ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра комп'ютерних наук

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «Розробка програмного забезпечення клієнтської частини для інтерактивної карти візуалізації протипаводкових заходів в Івано-Франківській області»

Виконав: студент 4 курсу, групи КНД–42
спеціальності
122 Комп'ютерні науки
Саміляк Іван Миколайович

Керівник Іщеряков Сергій Михайлович

Рецензент _____
(прізвище та ініціали)

Нормоконтроль _____
(прізвище та ініціали)

Київ – 2021

РЕФЕРАТ

Текстова частина бакалаврської роботи: 41с., 31 рис., 6 джерела.

Об`єкт дослідження – інтерактивна карта протипаводкових заходів.

Предмет дослідження – інтерактивний веб-додаток для відображення об’єктів та додаткової інформації на карті

Мета роботи – розробка зручного та багатофункціонального інтерфейсу інтерактивної карти об’єктів протипаводкової інфрастркутури Івано-Франківської області на базі веб-додатка.

Методи дослідження – створення ПЗ для відображення карти, робота з мовою програмування JavaScript, робота з програмними інтерфейсами OpenStreetMaps.

Сфера застосування – аналіз впливу готовності до сезонних паводків та аналізу наслідків

За результатами роботи отримано робочу версію інтерактивного веб-додатка з картою об’єктів протипаводкових заходів Івано-Франківської області. Реалізовано механізм зручного відображення та фільтрації додаткової інформації про об’єкти.

ВСТУП

Підготовка до сезону паводків в Івано-Франківській області це довга та кропітка робота багатьох спеціалістів. Щороку потрібно сформувати колосальний масив інформації про кількість опадів, захисні споруди та їхній стан, пошкодження інфраструктури після попередніх паводків. В таких обставинах з'явилась необхідність сформувати зручний портал з інтуїтивно зрозумілою картою для візуалізації даних, які надходять з різних джерел, в одному місці. Вона покликана спростити пошук інформації та пришвидшити прийняття рішень.

Для досягнення мети необхідно вирішити наступні завдання: проаналізувати практичну цінність проекта та сформувати конеретні потреби до його функціоналу, побудувати архітектуру проекту та сформувати принципи реалізації, детально описати процес реалізації.

В першому розділі ми детально проаналізуємо практичну цінність, яку має нести карта. Після чого сформуємо конкретні потреби до функціоналу майбутнього порталу. Також ми проведемо аналіз технологій, які можуть бути задіяні в реалізації.

В другому розділі буде описаний етап проектування продукту – загальний формат вхідних даних, опис клієнт-серверної комунікації, схема взаємодії зі сторонніми програмними інтерфейсами тощо.

В третьому розділі ми приступимо до практичної частини. Ми скористаємось інформацією та прийнятими рішеннями в перших двох розділах і спираючись на них опишемо процес розробки в усіх деталях – від налаштування програмного середовища до запуску та тестування готового продукту

1 ФОРМУВАННЯ ЗАГАЛЬНИХ ПОТРЕБ ДО ІНТЕРАКТИВНОЇ КАРТИ

1.1 Практична цінність

Паводки в Івано-Франківській області – явище постійне. З року в рік мости, дороги та дамби зазнають пошкоджень та руйнуються внаслідок паводків. Кожен з таких об’єктів постійно моніторять: спеціалісти перевіряють та фіксують у звітах стан об’єктів, оцінюють, які ремонтні роботи мають бути проведені до наступного сезону. Паралельно з цим аналізується потенційна матеріальна та соціальна шкода у разі несправності або значного пошкодження об’єкта.

При такій оцінці об’єкта враховують численні параметри. Для прикладу (рис. 1.1.1):

Рисунок 1.1.1 — Приклад аварійного моста

Міст між селами Болохів та Войнилів на автомобільній дорозі С090601 був пошкоджений паводком у 2020 році. Опори моста потребують капітального ремонту, вартість якого 14 000 000 гривень. У разі руйнування моста припиниться автомобільне сполучення до 11 сіл, в тому числі шкільних автобусів та екстрених служб. Чисельність населення в даному регіоні – 12,8 тисяч.

Даний приклад сформований з декількох джерел – висновок спеціалістів про технічний стан моста, оцінка ремонтних робіт від підрядника, аналіз впливу на інфраструктуру та соціальні функції спеціальних органів влади. Пізніше ця інформація співставляється з потенційними зонами затоплення, наданими метео-службами.

Вся інформація розосереджена по великій документній базі міської ради, що дуже уповільнює процес її аналізу та актуалізації. Зокрема, стає проблематичним процес розподілу коштів на відновлення або побудову нових об’єктів адже треба пріоритизувати різні проекти по соціальному та інфраструктурному значенню.

Важливу роль в масиві зібраної інформації має точне географічне положення об’єкта. Воно враховується при оцінці ризиків руйнування чи пошкодження. Ймовірні зони затоплення визначають спеціалісти метео станцій, які на основі зібраної статистики за останні десятиріччя та поточних показників рівня опадів наносять на карту потенційні зони застою річкових вод, розмиву берегів, тощо. Тобто, важливо співставляти візуально карти, надані метеорологами та карти об’єктів.

Враховуючи вище вказані деталі та при комунікації з представниками Івано-Франківської міської ради було запропоновано проект по створенню інтерактивної карти з нанесеними на ній географічними точками об'єктів, з повною інформацію по кожному з них та можливістю змінювати налаштування візуалізації.

1.2 Вибір технологій

1.2.1 Огляд Google Earth

При попередній комунікації з представниками Івано-Франківської ОДА як приблизний приклад для майбутнього проекту був приведений сервіс Google Earth. Це багатофункціональний веб-сервіс від компанії Google, основою якого слугує точно відтворена в електронному вигляді віртуальна модель Землі, яка складається із безлічі орбітальних знімків поверхні планети. Це ідеальна платформа для представлення ландшафту, автомобільних доріг, будинків, туристичних об’єктів, тощо.

Крім того ця платформа дозволяє наносити додаткові об’єкти та контури(рис. 1.2.1.1) на поверхню з їх детальним описом, який відображається у вигляді віджета на екрані. Також у віджет можна додавати фото. Саме на базі цього сервісу створювались перші прототипи візуалізації об’єктів.

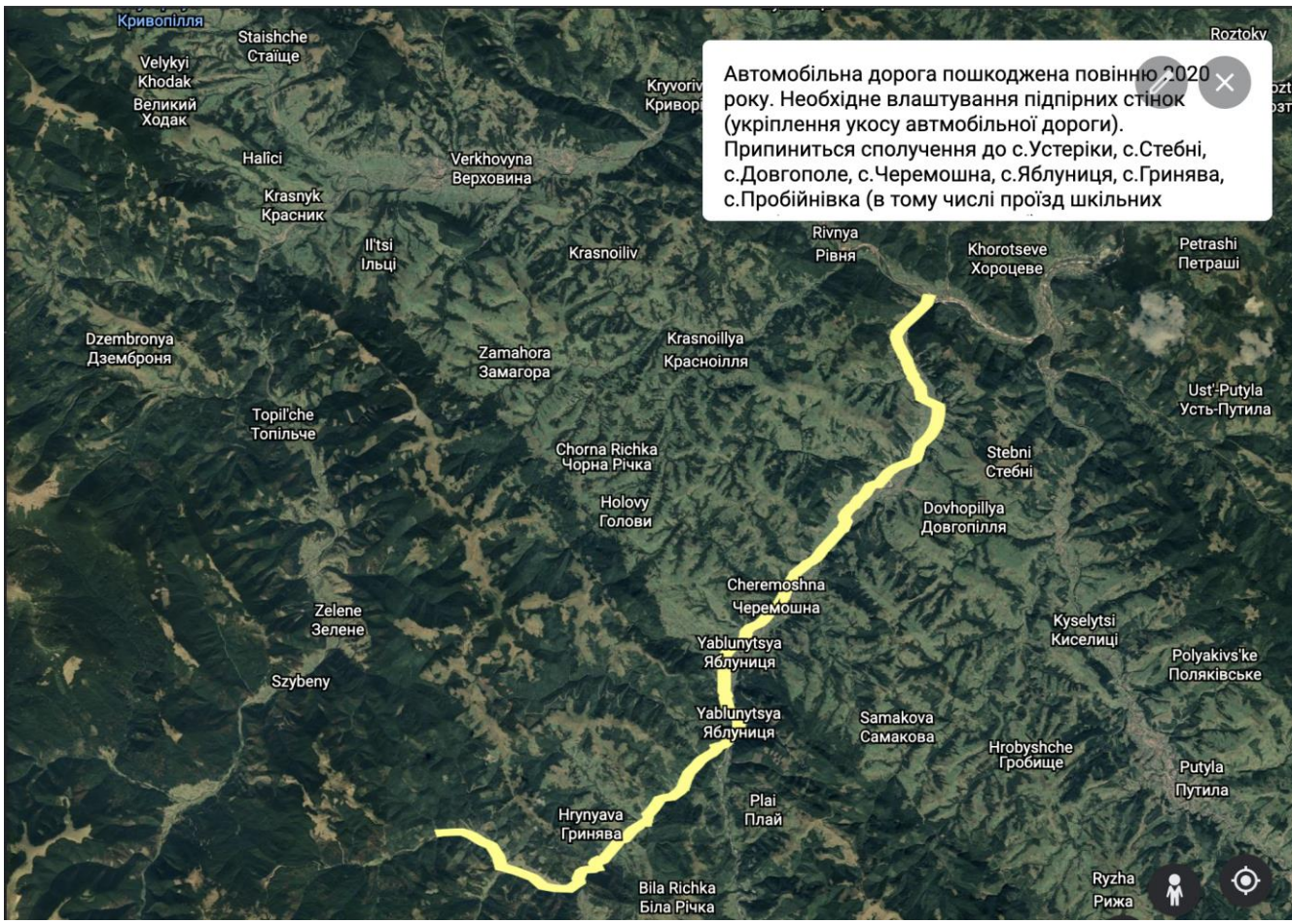


Рисунок 1.2.1.1 — прототип візуалізації

Проте, використання даного типу візуалізації показало невідповідності до базових вимог. Перш за все це зручність використання – проблемою стала нечитабельність великої кількості даних. Кожен окремий об’єкт це окремий пункт в боковому списку і щоб подивитись інформацію про нього потрібно було явно вибрати його серед усіх інших об’єктів. Як наслідок, став неможливим поширений сценарій використання серед спеціалістів – подивитись всі мости на певній ділянці русла річки та скласти їх список.

Наступною проблемою стало підтримання актуальності даних. Збереження всієї інформації в хмарному сховищі та її поширення для всіх споживачів з однієї точки не доступне в Google Earth. По суті вся інформація поширювалась у вигляді KML архіва.

KML-архів – це спеціальний тип файла для збереження географічних даних та зв’язаного з ним контенту, який використовується в Google Maps та Google Earth. Серед працівників поширювався актуальний файл, який треба було імпортувати в веб-додаток для його перегляду. Однак проблема була в тому що, що при найменших змінах доводиться створювати нову версію файла та знову поширювати її серед працівників. Імпорт оновленого файла створює ще одну карту з майже ідентичним набором об’єктів. В такому форматі роботи було легко заплутатись, яка версія актуальна. Це стало вагомим аргументом проти використання такого підходу.

Вирішальним недоліком стала неможливість створення фільтрів для об’єктів. Наприклад, при підготовці до розгляду фінансування таких об’єктів потрібно вибрати ті, вартість яких не перевищує 1 мільйона гривень. Дослідження функціоналу Google Earth не принесли інформації про те, як розрізняти об’єкти за різними параметрами.

Практична робота з такими візуалізаціями Google Earth довела те, що існуючий функціонал не задовольняє основні вимоги до інтерактивної карти. Наступним кроком стало дослідження інструментів для візуалізації карти з можливістю повного контролю над інтерактивними елементами.

