КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технології програмування

Кваліфікаційна робота

на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

ІНТЕРАКТИВНА БАЗА ОБ'ЄКТІВ З ГЕОКООРДИНАТАМИ. РОБОТА З ДАНИМИ

Виконав студент 4-го курсу
Максим РАСАХАЦЬКИЙ

(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Тарас ПАНЧЕНКО

(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування

«____» _____ 2023 р.,

протокол № ___

Завідувач кафедри

Микола НІКІТЧЕНКО _____

(підпис)

РЕФЕРАТ

Обсяг роботи 46 сторінок, 19 ілюстрацій, 4 таблиці, 12 джерел посилань.

ГЕОКООРДИНАТИ, СИСТЕМА КООРДИНАТ, БАЗА ДАНИХ, ІНФОРМАЦІЙНА СИСТЕМА

Об’єкт розробки: система обліку об’єктів з геокоординатами.

Метою роботи є створення сервісу для додавання об’єктів з геокоординатами в базу даних, який зможе працювати без доступу в інтернет, має мати зручні методи експорту та імпорту даних, в тому числі у форматі excel таблиць, підтримувати три різні системи координат та підтримувати стандарт APP-6D.

Інструменти розроблення: мови програмування Golang та C#, bash скрипти, контейнеризація docker, карти історій Miro, Microsoft Excel для вводу даних, бібліотека GDAL.

Результати роботи: була досліджена та розроблена карта історій користувача, було досліджено три різні системи координат, був досліджений стандарт військового символу APP-6D. Було розроблено сервіс фіксації об’єктів, які містять детальну інформацію про їх розташування разом з іншими деталями. Розроблено функціональність додавання точок як по одному елементу, так і за допомогою імпорту багатьох елементів з excel файлу. Розроблено сервіс конвертування координат, який повинен мати вбудовану функцію перетворення між системами геокоординат MGRS, WGS-84 та СК-42.

Взаємозв’язок з іншими роботами: сервіс розробляється командою студентів. В цій роботі описано дослідження та частину виконаної роботи студентом що відповідав за роботу з історіями користувача, роботу з даними та тестування.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

WGS-84 – World Geodetic System 1984, світова геодезична система 1984 року

MGRS – Military Grid Reference System, військова сіткова система координат

СК-42 – Система Координат 1942 року

APP-6D – Allied Procedural Publication 6, процесуальна публікація альянсу 6

API – Application Programming Interface, інтерфейс прикладного програмування

GDAL – Geospatial Data Abstraction Library, бібліотека абстрагування геопросторових даних

Regex – regular expression, регулярний вираз

EPSG – European Petroleum Survey Group, європейська група нафтогазових досліджень, назва комітету що розробила однойменний реєстр геодезичних датумів.

ВСТУП

Оцінка сучасного стану об’єкта розробки. На сьогоднішній день існує багато загроз для життя людини, з початком повномасштабних воєнних дій в Україні їх стає тільки більше. Однією з таких загроз є мінна небезпека. Міни залишаються на територіях після військових конфліктів, становлячи значний ризик для мирного населення [1].

Ключовим чинником в боротьбі з мінною загрозою є ефективний збір, обробка і аналіз даних про розташування мін. У цьому контексті, сучасні геоінформаційні системи можуть відіграти критичну роль. Вони дозволяють створити карти, на які можна нанести точки з вказівкою місць розташування мін, і тим самим стати незамінним інструментом для спеціалістів по розмінуванню.

Але просто додати точки в базу даних та відобразити на мапі мало – інакше це ніяк не може відрізнятися від сучасних business intelligence систем, які дозволяють досліджувати дані. Потрібен сервіс, який зможе працювати без доступу в інтернет, так як у військовий час з цим можуть бути проблеми, система має мати зручні методи експорту та імпорту даних, підтримувати різні системи координат, як сучасні військові, так і застарілі, які використовувались ще в радянські часи, та інші необхідні інтерактивні функції.

Актуальність роботи та підстави для її виконання. Дана робота має дозволяти в реальному часі оновлювати інформацію про знайдені об’єкти, наприклад міни, систематизуючи та оптимізуючи процес розмінування.

Подібний сервіс може значно сприяти збільшенню ефективності роботи по розмінуванню і зниженню мінної загрози для населення, особливо у регіонах, де відбуваються або недавно закінчилися військові конфлікти.

Мета й завдання роботи. Метою кваліфікаційної роботи є створення сервісу для додавання об’єктів з геокоординатами в базу даних, який зможе працювати без доступу в інтернет, має мати зручні методи експорту та імпорту даних, в тому числі у форматі excel таблиць, підтримувати три різні системи координат та підтримувати стандарт APP-6D.

Згідно з метою були поставлені наступні задачі

- Дослідити три різні системи координат WGS-84, MGRS, СК-42, можливості їх парсингу з користувацького вводу, зокрема за допомогою регулярних виразів, та конвертації одна в одну.
- Дослідити стандарт військового символу APP-6D, для візуалізації військових та цивільних установок на карті, використати ці іконки розробленим сервісом для відображення точок на мапі.
- Встановити необхідні функціональності для розробки системи, встановити їх пріоритет і зобразити їх в форматі карти історій користувача.
- Розробити сервіс фіксації об’єктів. Об’єкти мають містити детальну інформацію про розташування в трьох системах координат (WGS-84, MGRS, СК-42), час виявлення, коментар, тип об’єкту, відповідний код в форматі APP-6D, джерела звітності про об’єкт, посилання на фото, ідентифікатор ворожості, дані про найближчий населений пункт.
- Розробити функціональність додавання точок як по одному елементу, так і за допомогою імпорту багатьох елементів з excel файлу.
- Розробити сервіс конвертування координат, який повинен мати вбудовану функцію перетворення між системами геокоординат MGRS, WGS-84 та СК-42.

Об'єкт, методи й засоби розроблення. Об’єкти з їх геокоординатами.

Предмет дослідження. Розробка бази для збереження та обробки об’єктів з геокоординатами.

Можливі сфери застосування. Один із методів використання даного проєкту є контролювання замінованості територій. Сервіс дозволяє збирати інформацію про об’єкти, наприклад міни, збір таких даних є критично необхідним для вирішення задач безпеки, таких як розмінування. Однією із функціональностей сервісу є

надійність джерел інформації, що, наприклад, допомагає встановлювати пріорітетність задач для саперів.

Взаємозв'язок з іншими роботами. Сервіс розробляється командою студентів. В цій роботі описано дослідження та частину виконаної роботи студентом що відповідав за роботу з історіями користувача, роботу з даними та тестування.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1 Дослідження стандарту APP-6D

APP-6D (Allied Procedural Publication 6, процесуальна публікація альянсу) – це стандарт військового символу, розроблений НАТО для розпізнавання та візуалізації військових та цивільних установок на карті. Цей стандарт допомагає у забезпеченні універсальності та зрозумілості військових символів на міжнародному рівні [2].

APP-6D використовує специфічні форми, кольори та маркери для позначення різних типів військових установок, що включають окремі військові одиниці, формування, екіпірування та інші об'єкти або ситуації на військовій карті. Це дозволяє швидко та ефективно передавати важливу військову інформацію.

Переваги APP-6D:

1. Стандарт APP-6D використовується НАТО, що забезпечує універсальне розуміння і ідентифікацію військових символів.

2. Символи APP-6D дозволяють швидко і ефективно передавати важливу військову інформацію, зменшуючи ризик неправильного тлумачення.

3. APP-6D включає велику кількість символів, що покривають широкий спектр військових та цивільних установок, обладнання та ситуацій.

APP-6D – це потужний інструмент для військової візуалізації, хоча він вимагає відповідного розуміння та постійного оновлення для найефективнішого використання.

Стандарт APP-6D використовує графічні символи, які мають визначену структуру і дозволяють зображувати велику кількість інформації військового характеру. Графічні символи APP-6D складаються з різних елементів, кожен з яких має своє значення.

Основні елементи символів APP-6D включають:

1) Форма основного символу вказує на тип військового об'єкта.
2) Модифікатори – це знаки, що додаються до основного символу, щоб надати додаткову інформацію про об'єкт. Це можуть бути знаки, що вказують на рівень командування, додаткові можливості тощо.
3) Ідентифікатори типу – це код, який ідентифікує, до якої категорії належить об'єкт, його боєготовність, а також іншу важливу інформацію.
4) Кольори використовуються для представлення типу об'єкта (дружній, ворожий, нейтральний тощо) та його боєготовності.

