	26
3.3 Створення моделей	27
3.4 Вимоги від клієнтської частини	28
3.4.1 Зруйновані об'єкти.....	28
3.4.2 Зображення.....	32
3.5 Обробники запитів, сервіси та репозиторії	33
3.6. Контейнеризація і розгортання додатку	35
ВИСНОВОК ДО РОЗДІЛУ 3	37
РОЗДІЛ 4. ТЕСТУВАННЯ СИСТЕМИ	38
4.1 Керівництво користувача	38
4.2 Тестування серверної частини.....	38
ВИСНОВОК ДО РОЗДІЛУ 4	44
ВИСНОВКИ.....	45
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	46
ДОДАТОК 1. СХЕМА СТРУКТУРНА.....	46
ДОДАТОК 2. ДІАГРАМА КЛАСІВ. СХЕМА ФУНКЦІОНАЛЬНА.....	48
ДОДАТОК 3. АЛГОРИТМ ОБРОБКИ ЗАПИТІВ. СХЕМА ПРИНЦИПОВА .	50
ДОДАТОК 4. ТЕКСТ ПРОГРАМНОГО КОДУ.....	52

ПЕРЕЛІК СКОРОЧЕНЬ

HTML	HyperText Markup Language (Мова розмітки гіпертексту)
CSS	Cascading Style Sheets (Каскадні таблиці стилів)
HTTP	HyperText Transfer Protocol (Протокол передачі гіпертексту)
3D	Three-Dimensional (Тривимірний)
ORM	Object-Relational Mapping (Об'єктно-реляційне відображення)
ГІС	Географічна інформаційна система
GIS	Geographic Information System (Географічна інформаційна система)
JSON	JavaScript Object Notation (Позначення об'єктів у JavaScript)
SQL	Structured Query Language (Мова структурованих запитів)
NoSQL	No Structured Query Language (без Мови структурованих запитів)

ВСТУП

Розроблення геоінформаційної системи (ГІС) для візуалізації зруйнованих об'єктів є важливим та актуальним завданням у контексті відновлення України під час збройної агресії Росії. Дана дипломна робота обґрунтовує необхідність створення такої системи, аналізує існуючі аналоги, детально описує технічні аспекти розробки та тестування серверної частини веб-застосунку.

- **Актуальність та доцільність розроблення ГІС:** Викликані війною руйнування вимагають нових підходів до документації, оцінки збитків, планування відновлення та залучення міжнародної допомоги. Геоінформаційні системи стають ключовим інструментом для зберігання та аналізу даних про зруйновані об'єкти, надаючи можливість структурованого збереження інформації та хронологічного порівняння стану об'єктів.
- **Роль серверної частини:** Буде розроблено серверну частину веб-застосунку, що забезпечить ефективну обробку та збереження великих обсягів інформації.
- **Аналіз існуючих систем:** Буде проведено аналіз існуючих ГІС, виявлено переваги та недоліки кожної з них. Це дасть змогу визначити напрямки для вдосконалення розроблюваної системи.
- **Технічні аспекти розробки:** Буде оглянуто інструменти, такі як мова програмування, веб-фреймворк. Оглянуто способи зберігання географічних даних та зображень.
- **Тестування системи:** Буде проведено відповідне тестування основних функціональних можливостей системи та її відповідність поставленим вимогам.

					ІАЛЦ.467200.003 ПЗ	Арк.
						4
Зм.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність та доцільність розроблення геоінформаційних систем з візуалізації зруйнованих об'єктів

Російська збройна агресія проти України внесла зміни в життя мільйонів українців за даними директора ДП “Державний науково-дослідний інститут будівельних конструкцій”, доктора технічних наук, професора Геннадія Фаренюка станом на 31 жовтня 2023 року зруйновано або пошкоджено 160 тисяч будівель. У свою чергу Міністерство культури та інформаційної політики станом на 9 січня повідомило, що з них, – 872 об'єкти культурної спадщини[1, 2].

Разом із цим з'явилась необхідність:

- документації та оцінки масштабів руйнування об'єктів архітектури;
- планування відновлення об'єктів;
- залучення міжнародної уваги та гуманітарної допомоги на відновлення об'єктів;
- забезпечення правового та соціального регулювання;
- проведення наукових досліджень та аналізу руйнувань;

Документування руйнувань є важливим для фіксації воєнних злочинів. Збір доказів для міжнародних судів та трибуналів, які можуть використати дані для притягнення до відповідальності. Оцінка в цьому контексті дозволяє справедливо визначити покарання для злочинців та запросити відшкодування збитків.

Геоінформаційна система дозволяє в структурованому вигляді зберігати координати, документи, фотографії та іншу інформацію про архітектурні об'єкти. Також система дозволяє зберігати інформацію в хронологічному

					ІАЛЦ.467200.003 ПЗ	Арк.
						5
Зм.	Арк.	№ докум.	Підпис	Дата		

порядку, що дозволяє порівнювати об'єкти до і після руйнувань.

Планування відновлення зруйнованих об'єктів є не менш важливим завданням геоінформаційної системи. Держава та міжнародні установи можуть точно оцінити збитки, спланувати кошторис та час відновлення. За рахунок детального документування можна відслідковувати ефективність та якість проведеного відновлення. У випадку порушень проведення робіт,— навести докази.

Також варто не забувати про соціальний та культурний вплив таких систем. Вони дозволяють задокументувати руйнування соціально та культурно важливих об'єктів, звертати увагу до них, що також збільшує обсяги залучення допомоги[3, 4].

1.2 Роль серверної частини в геоінформаційній системі

Розроблювана геоінформаційна система матиме вигляд веб-застосунку, як і більшість веб-застосунків, вона складатиметься з клієнтської та серверної частин.

Клієнтська частина – це частина веб-застосунку, з якою взаємодіє і яку бачить користувач. Доступ до цієї частини здійснюється за допомогою браузера. Здебільшого використовується поєднання технологій HTML, CSS та JavaScript. Серверна частина – це частина веб-застосунку, яка відповідає за дані, збереження, обробку та доступ до них. Клієнтська частина відправляє запити до серверної частини на отримання даних, і серверна частина їх обробляє. Здебільшого запити йдуть через HTTP, обробляє їх сервер, який написаний на PHP, Java, Python, Javascript, Go чи будь-якій іншій мові програмування[5].

В даній роботі розглядається серверна частина, тому її розглянемо детальніше.

Збереження даних та доступ до них у геоінформаційній системі

Популярна архітектура серверної частини веб-застосунку виглядає як файлова система, реляційна база даних і сервер, який обробляє запити від клієнту.

					ІАЛЦ.467200.003 ПЗ	Арк.
						6
Зм.	Арк.	№ докум.	Підпис	Дата		

Та для геоінформаційних систем, які працюють з великими об'ємами даних різних типів, є необхідність створення додаткових архітектурних рішень, що нададуть більшу гнучкість та швидкість системі та дозволять надати користувачам великі об'єми даних у зручному вигляді.

У нашому випадку до файлової системи, бази даних і серверу буде доцільно:

- створити окреме сховище для зображень;
- контейнеризувати сервер, що дасть можливість працювати декільком екземплярів серверу одночасно і розподіляти навантаження;
- створити окреме сховище даних, для збереження детальної інформації про кожен об'єкт.

1.3 Огляд існуючих геоінформаційних систем зруйнованих об'єктів

Розроблювана геоінформаційна система має декілька аналогів:

- 1. Reukraine.shtab.net**
- 2. Misto.lun.ua**
- 3. Magneticonemt.com**

Розглянемо кожну систему окремо, виділимо переваги та недоліки, підведемо підсумки, визначимо чим розроблювана система вирізнятиметься з поміж інших.

Reukraine.shtab.net

Геоінформаційна система розроблена «Антикорупційним штабом» за підтримки Міжнародного фонду «Відродження», її місія, – «Забезпечити прозоре та підзвітне відновлення усіх зруйнованих та пошкоджених внаслідок загарбницької війни росії проти України об'єктів, залучити кошти на відбудову, моніторити їх використання і сприяти ефективній відбудові України.» [1, 6]

					ІАЛЦ.467200.003 ПЗ	Арк.
						7
Зм.	Арк.	№ докум.	Підпис	Дата		

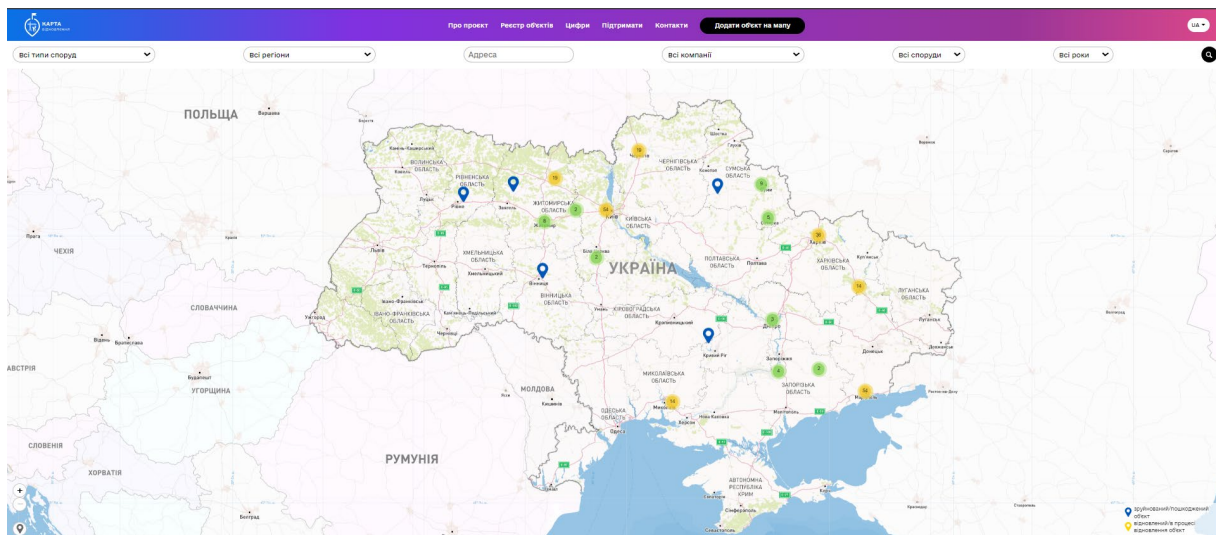


Рисунок 1.1 – Інтерактивна мапа «Карти відновлення» [6]

РЕЕСТР ОБ'ЄКТІВ

Що таке Реєстр об'єктів?

У Реєстрі «Карта Відновлення та Руїнувань» містяться дані про зруйновані/пошкоджені та відновлені об'єкти інфраструктури, які стали такими внаслідок російської агресії. Також, загальнодоступними та відкритими є дані щодо об'єктів, які наразі в процесі реконструкції.

[Розгорнути](#)

Всі типи споруд Всі регіони Адреса Всі компанії Всі споруди Всі роки [Додати об'єкт на мапу](#)

Показано 1 - 50 рядків. Всього 524

Показувати по: 50

Тип об'єкту	Регіон	Адреса	Дата руйнування	Вартість відновлення	Дія
Житлові будинки(багатоповерхівки/приватні будинки/гуртожитки)	Харківська область	м. Харків вул. Пушкінська 40	01 квітня 2022	-	Детальніше
Житлові будинки(багатоповерхівки/приватні будинки/гуртожитки)	Миколаївська область	Вулиця Артемієвська 9 місто Миколаїв	01 квітня 2022	-	Детальніше
Об'єкти культурної спадщини, культурні, архітектурні та ін)	Київська область	вулиця Терещенківська, 15, Київ, 01004	01 квітня 2022	-	Детальніше
Промисловість і об'єкти ЖК(енергетика, промисловість, комбінати та ін)	Миколаївська область	Вулиця Адміральська 38 Миколаїв	01 квітня 2022	1 €	Детальніше
Освітні заклади(дитячі садки, школи, університети, інститути)	Сумська область	Вул. Герасима Кондратьєва 105 Суми	01 квітня 2022	-	Детальніше
Релігійні об'єкти	Харківська область	вул. Сумська, 30, сквер Перемоги, м. Харків, Харківська область	01 квітня 2022	-	Детальніше
Адміністративні будівлі	Київська область	вулиця Володимирська 97 Київ 02000	01 квітня 2022	-	Детальніше
Об'єкти культурної спадщини, культурні, архітектурні та ін)	Київська область	Велика Васильківська, 103, Київ, 03150	01 квітня 2022	-	Детальніше
Житлові будинки(багатоповерхівки/приватні будинки/гуртожитки)	Київська область	м. Васильків вул. Декабристів 42	01 квітня 2022	-	Детальніше

Рисунок 1.2 – Реєстр об'єктів «Карти відновлення» [6]

Переваги:

- реєстр зберігає інформацію щодо всіх об'єктів України
- реєстр об'єктів з розширеними фільтрами;
- інтерактивна мапа України з позначенням кількості пошкоджених об'єктів за кластерами різних розмірів (областям, містах, селах, вулицях) та можливості фільтрувати об'єкти на мапі;
- можливість агрегації статистичних даних, щодо об'єктів;
- можливість додати власні об'єкти через просту форму;

Недоліки:

- реєстр зберігає тільки плоскі зображення;

- інтерактивна мапа не використовує технологію street view;
- відсутня можливість перегляду об'єктів у 3D;
- обмежений опис об'єкту;
- відсутня можливість редагування шарів мапи.

Misto.lun.ua

ЛУН Місто – організація, що займається розробкою інтерактивних мап, що сприяють сталому розвитку. Одною з таких мап є карта руйнувань Київщини, розглянемо її переваги та недоліки [7].