1.2.2 Вибір клієнтської платформи

Клієнтська платформа це мобільний додаток, браузер або завантажена на комп'ютер програма. Вибір залежить від того як буде використовуватись додаток та впливає на підхід до розробки додатка та складність його реалізації.

Спільно з замовником був проведений аналіз того як пристроїв на яких має запускатись майбутній додаток. В результаті було визначено, що пріоритетною платформою є персональний комп'ютер. Адже такий узагальнений масив інформації не потрібен в польових умовах, тому і версію для мобільних телефонів не було передбачено в першій версії карти.

Другим фактором стала простота використання. Додаток має запускатись на непідготовлених попередньо комп'ютерах для презентацій на зустрічах та нарадах. Тому був обраний варіант веб-додатка(сайт). Для нього потрібен тільки будь який із сучасних браузерів – Chrome, Firefox, Edge, Internet Explorer.

Крім задоволення вищеописаних потреб розробка сайту займає набагато менше часу ніж програма, яка інсталюється на ПК. Час розробки теж є важливим фактором, адже робота проводилась за кілька місяців до початку сезону паводків.

Підсумовуючи вище написане, результатом роботи мав стати веб-сайт. Для запуску однією обов'язковою умовою є наявність браузера з активною підтримкою та оновленнями.

1.2.3 Вибір мови програмування та супутніх технологій.

Розробка веб-сайтів є досить стандартизованою галуззю. Базовим інструментом є мова розмітки веб сторінок HTML(рис. 1.2.3.1).

```
<!DOCTYPE html>
<html>

    <head>
        <title>My First Webpage</title>
    </head>

    <body>
        <h1>
            My First Webpage
        </h1>
        <p>This is a paragraph...</p>
    </body>

</html>
```

Рисунок 1.2.3.1 — приклад розмітки веб-сторінки

HTML – це загальноприйнятий стандарт для розмітки елементів на веб сторінці. Він підтримуються всіма актуальними браузерами та на даний момент є безальтернативним інструментом у своїй сфері.

З його допомогою ми створимо основу для проекту – опишемо елементи сторінки та додамо метадані для них, які в майбутньому знадобляться для зміни їх зовнішнього вигляду та маніпуляцій з ними.

Для правильного позиціонування об’єктів, задання їм розміру та кольору, ми будемо використовувати CSS (Cascade Style Sheet) – каскадні таблиці стилів. Це ще один безальтернативний інструментв сфері функцій. які він виконує. CSS щільно інтегрований з HTML та дозволяє налаштувати зовнішній вигляд сайту дуже

гнучками та компленксними методами. В ньому використовуються раніше додані в розмітку метадані(класи та ідентифікатори). Також CSS код може бути доданий напряму в HTML код, без підключення його як стороннього ресурсу.

Для реалізації інтерактивної складової в веб сторінках існує одна основна мова програмування – JavaScript. Винятком можна назвати таку технологію як Java Applets, де використовувалась мова Java, але вона давно втратила актуальність і ніколи не була універсальним інструментом.

Для JavaScript існують надбудови: TypeScript, CoffeScript та інші. Вони створені для більш зручного написання та тестування коду, хоча і потребують більше часу в процесі розробки. Після оцінки майбутнього функціоналу, було вирішено використовувати класичний JavaScript без надбудов. Це дозволить швидше розробити прототип та спростити процес передачі кодової бази розробникам, не знайомим з синтаксисом надбудов.

Найважливішою частиною майбутнього додатку є інструмент для візуалізації та кастомізації карти. Після попереднього аналізу доступних варіантів, до етапу фінального порівняння було обрано Google Maps та OpenStreetMaps.

Це два найпопулярніших провайдера детальних географічних карт для веб сайтів з широкими можливостями кастомізації.

Порівняння обох інструментів проводилось через призму потреб конкретного проекту. Перевагами OpenStreetMaps стали: можливість безкоштовного використання, відповідність картографічним стандартам в умовних позначеннях та побудові карти, постійна підтримка актуальності картографічних даних місцевими ентузіастами. Перевагою Google Maps є можливість переглядати фотографії ландшафту із супутника(давність варіюється від 1 року до 3), але при цьому сервіс є платним.

В результаті між двома варіантами був обраний OpenStreetMaps через значну перевагу у своїх якостях, корисних для створення інтерактивної карти протипаводкових заходів.

1.3 Формування вимог до проекту

Паралельно з вибором базових технологій для реалізацію проекту, проводилось обговорення всіх деталей функціоналу майбутнього додатку. В процесі розробки змінювались невеликі деталі, але 90% проекту було заплановано на стадії обговорення ще до початку написання коду.

Візуально фокус мав бути зосереджений на карті, тому вона має займати 70% екрану та розташована з лівого боку. Решта вільного простору була відведена під панель для настройки відображення об’єктів на карті(рис. 1.3.1).

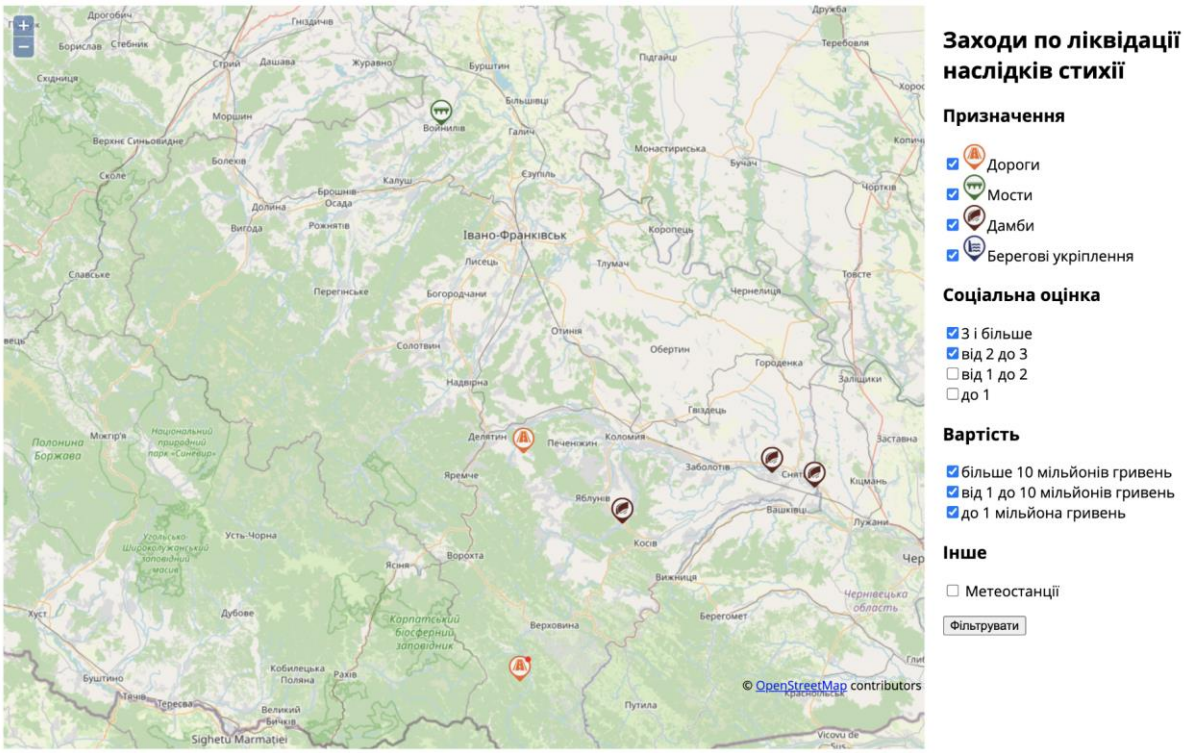


Рисунок 1.3.1 — загальний вигляд проекту

Такі пропорції були підібрані в ході експериментів то зворотнього зв’язку учасників процесу. Позиціонування всередині карти мало бути таким, щоб по замовчуванню відображалась Івано-Франківська область в максимально можливому розмірі по центру.

На карті мали відображатись об’єкти, кожен з яких належить до одного з нижче наведених типів:

- Дороги
- Мости
- Дамби
- Берегові укріплення
- Метеостанції

Кожен тип об’єкта має свою піктограму для відображення на карті(рис. 1.3.2).

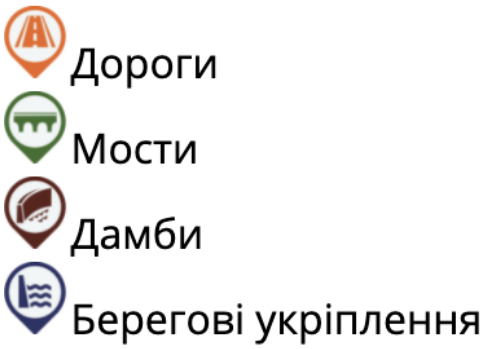


Рисунок 1.3.2 — типи об’єктів та їх піктограми

Позначки на карті мають бути одного розміру незалежно від того, наскільки приближена карта. Розмір позначок має бути достатнім для того. щоб ідентифікувати на ньому тип об’єкта, але не досить великим щоб значно закривати зміст карти.

При натиску лівою кнопкою миші на позначку об’єкта на карті має з’являтись інформаційний блок з необхідним набором даних про об’єкт(рис. 1.3.3). В ньому має міститись заголовок(назва об’єкта), вартість відновлювальних робіт, соціальна оцінка, визначена експертами, опис об’єкта та його фотографії(за наявності).

Рисунок 1.3.3 — приклад інформаційного блоку

Мініатюри фотографій в інформаційному блоці мають відкриватись у повному розмірі після натискання лівої кнопки миші на відповідну мініатюру. Винятком у форматі блоку з даними є метеостанції. Для них має відображатись історична статистика зафіксованих опадів у форматі таблиці.

Всі дані за домовленістю надані у форматі Excel таблиці. Вхідні дані обов’язково мають містити наступні поля:

- Назва об’єкту
- Географічні координати(довгота та широта в числовому форматі)
- Опис об’єкту
- Вартість відновлення
- Соціальна оцінка

Як виняток, для метеостанцій має бути сформована статистична таблиця яка і буде перенесена в інформаційний блок такого об’єкта. Фотографії були надані в

заархівованому вигляді, де вони були згруповані в директоріях, назва яких відповідає назві відповідного об’єкта.

Панель налаштування з правого боку від карти має містити фільтри для відображення об’єктів по значенню їх різних параметрів. Користувач має мати змогу вибрати список типів об’єктів для відображення, відфільтрувати їх по вартості та соціальній оцінці в заданих діапазонах. Фільтри мають застосовуватись при натисненні на кнопку підтвердження.

Сформована вище теоретична база знань про специфіку проекта, спектр технологій, які будуть задіяні в реалізації, та конкретний список вимог до результату формують завдання для даної наукової роботи:

- Дослідити процес створення веб-додатка на практиці
- Дослідити процес роботи географічними даними, картами та програмними аналогами
- Розробити прототип інтерактивної карти протипаводкових заходів

2 АРХІТЕКТУРА ПРОЕКТУ

2.1 Клієнт-серверна комунікація

Одним з основних та невід'ємних принципів розробки веб-проектів є клієнт-серверна комунікація. Її сенс полягає в тому, що в системі яка притримується такого принципу завжди є як мінімум два компонента - клієнт та сервер. Клієнт є ініціатором комунікації та відправляє запити на сервер. Сервер в свою чергу відправляє відповідь на запит клієнта. Зазвичай, багато клієнтів відправляють запити на один сервер(рис. 2.1.1).