Основний принцип APP-6D полягає у тому, щоб передати максимально можливу кількість інформації за допомогою графічних символів, використовуючи уніфікований набір знаків та кодів. Це дозволяє швидко розпізнавати військові об'єкти на карті та їхні характеристики, що є критично важливим у військових умовах.

1.1.1 Структура APP-6D

APP-6D складається з 20 або 30 символів [2].

Розглянемо її на прикладі коду 3002010000110700000011070000000

Таблиця 1.1 – Перші 10 символів коду APP-6D

Символ	3	0	0	2	0	1	0	0	0	0
Номер п/п	1	2	3	4	5	6	7	8	9	10
Значення	Версія		Ідентичність		Набір символів		Статус	Тип цілі	Підсилювач	

Наступні 10 цифр попарно ієрархічно допомагають визначити тип об’єкту, починаючи від того чи це об’єкт на колесах, закінчуючи моделями та модифікаторами.

Таблиця 1.2 – Другі 10 символів коду APP-6D

Символ	1	1	0	7	0	0	0	0	0	0
Номер п/п	11	12	13	14	15	16	17	18	19	20
Значення	Сутність		Тип сутності		Підтип сутності		Перший модифікатор		Другий модифікатор	

Треті 10 символів є опціональними, та за вимогою замовника в проекті не підтримуються.

Таблиця 1.3 – Треті 10 символів коду APP-6D

Символ	1	1	0	7	0	0	0	0	0	0
Номер п/п	21	22	23	24	25	26	27	28	29	30
Значення	Ідентифікатор оригінатору символіки			Набір символів оригінатору символіки	Значення символів задається оригінатором символіки					

1.1.2 Відображення APP-6D

Для відображення APP-6D на мапах використовують спеціальну символіку, де спочатку обирають форму символу та колір з таблиці головних ідентифікаторів, а потім наносять модифікатори і побічні ідентифікатори.

Кількість ідентифікаторів APP-6D складається з сотень графічних символів, які залежать від регіону, тому їх дослідження не проводилось, так як їх відмальовку було покладено на WEB бібліотеку для відмальовки APP-6D «milsymbol».

Рисунок 1.1 – Приклад символу APP6D. Символ означає що це наземний об’єкт, категорії «Танк», підкатегорії «Важкий Танк». Ідентичність: свій.

1.1.3 Приклад APP-6D

Розглянемо структуру APP-6D коду на конкретному прикладі, який може бути використаний і підтримується розробленою системою:

Код «10061500001202000000» складається з 20 цифр. Розділимо їх на частини для більшої зручності:

10 0 6 15 0 0 00

12 02 03 00 00

1) Перші дві цифри означають номер версії стандарту «10»
2) Третя цифра ідентифікує чи це реальний код, навчальний чи симуляція. «0» означає реальність
3) Четверта відповідає за тип (друг, ворог, невідомо, тощо), «6» означає що об’єкт ворожий
4) П’ята та шоста разом визначають головний набір іконок (повітря, земля, море), «15» означає сухопутне обладнання.
5) Сьома цифра відповідає за статус (наявний, пошкоджений, знищений), де «0» це наявний.
6) Восьма відповідає за тип цілі (Штаб-квартира, Цільова група, Імітаційна модель), «0» значить що тип невідомий
7) Дев’ята та десята відповідає за модифікатор типу (Підсилювач ешелону, мобільності, буксируваних масивів), «00» значить невідомо
8) «12» - Наземне обладнання
9) «02» - Танк

10) «03» - Важкий танк
11) «00» Модифікатор не задано
12) «00» Модифікатор не задано

1.2 Системи координат

1.2.1 Координати WGS-84

WGS-84 (World Geodetic System 1984) – це наразі найбільш поширена система координат, яка покриває планету прямокутною сіткою, де кожну точку можна задати певним значенням широти та довготи [3].

Координати широти і довготи використовуються в географічній системі координат для визначення точного положення на Землі. Вони часто використовуються в навігації, картографії, геодезії та інших географічних науках.

Широта визначає положення на північ або південь від екватора і вимірюється в градусах від 0° до 90°. 0° широти відповідає екватору, 90° північної широти відповідає Північному полюсу, а 90° південної широти — Південному полюсу. Широта відзначається як "N" (північна) або "S" (південна) після числового значення.

Довгота визначає положення на схід або захід від Початкового меридіану, який проходить через обсерваторій в Грінвічі, Лондон. Довгота вимірюється в градусах від 0° до 180°. 0° довготи відповідає Початковому меридіану. Значення довготи відзначаються як "E" (східна) або "W" (західна) після числового значення.

Використовуючи ці два параметри, можна визначити точне положення будь-якої точки на Землі.

1.2.2 Координати MGRS

MGRS (Military Grid Reference System) – це система військових географічних координат, яка використовується для точного визначення місць на Землі у військових операціях. Вона базується на системі WGS-84, але використовує специфічний формат представлення координат і додаткові правила для поділу земної поверхні на клітинки та підклітинки [4].

MGRS координати складаються з декількох компонентів:

1) Zone (зона): Поверхня Землі розділена на 60 зон широти, кожна зона має ширину 6°. Зона позначається цифрою від 1 до 60, де 1 відповідає довготі від 180° до 174° Західної довготи, а 60 - від 174° до 180° Східної довготи.
2) Grid Square (квадрат сітки): Кожна зона поділяється на 20 квадратів, розміром 8° по широті та 10° по довготі. Квадрат сітки позначається буквою від "C" до "X", пропускаючи літери "I" та "O".
3) Eastings (східність): Кожен квадрат сітки поділяється на 10000 одиниць що називаються східності, пронумерованих від 0 до 9999, починаючи відлік від західного краю квадрата.
4) Northings (північність): Кожний квадрат сітки також поділяється на 10000 одиниць що звуться північності, пронумерованих від 0 до 9999, рахуючи від південного краю квадрата.
5) Easting Separation Factor (множник східності): Цей компонент використовується для визначення точної ширини кожного квадрата сітки. Він вказує, скільки східностей потрібно перемістится вздовж паралелі для переходу від одного квадрата гріду до наступного [5].

1.2.3 Координати СК-42

Система координат СК-42 (Система координат 1942 року) - це географічна система координат, що використовувалася в СРСР та країнах Східної Європи, включаючи Україну, Росію, Білорусь, Казахстан та інші, до прийняття системи WGS-84 (World Geodetic System 1984). СК-42 була розроблена в 1942 році на основі вимірів, здійснених у період з 1924 по 1940 роки [6].

Система координат СК-42 базувалася на геодезичній сітці, побудованій на основі глобальних вимірів Землі. Основою СК-42 було використання широти і довготи для визначення географічного положення. Широта вимірювалася у градусах, хвилинах і секундах (ГГХХ:ММ:СС), а довгота також вимірювалася у градусах, хвилинах і секундах (ГГГХХ:ММ:СС).

СК-42 мав кілька різних проекцій, включаючи трансверсальну Меркаторову проекцію, альбертову проекцію та інші, залежно від регіону та конкретного використання. [7]

Після прийняття системи WGS-84 у більшості країн світу, СК-42 стала застарілою і припинила використовуватися в багатьох геодезичних та картографічних додатках.

Використовувана в даному проєкті система СК-42 складається з декількох зон Гауса-Крюгера. Ці зони в міжнародному форматі мають позначення EPSG:284xx, де xx це номер зони, починаючи з 02 і збільшуючись на один кожні шість градусів довготи.

Щоб дізнатися номер необхідної зони в форматі необхідно взяти значення координати довготи, поділити на 6, обрізати до цілого значення, та підставити в шаблон EPSG:284xx замість іксів, з ведучим нулем, якщо число є однозначним.

Наприклад зона в якій знаходиться координата (50.538, 25.752) розраховується таким чином: значення довготи 25.752 ділиться на шість, що дорівнює 4.292 і обрізається до цілого, що дорівнює 4. Підставивши з ведучим нулем у шаблон, отримуємо назву зони EPSG:28404.