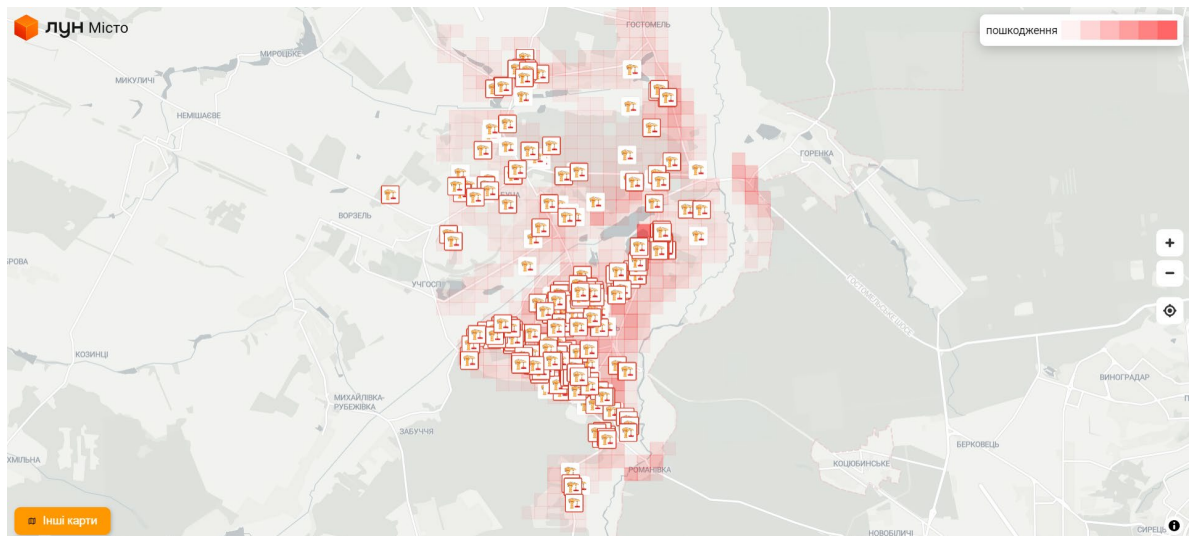


Рисунок 1.3 – Інтерактивна мапа «Руйнування Київщини» [7]

Переваги:

- інтерактивна мапа Київщини з градієнтним підсвіченням важкості пошкоджень;
- можливість моніторингу об'єктів;
- можливість перегляду об'єктів у 3D;
- детальний опис об'єктів

Недоліки:

- відсутній зручний реєстр з фільтрами по зруйнованим будинкам;
- відсутні фільтри на мапі;
- відсутня агрегація даних об'єктів;
- система обмежена Київською областю;

- не підтримується технологія street view;
- зберігаються тільки плоскі зображення;
- немає перевірки правдивості зображень, хоча є можливість повідомити про неточність;
- відсутній функціонал для додавання об'єктів, додавання об'єктів контролюється адміністраторами системи;
- відсутня можливість редагування шарів мапи;

Magneticone.mt.com

MagneticOne Municipal Technologies – компанія, що займається розробкою систем для муніципалітетів .

Однією з таких розробок є інтерактивна карта руйнувань регіону[8]. Саме її і розглянемо.

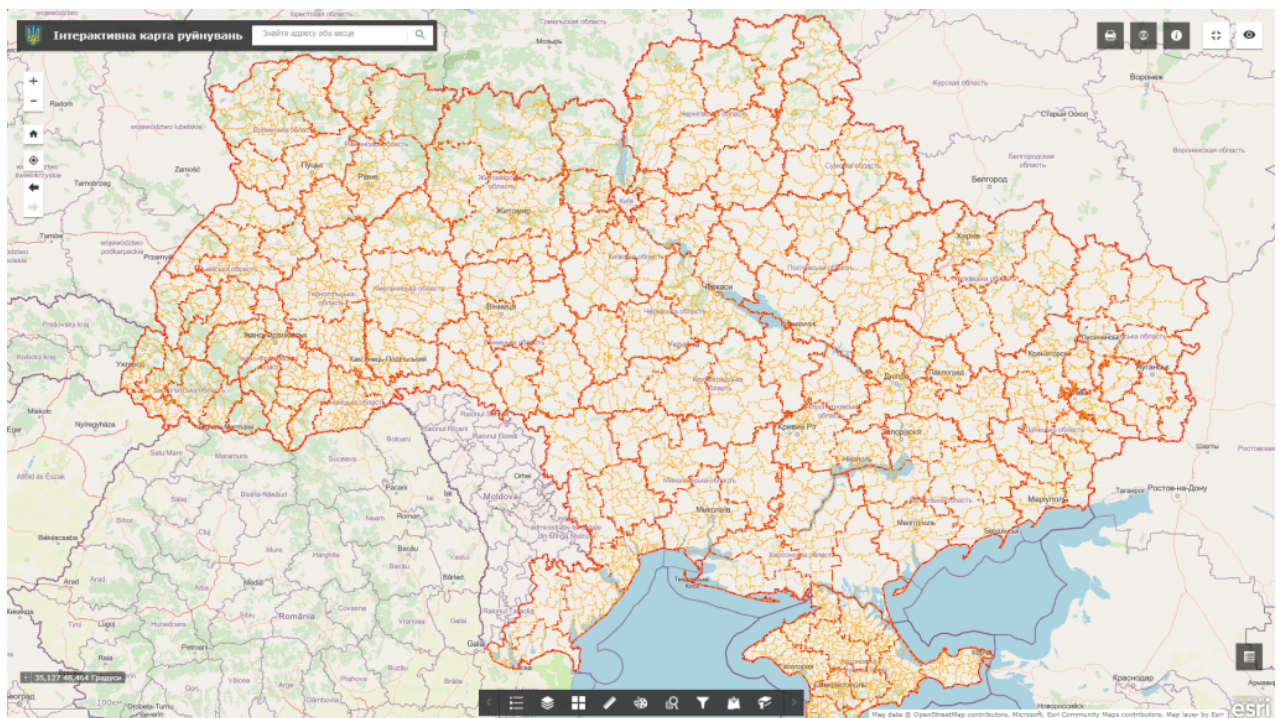


Рисунок 1.4 – Інтерактивна карта руйнувань регіону MagneticOne [8]

Переваги:

- інтерактивна мапа України, що відображає типи зруйнованих об'єктів та має фільтри для об'єктів;
- дашборд з реєстром об'єктів, таблицями та діаграми з статистичними

					ІАЛЦ.467200.003 ПЗ	Арк.
						10
Зм.	Арк.	№ докум.	Підпис	Дата		

даними.

- детальна характеристика об'єкту;
- можливість редагування шарів мапи;
- зручна система аудитів;
- статистика за областями з фільтрами;

Недоліки:

- система створюється для муніципалітетів, має обмежений доступ і є комерційним продуктом;
- не підтримується технологія street view;
- зберігаються тільки плоскі зображення;
- відсутня можливість перевірки точності зображень;
- відсутній функціонал для додавання об'єктів, додавання об'єктів контролюється адміністраторами системи;
- відсутня можливість перегляду об'єктів у 3D;

Підсумки

- всі системи використовують інтерактивні мапи для відображення об'єктів, наша система не виключення;
- дві з трьох систем надають різного роду статистичні візуалізації, у нашій системі буде надана можливість відображення налаштованих діаграм та таблиць;
- дві з трьох систем мають реєстр об'єктів, з відповідними фільтрами та детальною інформацією по кожному об'єкту, це важливий функціонал, його теж потрібно реалізувати;
- всі системи не підтримують 360 зображення, відео чи технологію street view, наша система має надати можливість завантажувати 360 зображення та переміщуватись між ними за допомогою street view, підтримка відео в такому випадку не доцільна;
- одна з систем надає можливість перегляду 3D макетів, але надання такого

					ІАЛЦ.467200.003 ПЗ	Арк.
						11
Зм.	Арк.	№ докум.	Підпис	Дата		

функціоналу не обов'язкове за умови використання комбінації 360 зображень та street view.

Отже, розроблювана система буде частково поєднувати функціонал трьох систем та буде надавати новий функціонал, з концентрацією на візуальній складові

					ІАЛЦ.467200.003 ПЗ	Арк.
						12
Зм.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВОК ДО РОЗДІЛУ 1

- 1. Актуальність розроблення геоінформаційних систем:** У зв'язку з російською агресією проти України, що призвела до значних руйнувань, існує критична потреба у документації, оцінці масштабів збитків, плануванні відновлення, залученні міжнародної уваги та допомоги, а також у правовому та соціальному регулюванні. Геоінформаційні системи відіграють ключову роль у цих процесах, забезпечуючи структуроване зберігання даних та їх хронологічне порівняння.
- 2. Роль серверної частини:** Серверна частина веб-застосунку відповідає за обробку даних та їх збереження. Вона має бути здатною ефективно обробляти великі об'єми інформації, що вимагає від розробників створення гнучких архітектурних рішень, таких як окремі сховища для зображень, контейнеризація серверів та спеціалізовані сховища даних.
- 3. Огляд існуючих систем:** Аналіз існуючих геоінформаційних систем виявив, що більшість з них використовують інтерактивні мапи та надають статистичні візуалізації. Однак, існують прогалини, такі як відсутність 3D-перегляду, обмежений опис об'єктів, відсутність перевірки достовірності зображень та обмежений доступ до додавання нових об'єктів. Нова система повинна вирішити ці недоліки, надаючи можливість завантаження 360-градусних зображень та інтеграції з технологією street view.

Поставлені задачі:

- Розробити систему з можливістю інтеграції з street view та 3D-візуалізацією.
- Забезпечити систему інструментами для перевірки достовірності інформації.
- Впровадити функціонал для легкого додавання та редагування

					ІАЛЦ.467200.003 ПЗ	Арк.
						13
Зм.	Арк.	№ докум.	Підпис	Дата		

об'єктів користувачами.

- Надати доступ до статистичних даних з можливістю налаштування візуалізацій.
- Забезпечити систему регламентованим доступом для збереження конфіденційної інформації.

					ІАЛЦ.467200.003 ПЗ	Арк.
						14
Зм.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2.

ОПИС ОБ'ЄКТА ПРОЄКТУВАННЯ

У даному розділі детально розглянемо, які технології будуть використані в серверній частині веб-застосунку. Технології поділимо на декілька категорій:

- Сервер;
- База даних;
- Контейнеризація;
- Додаткові технології.

2.1 Огляд серверних технологій

Далі буде детально описано роботу алгоритму косяка риб та генетичного алгоритму.

Важливо обрати найкращу технологію для серверної частини, щоб стек технологій не обмежували розробників, а навпаки давали повну свободу і дозволили повністю реалізувати необхідний функціонал. Сконцентруємось не на виборі мови, а конкретному фреймворці хоча й розділимо їх за мовами програмування

PHP

Laravel

Переваги:

- Laravel має інтуїтивний синтаксис та багато інструментів, які спрощують розробку.
- Велика кількість доступних пакетів та гарна документація.
- Міграції, ORM (Eloquent) та зручні інструменти для побудови запитів [9].

Недоліки:

					ІАЛЦ.467200.003 ПЗ	Арк.
						15
Зм.	Арк.	№ докум.	Підпис	Дата		

- В порівнянні з іншими фреймворками повільніший.
- Для великих проєктів може вимагати більше ресурсів і зусиль для підтримки продуктивності.

Python

Django

Переваги:

- Поставляється з великою кількістю вбудованих інструментів.
- Добре підходить для швидкої розробки і великих проєктів.
- Містить вбудовані засоби для захисту від поширених загроз.

Недоліки:

- Може бути не такою гнучкою, як мікрофреймворки.
- Може бути складно інтегрувати з іншими інструментами або фреймворками [10]

Flask

Переваги:

- Дуже легкий та простий для початку роботи.
- Легко налаштовується під конкретні потреби проєкту.
- Ідеально підходить для створення прототипів та малих проєктів.

Недоліки:

- Багато речей потрібно налаштовувати вручну.
- Може бути складним у підтримці при зростанні проєкту [10].

JavaScript

Express.js

Переваги:

- Легкий та швидкий фреймворк для Node.js.
- Відкритий до інтеграції з іншими бібліотеками та інструментами.
- Величезна кількість модулів та підтримка спільноти.

Недоліки:

					ІАЛЦ.467200.003 ПЗ	Арк.
						16
Зм.	Арк.	№ докум.	Підпис	Дата		

- Можливо, доведеться додатково налаштовувати багато речей.
- Може вирости занадто велика і заплутана в великих проєктах[11].

Next.js

Переваги:

- Добре підходить для SEO та швидкого завантаження сторінок.
- Вбудована підтримка багатьох сучасних функцій React.
- Може використовуватися як повний фреймворк для бекенду та фронтенду.

Недоліки:

- Може бути складнішим для налаштування у порівнянні з іншими рішеннями.
- Обмежений екосистемою React[12].

Go

Echo

Переваги:

- Дуже швидкий і ефективний завдяки Go.
- Інтуїтивний синтаксис і хороша документація.
- Добре справляється з великою кількістю одночасних запитів.

Недоліки:

- Менше доступних модулів у порівнянні з іншими фреймворками.
- Go порівняно з іншими мовами має вищий поріг входу[13, 14].

Вибір фреймворку для ГІС залежить від конкретних вимог проєкту. Якщо потрібна швидка розробка з багатим функціоналом і готовими рішеннями, Django або Laravel можуть бути хорошим вибором. Якщо ж важливі продуктивність і масштабованість, варто звернути увагу на Echo. Для гнучкості та інтеграції з сучасними фронтенд-інструментами можуть підійти Express.js або Next.js.

					ІАЛЦ.467200.003 ПЗ	Арк.
						17
Зм.	Арк.	№ докум.	Підпис	Дата		

У нашому випадку важлива продуктивність та масштабованість, тому обрано фреймворк Echo, хоча можна було б обрати будь-який інший Go фреймворк (Gin, mux, fiber, tinygo та ін.) або взагалі використати вбудовану бібліотеку net/http, адже більшість Go фреймворків концентруються на продуктивності, але надають різну обгортку.

2.2 Вибір бази даних

Вибір бази даних для геоінформаційних систем (ГІС) залежить від конкретних потреб системи, таких як обсяги даних, продуктивність, масштабованість, можливості просторового аналізу та інтеграція з іншими інструментами. Ось кілька популярних баз даних, які часто використовуються для ГІС:

PostGIS (на основі PostgreSQL)

Розширення для PostgreSQL, яке додає підтримку географічних об'єктів.

Переваги:

- Відкрите програмне забезпечення з потужними просторовими функціями.
- Підтримка складних геометричних і географічних операцій.
- Інтеграція з різними ГІС-інструментами (наприклад, QGIS, GeoServer).

Недоліки:

- Може бути складним у налаштуванні для початківців.

MySQL з розширенням для просторових даних

MySQL підтримує просторові дані з допомогою просторових розширень.

Переваги:

- Легкість у використанні та налаштуванні.
- Добре підходить для менш складних просторових задач.