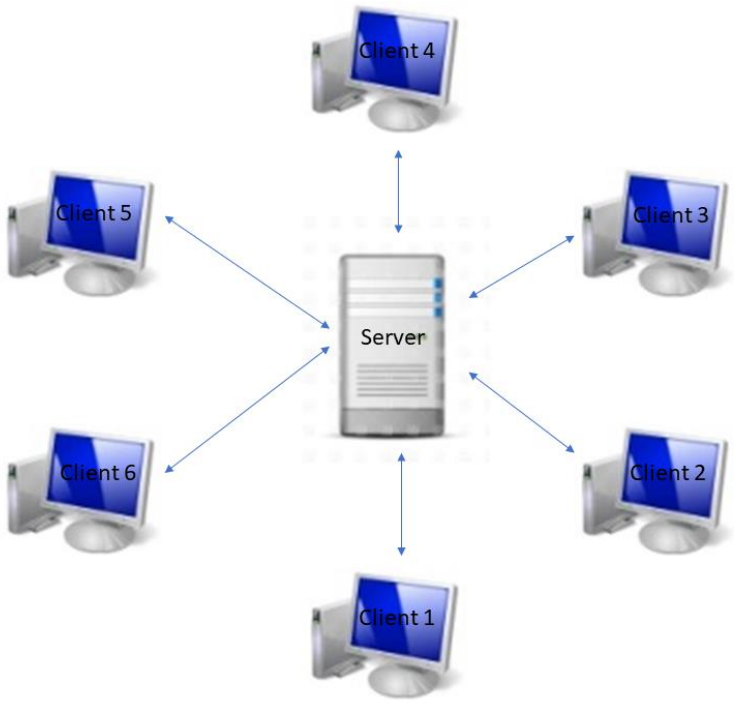


Рисунок 2.1.1 — схема клієнт-серверної комунікації

На практиці клієнтом в нашому випадку буде браузер на ПК, як ми встигли визначити в першому розділі. На етапі проектування нам треба визначити роль сервера в нашому проекті.

Реалізація серверної частини не входить в діапазон нашої роботи. Її створив Віталій Антонов та описав усі деталі в своїй науковій роботі. Однак, нам потрібно повністю описати, яка частина логіка буде реалізована в клієнтській частині, структуру ресурсів на сервері, так як робота з ними буде описана в клієнті.

Для початку нам треба проаналізувати, які ресурси потрібні для роботи веб-порталу. Серед них:

- Основа веб-проекту: HTML, CSS та JS файли. Це ресурси, які потрібні браузеру для запуску нашого веб-додатка

- Структуровані дані: назви об’єктів, вартість їх відновлення, соціальна оцінка тощо.

- Медіафайли: фотографії об’єктів для відображення в інформаційних віджетах.

Основні файли додатка завжди мають певну структуру, суттєві зміни в яку вносити не вийде. Тому деталям та тонкощам цієї частини роботи буде приділено у третьому розділі.

Набагато більше варіативності в пункті про структуровані дані. За найбільш поширеним підходом структуровані дані мають зберігатись в базі даних. Клієнтські додатки не можуть працювати напряму з базами даних. Тому схема ускладнюється ще одним компонентом - серверним додатком. Серверний додаток обробляє запити від клієнта і в свою чергу відправляє запити базі даних. Потім обробляє відповідь бази даних і результат відправляє назад клієнту.

Мета бази даних в такій узагальненій реалізації - швидко та ефективно фільтрувати структуровані дані серед великого їх масиву та видавати тільки потрібні дані.

У випадку з використанням бази даних потрібно планувати час на створення серверного додатка, дослідження комунікації з базою даних. Також це ускладнить деплоймент готового продукту, адже з’явиться два нових компонента на стороні сервера. Додатково це набагато збільшить затримку до візуалізації потрібних об’єктів.

Враховуючи потенційні витрати часу на таку систему, ми провели аналіз її доцільності та альтернативних варіантів.
Для початку ми отримали приблизну кількість об’єктів для відображення на карті - ~200 штук. По приблизних прогнозах об’єм цієї інформації - 250 - 300 KB

(кілобайт). При сучасних можливостях інтернет з’єднання та потужності пристроїв таку кількість інформації можна вважати мізерною. При цьому основним плюсом баз даних є економія часу на виборі тільки тієї інформації, яка потрібна для конкретної операції. Зазвичай розмір баз даних варіюється від сотень мегабайт до десяток гігабайт.

Маючи вхідні дані, можна визначити альтернативний підхід і він полягає у тому щоб завантажувати повний набір структурованих даних на старті веб-додатка із сервера, та оперувати ним з допомогою клієнтського коду.

Мала кількість даних не вплине на час старту веб-додатка, але при цьому ми уникнемо великої частини роботи по створенню серверного додатка та конфігурації бази даних.

Для медіафайлів було вирішено зробити спеціальну структуру файлового дерева для швидкого доступу до фотографій конкретного об’єкту. В спільній директорії фотографій мають бути субдиректорії з назвою унікального ідентифікатора об’єкта в якій і будуть розташовані фотографії(рис. 2.1.2).

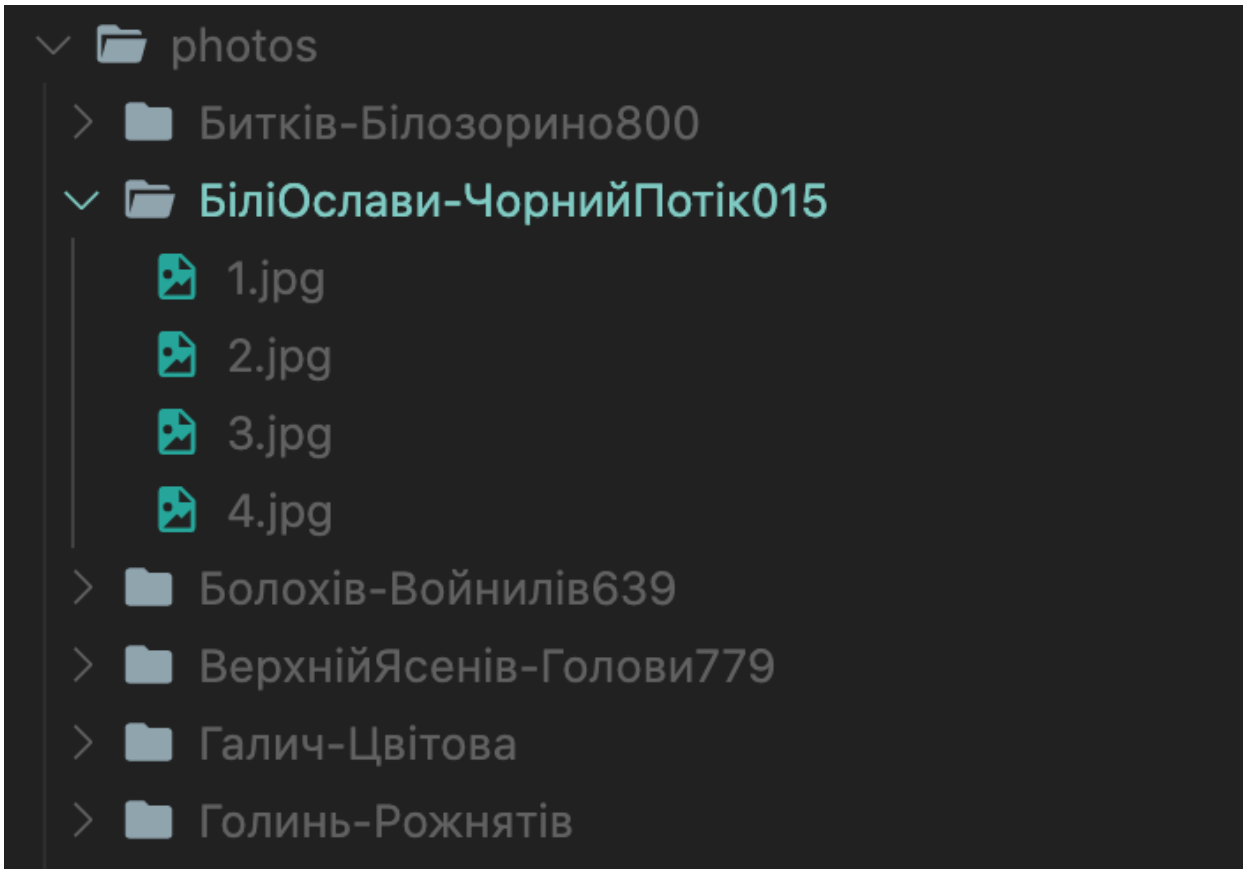


Рисунок 2.1.2 — приклад файлової структури медіафайлів

Унікальний ідентифікатор буде вказаний для кожного об’єкта в структурованих даних. Також в доповнення для роботи з фотографіями буде потрібна карта структури, яка буде описана в 3 розділі.

Таким чином ми визначили загальну схему того, які ресурси будуть завантажуватись клієнтською частиною із сервера, знайшли оптимальний спосіб завантаження структурованих даних та описали структуру медіафайлів.

2.2 Формат даних

Вхідні дані, які надійшли з ОДА, були у форматі XLSX(рис. 2.2.1). Це формат файлів який використовується в програмі Microsoft Excel, Google Sheets та OpenOffice. По суті своїй це структурована таблиця з можливістю використовувати формули. В наданих екземплярах формули не використовувались, що значно спрощувало задачу по зміні формату файлів.

10		2021 рік НОВЕ						
11	1	Нове будівництво водозахисної дамби на р.Прут в с.Сопів Печенізжинської селищної ради Івано-Франківської області	48.5278519	25.0049648	35000	відсутня	Внаслідок проходження паводків минулих років та стихій, що сталася на території Івано-Франківської області 12-24 червня 2020 року відбувалась руйнація ділянки корінного берега довжиною700 м.пог. на правому березі Прут в селі Сопів Коломийського району..В результаті інтенсивного руйнування корінного берега виникла загроза підтоплення території площею 176 га, об’єктів соціальної сфери та інфраструктури, газопроводу середнього тиску, трансформаторної підстанції та 210 приватних житлових будинків населеного пункту, в якому проживає 2552 жителів. У зв’язку з вищенаведеним існує необхідність будівництва водозахисної дамби в с.Сопів Коломийського району. Також існує загроза підтоплення та затоплення частини території с. Нижній Вербіж , розташованому нижче за течією.	1,2	0
12	2	Нове будівництво правобережної водозахисної дамби на р.Бистриця Солотвинська в с.Вовчинець Івано-Франківської міської ради (період проведення робіт 2021-2022 роки)	48.9489559	24.7311967	140000	в наявності, відсутис ОВД	Будівництво не розпочато, відбувається розмив правого берега р.Бистриці Солотвинської (2,6 км) та лівого корінного берега р.Бистриці Надвірнянської (1,0 км) нижче ж/д моста на загальній довжині 3,6 км. Існує загроза підтоплення 750 домоволодінь, 190 га с/г угідь та об’єктів соціальної сфери с.Вовчинець	0,7	0
13	3	Нове будівництво водозахисної дамби на р.Бистриця Солотвинська в с.Клузів Угринівської сільської ради Івано-Франківської області (період проведення робіт 2021-2022 роки)	48.9478074	24.7244161	150000	виготовляється ПКД	Будівництво не розпочато, відбувається розмив лівого корінного берега нижче ж/д моста в напрямку садового товариства на довжині 1,4 км. Існує загроза підтоплення 350 домоволодінь, 294 га с/г угідь та об’єктів соціальної сфери с.Клузів, с.Ямниця	0,7	0
14	1	Нове будівництво водозахисної дамби на р.Прут між селами Семаківці та Борщів Івано-Франківської області	48.4761761	25.2127591	31500	відсутня	Внаслідок проходження стихій, що сталася на території Івано-Франківської області 12-24 червня 2020 року відбувся перелив паводкових вод р. Прут через корінний берег на задамбовий простір, внаслідок чого було підтоплено частину території села Борщів та зруйновано ділянку водозахисної дамби в с. Борщів. Виникла загроза підтоплення території 740 га, об’єктів інфраструктури та 207 приватних житлових будинків населеного пункту, в яких проживає 570 жителів. У зв’язку з вищенаведеним існує необхідність будівництва водозахисної дамби між селами Семаківці Коломийського району та Борщів. Також існує загроза підтоплення та затоплення частини території смт. Заболотів, розташованому нижче за течією.	0,8	0

Рисунок 2.2.1 — приклад вхідних даних

Початковий формат даних не підходив до використання без підготовки - XSLX формат досить складний і візуалізувати його будь де крім створених для цього програм не доцільне завдання.