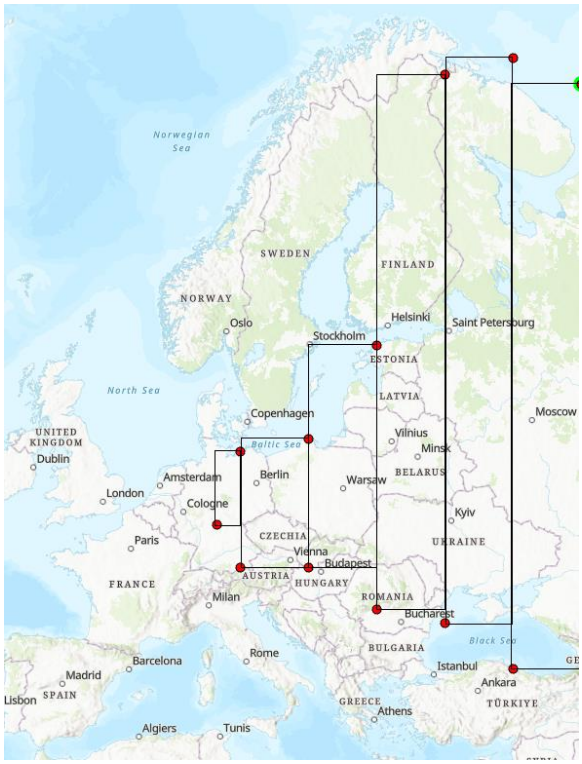

Рисунок 1.2 – Розташування зон СК-42, кожна зона позначена прямокутником. Починаючи зліва зони мають номер 2, 3, 4 і т.д. За межами прямокутників координат в системі не існує.

Як бачимо з рисунку 1.2 карта покриття світу зонами СК-42 є неповною, але так як програма розрахована на використання на території України, координати завжди будуть існувати, адже Україна повністю покрита зонами СК-42, що наприклад, не можна сказати про Фінляндію та Румунію.

1.3 Парсинг координат

1.3.1 Поняття парсингу

Парсинг – це процес аналізу і розбору тексту або даних зі строго визначеною структурою для отримання корисної інформації. Цей термін часто використовується в контексті програмування і обробки даних [8].

У широкому розумінні, парсинг означає розпізнавання та розбір вхідного тексту з метою виділення окремих компонентів, таких як слова, фрази, символи або структури даних, для подальшого використання. Парсинг може використовуватися для обробки різноманітних типів даних, включаючи текстові дані, файлові формати, мови програмування, мережеві протоколи та багато іншого.

Парсинг може включати такі етапи:

- Розбиття вхідного тексту на окремі лексичні одиниці або токени, такі як слова, числа, символи або роздільники.
- Встановлення синтаксичних правил, які описують логічну структуру даних. Це дозволяє перевірити, чи відповідає розбіраний текст визначеному синтаксису.
- Інтерпретація розпізнаних компонентів даних.

В даній роботі необхідно проводити парсинг ручного вводу координат від користувача в різних форматах.

Парсинг координат дозволяє перевірити правильність формату введених даних. Наприклад, можна перевірити, чи введені координати відповідають встановленому формату, чи правильно розділені числові значення та роздільники і т.д. Це допомагає уникнути помилок, що можуть виникнути через некоректне введення даних.

Введені користувачем координати зазвичай мають рядковий формат. Парсинг дозволяє отримати числові значення з цих рядків і перетворити їх до внутрішнього формату, який використовується в програмі або системі. Наприклад, можна перетворити рядок з координатами на цілочислені значення (для формату СК-42), або на значення з плаваючою комою (для довготи та широти).

Після парсингу, отримані координати можна використовувати для подальшої обробки або виконання різних дій. Наприклад, ці координати можуть бути використані для відображення місцезнаходження на карті, розрахунку відстаней, визначення напрямку та багатьох інших операцій, які вимагають роботи з географічними координатами.

Парсинг також дозволяє перевірити, чи введені координати потрапляють в допустимий діапазон значень. Наприклад, можна перевірити, чи введені координати входять в межі географічної області або в допустимі межі системи координат, з якою працює система.

1.3.2 Використання регулярних виразів для парсингу координат

Регулярні вирази (regex) - це механізм для пошуку та роботи з рядками тексту, що дозволяє здійснювати потужні операції зі збігами та замінами текстових шаблонів. Вони використовуються для виявлення та обробки текстових даних згідно з певними шаблонами або правилами [9].

Основна ідея регулярних виразів полягає у визначенні шаблону, який визначає певну послідовність символів, яку треба знайти або замінити. Вони дозволяють виконувати такі операції:

1) Пошук – регулярні вирази дозволяють здійснювати пошук текстових фрагментів, що відповідають певному шаблону. Наприклад, можна шукати всі слова, що починаються з певної літери або всі номери телефонів, які відповідають певному формату.
2) Валідація – регулярні вирази часто використовуються для перевірки коректності введених даних. Наприклад, можна перевірити, чи введений email відповідає стандартному формату або чи введений пароль має відповідати певним правилам.
3) Заміна – регулярні вирази дозволяють здійснювати заміну тексту згідно з певним шаблоном. Наприклад, можна замінювати всі входження певного слова на інше слово або видаляти певні символи з тексту.

4) Розбір тексту – регулярні вирази дозволяють розбивати текст на підрядки за певними шаблонами або здійснювати збір певних груп символів. Наприклад, можна розбити рядок на окремі слова або витягти числа з тексту.

Регулярні вирази використовуються в багатьох мовах програмування, текстових редакторах, базах даних, скриптових мовах та інших програмах, де необхідно виконувати роботу з текстовими даними.

Регулярні вирази використовуються для парсингу введеної користувачем довготи та широти шляхом зіставлення введеного тексту зі спеціальними шаблонами.

При парсингу координат використовуються наступні шаблони:

1) Для формату градусів, хвилин, секунд – регулярний вираз визначає структуру координат у форматі «градуси°хвилини'секунди"» з буквеним позначенням напрямку (N, S, W або E). Вираз знаходить числові значення градусів, хвилин, секунд та напрямку.
2) Для формату десяткового числа – регулярний вираз визначає структуру координат у форматі десяткового числа, де значення градусів та хвилин виражені як десяткові числа. Вираз знаходить числові значення довготи та широти.
3) Регулярні вирази можуть враховують різні варіації формату введених даних. Наприклад, вони можуть бути гнучкими щодо кількості пробілів між значеннями, можуть дозволяти використання як крапки, так і коми як роздільника десяткової частини.

Застосування регулярних виразів дозволяє виявляти та виділяти числові значення координат незалежно від їх формату, допомагаючи забезпечити точний та коректний парсинг введених даних.

1.4 Карта історій користувача

User Story Map (мапа історій користувача, карта розповіді користувача) - це візуальне представлення продукту, що допомагає краще розуміти та організовувати роботу в масштабах всієї команди. Створений Джеффом Паттоном, цей інструмент дозволяє командам бачити велику картину, в той час як вони працюють над окремими частинами продукту.

User Story Map відображає ключові взаємодії користувача з продуктом у вигляді сюжетної лінії, яка допомагає команді бачити і розуміти, як різні частини продукту взаємодіють між собою. Кожна "історія" (story) представляє собою частину користувацького досвіду, а "мапа" (map) допомагає відслідковувати, як ці історії пов'язані між собою, та в якій послідовності вони повинні бути реалізовані [10].

Спочатку, User Story Map створюється з вертикальними стовпцями, які представляють великі частини функціональності або великі етапи користувацького досвіду. Ці стовпці створюються на основі розуміння продукту та його ключових аспектів.

Після цього, додаються горизонтальні рядки, які представляють окремі користувацькі історії. Кожна історія представляє окремий аспект функціональності або частину користувацького досвіду. Ці історії розміщуються під відповідними стовпцями, відповідно до того, в якому вони мають бути реалізовані.

Використання User Story Map має ряд переваг: він допомагає командам краще розуміти продукт, визначати пріоритети та планувати роботу. Крім того, він допомагає забезпечити, що всі частини продукту будуть розроблені в збалансованій послідовності, що допомагає забезпечити найкращий можливий користувацький досвід.

Серед переваг використання User Story Map можна перерахувати наступні: [11]

- дозволяє візуально представити та розуміти велику картину продукту, включаючи функціональність і взаємодію з користувачем.

- допомагає командам встановити пріоритети для окремих функціональностей або користувацьких історій, враховуючи значимість для користувача та бізнес-цілей.
- сприяє командній роботі, оскільки всі члени команди працюють разом, щоб створити та оновити карту. Це зміцнює комунікацію та обмін знаннями між учасниками команди.
- допомагає команді зосередитися на користувацькому досвіді, наголошуючи на важливості зрозуміння того, як користувачі будуть використовувати продукт.
- дозволяє команді легко планувати та адаптувати свою роботу відповідно до змін у бізнес-цілях або потребах користувачів.