Недоліки:

					ІАЛЦ.467200.003 ПЗ	Арк.
						18
Зм.	Арк.	№ докум.	Підпис	Дата		

- Обмежені можливості просторового аналізу в порівнянні з PostGIS.

Microsoft SQL Server з функціями просторових даних

Комерційна реляційна СУБД з підтримкою просторових даних.

Переваги:

- Підтримка великих корпоративних рішень.
- Інтеграція з продуктами Microsoft.
- Висока продуктивність і надійність.

Недоліки:

- Вартість ліцензії[15]

Spatialite (на основі SQLite)

Розширення для SQLite, яке додає підтримку просторових даних.

Переваги:

- Легка, самодостатня база даних.
- Відкрите програмне забезпечення.
- Добре підходить для невеликих проєктів і мобільних додатків.

Недоліки:

- Обмежені можливості у порівнянні з більш масштабними СУБД[16].

MongoDB з GeoJSON

Документоорієнтована NoSQL база даних з підтримкою GeoJSON.

Переваги:

- Гнучкість NoSQL підходу.
- Добре підходить для роботи з неструктурованими даними.

Недоліки:

- Може бути складним для виконання складних просторових запитів[17].
- З вищеперелічених систем найкраще підходить PostGIS, адже:
 - не має проблем з масштабуванням;
 - має величезну кількість інтеграцій для GIS;
 - безкоштовна.

					ІАЛЦ.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		19

2.3 Контейнеризація серверу

Існує багато різних оркестраторів контейнерів, розглянемо найпопулярніші та оберемо один із них. Kubernetes є одним з найпопулярніших оркестраторів контейнерів і може бути відмінним вибором для розгортання та управління ГІС-додатками, але є й інші варіанти, які також варто розглянути. Ось кілька ключових варіантів контейнеризації для ГІС-додатків:

Kubernetes

Система оркестрації контейнерів з широким спектром можливостей для масштабування, автоматичного відновлення, балансування навантаження та інших функцій.

Переваги:

- Висока масштабованість і надійність.
- Активна спільнота та широкий спектр інтеграцій.
- Підтримка різних провайдерів хмарних послуг (AWS, Google Cloud, Azure тощо).
- Автоматичне масштабування та балансування навантаження.

Недоліки:

- Високий рівень складності налаштування та управління.
- Потребує значних ресурсів для запуску та моніторингу.

Docker Swarm

Вбудована оркестрація Docker, яка забезпечує просте розгортання та управління контейнерами.

Переваги:

- Легше в налаштуванні та використанні в порівнянні з Kubernetes.
- Інтегровано з Docker, що спрощує процес розгортання контейнерів.
- Добре підходить для менших середовищ і простих сценаріїв.

Недоліки:

					ІАЛЦ.467200.003 ПЗ	Арк.
						20
Зм.	Арк.	№ докум.	Підпис	Дата		

- Менш потужний і гнучкий у порівнянні з Kubernetes.
- Обмежені можливості масштабування для великих систем.

OpenShift

Платформа для управління контейнерами на базі Kubernetes, розроблена Red Hat.

Переваги:

- Забезпечує додаткові інструменти для розробки, безпеки та управління життєвим циклом додатків.
- Підтримка корпоративного рівня.
- Інтеграція з різними CI/CD інструментами.

Недоліки:

- Більш складний і може потребувати більше ресурсів для розгортання та підтримки.
- Комерційне рішення, що може включати додаткові витрати.

Nomad

Окрім контейнерів, Nomad може керувати будь-якими типами робочих навантажень, включаючи віртуальні машини та програми без контейнерів.

Переваги:

- Простий у налаштуванні та використанні.
- Гнучкість у підтримці різних типів робочих навантажень.

Недоліки:

- Менш розповсюджений у порівнянні з Kubernetes, менша спільнота.
- Менш інтегрований з екосистемою контейнеризації.

Для більшості ГІС-додатків Kubernetes буде найкращим вибором завдяки своїй потужності, гнучкості та широкому спектру підтримуваних функцій.

					ІАЛЦ.467200.003 ПЗ	Арк.
						21
Зм.	Арк.	№ докум.	Підпис	Дата		

Також, багато хмарних провайдерів підтримують Kubernetes Engine, що дозволить використати їх потужності[18, 19].

2.4 Додаткові технології

У першому розділі було зауважено, що є необхідність зберігання зображень.

Для цього прекрасно підходять Storage Buckets(від Google) чи S3 Storages (від Amazon) чи будь-яка інша схожа технологія/технології. У нашому випадку скористаємось Storage Buckets від Google[20, 21, 22].

					ІАЛЦ.467200.003 ПЗ	Арк.
						22
Зм.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВОК ДО РОЗДІЛУ 2

У процесі проектування серверної частини веб-застосунку було розглянуто різноманітні технології, які можуть бути застосовані для досягнення оптимальної продуктивності та масштабованості. Вибір фреймворку Echo для мови програмування Go є виправданим, оскільки він забезпечує високу швидкість обробки запитів та легкість у використанні, що є критично важливим для ГІС-додатків.

Для бази даних було обрано PostGIS, яка є розширенням PostgreSQL, забезпечуючи потужні просторові функції та відмінну інтеграцію з іншими ГІС-інструментами. Це рішення підтримує необхідність у складних геометричних та географічних операціях, що є незамінним для ефективної роботи з геоданими.

У сфері контейнеризації Kubernetes виокремлюється як найбільш підходящий оркестратор контейнерів, завдяки своїй масштабованості, надійності та підтримці багатьох хмарних провайдерів. Це дозволяє створювати високодоступні та гнучкі системи, які можуть ефективно масштабуватися відповідно до потреб користувачів.

Для зберігання зображень та детальної інформації, окремо від бази даних, було обрано S3 Storages від Amazon, що забезпечує надійне та масштабоване рішення для управління великими обсягами даних.

Загалом, вибір технологій для серверної частини веб-застосунку є важливим етапом розробки ГІС. Масштабованість та інтеграція з сучасними інструментами є важливим, адже це забезпечить стабільну та ефективну роботу ГІС-додатків, а також дозволить легко адаптуватися до змінних вимог та розширювати функціонал у майбутньому.

					ІАЛЦ.467200.003 ПЗ	Арк.
						23
Зм.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3.

ДЕТАЛІ РОЗРОБКИ СИСТЕМИ

Перейдемо до розгляду архітектури, проектування та розробки серверної частини.

3.1 Архітектура серверної частини

У даній роботі було використано The Clean Architecture[23].

3.1.1. REST API та Router з Middleware

Шар REST API є точкою входу в додаток. Клієнт надсилає на різні кінцеві точки HTTP-запити з різними заголовками та різними методами

Router (Маршрутизатор) відповідає за направлення вхідних HTTP-запитів від клієнта до відповідного handler'a на основі URL-запиту та методу (GET, POST, PUT, DELETE).

Middleware використовується для обробки запитів до їх передачі обробнику запитів. Здебільшого middleware вважається частиною маршрутизатора, але на відміну від маршрутизатора відповідає не тільки за направлення запиту, але й за його обробку, може відповідати за валідацію запитів, аутентифікацію, логування, у випадку Go за відновлення з панік і т.п.

3.1.2 Handler

Handler'и (Обробники запитів) виступають посередниками між маршрутизатором та сервісним шаром. Вони обробляють вхідні запити, взаємодіють із сервісним шаром для виконання бізнес-логіки та повертають відповіді маршрутизатору. Обробники запитів зазвичай:

- Парсять і валідують дані запиту.
- Викликають необхідні функції сервісу.
- Форматують і відправляють відповіді.

3.1.3. Service

					ІАЛЦ.467200.003 ПЗ	Арк.
						24
Зм.	Арк.	№ докум.	Підпис	Дата		

Сервісний шар містить основну бізнес-логіку додатку. Він виконує операції, такі як обробка та маніпуляція даними, зрідка валідує дані. Сервісний шар незалежний від веб-фреймворка, що полегшує його тестування та підтримку. Сервіси взаємодіють із шаром репозиторію для отримання або збереження даних.

3.1.4 Repository

Шар репозиторію відповідає за збереження і отримання даних. Він абстрагує джерело даних, надаючи уніфікований інтерфейс для сервісного шару. Репозиторій взаємодіє як з базами даних так і з іншими системами зберігання даних (наприклад, PostgreSQL, Google Cloud Storage чи BigQuery). Для роботи з базою даних обрано бібліотеку sqlx, для роботи зі сховком використано офіційну бібліотеку від Google go/storage. Шар репозиторію зазвичай:

- Виконує запити для отримання, вставки, оновлення або видалення даних.
- Мапує записи бази даних на структури Go (моделі).

3.1.5 Model

Моделі – це структури Go, що представляють дані, які використовуються в додатку – від обробника запитів до репозиторія. Вони визначають схему для даних, що зберігаються в базі даних або обробляються в додатку.

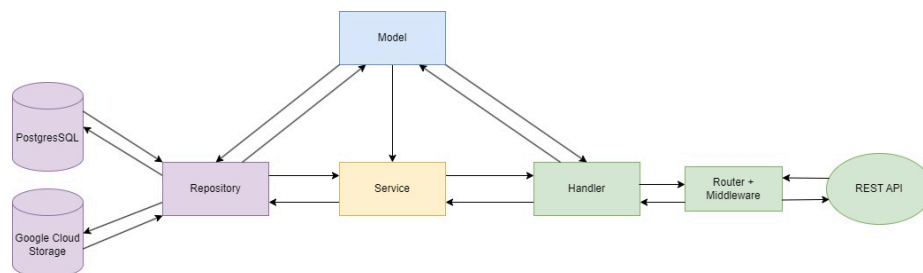


Рисунок 3.1 – Архітектура ГІС в підході Clean Architecture

3.2 Проєктування моделей

3.2.1 Зруйнований об'єкт

З аналізу подібних систем – зрозуміло, що існує необхідність зберігати детальну інформацію про кожен зруйнований об'єкт – назву, опис, повну адресу, дату руйнування, дату відновлення і т.п.

					ІАЛЦ.467200.003 ПЗ	Арк.
						25
Зм.	Арк.	№ докум.	Підпис	Дата		

Також, зберігатимемо фотографію зруйнованого об'єкта, для реалізації street view функціоналу потрібно буде зберігати не просто фотографію з локацією, а множину фотографії з множиною локацій. Які в подальшому «склеюватиметься» на клієнтській частині.

3.2.2 Фото зруйнованого об'єкта

Фото зберігатимемо шматками, для цього розіб'ємо весь панорманий простір навколо камери – сферами різного радіусу, для спостерігача – різними зумами камери. Надалі використовуватимемо термін зум замість радіус. Вважаємо кожен сферу дискретним станом ($Zoom = \{1, 2, 3, \dots\}$), самі сфери проєктуємо на 2D простір за допомогою циліндричної проєкції (також відома як проєкція Меркатора[24]) і розбиваємо на клітинки, у кожній клітинці є відповідна координата X та Y (рисунки 3.1). Важливо, що простір між сферами ніяк не зберігаємо, а переходи між зумами вирішуємо на клієнтській частині за рахунок простого приближення чи віддалення зображення.

Зміна кута зору майже нівелює головні недоліки циліндричної проєкції:

- Спотворення на полюсах.
- Лінійні спотворення.
- Вертикальний огляд.

Хоча існують інші недоліки, але вони не значні і не впливають сильно на сприйняття зображення

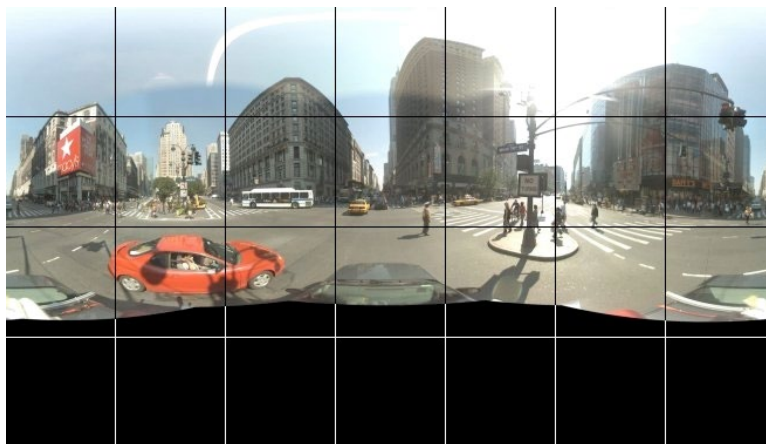


Рисунок 3.2 – циліндрична проєкція сфери розбита на клітинки[25]

3.3 Створення моделей

Таким чином маємо дві моделі: зруйнований об'єкт та його фотографії:

- destroyed_objects – таблиця з інформацією про зруйновані об'єкти
- images – таблиця з інформацією про фотографії

Таблиця 3.1 – Опис таблиці destroyed_objects

Назва колонки	Опис	Тип даних	Первинний	Обов'язкове
id	Ідентифікатор об'єкта	UUID	Так	Так
name	Назва об'єкта	VARCHAR(127)	Ні	Так
description	Опис об'єкта	TEXT	Ні	Так
type	Тип об'єкта	VARCHAR(31)	Ні	Так
region	Регіон об'єкта	VARCHAR(63)	Ні	Так
address	Адреса об'єкта	VARCHAR(127)	Ні	Так
lat	Широта об'єкта	FLOAT	Ні	Так
lng	Довгота об'єкта	FLOAT	Ні	Так
destruction_time	Час знищення об'єкта	TIMESTAMP	Ні	Так
restoration_time	Час відновлення об'єкта	TIMESTAMP	Ні	Ні
updated_at	Час останнього оновлення	TIMESTAMP	Ні	Так
created_at	Час створення	TIMESTAMP	Ні	Так

Таблиця 3.2 – Опис таблиці images

Назва колонки	Опис	Тип даних	Первинний	Обов'язкове
id	Ідентифікатор зображення	UUID	Так	Так
destroyed_object_id	Ідентифікатор знищеного об'єкта	UUID	Ні	Так
file_name	Ім'я файлу зображення	VARCHAR(31)	Ні	Так
path	Шлях до зображення	VARCHAR(127)	Ні	Так
lat	Широта зображення	FLOAT	Ні	Так
lng	Довгота зображення	FLOAT	Ні	Так
x	Координата X зображення	INT	Ні	Так
y	Координата Y зображення	INT	Ні	Так
zoom	Збільшення зображення	INT	Ні	Так
updated_at	Час останнього оновлення зображення	TIMESTAMP	Ні	Так
created_at	Час створення зображення	TIMESTAMP	Ні	Так

Також розглянемо структуру google cloud storage: Всі зображення зберігаємо в одному бакеті(у нашому випадку в diploma_images, у окремо створеному google cloud platform проєкті), при необхідності можна додати інші бакети для зберігання інших типів зображень. Сама схема зберігання у бакет виглядає так:

places/{lat}_{lon}/frames/{x}_{y}_{zoom}

Де:

- {lat} та {lon} - це широта та довгота місця, до якого відноситься зображення. Це дозволяє групувати всі зображення за географічним місцем.
- {x}, {y} та {zoom} - це параметри зображення. {x} та {y} - це координати зображення, а {zoom} - це рівень збільшення. Це дозволяє зберігати різні зображення одного й того ж місця з різними параметрами.