На цьому етапі стоїть завдання вибрати формат даних, який буде зручно обробляти програмним і ручним шляхом. Так як інформація має час від змінюватись та доповнюватись, треба передбачити зручний спосіб для цього.

Першом аналіз пройшов формат даних JSON.

JSON - формат даних що базується на тексті та основою якого є принцип “ключ-значення”(рис. 2.2.2). Підтримує основні типи даних: текст, числа, логічні значення(true, false), об’єкти та масиви.

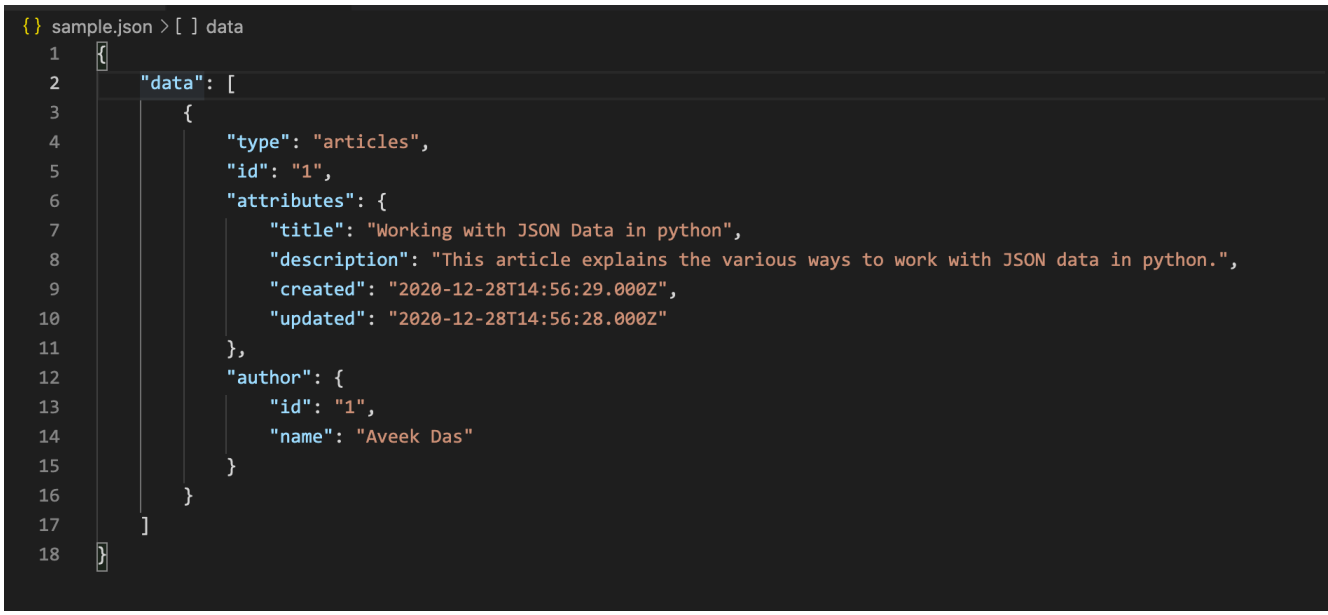


Рисунок 2.2.2 — приклад JSON формату

Однозначний плюс JSON - це стандарт клієнт-серверної комунікації. Його зручно обробляти програмним шляхом, його легко може прочитати людина. Однак доповнювати та змінювати його не дуже зручно для людей, які не знайомі з особливостями синтаксису. Існують ресурси, які спрощують роботу з таким форматом даних, але ми шукали більш звичний для кінцевих користувачів варіант.

Наступним був проаналізований формат CSV. Це тестовий формат табличних даних, який в незмінному вигляді не дуже зручний для прочитання людиною, але при цьому без проблем може бути оброблений програмним шляхом(рис. 2.2.3).

```
Family Name,Given Name,VIAF ID
Ackersdijck,Willem Cornelis,17959345
Adelung,Friedrich von,22963658
Afzelius,Arvid August,49972119
Amerling,Karel,13331054
Anton,Karl Gottlob von,183632821
Arwidsson,Adolf Ivar,8184878
Asbjørnsen,Peter Christen,116587918
Attems,Heinrich,37665468
Atterbom,Per Daniel Amadeus,46819248
Balabin,Viktor Petrovich,44473845
Banks,Joseph,46830189
Beck,Friedrich,44338671
Becker,Reinhold von,42101066
Bernhart,Johann Baptist,69674335
Bertram,Johann,32890043
Bilderdijk,Willem,14882166
Boisserée,Sulpiz,7483155
Bopp,Franz,61614118
Borovský,Karel Havlíček,100277614
Bosković,Jovan,161354270
Buslaev,Fyodor,10074560
Cenowa,Florian Stanislaw,44466031
```

Рисунок 2.2.3 — приклад CSV формату

Безумовним плюсом такого формату стало те, що його підтримують вище описані програми Excel, Google Sheet і т.д. Незважаючи на те, що в сирому вигляді файл був не зручним для сприйняття, кінцеві користувачі могли використовувати вже звичні програми для перегляду, зміни та доповнення файлів. При цьому ніяк не страждала логіка додатка.

Отже в результаті аналізу ми обрали формат даних, що задовольняє два основні критерія: зручність підтримки актуальності інформації та його обробки програмним шляхом.

2.3 Структура даних

В попередніх параграфах ми визначились з форматом даних - CSV файл. Після обговорень було вирішено розділити весь масив даних на 5 частин/файлів - в кожному з них будуть об’єкти одного типу. В результаті отримаємо п’ять файлів:

- bridges.csv
- dam.csv
- roads.csv
- shoreProtection.csv
- stations.csv

Таке розділення упростить задачу доповнення та актуалізації даних в майбутньому, адже однотипні об’єкти в одному файлі набагато легше розглядати ніж один великий масив даних.

Ще одна причина такого рішення полягає в тому що інформація про метеостанції структурно відрізняється від всіх інших. Тому її задля однорідності даних в одному джерелі треба виділити в окремий файл.

Наступний крок - визначити назви полів в таблиці. Важливим в цьому етапі є те, щоб привести всі типи об’єктів до однієї структури(за винятком метеостанцій).

Основною характеристикою об’єкта буде поле ‘Назва’. Це має бути унікальний ідентифікатор об’єкта, в рамках проекту в нього немає бути дублікатів. Воно має бути осмисленим для зручності використання в подальшому. Функціонально це поле потрібно для зв’язки з медіа файлами, адже їх набір характеризується якраз цим ідентифікатором. Наприклад:

Для об’єкта значення поля ‘Назва’ якого - ‘Ільці-Беркут’, медіафайли будуть знаходитись в під директорії ‘Ільці-Беркут’.

Наступні два поля - Longitude, Latitude. Широта та довгота - два параметра. що описують географічне положення і потрібні для створення точки на карті. По своїх

суті значення цих полів це невід’ємні десяткові числа з точність до 6 знаків після коми.

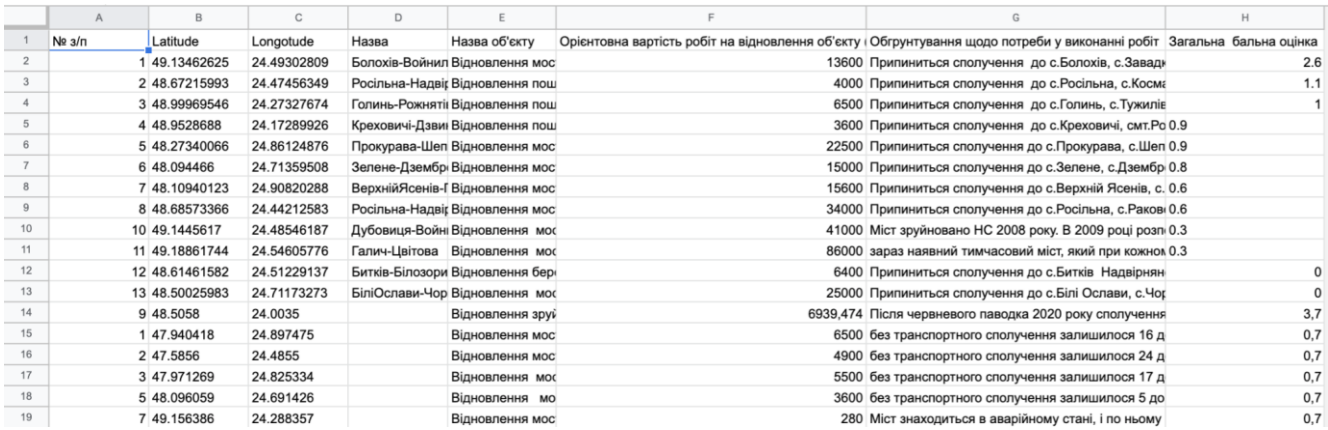
Наступне поле - ‘Назва об’єкта’. Воно несе декоративну функцію та буде відображатись як заголовок в інформаційному блоці.

‘Обгрунтування щодо потреби у виконанні робіт’ - більш точна назва для опису. Назва поля могла бути лаконічнішою, але в початкових даних потрібна повна ясність того які дані очікуються в полі.

Останніми будуть потрібні поля для фільтрів: ‘Загальна бальна оцінка’ і ‘Орієнтовна вартість робіт’. Вони означають соціальну оцінку від експертів та вартість побудови/відновлення об’єкта відповідно. Вони будуть використовуватись для фільтрації та відображення в інформаційних блоках.

В результаті отримуємо набір обов’язкових полів для кожного об’єкта(рис. 2.3.1):

- Назва об’єкта
- Longitude
- Latitude
- Обгрунтування щодо потреби у виконанні робіт
- Загальна бальна оцінка
- Орієнтовна вартість робіт



№ з/п	Latitude	Longotude	Назва	Назва об'єкту	Орієнтовна вартість робіт на відновлення об'єкту	Обгрунтування щодо потреби у виконанні робіт	Загальна бальна оцінка
1	49.13462625	24.49302809	Болохів-Войнил	Відновлення мос	13600	Припиниться сполучення до с.Болохів, с.Завад	2.6
2	48.67215993	24.47456349	Росільна-Надвід	Відновлення пош	4000	Припиниться сполучення до с.Росільна, с.Косм	1.1
3	48.99969546	24.27327674	Голинь-Рожняті	Відновлення пош	6500	Припиниться сполучення до с.Голинь, с.Тужилів	1
4	48.9528688	24.17289926	Крехович-Дзви	Відновлення пош	3600	Припиниться сполучення до с.Крехович, смт.Ро	0.9
5	48.27340066	24.86124876	Прокурава-Шеп	Відновлення мос	22500	Припиниться сполучення до с.Прокурава, с.Шеп	0.9
6	48.094466	24.71359508	Зелене-Дзембр	Відновлення мос	15000	Припиниться сполучення до с.Зелене, с.Дзембр	0.8
7	48.10940123	24.90820288	ВерхнійЯсенів-Г	Відновлення мос	15600	Припиниться сполучення до с.Верхній Ясенів, с.	0.6
8	48.68573366	24.44212583	Росільна-Надвід	Відновлення мос	34000	Припиниться сполучення до с.Росільна, с.Раков	0.6
9	49.1445617	24.48546187	Дубовиця-Войні	Відновлення мос	41000	Міст зруйновано НС 2008 року. В 2009 році розп	0.3
10	49.18861744	24.54605776	Галич-Цвітова	Відновлення мос	86000	зараз наявний тимчасовий міст, який при кожно	0.3
11	48.61461582	24.51229137	Битків-Білозори	Відновлення бер	6400	Припиниться сполучення до с.Битків Надвірнян	0
12	48.50025983	24.71173273	БіліОслави-Чор	Відновлення мос	25000	Припиниться сполучення до с.Білі Ослави, с.Чор	0
13	48.5058	24.0035		Відновлення зруй	6939,474	Після червневого паводка 2020 року сполучення	3,7
14	47.940418	24.897475		Відновлення мос	6500	без транспортного сполучення залишилося 16 д	0,7
15	47.5856	24.4855		Відновлення мос	4900	без транспортного сполучення залишилося 24 д	0,7
16	47.971269	24.825334		Відновлення мос	5500	без транспортного сполучення залишилося 17 д	0,7
17	48.096059	24.691426		Відновлення мо	3600	без транспортного сполучення залишилося 5 до	0,7
18	49.156386	24.288357		Відновлення мос	280	Міст знаходиться в аварійному стані, і по ньому	0,7

Рисунок 2.3.1 — кінцева структура

Тепер потрібно визначити структуру для статистичних даних з метеостанцій. Структура буде комплексна та складна - в одному файлі треба комбінувати багаторічні спостереження на багатьох метеостанціях.