Недоліками User Story Map можна вважати:

- витрата значного часу на планування, створення та оновлення карти.
- необхідність глибокого розуміння користувацького досвіду та контексту продукту. Якщо команда не правильно зрозуміла користувача у якомусь із контекстів, карта може стати неповною або неточною, що призведе до неточної реалізації.
- Зі зростанням продукту, карта може ставати все більш складною та великою, що ускладнює її управління. Це може вимагати від команди більше часу та ресурсів для оновлення та ведення карти.
- карта має бути постійно оновлювана, щоб відображати зміни у продукті та відповідати актуальним потребам користувачів. Це створює додаткове навантаження на команду.
- існує ризик, що команда, створюючи детальну карту, може втратити загальну мету продукту, зосереджуючись на окремих історіях або функціональності.

РОЗДІЛ 2
РОЗРОБКА СИСТЕМИ

2.1 Розробка User story map

Відповідно до мети роботи було створено 5 вертикальних стовпців що відповідають за великі частини функціональності: Core with API, WEB View, Excel compatibility, Login API.

Після чого під кожним вертикальним стовбцем були створений рядок із менших функціональностей, які могли в себе включати декілька менших.

До кожної з підфункціональностей були додані вертикально історії користувача у порядку їх подальшої імплементації.

На рисунку 2.1 зображено карту історій користувача, де червоним кольором (перший рядок) позначено активності, помаранчевим (другий рядок)

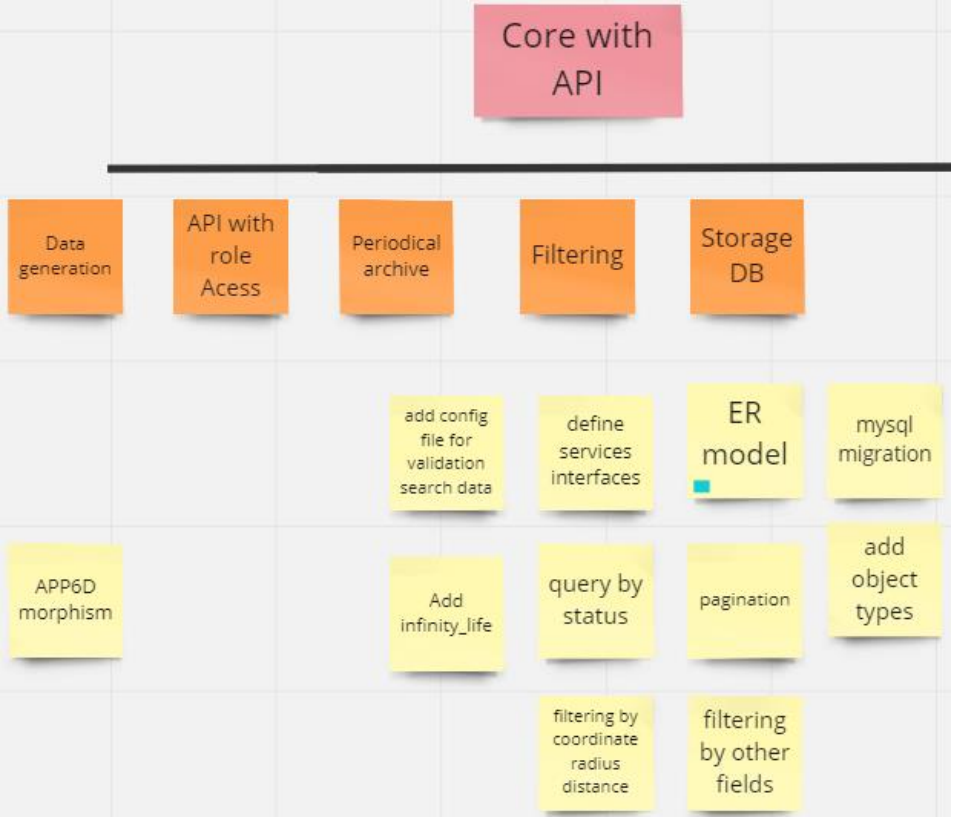

Рисунок 2.1 – Частина створеної User Story Map

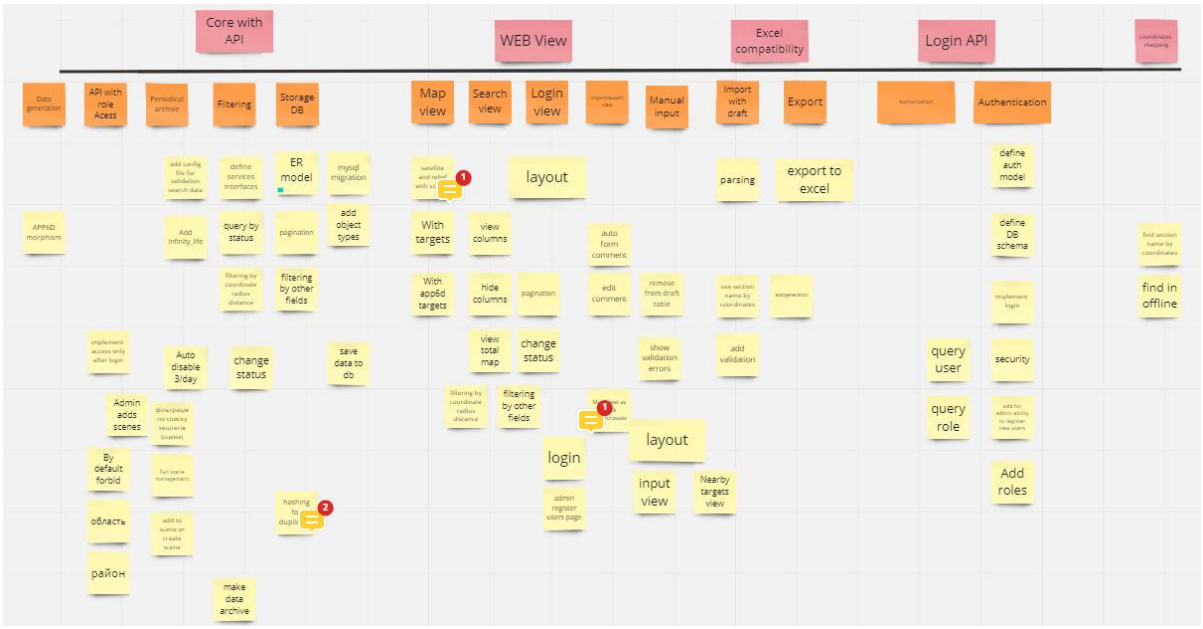

Рисунок 2.2 – Структура остаточної карти з усіма її історіями

2.1.2 Завершені історії користувача

Готова карта історій мала наступну структуру:

а) Ядро з API

1) Генерація даних
   - Морфізм APP6D
2) API з доступами за ролями
   - імплементувати доступ лише після логіну
   - адміністратор може додавати сцени
   - за замовчуванням доступ обмежений
   - керування сценами
   - зміна статусу
   - ручна архівація
3) Періодична архівація
   - додати конфігураційний файл
   - додати можливість вічного життя точкам
   - автоматично архівувати після спливу часу
4) Фільтрація

- задати інтерфейси сервісу
- запит за статусом
- фільтрація за радіусом відстані від точки
- фільтрація за списком сцен
- фільтрація за іншими полями

5) Сховище, База даних
   - ER модель
   - Mysql міграції
   - пагінація
   - збереження даних в базу даних
   - хешування дублікатів

б) WEB View

1) Інтерфейс мапи
   - відображення супутникового вигляду та вигляду мапи
   - відображення точок на мапі
   - відображення app6d іконок
2) Інтерфейс пошуку
   - приховування та показування колонок
   - показ всіх точок
   - пагінація
   - фільтрація за радіусом від точки
   - фільтрація за іншими полями
   - зміна статусу точки
3) Інтерфейс логіну
   - поля авторизації
   - авторизація
4) Інтерфейс імпорту та експорту
   - при імпорті створити чорнову таблицю зі змінами
   - автоматичне створення коментаря

- можливість редагування коментаря
- відображення всіх активних точок, імпортовані точки відображати поверх чорним кольором; користувач, працюючи з мапою – відмічає в чорновій таблиці точки, які не слід імпортувати на його думку
- відображення помилок валідації
- поля для ручного вводу даних

в) Сумісність з Excel

1) Імпорт даних
   - парсинг xslx файлу
   - валідація
2) Експорт даних
   - автогенерація xslx файлу з обраної відфільтрованої таблиці

г) Login API

1) Авторизація
   - API логіну
2) Аутентифікація
   - задання моделі аутентифікації
   - створення схеми бази даних
   - імплементація логіну
   - запит користувача та його ролі
   - створення ролей та їх функцій

д) Геокодування

1) використання реверсивне геокодування
2) доступ до офлайн реверсивного геокодування

2.2 Морфізм app6d кодів та категорій

Валідація даних — це процес перевірки коректності та повноти даних перед їх обробкою або збереженням. Цей процес важливий, оскільки він допомагає забезпечити надійність та якість даних, а також запобігти помилкам, що можуть виникнути внаслідок некоректного введення даних або їх обробки.