Така схема іменування дозволяє ефективно керувати зображеннями та швидко знаходити потрібні зображення.

3.4 Вимоги від клієнтської частини

У нашому випадку клієнтська частина буде отримувати об'єкти і їх фотографії через REST API.

3.4.1 Зруйновані об'єкти

Надамо CRUD функціонал фронт енду над зруйнованими об'єктами (Таблиці 3.3, 3.4).

					ІАЛЦ.467200.003 ПЗ	Арк.
						29
Зм.	Арк.	№ докум.	Підпис	Дата		

Таблиця 3.3 – Опис групи ендпойнтів destroyed_objects (* - опціональне поле, ** - вказано поля моделей, якщо повертається масив)

Назва ендпойнту	Опис	Вхідні дані		Вихідні дані**	
		Поле	Тип даних	Поле	Тип даних
Get Destroyed Objects	Повертає зруйновані об'єкти з бази даних	Page	Int	ID	String
		Per Page	Int	Name	String
		Name	*String	Description	String
		Type	*String	Type	String
		Region	*String	Region	String
		Start Destruction Time	*String	Address	String
		End Destruction Time	*String	Lat	Float64
		End Restoration Time	*String	Lng	Float64
		Restoration Time	*String	Destruction Time	String
				Restoration Time	String
				UpdatedAt	String
				CreatedAt	String

Кінець таблиці 3.3

Назва ендпойну	Опис	Вхідні дані		Вихідні дані	
		Поле	Тип даних	Поле	Тип даних
Create Destroyed Object	Створює зруйнований об'єкт в базі даних	Name	String		
		Description	String		
		Type	String		
		Region	String		
		Address	String		
		Lat	Float64		
		Lng	Float64		
		DestructionTime	String		
		RestorationTime	String		
Update Destroyed Object	Оновлює зруйнований об'єкт в базі даних	Name	String		
		Description	String		
		Type	String		
		Region	String		
		Address	String		
		Lat	Float64		
		Lng	Float64		
		DestructionTime	String		
		RestorationTime	String		
Delete Destroyed Object	Видаляє зруйнований об'єкт з бази даних	ID	String		

3.4.2 Зображення

Надамо CRUD функціонал фронт енду над зображеннями:

Таблиця 3.4 – Опис групи ендпойнтів (* - опціональне поле)

Назва ендпойнту	Опис	Вхідні дані		Вихідні дані	
		Поле	Тип даних	Поле	Тип даних
Get Image	Повертає фотографію з бази даних	Lat	Float64	ID	String
		Lng	Float64	Destroyed ObjectID	String
		X	Int	FileName	String
		Y	Int	Path	String
		Zoom	Int	Lat	String
				Lng	String
				X	Int
				Y	Int
				Zoom	Int
Create Image	Додає фотографію до бази даних і сховку	Destroyed ObjectID	String		
		FileName	String		
		Path	String		
		Lat	Float64		
		Lng	Float64		
		X	Int		
		Y	Int		
		Zoom	Int		

Кінець таблиці 3.4

Delete Image	Видаляє фотографію (ї) з бази даних і зі сховку (якщо не передати жодного поля, то ендпойнт поверне помилку)	ID	*String	
		Destroyed ObjectID	*String	

3.5 Обробники запитів, сервіси та репозиторії

Згідно Clean Architecture для кожної моделі ми маємо створити окремий обробник запитів, сервіс та репозиторій.

Відповідно до вимог клієнтської частини маємо два обробника з відповідними маршрутами (Рисунок 3.2), обробники запитів (Рисунки 3.3-3.4), сервіси (Рисунки 3.5-3.6) та репозиторії (Рисунки 3.7-3.9).

```
func (h *DestroyedObjectHandler) Register(v1 *echo.Group) { 1 usage
    destroyedObjects := v1.Group( prefix: "/destroyed_objects")
    destroyedObjects.GET( path: "", h.GetDestroyedObjects)
    destroyedObjects.POST( path: "", h.CreateDestroyedObject)
    destroyedObjects.PUT( path: "", h.UpdateDestroyedObject)
    destroyedObjects.DELETE( path: "", h.DeleteDestroyedObject)
}

func (h *ImageHandler) Register(v1 *echo.Group) { 1 usage 1 Masedko *
    images := v1.Group( prefix: "/images")
    images.GET( path: "", h.GetImage)
    images.POST( path: "", h.CreateImage)
    images.DELETE( path: "", h.DeleteImage)
}
```

Рисунок 3.2 – маршрути обробників запитів

```
Package: handler

type DestroyedObjectHandler struct {
    destroyedObjectService *service.DestroyedObjectService
}

Methods on (*DestroyedObjectHandler): Register(v1 *echo.Group)
GetDestroyedObjects(c echo.Context) error
CreateDestroyedObject(c echo.Context) error
UpdateDestroyedObject(c echo.Context) error
DeleteDestroyedObject(c echo.Context) error
```

Рисунок 3.3 – обробник запитів для зруйнованих об'єктів

```
Package: handler

type ImageHandler struct {
    imageService *service.ImageService
}

Methods on (*ImageHandler): GetImage(c echo.Context) error
CreateImage(c echo.Context) error
DeleteImage(c echo.Context) error
Register(v1 *echo.Group)
```

Рисунок 3.4 – обробник запитів для зображень

```
Package: service

type DestroyedObjectService struct {
    destroyedObjectRepo DestroyedObjectRepo
}

Methods on (*DestroyedObjectService): GetDestroyedObjects(req *dto.GetDestroyedObjectsRequest) ([]model.DestroyedObject, error)
CreateDestroyedObject(req *dto.CreateDestroyObjectRequest) error
UpdateDestroyedObject(req *dto.UpdateDestroyedObjectRequest) error
DeleteDestroyedObject(req *dto.DeleteDestroyedObjectRequest) error

'DestroyedObjectService' on pkg.go.dev ↗
```

Рисунок 3.5 – сервіс для зруйнованих об'єктів

```
Package: service

type ImageService struct {
    imageRepo ImageRepo
    imageBucket ImageBucket
}

Methods on (*ImageService): CreateImage(req *dto.CreateImageRequest, file *multipart.FileHeader) error
GetImageByLatLngXYZoom(req *dto.GetImageRequest) (dto.GetImage, error)
DeleteImage(req *dto.DeleteImageRequest) error
```

Рисунок 3.6 – сервіс для зображень

```
Package: repository

type DestroyedObjectRepo struct {
    db *database.DB
}

Methods on (*DestroyedObjectRepo): GetDestroyedObjects(page int, perPage int, name *string, objType *string, region *string,
startDestructionTime *string, endDestructionTime *string, startRestorationTime *string,
endRestorationTime *string) ([]model.DestroyedObject, error)
CreateDestroyedObject(object model.DestroyedObject) error
UpdateDestroyedObject(object model.DestroyedObject) error
DeleteDestroyedObject(id uuid.UUID) error
```

Рисунок 3.7 – репозиторій для зруйнованих об'єктів

```
Package: repository

type ImageRepo struct {
    db *database.DB
}

Methods on (*ImageRepo): CreateImage(image model.Image) error
GetImageByLatIngXYZZoom(lat float64, lng float64, x int, y int, zoom int) (dto.GetImage, error)
DeleteImage(id uuid.UUID) error
DeleteImagesByDestroyedObjectID(destroyedObjectID uuid.UUID) error
GetPathByID(id uuid.UUID) (string, error)
GetPathsByDestroyedObjectID(destroyedObjectID uuid.UUID) ([]string, error)
```

Рисунок 3.8 – репозиторій бази даних для зображень

```
Package: storage

type Bucket struct {
    *storage.BucketHandle
    Name string
}

Methods on (*Bucket): Upload(ctx context.Context, fileName string, file io.Reader) (string, error)
Delete(ctx context.Context, fileName string) error
```

Рисунок 3.9 – репозиторій сховку для зображень

3.6. Контейнеризація і розгортання додатку

Контейнеризувати і розгортати додаток будемо за допомогою зв'язки Docker та Google Kubernetes Engine(GKE) в шість етапів:

1. Спочатку створимо Dockerfile і Docker image з додатку
2. Створимо регіональний кластер GKE
3. Підключимось до GKE та налаштуємо Google Container Registry (GCR)

4. Створимо деплойменти та сервіси (створимо .yaml маніфести та застосуємо їх), створимо instance PostgreSQL з GIS модифікаціями.
5. Створимо Google Cloud Storage(GCS) bucket для зберігання зображень
6. Налаштуємо load balancer та ingress для зовнішнього доступу до сервісу

					ІАЛЦ.467200.003 ПЗ	Арк.
						36
Зм.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВОК ДО РОЗДІЛУ 3

В процесі розробки серверної частини застосовано архітектурний підхід Clean Architecture, що забезпечує модульність, масштабованість та легкість у підтримці додатку. Детально розглянуто структуру REST API, роль маршрутизатора, обробників запитів, сервісного шару, репозиторію та моделей.

Розглянуто проєктування моделей і відповідних обробників запитів, сервісів та репозиторіїв.

Розглянуто метод контейнеризації кінцевого додатку разом з базою даних.

					ІАЛЦ.467200.003 ПЗ	Арк.
						37
Зм.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 4.

ТЕСТУВАННЯ СИСТЕМИ

4.1 Керівництво користувача

Керівництво користувача не є частиною серверу і розглянуто в індивідуальній частині дипломного проєкту "Геоінформаційна система підтримки зруйнованих об'єктів за допомогою користувацького інтерфейсу".

4.2 Тестування серверної частини

Важливою частиною розробки програмного забезпечення є тестування. Метою тестування є оцінка відповідності вимог і кінцевого продукту.

Проведетемо тестування кожного ендпойнту, кожен тестовий сценарій буде складатиметься з:

- Опису тесту
- Вхідних даних
- Порядок дій
- Очікуваного результату
- Відповідності очікуваного результату з фактичним

Таблиця 4.1 – Тестування створення зруйнованого об'єкту

Опис тесту	POST запит від клієнту передає інформацію про зруйнований об'єкт. Сервер приймає запит, створює об'єкт в базі даних та повертає 201 HTTP статус код
Вхідні дані	Інформація про зруйнований об'єкт

Продовження таблиці 4.1

Порядок дій	1. Валідація вхідних даних. 2. Відправлення запиту до бази даних зі створення зруйнованого об'єкту. Повернення 201 HTTP статус коду.
Очікуваний результат	Інформація про зруйнований об'єкт збережено до бази даних
Фактичний результат відповідає очікуваному результату	Так

Таблиця 4.2 – Тестування оновлення зруйнованого об'єкту

Опис тесту	PUT запит від клієнту передає інформацію про оновлений зруйнований об'єкт. Сервер приймає запит, оновлює об'єкт в базі даних та повертає 204 HTTP статус код
Вхідні дані	Інформація про оновлений зруйнований об'єкт.
Порядок дій	1. Валідація вхідних даних. 2. Відправлення запиту до бази даних на оновлення зруйнованого об'єкту. 3. Повернення 204 HTTP статус коду.
Очікуваний результат	Інформація про оновлений зруйнований об'єкт збережено до бази даних
Фактичний результат відповідає очікуваному результату	Так

Таблиця 4.3 – Тестування отримання зруйнованих об'єктів

Опис тесту	GET запит від клієнту передає фільтри для отримання зруйнованих об'єктів. Сервер приймає запит, повертає масив об'єктів з бази даних та 200 HTTP статус код
Вхідні дані	Фільтри для зруйнованих об'єктів
Порядок дій	1. Валідація вхідних даних. 2. Відправлення запиту до бази даних на отримання зруйнованих об'єктів. 3. Повернення 200 HTTP статус коду та масиву зруйнованих об'єктів.
Очікуваний результат	Масив з інформацією про зруйновані об'єкти повернено клієнту
Фактичний результат відповідає очікуваному результату	Так

Таблиця 4.4 – Тестування видалення зруйнованого об'єкта

Опис тесту	DELETE запит від клієнту передає ідентифікатор зруйнованого об'єкту. Сервер приймає запит, видаляє об'єкт з бази даних та повертає 204 HTTP статус код
Вхідні дані	Ідентифікатор зруйнованого об'єкту
Порядок дій	1. Валідація вхідних даних. 2. Відправлення запиту до бази даних на видалення зруйнованого об'єкту. 3. Повернення 204 HTTP статус коду.