Для кожної конкретної метеостанції має бути сформована таблиця і ось її поля(рис. 2.3.2):

- Рік
- Дата (дата спостереження)
- сума опадів (в міліметрах)
- витрата (м.куб/с)
- рівень (см)

Рік	дата	сума опадів мм	витрата м.куб/с	рівень см
2008	25.07	82.3	273	420
2009	12.04	28.2	83.2	260
2010	08.07	79.2	257	410
2011	28-29.06	49.6	64.2	236
2012	06.04	2.5	30.2	186
2013	09.03	7.2	48.6	222
2014	15.05	73.0	97.0	280
2015	18.04	27.6	57.0	230
2016	16.02	7.6	36.5	200
2017	16.12	31.2	59.9	234
2018	30.06	63.0	88.2	270
2019	30.06	63.0	88.2	270
2020	23.06	119.4	257	510

Рисунок 2.3.2 — приклад статистики метеостанції

Так як метеостанція не одна, треба придумати спосіб помістити декілька таблиць. Для цього може слугувати роздільний рядок з базовою інформацією(рис. 2.3.3) про метеостанцією. В нього ми помістимо координати та назву станції

12	2017	16.12	18.8	12.4	241	
13	2018	30.06	30.1	16.0	250	
14	2019	22.06	34.5	10.8	236	
15	2020	24.06	20.4	33.4	285	
16	2	р.Прут-с.Татарів; приведення - 90 см	Довгота - 48.366	Широта - 24.550000		
17	Рік	дата	сума опадів мм	витрата м.куб/с	рівень см	
18	2008	25.07	82.3	273	420	
19	2009	12.04	28.2	83.2	260	
20	2010	08.07	79.2	257	410	
21	2011	28-29.06	49.6	64.2	236	
22	2012	06.04	2.5	30.2	186	

Рисунок 2.3.3 — приклад розділення метеостанції

В цьому параграфі ми повністю описали структури даних в яких врахували усі обов’язкові елементи, уніфікували інформацію більшості типів об’єктів. В такому форматі файли можна з мінімальними витратами часу доповнювати та змінювати.

2.4 Принципи розробки фронтенду

Після проектування основновних принципів комунікації та дизайну моделі даних, можна сформувати основні принципи, яких ми будемо притримуватись при розробці клієнтської частини.

Основним принципом хорошого тону в розробці ПЗ це використання системи контролю версій. В нашому випадку це буде Git - найпоширеніша та найзручніша у використанні. Ми будемо використовувати її по найпростішому сценарію: після реалізації логічно закінченої частини роботи ми будемо формувати зміни в один коміт та завантажувати його в віддалений репозиторій на сервісі Github.

При первинному обговоренні, було вирішено що дизайн інтерфейсу не в пріоритеті. Саме тому ми будемо використовувати стандартні шрифти, для

виділення важливих елементів - жирний та збільшений шрифт замість загальноприйнятого підходу з виділенням кольором, фокусом і т.д. Стилі кнопок будуть за замовчуванням.

Враховуючи мінімалістичний підхід до дизайну, CSS код немає сенсу виносити в окремий файл - нечисленні стильові правила ми будемо прописувати в атрибут style відповідних блоків HTML(рис. 2.4.1).

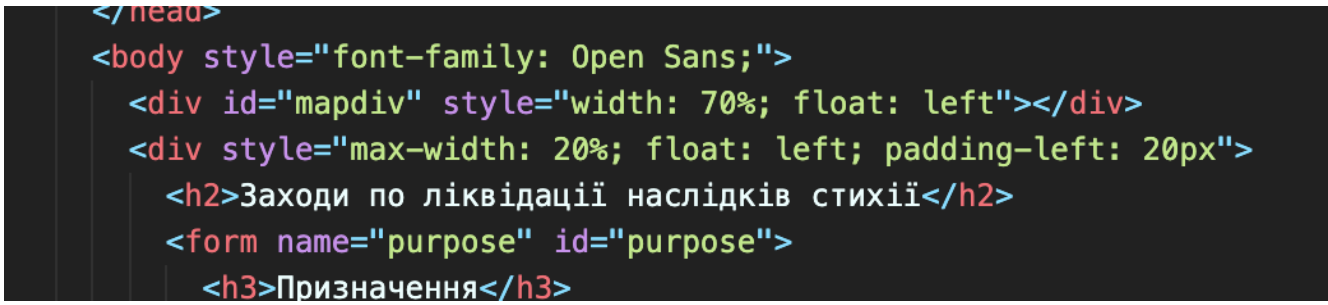


Рисунок 2.4.1 — приклад стилів в HTML коді

Також треба визначитись бібліотеками версій та стандартів. При написанні JavaScript коду ми будемо притримуватись стандарту EcmaScript 2015. Це дозволить розширити спектр браузерів, які будуть підтримувати наш веб-додаток.

Додати віджет карти OpenStreetMaps нам допоможе бібліотека OpenLayers. Це проект з відкритим кодом, що допомагає інтегрувати та кастомізувати географічні карти. Функціонал цієї бібліотеки допоможе нам додати функціональні інформаційні блоки, яких немає в стандартному інтерфейсі OpenStreetMaps.

Для роботи з даними ми будемо використовувати csvtojson. Це бібліотека, яка допоможе розпарсити файли для роботи з даними, які в них зберігаються.

2.5 Висновки до розділу

В цьому розділі ми провели попереднє планування реалізації проекту. Визначили ключові принципи щоб на їх основі створити продукт за мінімальний час з перспективою на доопрацювання в майбутньому. Також на стадії проектування ми врахували всі сценарії підтримки продукту - зміни та доповнення даних.

3 ОПИС РЕАЛІЗАЦІЇ ПРОЕКТУ

3.1 Підготовка програмного середовища

На етапі підготовки нам потрібно буде зробити дві речі: налаштувати кодовий редактор та встановити необхідне програмне забезпечення.

Серед кодових редакторів є багато альтернатив - ціла екосистема IntelliJ від JetBrains де кожен інструмент розрахований на конкретну мову програмування та має надширокий функціонал автодоповнення, аналізу коду та налаштувань запуску. Всі продукти від JetBrains доступні безкоштовно за студентською ліцензією.

Є цілий набір більш простих інструментів: SublimeText, Brackets, Atom та Visual Studio Code. На останній варіант і упав наш вибір - він розповсюджується на безкоштовній основі, його функціонал можна доповнити безліччю плагінів, а інтерфейс простий і зрозумілий(рис. 3.1.1) на відміну від комплексних інструментів від IntelliJ.

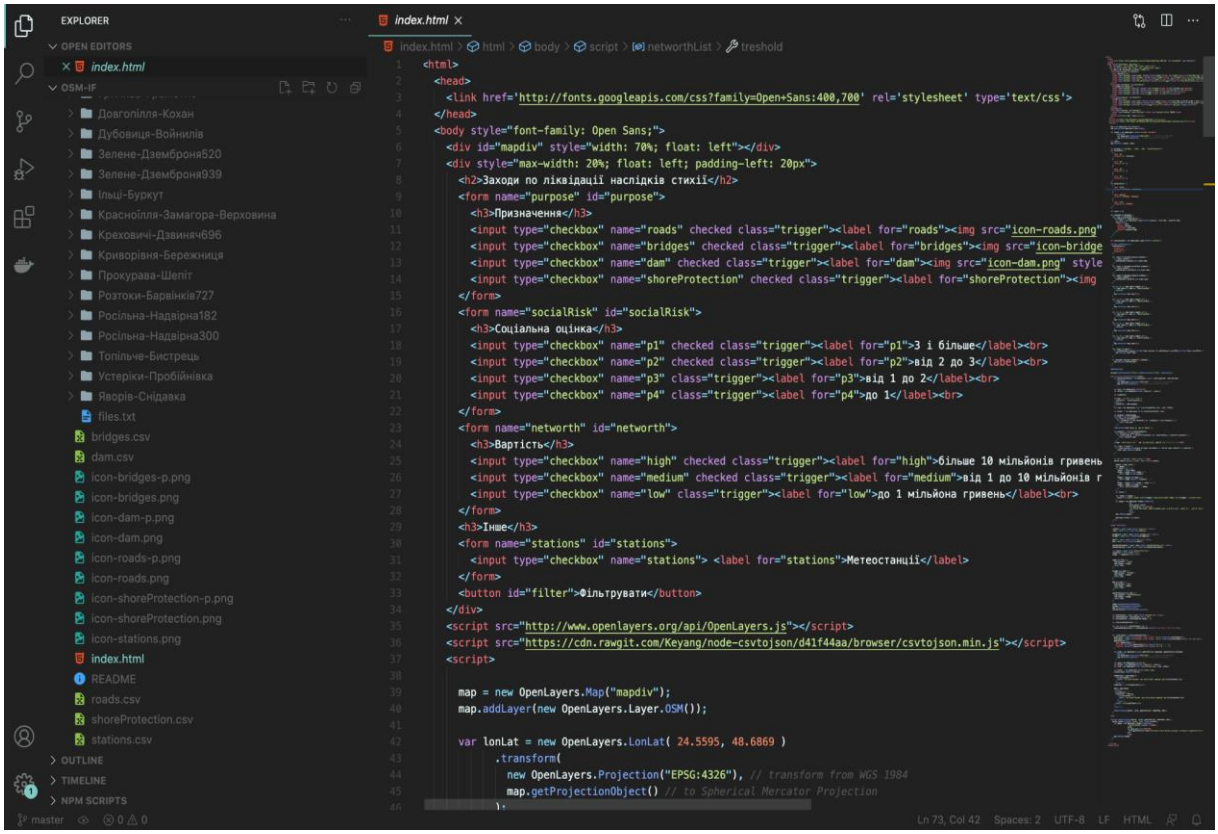


Рисунок 3.1.1 — інтерфейс Visual Studio Code

При дефолтних налаштуваннях цього редактора після установки, він вже підходить для розробки запланованого проекту. Однак для зручності ми інсталюємо плагін для зручного переглядання CSV файлів прямо з кодового редактора.

Для цього в лівому боковому меню потрібно знайти пункт `Extensions` та в полі пошуку що з’явилось ввести фразу `CSV to table` та встановити знайдений плагін.

Це допоможе нам не витрачати час на відкриття файлів в Excel при розробці та тестуванні додатка.

Після того як конфігурація кодового редактора готова, можна приступити до установки усіх необхідних програм. Для самого процесу розробки нам потрібен NodeJS. З його допомогою ми створимо емулятор сервера з функцією автоматичного оновлення при змінах в кодовій базі.