Валідація даних може включати такі перевірки:

- Перевірка типів даних
- Перевірка діапазону значень
- Перевірка формату значень
- Перевірка обов'язковості заповненості даних (що поля не пусті)
- Перевірка унікальності даних (наприклад для унікальних ідентифікаторів)

Валідація даних може бути реалізована на різних рівнях, включаючи клієнтську сторону (у веб-браузері користувача), серверну сторону (на веб-сервері), а також при взаємодії з базою даних. В даній роботі валідація реалізована на серверній стороні.

Морфізм даних часто використовується для позначення перетворення або мапінгу даних з одного формату або структури в інший. Такі перетворення можуть включати декодування, кодування, серіалізацію, десеріалізацію, преобразування типів даних та інші подібні процеси.

Зазвичай, при розробці програмного забезпечення, морфізми даних є важливими, тому що вони допомагають маніпулювати даними таким чином, що вони можуть бути коректно інтерпретовані різними системами, мовами програмування або інтерфейсами. Це особливо важливо в сучасних високорівневих та розподілених системах, де дані часто передаються і обмінюються між різними компонентами, сервісами або мережами.

2.2.1 Збережені дані

Щоб пов’язати зручні для користувача назви категорій об’єкту, та дозволяти адміністратору системи додавати або видаляти об’єкти до системи, були створені конфігураційні файли типу JSON для їх легкого читання та зберігання, що дозволяє швидко міняти категорії без перезапуску системи.

JSON (JavaScript Object Notation) - це легковаговий формат даних, оснований на тексті, що використовується для збереження та обміну даними. Цей формат є мовою незалежним і може бути використаний у багатьох мовах програмування, включаючи JavaScript, Python, PHP, Java та багатьох інших.

Основні структурні складові JSON включають об'єкти і масиви:

Об'єкти у JSON представлені у вигляді набору пар "ключ/значення", огорнутих у фігурні дужки «{}». Ключі це символьні рядки, а значення можуть приймати значення символьних рядків, чисел, булевих значень, об'єктів, масивів або типу null.

Масиви у JSON представляють собою упорядковані списки значень, огорнуті у квадратні дужки []. Значеннями можуть бути символьні рядки, числа, об'єкти, інші масиви, булеві значення або null.

Переваги використання JSON

- JSON є стандартом відкритого формату, що підтримується багатьма мовами програмування, що робить його універсальним рішенням для обміну даними між різними системами.
- Дані у форматі JSON легко читаються і розуміються людьми, що спрощує розробку та налагодження.
- Багато мов програмування мають вбудовані функції для аналізу JSON, що дозволяє легко перетворити дані JSON на структури даних, які використовуються в цих мовах.

- JSON був розроблений на основі синтаксису JavaScript, тому він може бути безпосередньо використаний в JavaScript. Це робить JSON особливо корисним для веб-розробки, де JavaScript є основною мовою програмування на клієнтській стороні.
- Завдяки своїй текстовій природі, JSON легко передається через мережу і може бути переданий між сервером і клієнтом без необхідності використання спеціальних механізмів перетворення.
- JSON може представляти досить складні структури даних, включаючи вкладені об'єкти та масиви, що робить його підходящим для використання в сучасних додатках, які часто вимагають передачі складних структур даних.

Категорії записані в конфігураційний файл складаються з назви, app-6d коду, кількості днів скільки цей об’єкт буде активним та списку регулярних виразів для розпізнавання категорії з текстових рядків.

```
[
  {
    "назва": "Міна",
    "app6d": "10061000001413000000",
    "активність-днів": 90,
    "oivt_regexes": [
      "^мінне.*",
      "^міна.*",
      "^міни.*",
      "^.*мінован.*"
    ]
  },
    {
    "назва": "Літак",
    "app6d": "10060100001101000000",
    "активність-днів": 5,
    "oivt_regexes": [
      "^аеро.*",
      "^.*аеро.*"
    ]
  }
],
```

Рисунок 2.2 – Приклад JSON конфігурації для категорій

2.2.2 Морфізм даних

Оскільки збережені APP-6D коди мають заздалегідь визначене значення ідентифікації ворожості, а імпортовані дані можуть як містити, так і не містити необхідні дані, необхідно реалізувати морфізм даних.

При імпорті даних до системи були розглянуті три поля, що відповідають за категорію об’єкту: «Назва категорії», «APP-6D» та «Ворожість». Оскільки кожне з них може бути як присутнім так і відсутнім або не валідним, маємо 2 в степені 3 можливих комбінацій, що дорівнює 8.

Було розроблено специфікацію, яка відповідає за порядок дій при наявності тих чи інших даних. Специфікація наведена у вигляді таблиці.

Таблиця 2.4 – Порядок дій за різних комбінацій наявних даних

Наявність даних			**Дії**	**За яких умов повертати помилку**
Назва категорії	**APP-6D**	**Ворожість**		
Так	Так	Так	Вставка ворожості в наявний APP-6D; Перевірка наявності іншої категорії за наявним APP-6D	Якщо збережена категорія відповідна APP-6D не співпадає з наданою
Так	Так	**Ні**	Автоматичне встановлення ворожості як «Ворог»; Перевірка наявності іншої категорії за наявним APP-6D	Якщо збережена категорія відповідна APP-6D не співпадає з наданою
Так	**Ні**	Так	Пошук збереженого APP-6D коду за наданою категорією співпадінням та регулярними виразами	Якщо APP-6D код не знайдено

Наявність даних			**Дії**	**За яких умов повертати помилку**
Назва категорії	**APP-6D**	**Ворожість**		
Так	**Ні**	**Ні**	Автоматичне встановлення ворожості як «Ворог»; Пошук збереженого APP-6D коду за наданою категорією співпадінням та регулярними виразами	Якщо APP-6D код не знайдено
Ні	Так	Так	Вставка ворожості в наявний APP-6D; Пошук збереженої категорії за APP-6D кодом виключаючи ворожість	Якщо категорія не знайдена
Ні	Так	**Ні**	Автоматичне встановлення ворожості як «Ворог»; Пошук збереженої категорії за APP-6D кодом виключаючи ворожість	Якщо категорія не знайдена
Ні	**Ні**	Так	Повертається помилка про недостатність даних	
Ні	**Ні**	**Ні**	Повертається помилка про недостатність даних	

При провалі перевірки на валідність категорії, коду або ідентичності, буде також повернута помилка про провалену валідацію.

2.2 Валідація кодів та категорій

Необхідно перевірити назву категорії, APP-6D та ворожість на валідність.

Ідентичність ворожості для зручності користувача відображається текстовим рядком, не дивлячись на те що APP-6D сприймає їх як одну цифру. Відповідність рядків є налаштовуваною і також задається в конфігураційному файлі.

```
[
  { "id": 0, "name": "Очікується" },
  { "id": 1, "name": "Невідомо" },
  { "id": 2, "name": "Схоже друг" },
  { "id": 3, "name": "Друг" },
  { "id": 4, "name": "Нейтральний" },
  { "id": 5, "name": "Підозрілий" },
  { "id": 6, "name": "Ворог" }
]
```

Рисунок 2.3 – Відповідність текстових рядків ідентичності до числа APP-6D

Розробляємо перевірку наступним чином

а) Імпортоване тестове значення ворожості має співпадати зі збереженим у конфігураційному файлі

б) APP-6D має бути перевіреним наступним чином

1) Рядок складається лише з цифр
2) Рядок має довжину 20 символів
3) Номер версії завжди дорівнює 10
4) Третій символ завжди дорівнює 0 (реальність)

в) Категорія має співпадати зі збереженим категорія в конфігураційному файлі

2.2.3 Розробка функціоналу морфізму та валідації даних

Мова програмування: Golang

Go, відома також як Golang, - це статично типізована, компільована мова програмування, яку розробили в Google. Вона була створена, щоб вирішити

проблеми, які виникають при роботі з великими системами, особливо щодо продуктивності та швидкості.

Go має набір можливостей, що робить його зручним для розробки веб-сервісів:

- Go має потужну стандартну бібліотеку, яка включає в себе засоби для роботи з мережею, веб-серверами, криптографією, обробкою даних і багато іншого. Це робить Go мовою, яка може використовуватися для створення розгорнутих веб-сервісів без необхідності встановлення додаткових бібліотек або фреймворків.
- Go має вбудовану підтримку горутин (goroutines) та каналів, що дозволяє ефективно розподіляти обчислювальні ресурси між різними задачами, що виконуються паралельно. Це особливо корисно для веб-сервісів, які часто потребують обробки великої кількості одночасних запитів.