Продовження таблиці 4.4

Очікуваний результат	Зруйнований об'єкт видалено з бази даних
Фактичний результат відповідає очікуваному результату	Так

Таблиця 4.5 – Тестування створення зображення

Опис тесту	POST запит від клієнту передає інформацію про зображення і саме зображення. Сервер приймає запит, зберігає зображення до сховку, створює об'єкт в базі даних та повертає 201 HTTP статус код
Вхідні дані	Зображення та інформація до зображення
Порядок дій	1. Валідація вхідних даних. 2. Відправлення зображення до сховку. 3. Відправлення запиту до бази даних зі створення зображення об'єкту. 4. Повернення 201 HTTP статус коду.
Очікуваний результат	Зображення збережено до сховку. Інформацію про зображення збережено до бази даних
Фактичний результат відповідає очікуваному результату	Так

Таблиця 4.6 – Тестування отримання зображення

Опис тесту	GET запит від клієнту передає інформацію про зображення. Сервер приймає запит, повертає об'єкт з бази даних та 200 HTTP статус код
Вхідні дані	Інформація про зруйнований об'єкт
Порядок дій	1. Валідація вхідних даних. 2. Відправлення запиту до бази даних на отримання зображення. 3. Повернення 200 HTTP статус коду та інформації про зображення (включно з шляхом до зображення для його завантаження зі сховку)
Очікуваний результат	Інформацію про зображення повернено клієнту
Фактичний результат відповідає очікуваному результату	Так

Таблиця 4.7 – Тестування видалення зображень

Опис тесту	DELETE запит від клієнту передає ідентифікатор зображення чи зруйнованого об'єкту. Сервер приймає запит і видаляє об'єкти зі сховку та бази даних і повертає 204 HTTP статус код.
Вхідні дані	Ідентифікатор зображення чи зруйнованого об'єкту

Продовження таблиці 4.7

Порядок дій	1. Валідація вхідних даних. 2. Відправлення запиту до сховку на видалення зображень 3. Відправлення запиту до бази даних на видалення зображень. 4. Повернення 204 HTTP статус коду.
Очікуваний результат	Зображення видалено зі сховку. Інформацію про зображення видалено з бази даних.
Фактичний результат відповідає очікуваному результату	Так

ВИСНОВОК ДО РОЗДІЛУ 4

У розділі було розглянуто ключові аспекти тестування серверної частини геоінформаційної системи підтримки зруйнованих об'єктів. Всі тестування проводилися для перевірки відповідності розробленого функціоналу вимогам, забезпечення коректної роботи серверної частини та підтвердження її надійності.

Було проведено тестування основних ендпойнтів: створення, оновлення, отримання та видалення зруйнованих об'єктів і зображень, що супроводжують ці об'єкти. Усі тестові сценарії склалися з чітко визначених етапів: валідації вхідних даних, відправлення відповідних запитів до бази даних та повернення відповідних HTTP статус кодів.

Результати тестування показали, що серверна частина системи функціонує згідно з очікуваннями:

Створення нових зруйнованих об'єктів і зображень, оновлення їх інформації та їхнє видалення з бази даних виконуються коректно.

Отримання списків зруйнованих об'єктів та окремих зображень здійснюється швидко та точно відповідно до заданих фільтрів та параметрів.

Тестування підтвердило, що всі описані операції успішно виконуються, а фактичні результати повністю відповідають очікуваним. Це забезпечує високу якість та надійність розробленої геоінформаційної системи, роблячи її готовою до реального використання.

					ІАЛЦ.467200.003 ПЗ	Арк.
						44
Зм.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

Під час виконання дипломної роботи було розроблено серверну частину геоінформаційної системи для візуалізації зруйнованих об'єктів. Дана ГІС, разом із клієнтською частиною надає можливість інтерактивно взаємодіяти з мапою та реєстром зруйнованих об'єктів.

У першому розділі було розглянуто актуальність розроблення такої ГІС, роль серверної частини в її роботі та також розглянуто аналоги системи як `reukraine.shtab.net`, `misto.lun.ua` чи `MagneticOne`, виявлено їх переваги та недоліки, на їх основі було сформульовано вимоги та задачі, які повинна вирішувати система.

У другому розділі описано найбільш актуальні технології для розроблення ГІС. Оцінено переваги й недоліки кожної можливої технології. Взято до уваги не тільки швидкість розробки, а і можливість для розростання системи. Було обрано мову програмування `Go`, веб-фреймворк `Echo`, реляційна база даних `PostgreSQL`(з надстройками `PostGIS`), оркестратор `Kubernetes` та сховище даних `Google Cloud Storage`.

У третьому розділі розглянуто архітектуру серверної частини,. Розглянуто підхід `Clean Architecture` в контексті мови програмування `Go`, веб-фреймворку `Echo` та загальних вимог системи. Розглянуто головні елементи такої архітектури: `Handler`, `service`, `repository`, `model`. Також було розглянуто вимоги клієнтської частини та проєктування моделей згідно архітектурного підходу.

У четвертому розділі описано тестування додатку та його результати. Було перевірено, що всі ендпойнти працюють коректно і відповідають початковим вимогам.

Технічне завдання дипломного бакалаврського проєкту виконане повністю.

					ІАЛЦ.467200.003 ПЗ	Арк.
						45
Зм.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. В Україні запрацювала карта руйнувань: як додати об'єкт та чим це допоможе. *Новини України - останні новини України сьогодні - УНІАН*. URL: <https://www.unian.ua/techno/v-ukrajini-zapracyuvala-karta-ruynuvan-yak-dodati-ob-yekt-ta-chim-ce-dopomozhe-11876043.html> (дата звернення: 26.05.2024).
2. 872 пам'ятки пошкоджені в Україні через війну – Мінкульт. *Суспільне | Новини*. URL: <https://suspilne.media/culture/657662-872-pamatki-poskodzeni-v-ukraini-cerez-vijnu-minkult/> (дата звернення: 26.05.2024).
3. Зруйнована культурна спадщина України. *mvs.gov.ua*. URL: <https://mvs.gov.ua/news/zruinovana-kulturna-spadshhina-ukrayini> (дата звернення: 26.05.2024).
4. З'явилася онлайн-мапа зруйнованих новобудов у Бучі, Ірпені й Гостомелі. *The Village Україна*. URL: <https://www.village.com.ua/village/city/city-news/328615-z-yavilasya-onlayn-mapa-zruynovanih-novobudov-u-buchi-irpeni-ta-gostomeli> (дата звернення: 26.05.2024).
5. Чим відрізняється Frontend та Backend розробка? Що обрати?. *Front-end Academy*. URL: <https://frontend.lviv.ua/chym-vidriznyayetsya-frontend-ta-backend-rozrobka-shho-obraty> (дата звернення: 26.05.2024).
6. Карта відновлення та руйнувань. *Карта Відновлення та Руйнувань*. URL: <https://reukraine.shtab.net/about> (дата звернення: 26.05.2024).
7. Про ЛУН місто. *ЛУН Місто – Громадська організація дослідження комфорту та якості життя в українських містах*. URL: <https://misto.lun.ua/go> (дата звернення: 26.05.2024).

					ІАЛЦ.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		46

8. Інтерактивна карта руйнувань регіону. *MagneticOne Municipal Technologies*. URL: <https://magneticonemt.com/mlmt-interaktyvna-karta-rujnuvan/> (дата звернення: 26.05.2024).
9. Django vs flask vs laravel | what are the differences?. *StackShare*. URL: <https://stackshare.io/stackups/django-vs-flask-vs-laravel> (дата звернення: 26.05.2024).
10. From python to go: the top 9 backend languages to watch in 2024 - arc talent career blog. *Arc Talent Career Blog*. URL: <https://arc.dev/talent-blog/top-9-backend-languages/> (дата звернення: 26.05.2024).
11. Express - це фреймворк для веб-застосунків, побудованих на node.js. *Express - Node.js web application framework*. URL: <https://expressjs.com/uk/> (дата звернення: 26.05.2024).
12. Laravel - the PHP framework for web artisans. *Laravel - The PHP Framework For Web Artisans*. URL: <https://laravel.com/docs/11.x/readme> (дата звернення: 26.05.2024).
13. Next.js by vercel - the react framework. *Next.js by Vercel - The React Framework*. URL: <https://nextjs.org/> (дата звернення: 26.05.2024).
14. High performance, extensible, minimalist Go web framework | Echo. *High performance, extensible, minimalist Go web framework | Echo*. URL: <https://echo.labstack.com/> (дата звернення: 26.05.2024).
15. Kisivuli Q. How to design a backend architecture for your web application. *Medium*. URL: <https://medium.com/@queenskisivuli/how-to-design-a-backend-architecture-for-your-web-application-2b7188a497ab> (дата звернення: 26.05.2024).
16. Cross Compare SQL Server 2008 Spatial, PostgreSQL/PostGIS 1.3-1.4, MySQL 5-6. *Boston GIS: Geographic Information Systems Web Mapping OpenGIS and open source Solutions*. URL: https://www.bostongis.com/PrinterFriendly.aspx?content_name=sqlserver2008_postgis_mysql_compare (дата звернення: 26.05.2024).

					ІАЛЦ.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		47

17. Geospatial queries - mongodb manual v7.0. *MongoDB: The Developer Data Platform | MongoDB.*

URL: <https://www.mongodb.com/docs/manual/geospatial-queries/> (дата звернення: 26.05.2024).

18. Powell R. Docker Swarm vs Kubernetes: how to choose a container orchestration tool. *CircleCI.* URL: <https://circleci.com/blog/docker-swarm-vs-kubernetes/> (дата звернення: 26.05.2024).

19. SpatiaLite: SpatiaLite. *Test page: GaiaTool CGI map server.* URL: <https://www.gaia-gis.it/fossil/libspatialite/home> (дата звернення: 26.05.2024).

20. Nomad by HashiCorp. Nomad by HashiCorp. URL: <https://www.nomadproject.io/> (дата звернення: 26.05.2024).

21. Amazon simple storage service (S3) – cloud storage – AWS. Amazon Web Services, Inc. URL: <https://aws.amazon.com/s3/faqs/> (дата звернення: 26.05.2024).

22. BigQuery enterprise data warehouse. Google Cloud. URL: <https://cloud.google.com/bigquery> (дата звернення: 26.05.2024).

23. Cloud storage. Google Cloud. URL: <https://cloud.google.com/storage> (дата звернення: 26.05.2024).

24. Система координат mgrs та utm. Спротив G7. Сили територіальної оборони ЗСУ. URL: <https://sprotyvg7.com.ua/lesson/sistema-koordinat-mgrs-ta-utm> (дата звернення: 01.06.2024).

25. Nanchiang R. How to implement clean architecture in Golang (EN). Medium. URL: <https://medium.com/@rayato159/how-to-implement-clean-architecture-in-golang-en-f50d66378ebf> (дата звернення: 01.06.2024).

26. Street view service | maps javascript API | google for developers. Google for Developers. URL: <https://developers.google.com/maps/documentation/javascript/streetview> (дата звернення: 01.06.2024).

					ІАЛЦ.467200.003 ПЗ	Арк.
Зм.	Арк.	№ докум.	Підпис	Дата		48

ДОДАТОК 1

**Геоінформаційна система для підтримки обчислень з
візуалізації зруйнованих об'єктів**

Схема структурна
ІАЛЦ.467200.004 Е1

Аркушів 1

Київ 2024 р



					ІАЛЦ.467200.004 Е1			
		№ докум.	Підпис	Дата				
Розробив	Гладкий А.І.				Геоінформаційна система для підтримки обчислень з візуалізації зруйнованих об'єктів Схема структурна		Літ.	Аркуш
Перевірив	Гончаренко О.О.							Аркушів
							1	1
Н.Контр	Виноградов Ю.М.						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, гр. ІО-01	
Затвердив								