Пакет для інсталювання NodeJS можна завантажити на його офіційному сайті від будь яку платформу. Далі, слідуючи інструкціям в інсталяторі, установити його.

Перевірити правильність установки слід з допомогою наступної команди у в будованому в операційну систему терміналі (в нашому випадку Terminal на MacOS):

```
➜ osm-if git:(master) node -v
v10.15.3
```

В результаті запуску команди node -v ми отримали встановлену версію програми. Це свідчить про правильну установку і ми можемо продовжувати далі налаштовувати середовище.

Наступний крок полягає в установці емулятора сервера з функцією автооновлення. Для цього ми використаємо пакетний менеджер NPM - він уже установлений разом із NodeJS. Для установки треба запустити наступну команду:

```
➜ osm-if git:(master) npm install -g live-server
```

Після запуску команди має початися процес завантаження та установки пакета live-server. Атрибут -g в команді вказує на глобальну установку пакету а не контекстну для одного конкретного проекту.

Для тестування проекту нам потрібно встановити браузер Google Chrome - в нього найширший функціонал по відладці веб-додатків. що допоможе у процесі розробки.

Завантажити його інсталятор можна на офіційному сайті Google Chrome.

Одним з найважливіших пунктів є установка системи контролю версій git(рис. 3.1.2). В MacOS та дистрибутивах Linux вона встановлена за замовчуванням, а в Windows її треба завантажити окремо.

```
→  osm-if git:(master) git add .
→  osm-if git:(master) ✗ git commit -m "change data in csv file"
[master 2749641] change data in csv file
 1 file changed, 4 insertions(+)
→  osm-if git:(master) git push
```

Рисунок 3.1.2 — приклад використання Git

Також для того щоб зберігати віддалена та поширювати код нам потрібно буде створити аккаунт та приватний репозиторій в сервісі Github(рис. 3.1.3).

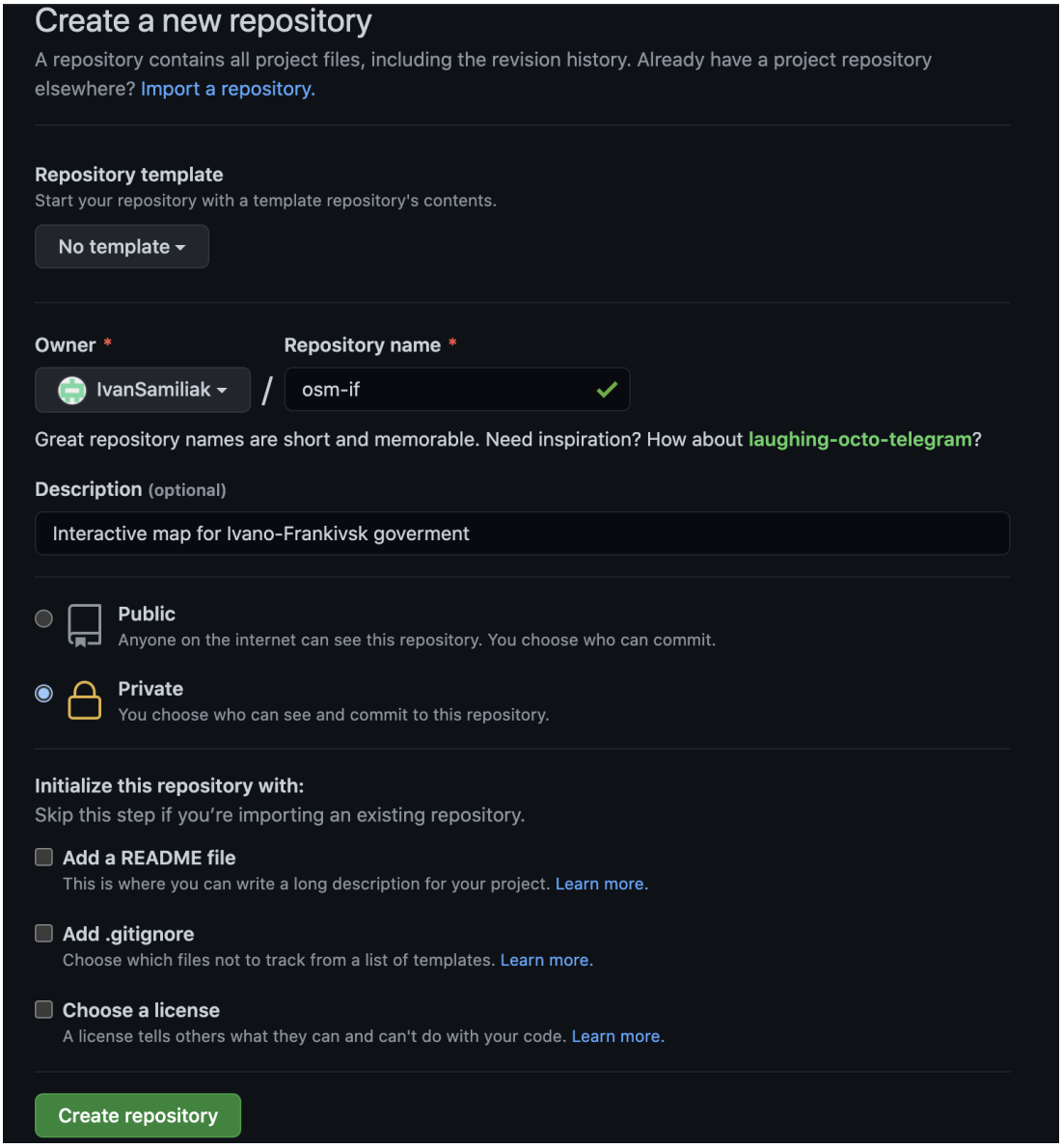


Рисунок 3.1.3 — процес створення репозиторія в Github

В цьому параграфі ми описали підготовку середовищі розробки та принципи використання системи контролю версій. Далі нам потрібно створити базовій веб додаток, додати карту на нього і створити елементи фільтрації для подальшої реалізації їх функціоналу.

3.2 Створення базового інтерфейсу

Для початку роботи потрібно створити окрему директорію(папку) на ПК де буде зберігатися весь код. В нашому випадку директорія була названа **osm-if**, однойменну з назвою репозиторія.

Основним файлом в нас буде **index.html**. В ньому буде описана розмітка веб-сторінки, там же будуть описані каскадні таблиці стилів та логіка клієнтського коду на JavaScript.

Основою для нас будуть кілька обов'язкових елементів для HTML файла(рис. 3.2.1). Серед низ заголовок який описує тип файла, елемент html в якому ми будемо описувати весь контент і вкладені в нього елементи body, head.

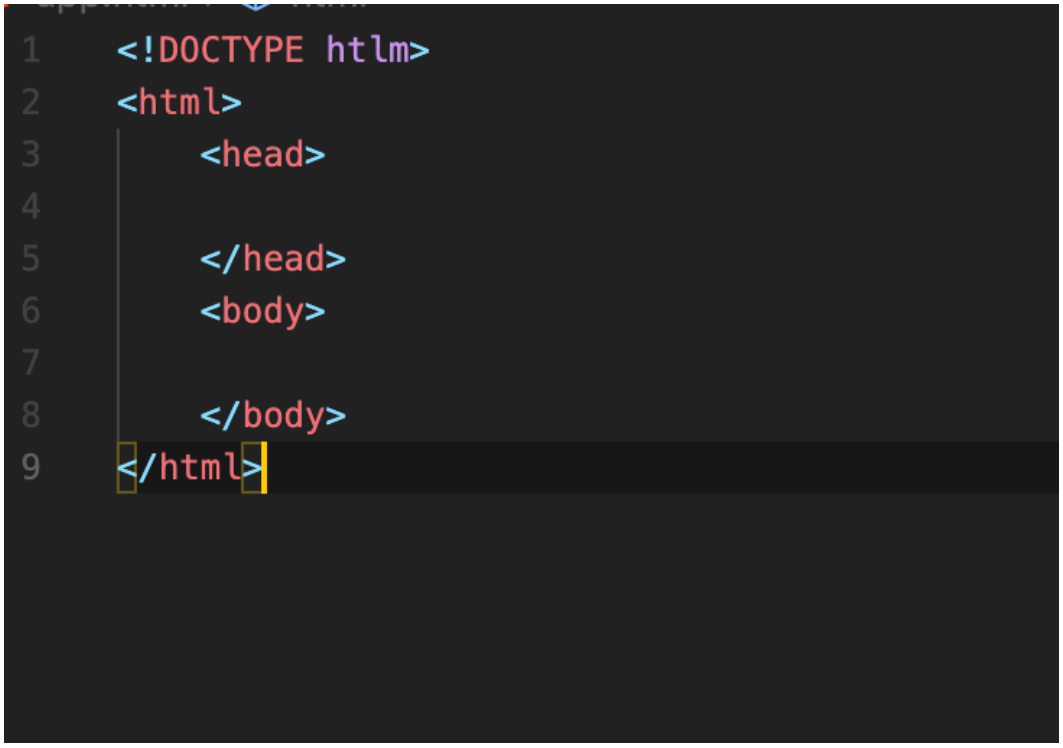


Рисунок 3.2.1 — код базової веб-сторінки

Наступний елемент - карта. Для початку нам треба просто її відобразити на сайті та правильно позиціонувати з фокусом на Івано-Франківську область. Бібліотека підключається до веб-сторінки елементом **script** в якому атрибут *href* буде містити URL-посилання на код бібліотеки:

<script src="http://www.openlayers.org/api/OpenLayers.js"></script>

Цей елемент треба розмістити в кінці елемента **body**, а вже над ним розміщувати решту контенту. Далі нам потрібно додати примітивний блочний елемент на сторінку всередині якого буде відображатись карта. Для цього підійде досить універсальний **div** в який треба додати ідентифікатор(id). Він нам знадобиться в подальшому для зв’язки цього елемента з кодом бібліотеки. В цьому ж елементі треба прописати його стилі. По технічному завданню, карта має займати 70% екранного простору. Тому додаємо поле *width* із значенням 70%. Для того щоб карта знаходилась завжди з лівого боку та не переміщувалася залежно від розміру екрана, додамо атрибут *float: left.*

Тепер коли місце для карти визначене та підключена бібліотека, нам потрібно написати кілька рядків коду щоб відобразити карту(рис. 3.2.2).

```
<body>
    <div id="mapdiv" style="width: 70%; float: left"></div>
    <script src="http://www.openlayers.org/api/OpenLayers.js"></script>
    <script>
        map = new OpenLayers.Map("mapdiv");
        map.addLayer(new OpenLayers.Layer.OSM());
    </script>
</body>
```

Рисунок 3.2.2 — підключення бібліотека OpenLayers

В інтеграційному коді ми використовуємо ідентифікатор раніше створеного створеного елемента щоб вказати в якому конкретно блоці треба відображати карту.

Також там ми додаємо базовий шар карти - в нашому випадку це OSM(Open Street Maps).

Далі треба правильно відпозиціонувати карту. Для цього нам потрібні координати центра відображуваної області та кратність зуму. Всі значення були підібрані емпіричним шляхом на офіційній веб-версії сайту OpenStreetMaps для однорідності результатів на тестах та використанні в коді. В ході підбору оптимальних значень ми отримали: координати - 24.5595, 48.6869 та кратність зуму - 9.