Go є компільованою мовою, що означає, що виконуваний код виконується безпосередньо на апаратному рівні, що робить його дуже швидким порівняно з інтерпретованими мовами, такими як Python або JavaScript.

Go відома своєю відмінною підтримкою паралельної обробки, що робить її ідеальною мовою для розробки великомасштабних, високопродуктивних систем.

Також Go є статично типізованою мовою, що зменшує шанси виникнення помилок типів у коді. Програми написані цією мовою компілюється дуже швидко, що робить процес розробки більш ефективним і доволяє швидше розгортати готові рішення наприклад в Docker.

Для обробки JSON при імпорті та прочитанні конфігураційних файлів використовувася пакет Golang encoding/json. Цей пакет надає засоби для серіалізації (також відомої як "маршалінг") структур даних Go до формату JSON та десеріалізації (або "анмаршалінгу") даних JSON назад в структури Go.

```
type ObjectCategory struct {
  Name       string   `json:"назва"`
  ActiveDays int      `json:"активність-днів"`
  App6d      string   `json:"app6d"`
  Regexes    []string `json:"oivt_regexes"`
}
```

Рисунок 2.4 – Використання encoding/json для структури даних конфігураційного файлу

Для морфізму та валідації необхідних полів була створена окрема структура Fields, та написані методи ресівери setIdentity, setCode, trySetCategory, equalWithoutIdentity, getIdentityFromCode та ValidateCode, що відповідають за функціонал вказаний в таблиці 2.4 (додаток А).

Перевірка роботи морфізму та валідації відбувалася через існуючий інтерфейс імпорту даних з екселю «/excel-import/excel», який був також розроблений мовою Golang і знаходиться на тому ж сервісі.

Помилки валідації та морфізму повертаються у вигляді масиву текстових рядків (так як помилок може бути декілька) для кожного імпортованого рядка. Для повернення через веб-інтерфейс помилки додаються до морфованих даних в форматі JSON як окремі поля «errors».

```
"values": {
  "lon": "37.3157",
  "lat": "48.0084",
  "height": "0",
  "sk42_x": "5320836",
  "sk42_y": "7374444",
  "mgrs": "37U CP 74381 18606",
  "detectionTime": "2023-03-06 18:54:00",
  "oivt": "string",
  "reliabilityCategory": "A1",
  "source": "Петро Петрович",
  "link": "string",
  "comment": "string",
  "identity": "Ворог",
  "app6dCode": "10061500001202000000",
  "district": "Покровський район",
  "region": "Донецька область",
  "mostCloseSettlement": "Іллінка",
  "objectCategory": "Козацький підводний човен",
  "summaryComment": "string",
  "id": "00f1c0d8-ec79-4e6e-8ba6-befbd8261f34"
},
"errors": "не вдалося провалідувати app6d код 10061500001202000000, встановлено шаблон."
```


Рисунок 2.5 – Приклад результату морфізму та валідації. Поле errors містить помилку валідації

2.3 Парсинг координат

2.3.1 Парсинг WGS-84 координат

У даній роботі був розроблений парсер на мові C#, який призначений для парсингу координат у форматі WGS-84.

Парсер складається з двох частин: одна реалізована з використанням можливостей вбудованої бібліотеки Coordinatesharp, яка забезпечує функціональність парсингу та обробки географічних координат. Ця бібліотека має вбудовані методи для розбору різних форматів координат, включаючи формат WGS-84. Але вона не сприймає введення у форматі градусів, мінут та секунд, лише широту та довготу. Була написана булева функція для парсингу (малюнок 2.5).

```
public static bool TryParseWgs84(string str, out CoordinateWGS84 c)
{
    return
        TryParseWgs84WithLibraryParser(str, out c) ||
        TryParseWgs84Regex(str, out c);
}

private static bool TryParseWgs84WithLibraryParser(string str, out CoordinateWGS84 c)
{
    c = new CoordinateWGS84(0, 0);
    if (!CoordinateSharp.Coordinate.TryParse(str, out var coordinate))
        return false;

    c = new CoordinateWGS84(coordinate.Latitude.ToDouble(), coordinate.Longitude.ToDouble());
    return true;
}
```

Рисунок 2.5 – Функція TryParseWgs84 для виклику парсингу, бібліотечним методом TryParseWgs84WithLibraryParser, яка при невдачі викликає парсинг за регулярним виразом

Однак, у випадку, якщо вбудованому парсеру не вдається правильно розпізнати формат або розібрати введені координати, використовується окремий парсинг за допомогою регулярних виразів.

Для цього був створений спеціальний регулярний вираз, який здатний виявляти та виділяти числові значення широти та довготи з введеного рядка. Вираз зображений на рисунку 2.6.

```
(\d{1,2}[,\.]\d+)[,\.]?\s*(\d{1,2}[,\.]\d+)
```

Рисунок 2.6 – Регулярний вираз для розбору координат WGS-84

Вираз складається з наступних частин:

1) `(\d{1,2}[,\.]\d+)` – група захоплення першої координати
    а) \d – відповідає цифрі
    б) {1,2} – повторює попередню лексему від 1 до 2 разів, якомога більше разів, віддаючи назад стільки, скільки потрібно
    в) `[,\.]` – співпадіння з одним символом зі списку, тобто співпадіння із символом коми або крапки

г) \d – відповідає цифрі

д) + – повторює попередню лексему необмежену кількості разів, стільки разів, скільки це можливо, повертаючи його за потреби

2) [,\.]?\s* - група захоплення розділяючої коми

а) `[,\.]` – співпадіння з одним символом зі списку, тобто співпадіння із символом коми або крапки

б) ? – збігається з попереднім токеном у діапазоні від нуля до одиниці якомога більше разів, повертаючи його за потреби

в) \s – відповідає будь-якому пробілу

г) * – збігається з попереднім токеном від нуля до необмеженої кількості разів, стільки разів, скільки можливо, повертаючи його за потреби

2) 3) `(\d{1,2}[,\.]\d+)` - група захоплення другої координати

а) \d – відповідає цифрі

б) {1,2} – повторює попередню лексему від 1 до 2 разів, якомога більше разів, віддаючи назад стільки, скільки потрібно

в) `[,\.]` – співпадіння з одним символом зі списку, тобто співпадіння із символом коми або крапки

г) \d – відповідає цифрі

д) + – повторює попередню лексему необмежену кількості разів, стільки разів, скільки це можливо, повертаючи його за потреби

Цей регулярний вираз дозволяє виявляти два числа з плаваючою комою, з максимум трьома знаками до коми, та необмеженим після. Кома всередині такого числа може бути символом коми або крапки. Два таких числа можуть бути розділені комою, комою з довільною кількістю пробілів, або просто довільною кількістю пробілів. Реалізована функція TryParseWgs84Regex, що зображена на рисунку 2.7, спочатку розпізнає два числа як рядки, замінює дробовий розділювач на крапку і розпізнає числа в тип з плаваючою комою.

```
private static bool TryParseWgs84Regex(string str, out CoordinateWGS84 c)
{
    c = new CoordinateWGS84(0,0);
    var rx = new Regex(@"(\d{1,2}[,\.]\d+)[,\.]?\s*(\d{1,2}[,\.]\d+)",
    RegexOptions.Compiled | RegexOptions.IgnoreCase);

    var matches = rx.Matches(str);
    if (matches.Count == 0) return false;

    var groups = matches[0].Groups;
    if (groups.Count != 3) return false;

    var lat = double.Parse(groups[1].Value.Replace(",", "."));
    var longi = double.Parse(groups[2].Value.Replace(",", "."));

    c = new CoordinateWGS84(lat, longi);
    return true;
}
```

Рисунок 2.7 – Булева функція TryParseWgs84Regex що реалізує перевірку регулярним виразом

2.3.2 Парсинг MGRS координат

Парсинг MGRS координат був повністю покладений на бібліотеку Coordinatesharp так як варіаій написання координати не багато, і відрізняється тільки внутрішніми пробілами, з чим справляється парсер. Реалізований парсер зображений на рисунку 2.8.

```
public static bool TryParseMgrs(string str, out CoordinateMGRS c)
{
    c = new CoordinateMGRS(_zeroMGRS);
    return TryParseMgrsWithLibraryParser(str, out c);
}

private static bool TryParseMgrsWithLibraryParser(string str, out CoordinateMGRS c)
{
    c = new CoordinateMGRS(_zeroMGRS);
    if (!CoordinateSharp.MilitaryGridReferenceSystem.TryParse(str, out var coordinate))
        return false;

    c = new CoordinateMGRS(coordinate);
    return true;
}
```