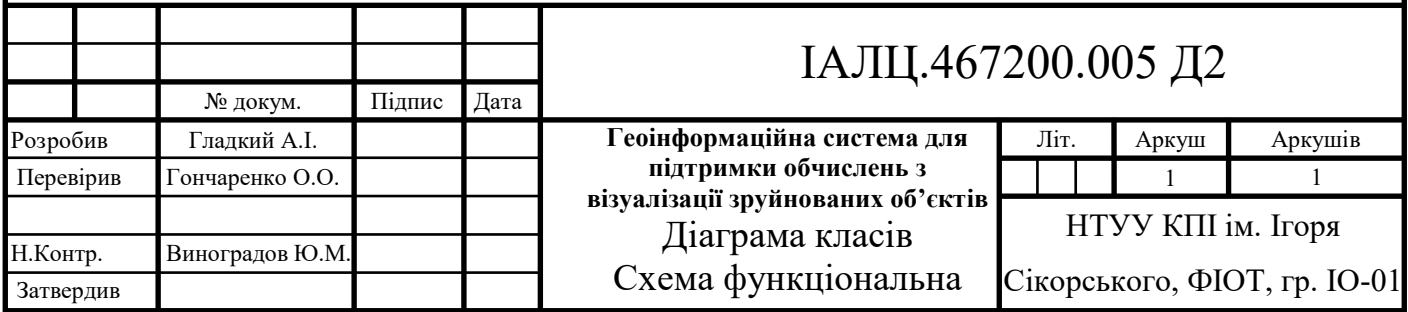
ДОДАТОК 2

**Геоінформаційна система для підтримки обчислень з візуалізації
зруйнованих об'єктів**

**Діаграма класів
Схема функціональна
ІАЛЦ.467200.005 Д2**

Аркушів 1

Київ 2024 р



ДОДАТОК 3

**Геоінформаційна система для підтримки обчислень з візуалізації
зруйнованих об'єктів**

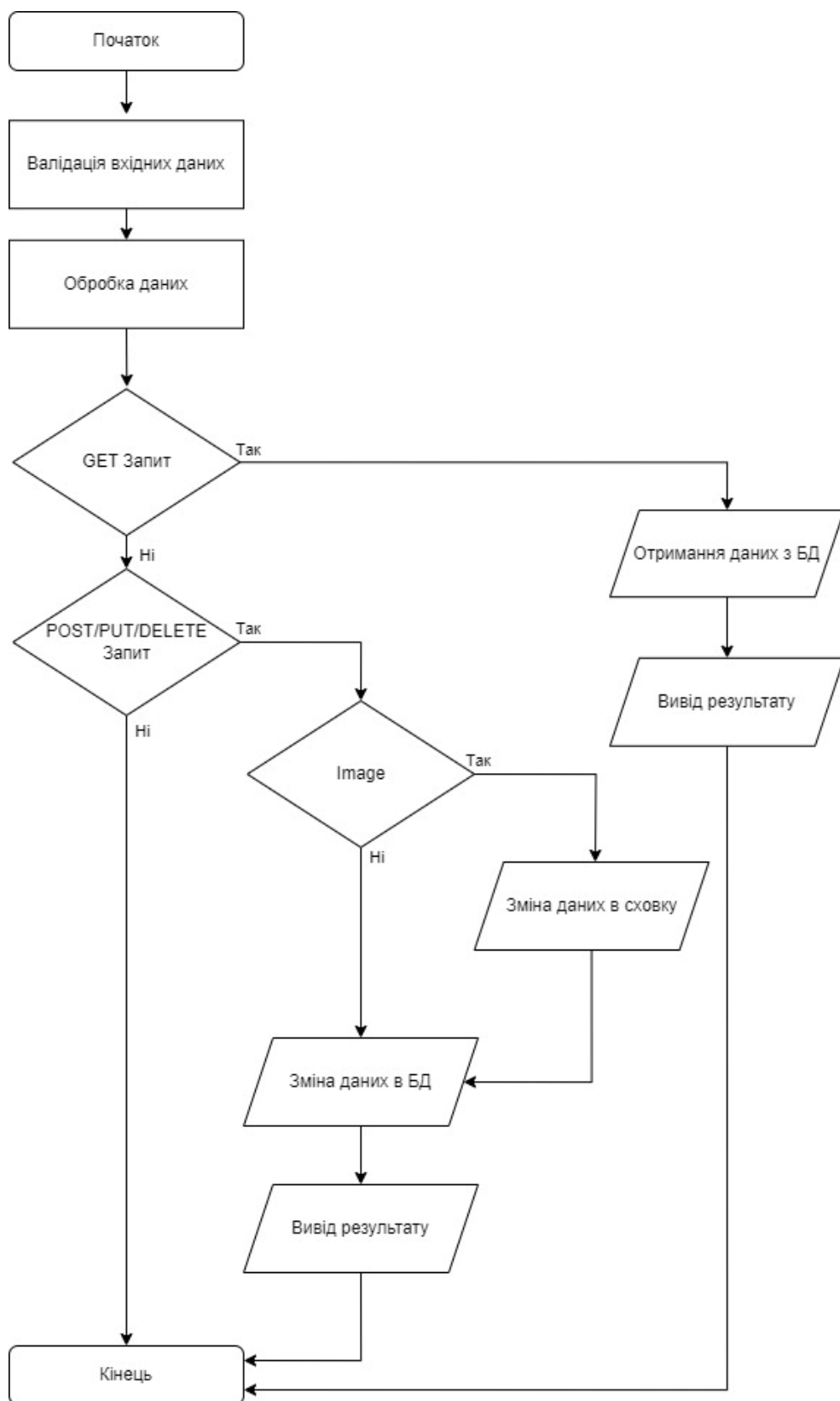
Алгоритм обробки запитів

Схема принципова

ІАЛЦ.467200.006 ДЗ

Аркушів 1

Київ 2024 р



					ІАЛЦ.467200.006 ДЗ				
		№ докум.	Підпис	Дата					
Розробив	Гладкий А.І.				Геоінформаційна система для підтримки обчислень з візуалізації зруйнованих об'єктів Алгоритм обробки запитів Схема принципова	Літ.	Аркуш	Аркушів	
Перевірив	Гончаренко О.О.						1	1	
						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, гр. ІО-01			
Н.Контр	Виноградов Ю.М.								
Затвердив									

ДОДАТОК 4

Геоінформаційна система для підтримки обчислень з
візуалізації

Текст програмного коду
ІАЛЦ.467200.007 Д4

Аркушів 31

Київ 2024 р

```

package main

import "github.com/Masedko/go-backend/internal/app"

func main() {
    a := app.New()
    a.Run()
}

package config

import (
    "github.com/spf13/viper"

    "github.com/Masedko/go-backend/internal/core/errors"
)

type Load struct {
    Path string
    Name string
    Type string
}

type Server struct {
    Host string `mapstructure:"host"`
    Port int    `mapstructure:"port"`
}

type Database struct {
    Host      string `mapstructure:"host"`
    User      string `mapstructure:"user"`
    Password  string `mapstructure:"password"`
    DBName    string `mapstructure:"db-name"`
    Port      int    `mapstructure:"port"`
    SSLMode   string `mapstructure:"ssl-mode"`
}

```

					ІАЛЦ.467200.007 Д4			
		№ докум.	Підпис	Дата				
Розробив		Гладкий А.І.			Геоінформаційна система для підтримки обчислень з візуалізації зруйнованих об'єктів Текст програмного коду	Літ.	Аркуш	Аркушів
Перевірив		Гончаренко О.О.					1	31
						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, гр. ІО-01		
Н.Контр.		Виноградов Ю.М.						
Затвердив								

```

type Storage struct {
    BucketNames []string `mapstructure:"bucket-names"`
}

type Config struct {
    Server    `mapstructure:"server"`
    Database  `mapstructure:"database"`
    Storage   `mapstructure:"storage"`
}

func LoadConfig(load Load) (*Config, error) {
    v := viper.New()
    v.SetConfigName(load.Name)
    v.SetConfigType(load.Type)
    v.AddConfigPath(load.Path)

    err := v.ReadInConfig()
    if err != nil {
        return nil, errors.Error{
            Err:  err,
            Desc: "Cannot read config",
        }
    }

    var config Config
    err = v.Unmarshal(&config)
    if err != nil {
        return nil, errors.Error{
            Err:  err,
            Desc: "Cannot unmarshal config",
        }
    }

    return &config, nil
}

package app

import (
    "fmt"

    "github.com/Masedko/go-backend/config"
    pkgerrors "github.com/Masedko/go-backend/internal/core/errors"

```

					ІАЛЦ.467200.007 Д4	Арк.
						2
Зм.	Арк.	№ докум.	Підпис	Дата		

```

"github.com/Masedko/go-backend/internal/core/handler"
"github.com/Masedko/go-backend/internal/core/logger"
"github.com/Masedko/go-backend/internal/core/router"
"github.com/Masedko/go-backend/internal/core/service"
"github.com/Masedko/go-backend/internal/data/database"
"github.com/Masedko/go-backend/internal/data/repository"
"github.com/Masedko/go-backend/internal/data/storage"
)

```

```

type App struct {
    Logger *logger.Logger
    Config *config.Config
    Router *router.Router
}

```

```

func New() *App {
    log := logger.NewLogger()
    cfg, err := config.LoadConfig(config.Load{
        Path: "./config",
        Name: "config",
        Type: "yaml",
    })
    if err != nil {
        log.FatalWithDesc(err)
    }

    db, err := database.NewDB(database.Config{
        Host:      cfg.Database.Host,
        User:      cfg.Database.User,
        Password:  cfg.Database.Password,
        DBName:    cfg.Database.DBName,
        SSLMode:   cfg.Database.SSLMode,
    })
    if err != nil {
        log.FatalWithDesc(err)
    }
    log.Info().Msg("Database connected")

    s, err := storage.New()
    if err != nil {
        log.FatalWithDesc(err)
    }
    log.Info().Msg("Storage connected")
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						3
Зм.	Арк.	№ докум.	Підпис	Дата		

```

if len(cfg.Storage.BucketNames) == 0 {
    log.FatalWithDesc(pkgerrors.NewError("Bucket name is not set", nil))
}
imageBucket := storage.NewBucket(s, cfg.Storage.BucketNames[0])

r := router.New(log)
v1 := r.Group("/api/v1")

imageRepo := repository.NewImageRepository(db)
destroyedObjectRepo := repository.NewDestroyedObjectRepository(db)

imageService := service.NewImageService(imageRepo, imageBucket)
destroyedObjectService := service.NewDestroyedObjectService(destroyedObjectRepo)

imageHandler := handler.NewImageHandler(imageService)
destroyedObjectHandler := handler.NewDestroyedObjectHandler(destroyedObjectService)

imageHandler.Register(v1)
destroyedObjectHandler.Register(v1)

return &App{
    Logger: log,
    Config: cfg,
    Router: r,
}
}

func (app *App) Run() {
    err := app.Router.Start(fmt.Sprintf("%s:%d", app.Config.Server.Host,
app.Config.Server.Port))
    if err != nil {
        app.Logger.ErrorWithDesc(pkgerrors.Error{
            Err: err,
            Desc: "Cannot start web app",
        })
    }
}

package errors

type Error struct {
    Desc string
    Err error
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						4
Зм.	Арк.	№ докум.	Підпис	Дата		

```

}

func (e Error) Error() string {
    return e.Err.Error()
}

func (e Error) Description() string {
    return e.Desc
}

func NewError(desc string, err error) Error {
    return Error{
        Desc: desc,
        Err:  err,
    }
}

package handler

import (
    "net/http"

    "github.com/labstack/echo/v4"

    pkgerrors "github.com/Masedko/go-backend/internal/core/errors"
    "github.com/Masedko/go-backend/internal/core/model"
    "github.com/Masedko/go-backend/internal/core/model/dto"
    "github.com/Masedko/go-backend/internal/core/service"
)

type DestroyedObjectService interface {
    GetDestroyedObjects(req *dto.GetDestroyedObjectsRequest) ([]model.DestroyedObject,
error)
    CreateDestroyedObject(req *dto.CreateDestroyObjectRequest) error
    UpdateDestroyedObject(req *dto.UpdateDestroyedObjectRequest) error
    DeleteDestroyedObject(req *dto.DeleteDestroyedObjectRequest) error
}

type DestroyedObjectHandler struct {
    destroyedObjectService *service.DestroyedObjectService
}

func NewDestroyedObjectHandler(destroyedObjectService *service.DestroyedObjectService)
*DestroyedObjectHandler {

```

					ІАЛЦ.467200.007 Д4	Арк.
						5
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        return &DestroyedObjectHandler{
            destroyedObjectService: destroyedObjectService,
        }
    }

func (h *DestroyedObjectHandler) GetDestroyedObjects(c echo.Context) error {
    req := &dto.GetDestroyedObjectsRequest{}
    if err := c.Bind(req); err != nil {
        return c.JSON(http.StatusUnprocessableEntity, pkgerrors.NewError("Failed to
bind request", err))
    }
    if err := c.Validate(req); err != nil {
        return c.JSON(http.StatusUnprocessableEntity, pkgerrors.NewError("Failed to
validate request", err))
    }
    objects, err := h.destroyedObjectService.GetDestroyedObjects(req)
    if err != nil {
        return c.JSON(http.StatusUnprocessableEntity, pkgerrors.NewError("Failed to get
destroyed objects", err))
    }
    return c.JSON(http.StatusOK, dto.GetDestroyedObjectsResponse{
        DestroyedObjects: objects,
    })
}

func (h *DestroyedObjectHandler) CreateDestroyedObject(c echo.Context) error {
    req := &dto.CreateDestroyObjectRequest{}
    if err := c.Bind(req); err != nil {
        return c.JSON(http.StatusUnprocessableEntity, pkgerrors.NewError("Failed to
bind request", err))
    }
    if err := c.Validate(req); err != nil {
        return c.JSON(http.StatusUnprocessableEntity, pkgerrors.NewError("Failed to
validate request", err))
    }
    err := h.destroyedObjectService.CreateDestroyedObject(req)
    if err != nil {
        return c.JSON(http.StatusUnprocessableEntity, pkgerrors.NewError("Failed to
create destroyed object", err))
    }
    return c.JSON(http.StatusCreated, dto.CreateDestroyedObjectResponse{})
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						6
Зм.	Арк.	№ докум.	Підпис	Дата		

```

func (h *DestroyedObjectHandler) UpdateDestroyedObject(c echo.Context) error {
    req := &dto.UpdateDestroyedObjectRequest{}
    if err := c.Bind(req); err != nil {
        return c.JSON(http.StatusUnprocessableEntity, pkgerrors.NewError("Failed to
bind request", err))
    }
    if err := c.Validate(req); err != nil {
        return c.JSON(http.StatusUnprocessableEntity, pkgerrors.NewError("Failed to
validate request", err))
    }
    err := h.destroyedObjectService.UpdateDestroyedObject(req)
    if err != nil {
        return c.JSON(http.StatusUnprocessableEntity, pkgerrors.NewError("Failed to
update destroyed object", err))
    }
    return c.JSON(http.StatusNoContent, dto.UpdateDestroyedObjectResponse{})
}

func (h *DestroyedObjectHandler) DeleteDestroyedObject(c echo.Context) error {
    req := &dto.DeleteDestroyedObjectRequest{}
    if err := c.Bind(req); err != nil {
        return c.JSON(http.StatusUnprocessableEntity, pkgerrors.NewError("Failed to
bind request", err))
    }
    if err := c.Validate(req); err != nil {
        return c.JSON(http.StatusUnprocessableEntity, pkgerrors.NewError("Failed to
validate request", err))
    }
    err := h.destroyedObjectService.DeleteDestroyedObject(req)
    if err != nil {
        return c.JSON(http.StatusUnprocessableEntity, pkgerrors.NewError("Failed to
delete destroyed object", err))
    }
    return c.JSON(http.StatusNoContent, dto.DeleteDestroyedObjectResponse{})
}

package handler

import (
    "net/http"

    "github.com/labstack/echo/v4"

    pkgerrors "github.com/Masedko/go-backend/internal/core/errors"

```

					ІАЛЦ.467200.007 Д4	Арк.
						7
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        "github.com/Masedko/go-backend/internal/core/model/dto"
        "github.com/Masedko/go-backend/internal/core/service"
    )

type ImageHandler struct {
    imageService *service.ImageService
}

func NewImageHandler(imageService *service.ImageService) *ImageHandler {
    return &ImageHandler{
        imageService: imageService,
    }
}

func (h *ImageHandler) GetImage(c echo.Context) error {
    req := &dto.GetImageRequest{}
    if err := c.Bind(req); err != nil {
        return c.JSON(http.StatusUnprocessableEntity, pkgerrors.NewError("Failed to bind request", err))
    }
    if err := c.Validate(req); err != nil {
        return c.JSON(http.StatusUnprocessableEntity, pkgerrors.NewError("Failed to validate request", err))
    }
    dtoImageGet, err := h.imageService.GetImageByLatLngXYZZoom(req)
    if err != nil {
        return c.JSON(http.StatusUnprocessableEntity, pkgerrors.NewError("Failed to get images", err))
    }

    return c.JSON(http.StatusOK, dto.GetImageResponse{
        ID:                dtoImageGet.ID,
        DestroyedObjectID: dtoImageGet.DestroyedObjectID,
        FileName:          dtoImageGet.FileName,
        Path:               dtoImageGet.Path,
        Lat:                dtoImageGet.Lat,
        Lng:                dtoImageGet.Lng,
    })
}

func (h *ImageHandler) CreateImage(c echo.Context) error {
    req := &dto.CreateImageRequest{}
    if err := c.Bind(req); err != nil {

```

					ІАЛЦ.467200.007 Д4	Арк.
						8
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        return c.JSON(http.StatusUnprocessableEntity, pkgerrors.NewError("Failed to
bind request", err))
    }

    if err := c.Validate(req); err != nil {
        return c.JSON(http.StatusUnprocessableEntity, pkgerrors.NewError("Failed to
validate request", err))
    }
    file, err := c.FormFile("file")
    if err != nil {
        return c.JSON(http.StatusUnprocessableEntity, pkgerrors.NewError("Failed to get
file", err))
    }

    err = h.imageService.CreateImage(req, file)
    if err != nil {
        return c.JSON(http.StatusUnprocessableEntity, pkgerrors.NewError("Failed to
create image", err))
    }

    return c.JSON(http.StatusCreated, dto.CreateImageResponse{})
}

func (h *ImageHandler) DeleteImage(c echo.Context) error {
    req := &dto.DeleteImageRequest{}
    if err := c.Bind(req); err != nil {
        return c.JSON(http.StatusUnprocessableEntity, pkgerrors.NewError("Failed to
bind request", err))
    }
    if err := c.Validate(req); err != nil {
        return c.JSON(http.StatusUnprocessableEntity, pkgerrors.NewError("Failed to
validate request", err))
    }
    err := h.imageService.DeleteImage(req)
    if err != nil {
        return c.JSON(http.StatusUnprocessableEntity, pkgerrors.NewError("Failed to
delete image", err))
    }

    return c.JSON(http.StatusNoContent, dto.DeleteImageResponse{})
}
package handler