Для позиціонування треба створити об’єкт OpenLayers.LonLat в картографічній проекції EPSG:4326. Цей об’єкт є програмною абстракцією над набором координат що складається з довготи і широти (24.5595, 48.6869 відповідно в нашому випадку). Далі з допомогою метода **setCenter** в об’єкті карти вказуємо новостворений об’єкт як перший аргумент функції. Другим аргументом має бути кратність зуму, який для нас складає 9(рис. 3.2.3).

```
<!DOCTYPE htlm>
<html>
    <head>

    </head>
    <body>
        <div id="mapdiv" style="width: 70%; float: left"></div>
        <script src="http://www.openlayers.org/api/OpenLayers.js"></script>
        <script>
            map = new OpenLayers.Map("mapdiv");
            map.addLayer(new OpenLayers.Layer.OSM());

            var lonLat = new OpenLayers.LonLat( 24.5595, 48.6869 )
            .transform(
              new OpenLayers.Projection("EPSG:4326"), // transform from WGS 1984
              map.getProjectionObject() // to Spherical Mercator Projection
            );
            var zoom=9;
            map.setCenter (lonLat, zoom);
        </script>
    </body>
</html>
```

Рисунок 3.2.3 — код відображення карти

В результаті в 22 рядка коду мають відобразити карту Івано-Франківської області на 70 відсотків екрану. Для тестування треба запустити раніше встановлену програму live-server з допомогою вбудованого в редактор терміналу.

```
➜ osm-if git:(master) live-server

Serving "/Users/samiliak/code/osm-if" at http://127.0.0.1:8080
Ready for changes
```

В результаті ми отримуємо процес запущений в окремому терміналі, який емулює сервер та миттєво відображає зміни в коді в запущеному в браузері додатку.

Паралельно з тим як ми запустили програму у вас відкривається вікно браузера з запущеним сайтом(рис. 3.2.4).

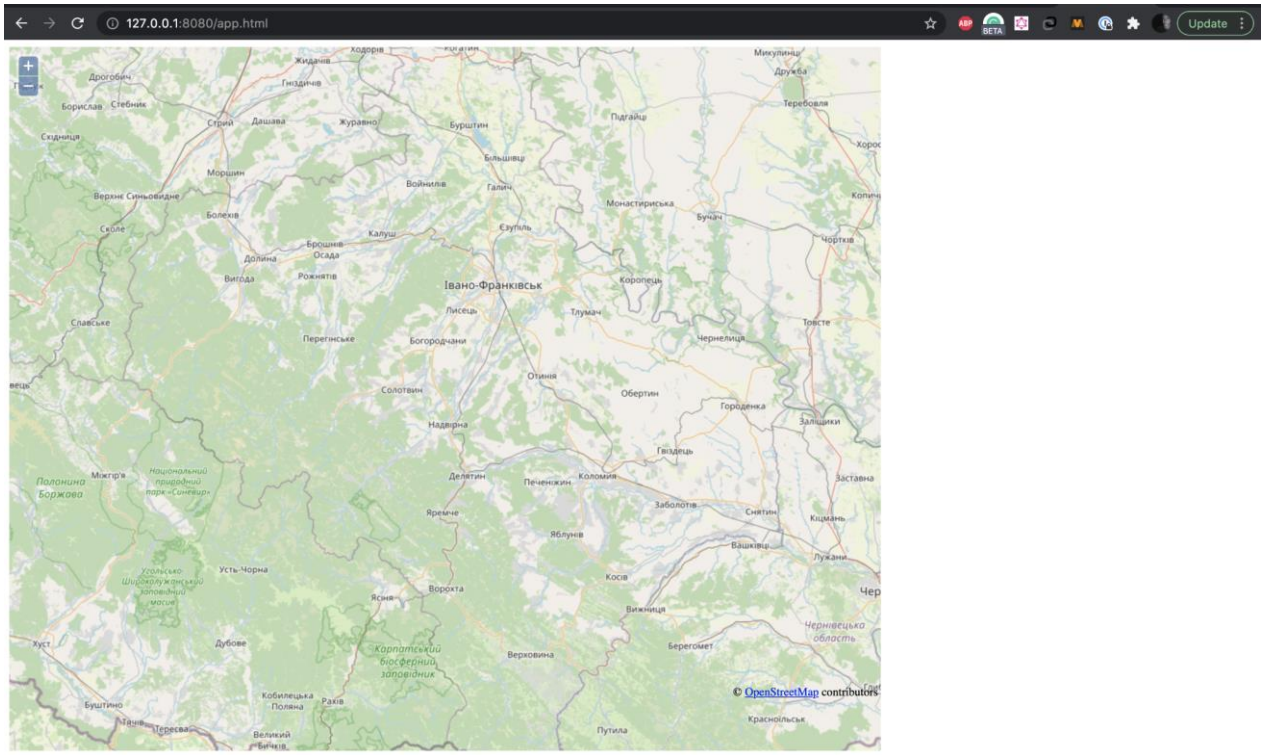


Рисунок 3.2.4 — проміжний результат

Так як проміжну мету з картою ми виконали, можна приступати до інтерфейсу фільтрів. В їх реалізації ми будемо використовувати елементи **form, input, label.**

Form - елемент, створений для групування інтерактивних елементів. Інтерактивні елементи в різних формах розглядаються окремо. В нашому випадку ми будемо їх використовувати для незалежного комбінування фільтрів по різних параметрах.

Input - інтерактивний елемент, який створює інтерфейси різних типів для взаємодії користувача із додатком. Цей елемент в основному характеризується своїм атрибутом *type*. Його допустимі значення це *text, number, checkbox* і т.д.(рис. 3.2.5)

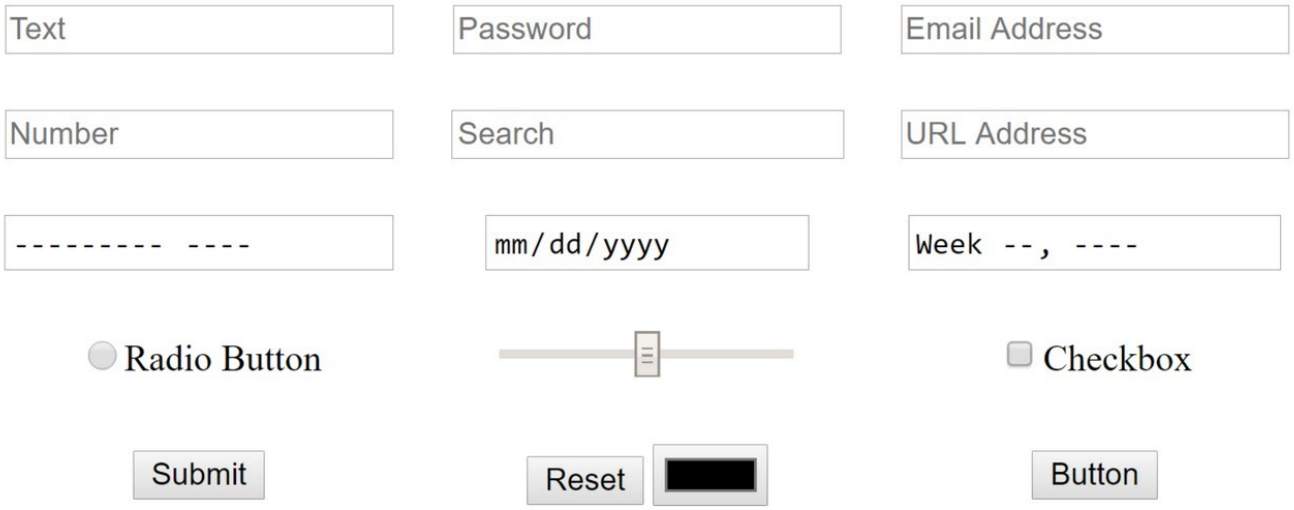


Рисунок 3.2.5 — типи елемента **input**

В нашому випадку ми будемо використовувати тільки тип checkbox, так як всі наші фільтри виражені у комбінації різних діапазонів масиву даних.

Label - елемент, що описує input. Label можна прив'язати до одного конкретного input і він буде слугувати описом інтерактивного елемента. Ми будемо його використовувати для вказання діапазонів, доступних для вибору.

Враховуючи специфіку вищеописаних елементів, ми створимо наступну структуру: для кожного фільтра (типу об’єкта, соціальної оцінки та вартості) буде створена індивідуальна форма. В кожну форму будуть додані діапазони фільтрації по конкретному параметру.

Окремим блоком буде доданий чекбокс для відображення метеостанцій, що було уточненням після першої ітерації розробки.

```
<form name="socialRisk" id="socialRisk">
  <h3>Соціальна оцінка</h3>
  <input type="checkbox" name="p1" checked class="trigger">
  <label for="p1">3 і більше</label><br>
  <input type="checkbox" name="p2" checked class="trigger">
  <label for="p2">від 2 до 3</label><br>
  <input type="checkbox" name="p3" class="trigger">
  <label for="p3">від 1 до 2</label><br>
  <input type="checkbox" name="p4" class="trigger">
  <label for="p4">до 1</label><br>
</form>
```

Рисунок 3.2.6 — код форми фільтра

Основну структуру форми було доповнено додатковими деталями(рис. 3.2.6). Елемент **form** має атрибути *name* та *id*, які слугують для подальшої ідентифікації та збору результатів заповненої форми.

Елемент **h3** - заголовок третього рівня. який описує назву фільтра.

Атрибути елемента **input**:

- *type=”checkbox”* - вказує на тип інтерфейса, в цьому випадку форма має два стани: активна і не активна. Активна форма відобразить відповідний діапазон об’єктів на карті

- *name=”<name>”* - атрибут що присвоює ім’я конкретному інтерфейсу для подальшої перевірки його стану.

- *checked* - якщо такий атрибут присутній то чекбокс буде активний за замовчуванням. В іншому випадку він буде неактивний. В вище напеденому прикладі при завантаженні сторінки будуть активні діапазони “3 і більше” і “від 2 до 3”, а решта - ні.

В елемента **label** є тільки атрибут *for*, який з’єдний лейбл з інпутом по імені цього інпута. Вміст елемента **label** - опис діапазона.

Елемент **br** слугує як знак переносу стрічки і в нашому випадку позиціонує інпути.

Фінальний штрих в інтерфейсі фільтрів - кнопка, яка застосовує налаштування. В ній є тільки атрибут *id*, який потрібен для зчитування подій кліків на цю кнопку. Код кнопки:

```
<button id="filter">Фільтрувати</button>
```

Всі елементи **form** були поміщені в елемент **div** для їх правильного розміщення на екрані. Код верхньорівневого блоку:

```
<div style="max-width: 20%; float: left; padding-left: 20px">
…
</div>
```

В стилях налаштована ширина та відступи від карти. Результат на рисунку 3.2.7.