Рисунок 2.8 – Булеві функції що реалізують парсинг MGRS

2.3.3 Парсинг СК-42 координат

Парсинг СК-42 є найскладнішим з усіх трьох, так як не існує готового рішення яке можна задіяти в проекті. Парсинг був розроблений по аналогії з парсингом за регулярним виразом WGS-84. Для цього був створений окремий регулярний вираз.

```
(?i)(?:x[=:]\s*|\b)(?<x>\d{1,2}-?\d{5})(?:\s*[,;]\s*|\s+)
(?:y[=:]\s*|\b)(?<y>\d{1,2}-?\d{5})\b
```

Рисунок 2.9 – Регулярний вираз для розбору координат СК-42

Створений вираз складається з наступних частин:

1) (?i): Це модифікатор регістру, який вказує, що пошук повинен бути незалежним від регістру.
2) (?:x[=:]\s*|\b)
    a) (?:) використовується для групування шаблону, але не створює змінної збереження результатів. У цьому випадку, вона використовується для визначення початку шаблону, де координата x може починатись.

б) x[=:]\s*:

- X - літера x, яка вказує на координату x.
- [=:] - символи дорівнює або двокрапка, які можуть служити як роздільник між літерою та значенням координати x.
- \s* - нуль або більше пробільних символів (пробіл, табуляція, перенос рядка).
- \b - вказує на границю слова, тобто відповідає, коли попередній символ є неалфавітно-цифровим символом, а наступний символ є алфавітно-цифровим символом. В даному випадку, \b використовується для визначення початку слова перед значенням координати x.

3) (?<x>\d{1,2}-?\d{5})

а) (?<x>...) - визначає групування та надає ім'я групі. Вона використовується для виявлення та збереження значення координати x.

б) \d{1,2} - відповідає одній або двом десятковим цифрам, що вказують на числове значення координати.

в) -? - необов'язковий дефіс, який дозволяє наявність або відсутність мінуса.

г) \d{5} - відповідає п'яти десятковим цифрам, що вказують на числове значення координати.

4) (?:\s*[,;]\s*|\s+) - Ця група визначає роздільники між значеннями координат x та y.

а) \s*[,;]\s* - відповідає нульовій або більшій кількості пробільних символів, за якими слідує кома, або крапка з комою, і знову нульова або більша кількість пробільних символів.

б) | - це оператор альтернативи, який вказує, що можливий альтернативний шаблон.

в) \s+ - відповідає одному або більше пробільним символам.

5) (?:y[=:]\s*|\b) - ця група аналогічна групі (?:x[=:]\s*|\b), але використовується для визначення початку шаблону, де може починатись координата y.

6) (?<y>\d{1,2}-?\d{5})\b - Ця група аналогічна групі (?<x>\d{1,2}-?\d{5}), але використовується для виявлення та збереження значення координати y.

а) (?<y>...) - <y> - це ім'я групи для значення координати y.

б) \d{1,2} - відповідає одній або двом десятковим цифрам, що вказують на числове значення координати.

в) -? - необов'язковий дефіс, який дозволяє наявність або відсутність мінуса.

г) \d{5} - відповідає п'яти десятковим цифрам, що вказують на хвилини.

д) \b: вказує на границю слова, тобто відповідає, коли наступний символ є неалфавітно-цифровим символом.

Після парсингу координат, отримані числові значення широти та довготи можуть бути використані для подальшої обробки, такої як відображення на карті, розрахунок відстаней, пошук найближчих точок та інше.

```
public static bool TryParseSK42(string str, out CoordinateSK42 c)
{
    c = null;
    return TryParseSK42Regex(str, out c);
}

private static bool TryParseSK42Regex(string str, out CoordinateSK42 c)
{
    c = new CoordinateSK42(0,0);
    var rx = new Regex(@"(?i)(?:x[=:]\s*|\b)(?<x>\d{1,2}-?\d{5})(?:\s*[,;]\s*|\s+)(?:y[=:]\s*|\b)(?<y>\d{1,2}-?\d{5})\b",
        RegexOptions.Compiled | RegexOptions.IgnoreCase);

    var matches = rx.Matches(str);
    if (matches.Count == 0)
        return false;

    var groups = matches[0].Groups;
    if (groups.Count != 3)
        return false;

    var strX = groups[1].Value.Replace("-", "");
    var strY = groups[2].Value.Replace("-", "");

    if (!(int.TryParse(strX, out var x) && int.TryParse(strY, out var y)))
        return false;

    c = new CoordinateSK42(x, y);
    return true;
}
```

Рисунок 2.10 – Булеві функції що реалізують парсинг СК-42

2.4 Конвертування координат

Для точного конвертування координат між системами WGS-84, СК-42 та MGRS в цій науковій роботі було використано мову програмування C# та сторонні бібліотеки, CoordinateSharp та GDAL.

Так як WGS-84 та MGRS можуть бути розпізнані та обробляються бібліотекою CoordinateSharp, ця бібліотека дозволяє конвертувати їх одна в одну.

Координату СК-42 може конвертувати тільки бібліотека GDAL і тільки між СК-42 та WGS-84. Тобто для конвертування з СК-42 в MGRS потрібно буде провести подвійну конвертацію.

Бібліотеку CoordinateSharp можна знайти серед пакетів nuget доступних в інтернет, або зазвичай прямо з інтегрованого середовища розробки.

Бібліотека GDAL має декілька імплементацій що розташовані серед репозиторію пакетів nuget. Спочатку конвертація використовувала пакет під назвою GDAL, але згодом виявилось що ця бібліотека не працює з фреймворком .NET core, на якому базується проєкт, тому було обрано інший пакет під назвою

MaxRev.Gdal.Core та допоміжний до нього пакет MaxRev.Gdal.LinuxRuntime.Minimal для запуску на операційній системі Linux.

Точність конвертування визначалася розміром похибки в метрах при вимірюванні відстані від оригінальної точки та конвертованої. Для СК-42 необхідно проводити подвійну конвертацію, так як жодна з існуючих програм у відкритому доступі не має можливості напряму відобразити на мапі СК-42 координати.

Для початку було розроблено простий клас для роботи з координатами довготи та широти CoordinateWGS84.

```
public class CoordinateWGS84
{
    public double Longitude;
    public double Latitude;

    public double X => Longitude;
    public double Y => Latitude;

    public CoordinateWGS84(double latitude, double longitude)
    {
        Longitude = longitude;
        Latitude = latitude;
    }
}
```

Рисунок 2.11 – Структура класу для роботи з WGS-84 координатами

Розроблено функції для конвертування WGS-84 та MGRS одна в одну, такі функції не потребують етапу валідації, бо вся валідація координат відбулася на етапі парсингу через бібліотеку. Якби координати в тій чи іншій системі були невалідні, бібліотека повернула би виключення, яке можна обробити і повернути користувачу.

```
public static CoordinateMGRS WGS84_to_MGRS(CoordinateWGS84 wgs)
{
    var c = new CoordinateSharp.Coordinate(wgs.Latitude, wgs.Longitude);
    return new CoordinateMGRS(c.MGRS);
}

public static CoordinateWGS84 MGRS_to_WGS84(CoordinateMGRS mgrs)
{
    var coord = MilitaryGridReferenceSystem.MGRStoLatLong(mgrs.Value);
    return new CoordinateWGS84(coord.Latitude.ToDouble(), coord.Longitude.ToDouble());
}
```

Рисунок 2.12 – Функції для конвертування WGS-84 та MGRS

Для роботи з системою координат СК-42 була використана бібліотека GDAL. GDAL надає набір інструментів для роботи з великою кількістю систем координат, однак робота з цією бібліотекою є набагато складнішою, так як це обгортка над однойменною бібліотекою написаною мовою C. Тому бібліотека використовується лише для конвертації в СК-42 та навпаки [12].

Так як координати СК-42 існують лише в прямокутниках заданих стандартом, необхідно провести валідацію точок по довготі та широті, перед тим як конвертувати їх. Якщо координата не існує в системі СК-42 бібліотека-конвертер все одно поверне координати, тільки вони будуть теж невалідними.