```

					ІАЛЦ.467200.007 Д4	Арк.
						9
Зм.	Арк.	№ докум.	Підпис	Дата		

```

import (
    "github.com/labstack/echo/v4"
)

func (h *DestroyedObjectHandler) Register(v1 *echo.Group) {
    destroyedObjects := v1.Group("/destroyed_objects")
    destroyedObjects.GET("", h.GetDestroyedObjects)
    destroyedObjects.POST("", h.CreateDestroyedObject)
    destroyedObjects.PUT("", h.UpdateDestroyedObject)
    destroyedObjects.DELETE("", h.DeleteDestroyedObject)
}

func (h *ImageHandler) Register(v1 *echo.Group) {
    images := v1.Group("/images")
    images.GET("", h.GetImage)
    images.POST("", h.CreateImage)
    images.DELETE("", h.DeleteImage)
}

package logger

import (
    "errors"
    "os"

    "github.com/rs/zerolog"

    pkgerrors "github.com/Masedko/go-backend/internal/core/errors"
)

type Logger struct {
    zerolog.Logger
}

func NewLogger() *Logger {
    return &Logger{
        zerolog.New(os.Stdout).With().Timestamp().Logger(),
    }
}

func (l *Logger) ErrorWithDesc(err error) {
    var e pkgerrors.Error
    if errors.As(err, &e) {
        l.Error().Err(e.Err).Msg(e.Desc)
    }
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						10
Зм.	Арк.	№ докум.	Підпис	Дата		

```
        } else {
            l.Error().Err(err).Msg("Unknown error")
        }
    }
}

func (l *Logger) FatalWithDesc(err error) {
    var e pkgerrors.Error
    if errors.As(err, &e) {
        l.Fatal().Err(e.Err).Msg(e.Desc)
    } else {
        l.Fatal().Err(err).Msg("Unknown error")
    }
}
}

package validator

import "github.com/go-playground/validator/v10"

type Validator struct {
    validator *validator.Validate
}

func New() *Validator {
    return &Validator{
        validator: validator.New(),
    }
}

func (v *Validator) Validate(i interface{}) error {
    return v.validator.Struct(i)
}

package middleware

import (
    "github.com/labstack/echo/v4"
    "github.com/labstack/echo/v4/middleware"

    "github.com/Masedko/go-backend/internal/core/logger"
)

func NewLogger(logger *logger.Logger) echo.MiddlewareFunc {
    return middleware.RequestLoggerWithConfig(middleware.RequestLoggerConfig{
        LogURI:    true,
        LogStatus: true,
    })
}
```

					ІАЛЦ.467200.007 Д4	Арк.
						11
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        LogValuesFunc: func(c echo.Context, v middleware.RequestLoggerValues) error {
            logger.Info().
                Str("URI", v.URI).
                Int("status", v.Status).
                Msg("request")
            return nil
        },
    })
}

package middleware

import (
    "github.com/labstack/echo/v4"
    "github.com/labstack/echo/v4/middleware"
    "github.com/labstack/gommon/log"
)

func NewRecovery() echo.MiddlewareFunc {
    return middleware.RecoverWithConfig(middleware.RecoverConfig{
        StackSize: 1 << 10, // 1 KB
        LogLevel:  log.ERROR,
    })
}

package dto

import (
    "github.com/google/uuid"

    "github.com/Masedko/go-backend/internal/core/model"
)

type GetDestroyedObjectsRequest struct {
    Page          int    `json:"page" validate:"required"`
    PerPage       int    `json:"per_page" validate:"required"`
    Name          *string `json:"name" validate:"optional"`
    Type          *string `json:"type" validate:"optional"`
    Region        *string `json:"region" validate:"optional"`
    StartDestructionTime *string `json:"start_destruction_time" validate:"optional"`
    EndDestructionTime  *string `json:"end_destruction_time" validate:"optional"`
    StartRestorationTime *string `json:"start_restoration_time" validate:"optional"`
    EndRestorationTime  *string `json:"end_restoration_time" validate:"optional"`
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						12
Зм.	Арк.	№ докум.	Підпис	Дата		

```

type CreateDestroyObjectRequest struct {
    Name          string `json:"name" validate:"required"`
    Description    string `json:"description" validate:"required"`
    Type          string `json:"type" validate:"required"`
    Region        string `json:"region" validate:"required"`
    Address       string `json:"address" validate:"required"`
    Lat           float64 `json:"lat" validate:"required"`
    Lng           float64 `json:"lng" validate:"required"`
    DestructionTime string `json:"destruction_time" validate:"required"`
    RestorationTime *string `json:"restoration_time" validate:"optional"`
}

type UpdateDestroyedObjectRequest struct {
    ID            uuid.UUID `json:"id" validate:"required"`
    Name          string   `json:"name" validate:"required"`
    Description    string   `json:"description" validate:"required"`
    Type          string   `json:"type" validate:"required"`
    Region        string   `json:"region" validate:"required"`
    Address       string   `json:"address" validate:"required"`
    Lat           float64   `json:"lat" validate:"required"`
    Lng           float64   `json:"lng" validate:"required"`
    DestructionTime string   `json:"destruction_time" validate:"required"`
    RestorationTime *string   `json:"restoration_time" validate:"optional"`
}

type DeleteDestroyedObjectRequest struct {
    ID uuid.UUID `json:"id" validate:"required"`
}

type GetDestroyedObjectsResponse struct {
    DestroyedObjects []model.DestroyedObject `json:"destroyed_objects"`
}

type CreateDestroyedObjectResponse struct{}

type UpdateDestroyedObjectResponse struct{}

type DeleteDestroyedObjectResponse struct{}

package dto

import "github.com/google/uuid"

type GetImage struct {

```

					ІАЛЦ.467200.007 Д4	Арк.
						13
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        ID                uuid.UUID `json:"id"`
        DestroyedObjectID uuid.UUID `json:"destroyed_object_id"`
        FileName           string   `json:"file_name"`
        Path               string   `json:"path"`
        Lat                float64  `json:"lat"`
        Lng                float64  `json:"lng"`
        X                  int       `json:"x"`
        Y                  int       `json:"y"`
        Zoom               int       `json:"zoom"`
    }

type GetImageRequest struct {
    Lat float64 `json:"lat" validate:"required"`
    Lng float64 `json:"lng" validate:"required"`
    X    int     `json:"x" validate:"required"`
    Y    int     `json:"y" validate:"required"`
    Zoom int     `json:"zoom" validate:"required"`
}

type CreateImageRequest struct {
    DestroyedObjectID uuid.UUID `json:"destroyed_object_id" validate:"required"`
    FileName           string   `json:"file_name" validate:"required"`
    Path               string   `json:"path" validate:"required"`
    Lat                float64  `json:"lat" validate:"required"`
    Lng                float64  `json:"lng" validate:"required"`
    X                  int       `json:"x" validate:"required"`
    Y                  int       `json:"y" validate:"required"`
    Zoom               int       `json:"zoom" validate:"required"`
}

type DeleteImageRequest struct {
    ID                *uuid.UUID `json:"id" validate:"optional"`
    DestroyedObjectID *uuid.UUID `json:"destroyed_object_id" validate:"optional"`
}

type GetImageResponse struct {
    ID                uuid.UUID `json:"id"`
    DestroyedObjectID uuid.UUID `json:"destroyed_object_id"`
    FileName           string   `json:"file_name"`
    Path               string   `json:"path"`
    Lat                float64  `json:"lat"`
    Lng                float64  `json:"lng"`
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						14
Зм.	Арк.	№ докум.	Підпис	Дата		

```

type CreateImageResponse struct{}

type DeleteImageResponse struct{}

package model

import (
    "time"

    "github.com/google/uuid"
    "github.com/lib/pq"
)

type DestroyedObject struct {
    ID            uuid.UUID `json:"id"`
    Name          string    `json:"name"`
    Description    string    `json:"description"`
    Type          string    `json:"type"`
    Region        string    `json:"region"`
    Address       string    `json:"address"`
    Lat           float64   `json:"lat"`
    Lng           float64   `json:"lng"`
    DestructionTime time.Time `json:"destruction_time"`
    RestorationTime pq.NullTime `json:"restoration_time"`
    UpdatedAt     time.Time `json:"updated_at"`
    CreatedAt     time.Time `json:"created_at"`
}

package model

import (
    "time"

    "github.com/google/uuid"
)

type Image struct {
    ID            uuid.UUID `json:"id"`
    DestroyedObjectID uuid.UUID `json:"destroyed_object_id"`
    FileName      string    `json:"file_name"`
    Path         string    `json:"path"`
    Lat          float64   `json:"lat"`
    Lng          float64   `json:"lng"`
    X            int       `json:"x"`
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						15
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        Y                int        `json:"y"`
        Zoom             int        `json:"zoom"`
        UpdatedAt        time.Time  `json:"updated_at"`
        CreatedAt        time.Time  `json:"created_at"`
    }
package router

import (
    "github.com/labstack/echo/v4"
    echomiddleware "github.com/labstack/echo/v4/middleware"

    "github.com/Masedko/go-backend/internal/core/logger"
    "github.com/Masedko/go-backend/internal/core/middleware"
    "github.com/Masedko/go-backend/internal/core/middleware/validator"
)

type Router struct {
    *echo.Echo
}

func New(log *logger.Logger) *Router {
    e := echo.New()
    e.Pre(echomiddleware.RemoveTrailingSlash())
    e.Use(echomiddleware.CORSWithConfig(echomiddleware.CORSConfig{
        AllowOrigins: []string{"*"},
        AllowHeaders: []string{echo.HeaderOrigin, echo.HeaderContentType,
echo.HeaderAccept, echo.HeaderAuthorization},
        AllowMethods: []string{echo.GET, echo.HEAD, echo.PUT, echo.PATCH, echo.POST,
echo.DELETE},
    })))
    e.Validator = validator.New()
    e.Use(middleware.NewLogger(log))
    e.Use(middleware.NewRecovery())
    return &Router{e}
}
package service

import (
    "time"

    "github.com/google/uuid"
    "github.com/lib/pq"

```

					ІАЛЦ.467200.007 Д4	Арк.
						16
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        "github.com/Masedko/go-backend/internal/core/model"
        "github.com/Masedko/go-backend/internal/core/model/dto"
    )

type DestroyedObjectRepo interface {
    GetDestroyedObjects(
        page int, perPage int,
        name *string,
        objType *string,
        region *string,
        startDestructionTime *string, endDestructionTime *string,
        startRestorationTime *string, endRestorationTime *string,
    ) ([]model.DestroyedObject, error)
    CreateDestroyedObject(object model.DestroyedObject) error
    UpdateDestroyedObject(object model.DestroyedObject) error
    DeleteDestroyedObject(id uuid.UUID) error
}

type DestroyedObjectService struct {
    destroyedObjectRepo DestroyedObjectRepo
}

func NewDestroyedObjectService(destroyedObjectRepo DestroyedObjectRepo)
*DestroyedObjectService {
    return &DestroyedObjectService{
        destroyedObjectRepo: destroyedObjectRepo,
    }
}

func (s *DestroyedObjectService) GetDestroyedObjects(req *dto.GetDestroyedObjectsRequest)
([]model.DestroyedObject, error) {
    return s.destroyedObjectRepo.GetDestroyedObjects(
        req.Page, req.PerPage,
        req.Name, req.Type, req.Region,
        req.StartDestructionTime, req.EndDestructionTime,
        req.StartRestorationTime, req.EndRestorationTime,
    )
}

func (s *DestroyedObjectService) CreateDestroyedObject(req *dto.CreateDestroyObjectRequest)
error {
    destructionTime, err := time.Parse(time.DateTime, req.DestructionTime)
    if err != nil {

```

					ІАЛЦ.467200.007 Д4	Арк.
						17
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        return err
    }
    var restorationTime pq.NullTime
    if req.RestorationTime != nil {
        restorationTime.Time, err = time.Parse(time.DateTime, *req.RestorationTime)
        if err != nil {
            return err
        }
        restorationTime.Valid = true
    }
    return s.destroyedObjectRepo.CreateDestroyedObject(model.DestroyedObject{
        ID:            uuid.New(),
        Name:           req.Name,
        Description:    req.Description,
        Type:           req.Type,
        Region:         req.Region,
        Address:        req.Address,
        Lat:            req.Lat,
        Lng:            req.Lng,
        DestructionTime: destructionTime,
        RestorationTime: restorationTime,
        UpdatedAt:      time.Now(),
        CreatedAt:      time.Now(),
    })
}

func (s *DestroyedObjectService) UpdateDestroyedObject(req *dto.UpdateDestroyedObjectRequest)
error {
    destructionTime, err := time.Parse(time.DateTime, req.DestructionTime)
    if err != nil {
        return err
    }
    var restorationTime pq.NullTime
    if req.RestorationTime != nil {
        restorationTime.Time, err = time.Parse(time.DateTime, *req.RestorationTime)
        if err != nil {
            return err
        }
        restorationTime.Valid = true
    }
    return s.destroyedObjectRepo.UpdateDestroyedObject(model.DestroyedObject{
        ID:            req.ID,
        Name:           req.Name,

```

					ІАЛЦ.467200.007 Д4	Арк.
						18
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        Description:    req.Description,
        Type:           req.Type,
        Region:         req.Region,
        Address:         req.Address,
        Lat:             req.Lat,
        Lng:             req.Lng,
        DestructionTime: destructionTime,
        RestorationTime: restorationTime,
        UpdatedAt:       time.Now(),
    })
}

func (s *DestroyedObjectService) DeleteDestroyedObject(req *dto.DeleteDestroyedObjectRequest)
error {
    return s.destroyedObjectRepo.DeleteDestroyedObject(req.ID)
}

package service

import (
    "context"
    "io"
    "mime/multipart"

    "github.com/google/uuid"

    pkgerrors "github.com/Masedko/go-backend/internal/core/errors"
    "github.com/Masedko/go-backend/internal/core/model"
    "github.com/Masedko/go-backend/internal/core/model/dto"
)

type ImageRepo interface {
    CreateImage(image model.Image) error
    GetImageByLatLngXYZZoom(lat float64, lng float64, x int, y int, zoom int) (dto.GetImage,
error)
    DeleteImage(id uuid.UUID) error
    DeleteImagesByDestroyedObjectID(destroyedObjectID uuid.UUID) error
    GetPathByID(id uuid.UUID) (string, error)
    GetPathsByDestroyedObjectID(destroyedObjectID uuid.UUID) ([]string, error)
}

type ImageBucket interface {
    Upload(ctx context.Context, fileName string, file io.Reader) (string, error)
    Delete(ctx context.Context, fileName string) error
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						19
Зм.	Арк.	№ докум.	Підпис	Дата		

```

}

type ImageService struct {
    imageRepo    ImageRepo
    imageBucket  ImageBucket
}

func NewImageService(imageRepo ImageRepo, imageBucket ImageBucket) *ImageService {
    return &ImageService{
        imageRepo:    imageRepo,
        imageBucket:  imageBucket,
    }
}

func (s *ImageService) CreateImage(req *dto.CreateImageRequest, file *multipart.FileHeader) error {
    src, err := file.Open()
    if err != nil {
        return err
    }
    defer src.Close()

    return s.imageRepo.CreateImage(model.Image{
        ID:             uuid.New(),
        Lat:            req.Lat,
        Lng:            req.Lng,
        X:              req.X,
        Y:              req.Y,
        Zoom:           req.Zoom,
        DestroyedObjectID: req.DestroyedObjectID,
    })
}

func (s *ImageService) GetImageByLatLngXYZZoom(req *dto.GetImageRequest) (dto.GetImage, error) {
    return s.imageRepo.GetImageByLatLngXYZZoom(req.Lat, req.Lng, req.X, req.Y, req.Zoom)
}

func (s *ImageService) DeleteImage(req *dto.DeleteImageRequest) error {
    if req.ID != nil {
        path, err := s.imageRepo.GetPathByID(*req.ID)
        if err != nil {
            return err
        }
    }
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						20
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    }

    err = s.imageBucket.Delete(context.Background(), path)
    if err != nil {
        return err
    }

    err = s.imageRepo.DeleteImage(*req.ID)
    if err != nil {
        return err
    }
} else if req.DestroyedObjectID != nil {
    imagePaths, err :=
s.imageRepo.GetPathsByDestroyedObjectID(*req.DestroyedObjectID)
    if err != nil {
        return err
    }
    for _, imagePath := range imagePaths {
        err = s.imageBucket.Delete(context.Background(), imagePath)
        if err != nil {
            return err
        }
    }
    err = s.imageRepo.DeleteImagesByDestroyedObjectID(*req.DestroyedObjectID)
    if err != nil {
        return err
    }
} else {
    return pkgerrors.NewError("Specify ID or DestroyedObjectID", nil)
}
return nil
}

package database

import (
    "fmt"

    "github.com/jmoiron/sqlx"
    _ "github.com/lib/pq"
)

const driverName = "postgres"

type DB struct {

```

					ІАЛЦ.467200.007 Д4	Арк.
						21
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        *sqlx.DB
    }

type Config struct {
    Host      string
    User      string
    DBName    string
    Password  string
    SSLMode   string
}

func NewDB(cfg Config) (*DB, error) {
    dataSourceName := fmt.Sprintf("host=%s user=%s dbname=%s password=%s sslmode=%s",
        cfg.Host, cfg.User, cfg.DBName, cfg.Password, cfg.SSLMode,
    )
    sqlxDB, err := sqlx.Connect(driverName, dataSourceName)
    if err != nil {
        return nil, err
    }
    return &DB{sqlxDB}, nil
}

package repository

import (
    "github.com/google/uuid"

    "github.com/Masedko/go-backend/internal/core/model"
    "github.com/Masedko/go-backend/internal/data/database"
)

type DestroyedObjectRepo struct {
    db *database.DB
}

func NewDestroyedObjectRepository(db *database.DB) *DestroyedObjectRepo {
    return &DestroyedObjectRepo{
        db: db,
    }
}

func (r *DestroyedObjectRepo) GetDestroyedObjects(
    page int,
    perPage int,

```

					ІАЛЦ.467200.007 Д4	Арк.
						22
Зм.	Арк.	№ докум.	Підпис	Дата		

```

    name *string,
    objType *string,
    region *string,
    startDestructionTime *string,
    endDestructionTime *string,
    startRestorationTime *string,
    endRestorationTime *string,
) ([]model.DestroyedObject, error) {
    var objects []model.DestroyedObject
    var conditions []string
    var args []interface{}
    var argID int

    argID++
    conditions = append(conditions, "LIMIT "+string(rune(argID)))
    args = append(args, perPage)

    argID++
    conditions = append(conditions, "OFFSET "+string(rune(argID)))
    args = append(args, (page-1)*perPage)

    if name != nil {
        argID++
        conditions = append(conditions, "name = "+string(rune(argID)))
        args = append(args, *name)
    }

    if objType != nil {
        argID++
        conditions = append(conditions, "type = "+string(rune(argID)))
        args = append(args, *objType)
    }

    if region != nil {
        argID++
        conditions = append(conditions, "region = "+string(rune(argID)))
        args = append(args, *region)
    }

    if startDestructionTime != nil {
        argID++
        conditions = append(conditions, "destruction_time >= "+string(rune(argID)))
        args = append(args, *startDestructionTime)
    }

```

					ІАЛЦ.467200.007 Д4	Арк.
						23
Зм.	Арк.	№ докум.	Підпис	Дата		

```

}

if endDestructionTime != nil {
    argID++
    conditions = append(conditions, "destruction_time <= "+string(rune(argID)))
    args = append(args, *endDestructionTime)
}

if startRestorationTime != nil {
    argID++
    conditions = append(conditions, "restoration_time >= "+string(rune(argID)))
    args = append(args, *startRestorationTime)
}

if endRestorationTime != nil {
    argID++
    conditions = append(conditions, "restoration_time <= "+string(rune(argID)))
    args = append(args, *endRestorationTime)
}

baseQuery := `
    SELECT
        id,
        name,
        description,
        type,
        region,
        address,
        lat,
        lng,
        destruction_time,
        restoration_time,
        updated_at,
        created_at,
    FROM destroyed_objects`
err := r.db.Select(&objects, baseQuery)
if err != nil {
    return nil, err
}

return objects, nil
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						24
Зм.	Арк.	№ докум.	Підпис	Дата		

```

func (r *DestroyedObjectRepo) CreateDestroyedObject(object model.DestroyedObject) error {
    _, err := r.db.Exec(`
        INSERT INTO destroyed_objects (
            id,
            name,
            description,
            type,
            region,
            address,
            lat,
            lng,
            destruction_time,
            restoration_time,
            updated_at,
            created_at
        ) VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10, $11, $12)`,
        object.ID,
        object.Name,
        object.Description,
        object.Type,
        object.Region,
        object.Address,
        object.Lat,
        object.Lng,
        object.DestructionTime,
        object.RestorationTime,
        object.UpdatedAt,
        object.CreatedAt,
    )
    if err != nil {
        return err
    }

    return nil
}

```

```

func (r *DestroyedObjectRepo) UpdateDestroyedObject(object model.DestroyedObject) error {
    _, err := r.db.Exec(`
        UPDATE destroyed_objects SET
            name = $1,
            description = $2,
            type = $3,
            region = $4,

```

					ІАЛЦ.467200.007 Д4	Арк.
						25
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        address = $5,
        lat = $6,
        lng = $7,
        destruction_time = $8,
        restoration_time = $9,
        updated_at = $10
WHERE id = $11`,
object.Name,
object.Description,
object.Type,
object.Region,
object.Address,
object.Lat,
object.Lng,
object.DestructionTime,
object.RestorationTime,
object.UpdatedAt,
object.ID,
)
if err != nil {
    return err
}

return nil
}

func (r *DestroyedObjectRepo) DeleteDestroyedObject(id uuid.UUID) error {
    _, err := r.db.Exec(`
        DELETE FROM destroyed_objects
        WHERE id = $1`,
        id,
    )
    if err != nil {
        return err
    }

    return nil
}

package repository

import (
    "github.com/google/uuid"

```

					ІАЛЦ.467200.007 Д4	Арк.
						26
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        "github.com/Masedko/go-backend/internal/core/model"
        "github.com/Masedko/go-backend/internal/core/model/dto"
        "github.com/Masedko/go-backend/internal/data/database"
    )

type ImageRepo struct {
    db *database.DB
}

func NewImageRepository(db *database.DB) *ImageRepo {
    return &ImageRepo{
        db: db,
    }
}

func (r *ImageRepo) CreateImage(image model.Image) error {
    _, err := r.db.Exec(`
        INSERT INTO images (
            id,
            destroyed_object_id,
            file_name,
            path,
            lat,
            lng,
            x,
            y,
            zoom,
            updated_at,
            created_at
        ) VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10, $11)`,
        image.ID, image.DestroyedObjectID, image.FileName, image.Path, image.Lat,
image.Lng, image.X, image.Y, image.Zoom, image.UpdatedAt, image.CreatedAt,
    )
    if err != nil {
        return err
    }

    return nil
}

func (r *ImageRepo) GetImageByLatLngXYZZoom(lat float64, lng float64, x, y, zoom int)
(dto.GetImage, error) {
    var image dto.GetImage

```

					ІАЛЦ.467200.007 Д4	Арк.
						27
Зм.	Арк.	№ докум.	Підпис	Дата		

```

err := r.db.Get(&image, `
    SELECT
        id,
        destroyed_object_id,
        file_name,
        path,
        lat,
        lng,
        x,
        y,
        zoom,
        -- Haversine formula
        ( acos( cos( radians($1) ) * cos( radians( lat ) ) * cos( radians( lng )
- radians($2) ) + sin( radians(:$1) ) * sin( radians( lat ) ) ) ) AS distance
    FROM images
    WHERE AND x = $3 AND y = $4 AND zoom = $5
    ORDER BY distance
    LIMIT 1`, lat, lng, x, y, zoom,
)
if err != nil {
    return dto.GetImage{}, err
}

return image, nil
}

func (r *ImageRepo) DeleteImage(id uuid.UUID) error {
    _, err := r.db.Exec(`
        DELETE FROM images
        WHERE id = $1`,
        id,
    )
    if err != nil {
        return err
    }

    return nil
}

func (r *ImageRepo) DeleteImagesByDestroyedObjectID(destroyedObjectID uuid.UUID) error {
    _, err := r.db.Exec(`
        DELETE FROM images
        WHERE destroyed_object_id = $1`,

```

					ІАЛЦ.467200.007 Д4	Арк.
						28
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        destroyedObjectID,
    )
    if err != nil {
        return err
    }

    return nil
}

func (r *ImageRepo) GetPathByID(id uuid.UUID) (string, error) {
    var path string
    err := r.db.Get(&path, `
        SELECT path
        FROM images
        WHERE id = $1`,
        id,
    )
    if err != nil {
        return "", err
    }

    return path, nil
}

func (r *ImageRepo) GetPathsByDestroyedObjectID(destroyedObjectID uuid.UUID) ([]string,
error) {
    var path []string
    err := r.db.Get(&path, `
        SELECT path
        FROM images
        WHERE destroyedObjectID = $1`,
        destroyedObjectID,
    )
    if err != nil {
        return nil, err
    }

    return path, nil
}
}

package storage

import (
    "context"

```

					ІАЛЦ.467200.007 Д4	Арк.
						29
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        "fmt"
        "io"

        "cloud.google.com/go/storage"
    )

type Bucket struct {
    *storage.BucketHandle
    Name string
}

// NewBucket Creates Bucket that can be used to upload and download files to a single bucket
func NewBucket(c *Client, bucketName string) *Bucket {
    bkt := c.client.Bucket(bucketName)
    return &Bucket{bkt, bucketName}
}

func (b *Bucket) Upload(ctx context.Context, fileName string, file io.Reader) (string, error) {
    {
        obj := b.Object(fileName)
        wc := obj.NewWriter(ctx)
        if _, err := io.Copy(wc, file); err != nil {
            return "", fmt.Errorf("cannot copy file: %v", err)
        }
        err := wc.Close()
        if err != nil {
            return "", fmt.Errorf("cannot close file writer: %v", err)
        }
        return fmt.Sprintf("%s/%s", b.Name, fileName), nil
    }
}

func (b *Bucket) Delete(ctx context.Context, fileName string) error {
    {
        obj := b.Object(fileName)
        err := obj.Delete(ctx)
        if err != nil {
            return fmt.Errorf("cannot delete file: %v", err)
        }
        return nil
    }
}

package storage

import (
    "context"

```

					ІАЛЦ.467200.007 Д4	Арк.
						30
Зм.	Арк.	№ докум.	Підпис	Дата		

```

        "cloud.google.com/go/storage"
    )

type Client struct {
    client *storage.Client
}

// New Creates Client that can be used to access several buckets
func New() (*Client, error) {
    ctx := context.Background()
    c, err := storage.NewClient(ctx)
    if err != nil {
        return nil, err
    }
    return &Client{c}, nil
}

```

					ІАЛЦ.467200.007 Д4	Арк.
						31
Зм.	Арк.	№ докум.	Підпис	Дата		