Для того щоб уникнути цього, був створений список прямокутників, в яких можуть існувати СК-42 координати, і які використовуються в даному проєкті.

```
private static List<Rectangle> _sk42Limits = new()
{
    new Rectangle(50.2100, 9.9300, 54.1800, 12.0000),
    new Rectangle(47.7500, 12.0000, 54.8300, 18.0000),
    new Rectangle(47.7400, 18.0000, 59.2900, 24.0000),
    new Rectangle(45.2100, 24.0000, 69.4700, 30.0000),
    new Rectangle(44.3800, 30.0000, 69.9600, 36.0000),
    new Rectangle(41.4300, 36.0000, 69.1700, 42.0000),
    new Rectangle(38.8400, 42.0000, 80.8600, 48.0000),
    new Rectangle(37.0000, 48.0000, 84.0000, 54.0000),
    new Rectangle(36.6000, 54.0000, 84.0000, 60.0000),
};
```

Рисунок 2.13 – Список прямокутників що покривають валідні зони для СК-42 у вигляді WGS-84 координат

Також для роботи з СК-42 необхідно вирахувати стандарт EPSG. Для цього необхідно взяти значення координати довготи, поділити на 6, обрізати до цілого значення, та підставити в шаблон EPSG:284xx, де підставити отримане число замість іксів, з ведучим нулем, якщо число є однозначним.

```
const int EPSG_SK42_BASE_CODE = 28400;
var EPSG_SK42_ZONE_CODE = EPSG_SK42_BASE_CODE + (int)Math.Ceiling(wgs.X / 6);
var outputSrs = new OSGeo.OSR.SpatialReference("");
outputSrs.ImportFromEPSG(EPSG_SK42_ZONE_CODE);
```

Рисунок 2.14 – Реалізація визначення стандарту EPSG

Після чого було розроблено функції для конвертування з та в СК-42 та WGS-84 та для перевірки валідності координат.

Функції було поєднано в клас CoordinateTransformer, який дозволяє конвертувати координати з будь-якої в будь яку. Цей клас використовується класом Coordinate який має функції конструктори з будь якого з трьох видів координат, які викликають конвертацію всередині себе, та заповняють поля з координатами в різних системах.

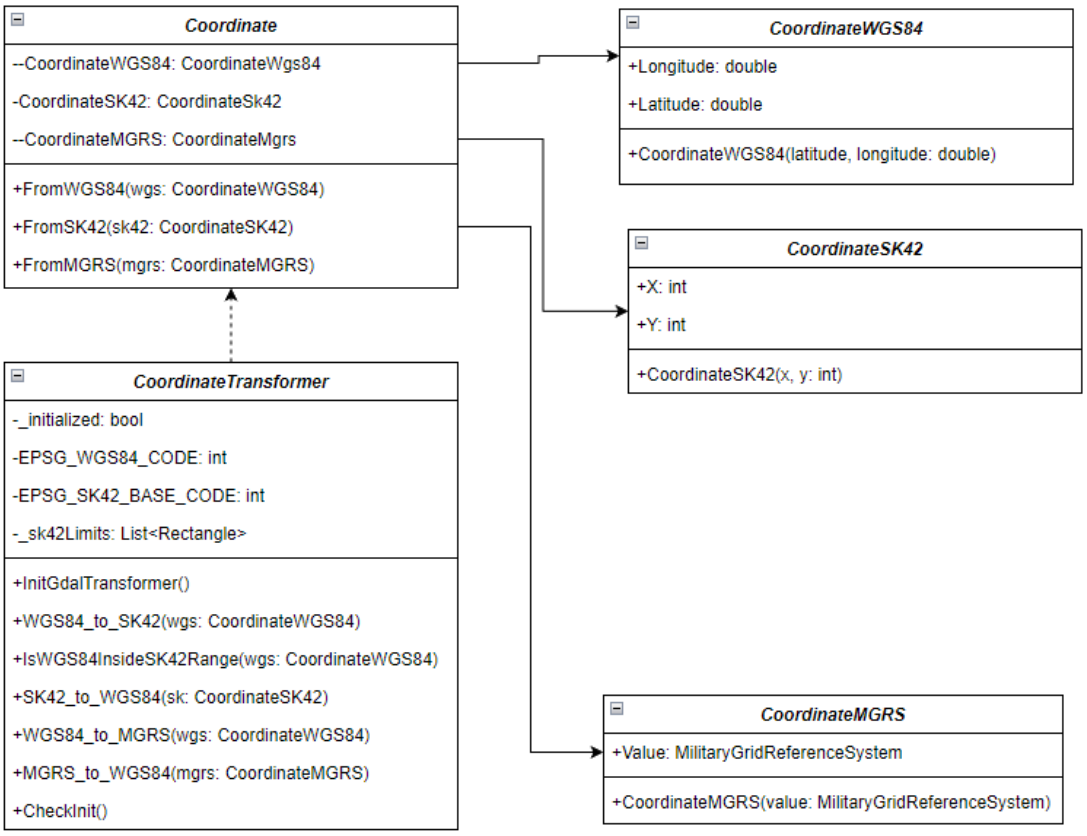

Рисунок 2.15 – Діаграма класів що відповідають за конвертацію координат

Отже написані класи можна використовувати наступним чином: створити об’єкт класу Coordinate, та присвоїти їх результат виконання однієї з функцій FromWGS84, FromSK42 або FromMGRS, в залежності від того який тип координат нам дано. Після чого конструктори конвертують координати через CooerdinateTransformer, запишуть результат в властивості класу Coordinate і повернуть новий її екземпляр.

ВИСНОВКИ

Досліджена та розроблена карта історій користувача, яка допомогла краще зрозуміти та організовувати роботу над проєктом.

Досліджено три різні системи координат WGS-84, MGRS, СК-42, та можливості їх парсингу та конвертації. Зокрема було використано можливості регулярних виразів для розбору координат.

Досліджений стандарт військового символу APP-6D, розроблений НАТО для розпізнавання та візуалізації військових та цивільних установок на карті, символ використовується розробленим сервісом для відображення іконок на мапі.

Розроблено сервіс фіксації об’єктів. Об’єкти містять детальну інформацію про розташування в трьох системах координат (WGS-84, MGRS, СК-42), час виявлення, коментар, тип об’єкту, відповідний код в форматі APP-6D, джерела звітності про об’єкт, посилання на фото, ідентифікатор ворожості, дані про найближчий населений пункт.

Розроблено функціональність додавання точок як по одному елементу, так і за допомогою імпорту багатьох елементів з excel файлу.

Розроблено сервіс конвертування координат, який повинен мати вбудовану функцію перетворення між системами геокоординат MGRS, WGS-84 та СК-42.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Україна – найбільш замінована країна світу: скільки української території забруднено мінами [Електронний ресурс] – Режим доступу до ресурсу: https://www.slovoidilo.ua/2023/03/02/infografika/bezpeka/ukrayina-najbilsh-zaminovana-krayina-svitu-skilky-ukrayinskoyi-terytoriyi-zabrudneno-minamy.
2. NATO STANDARD APP-6. // NATO JOINT MILITARY SYMBOLOGY / NORTH ATLANTIC TREATY ORGANIZATION, 2017. – 922 с. – (Edition D Version 1).
3. WORLD GEODETIC SYSTEM 1984 (WGS 84) [Електронний ресурс] // Office of Geomatics, National Geospatial-Intelligence Agency. – 2022. – Режим доступу до ресурсу: https://earth-info.nga.mil/?dir=wgs84&action=wgs84.
4. Smart Soldier: Understanding the Military Grid Reference System [Електронний ресурс] // Army Knowledge Centre. – 2022. – Режим доступу до ресурсу: https://cove.army.gov.au/article/smart-soldier-understanding-military-grid-reference-system
5. UNIVERSAL GRIDS AND GRID REFERENCE SYSTEMS, 2014. – 101 с. – (Version 2.0.0).
6. Система координат 1942 года, «СК-42» [Електронний ресурс] // GEOSTART – Режим доступу до ресурсу: https://geostart.ru/post/326.
7. Серпухов В.И. Курс общей геологии / Серпухов В.И., Билибина Т.В.., 1976. – 536 с.
8. Frost, R. Modular and Efficient Top-Down Parsing for Ambiguous Left-Recursive Grammars / Frost, R., Hafiz, R, Callaghan, P.., 2007. – 120 с.
9. Leung H. Regular Languages and Finite Automata / Hing Leung., 2010. – 12 с.
10. Cohn M. User Stories Applied / Mike Cohn., 2004 – 291 с.;
11. Cohn M. Advantages of User Stories for Requirements / Mike Cohn // Agile Estimating and Planning / Mike Cohn., 2006 – 368 с.;
12. Warmerdam F. GDAL Documentation / F. Warmerdam, E. Rouault., 2023. – 1146 с.