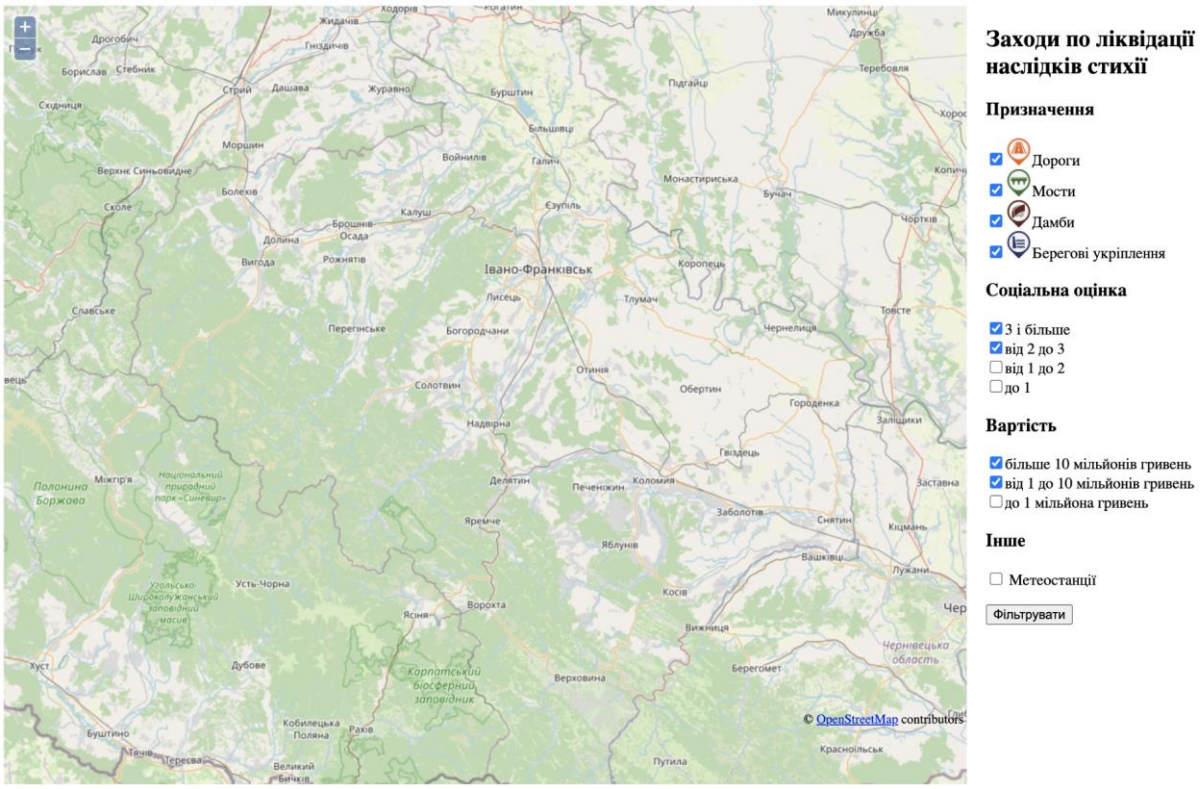


Рисунок 3.2.7 — інтерфейс фільтрів

3.3 Реалізація функцій фільтрації

Один з принципів бібліотеки OpenLayers полягає в трьохрівневій системі об’єктів: карта, шар та об’єкти. Об’єкти можуть бути додані тільки на шар, а вже шар додається на карту, тим самим відображаючи всі вкладені в нього об’єкти на карті.

Видаляти об’єкти з шару досить проблематично, тому нам потрібно оперувати цілими шарами при зміні кількості відображених об’єктів. Для цього ми створимо шари для всіх можливих комбінацій діапазонів(рис. 3.3.1).

```
for (purpose of purposes) {
  for (risk of socialRisk) {
    for(networth of networthList) {
      var layer = new OpenLayers.Layer.Markers(purpose + risk.name + networth.name);
      layers.push({
        layer : layer,
        purpose: purpose,
        socialRisk: risk.name,
        networth: networth.name
      });
    }
  }
}
```

Рисунок 3.3.1 — створення шарів для фільтрації

Шари будуть заповнені об’єктами, що задовольняють характеристики шару по описаним діапазонам. Наприклад, об’єкт - міст з вартістю відновлення 500 000 тисяч гривень та соціальною оцінкою 2,5 буде додано в шар з комбінацією діапазонів “Мости”, “від 2 до 3”, “до 1 мільйона гривень”.

Процес заповнення шарів на основі CSV файлів буде описано в наступному параграфі.

Маючи структуровану систему шарів для відображення, нам треба зчитати вибрані діапазони з інтерфейсу та відобразити тільки ті шари, які задовольняють всі умови, вибрані користувачем. Для цієї операції ми створимо функцію updateLayers(рис. 3.3.2).

```
enabledLayers={
  purpose: [],
  socialRisk: [],
  networth: []
};

for (input of document.purpose.elements) {
  if (input.checked) {
    enabledLayers.purpose.push(input.name)
  }
}
for (input of document.socialRisk.elements) {
  if (input.checked) {
    enabledLayers.socialRisk.push(input.name)
  }
}
for (input of document.networth.elements) {
  if (input.checked) {
    enabledLayers.networth.push(input.name)
  }
}
```

Рисунок 3.3.2 — зчитування інформації

Використовуючи раніше додані в інпути та форми ідентифікатори, ми вибираємо активовані чекбокси та визначаємо допустимі діапазони для кожного фільтра.

Тепер нам треба співставити дану інформацію із списком шарів з об’єктами - нам треба відфільтрувати тільки ті шари, які задовільняють вибрані діапазони(рис. 3.3.3). Відфільтрований список використаємо щоб додати всі потрібні шари на карту.

```
  for (layer of layers) {
    if (enabledLayers.purpose.includes(layer.purpose)
        && enabledLayers.socialRisk.includes(layer.socialRisk)
        && enabledLayers.networth.includes(layer.networth)) {
      map.addLayer(layer.layer)
    }
  }

  if (document.stations.elements[0].checked) {
    map.addLayer(stationsLayer);
  }
}
```

Рисунок 3.3.3 — фільтрація шарів

Крім трьох основних фільтрів, також була реалізована фільтрація метеостанцій.

В цьому параграфі ми описали механізм фільтрації об’єктів на карті. Розробили систему шарів для операцій з фільтрами. Описали логіку зчитування користувацького вводу та фільтрації шарів по цих даних.

3.4 Зчитування даних з сервера

Як вже було описано в розділі про архітектуру проекту, всі структуровані дані зберігаються в 5 CSV файлах. На стадії розробки файли доступні на локальній машині з допомогою інструмента live-server. На реальному сервері принцип доступу до файлів не буде відрізнятись.

Для локальної машини файли доступні по URL http://localhost/roads.csv(рис. 3.4.1) (для доріг).

```
roadsCsv = await (await fetch('roads.csv')).text();
roads = await csv().fromString(roadsCsv);

bridgesCsv = await (await fetch('bridges.csv')).text();
bridges = await csv().fromString(bridgesCsv);

damCsv = await (await fetch('dam.csv')).text();
dam = await csv().fromString(damCsv);

shoreProtectionCsv = await (await fetch('shoreProtection.csv')).text();
shoreProtection = await csv().fromString(shoreProtectionCsv);
```

Рисунок 3.4.1 — завантаження файлів

Для завантаження файлів використовуються асинхронні функції, порядок виконання яких контролює оператор await. Крім того асинхронними операціями був парсинг CSV файлів. Ця операція не входить до вбудованих функцій JavaScript тому ми заплановано використовуємо бібліотеку csvtojson.

Після парсингу ми отримуємо масив об’єктів і тепер його треба розсортувати по відповідних шарах для подальної фільтрації, яка була описана в попередньому параграфі(рис. 3.4.2).

```
var markerCoordinates = new OpenLayers.LonLat( item.Longotude, item.Latitude )
  .transform(
    new OpenLayers.Projection("EPSG:4326"), // transform from WGS 1984
    map.getProjectionObject() // to Spherical Mercator Projection
  );

var size = new OpenLayers.Size(28,28);
var offset = new OpenLayers.Pixel(-(size.w/2), -size.h);

var iconSuffix;

if(item['Пріоритет'] === 'true'){
  iconSuffix = `${item.purpose}-p`;
} else {
  iconSuffix = item.purpose;
}
var icon = new OpenLayers.Icon(`icon-${iconSuffix}.png`, size, offset);

var marker  = new OpenLayers.Marker(markerCoordinates, icon);
```

Рисунок 3.4.2 — створення маркера

Для кожного об’єкта потрібно створити маркер. Для цього ми дістали із даних об’єкта довготу та широту. Також тут ми визначаємо розміри маркера, які по технічному завданню мають бути статичні.

Крім того по типу об’єкта ми вибираємо файл маркера - зображення позначки на карті яке залежить від типу об’єкта.

Наступним кроком ми співставляємо значення ключових параметрів з діапазонами фільтрів та привязуємо маркер до відповідного фільтраційного шару(рис. 3.4.3).

```
var purpose = item.purpose;
var risk = function(riskValue){
  for (risk of socialRisk) {
    if (riskValue >= risk.treshold[0] &&  riskValue <= risk.treshold[1]) {
      return risk.name
    }
  }
}(parseFloat(item['Загальна  бальна оцінка']));

var networth = function(networthValue){
  for (networth of networthList) {
    if (networthValue >= networth.treshold[0] &&  networthValue <= networth.treshold[1]) {
      return networth.name
    }
  }
}(item['Орієнтовна вартість робіт на відновлення об’єкту (тис']['грн'][')'] * 1000);

for (layer of layers) {
  if (layer.purpose === purpose && layer.socialRisk === risk && layer.networth === networth) {
    layer.layer.addMarker(marker)
  }
}
```

Рисунок 3.4.3 — прив’язка маркера до шару

Таким чином ми наповнили нашу карту маркерами та інтегрували їх з системою фільтрації. Тепер на карті з’являється тільки ті об’єкти, які відповідають усім відзначеним діапазонам ключових значень.

3.5 Відображення інформаційних блоків

Для відображення інформаційних блоків потрібно додати обробник подій на маркер. При кліку на маркер система має дістати з оперативної пам’яті контекст маркера та сформувати інформаційний блок із назви, опису об’єкта, його параметрів та посилань на фотографії(рис. 3.5.1).

```
marker.events.register('click', this, function(event){

  object = item['Назва'];
  var images = []
  if (object !== '') {
    images  = item.images
    images = images.filter(image => {
      return image.includes(object)
    })
    images = images.map(image => {
      return image.replace(`./${object}`, '')
    })
    images = images.filter(image => image !== "")
    images = images.map(image => {
      return `photos/${object}` + image;
    })
  }
  var block='';

  for (image of images) {
    block+=`<a target="_blank" href="${image}"><img style="width: 400px" src="${image}" /></a><br/><br/>`
  }
  var popup = new OpenLayers.Popup.FramedCloud(
                  id,
                  event.object.lonlat,
                  new OpenLayers.Size(300,300),
                  `<h3 style='max-width: 300px'>${item["Назва об'єкту"]}</h3>
                  <p>Вартість робіт по відновленню:
                    ${(item['Орієнтовна вартість робіт на відновлення об’єкту (тис']['грн']['')'] * 1000).toLocaleString()}
                     грн. </p> <p> Соціальна оцінка: ${item['Загальна  бальна оцінка']} </p>
                     <p style='max-width: 300px'>${item["Обгрунтування щодо потреби у виконанні робіт"]}</p>${block}`,
                  null,
                  true);
  map.addPopup(popup);
```

Рисунок 3.5.1 — створення інформаційного блоку

Фотографії були отримані завдяки раніше описаній структурі файлів. Так з допомогою одного ідентифікатора можна було отримати всі фотографії об’єкта незалежно від їх кількості.

В інформаційний блок також додана функція збільшення фотографій при кліку на них, адже мініатюри не у всіх випадках достатньо.

ВИСНОВКИ

В даній науковій роботі був описаний повний цикл розробки продукту: від узгодження деталей майбутнього продукту до опису тонкощів взаємодії з сторонніми програмними інтерфейсами. Проведені численні дослідження, результати яких напряму вплинули на проект.

Зокрема, проаналізована практична цінність проекту - дослідження регіональних особливостей, специфіки роботи з даними та проблематики їх консолідації дозволило автоматизувати рутинну роботу та пришвидшити узагальнення та пошук даних.

Був проведений аналіз доступних технологій, на основі якого був зроблений вибір, який поєднує простоту та швидкість реалізації і підтримки.

Також були сформовані конкретні функціональні вимоги до проекту спільно із замовниками прототипу, що дозволило зекономити час на його реалізації та подальших змінах.

Створена архітектура проекту, яка дала розуміння багатьох технічних аспектів ще до написання коду, дозволила спланувати роботу і підготувати базу для подальших удосконалень.

Детально описаний процес реалізації - з конкретними етапами процесу, проміжним тестуванням та з урахуванням усіх загальноприйнятих практик.

На даний момент прототип використовують за прямим призначенням в Івано-Франківській ОДА, що підкреслює його актуальність та цінність.

Таким чином ми виконали задачі, які ми ставили перед собою в початковій частині наукової роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Проект на Github - https://github.com/IvanSamiliak/osm-if
2. Демонстраційний прототип - http://osm-if.herokuapp.com
3. Документація OpenLayers - https://openlayers.org/workshop/en/basics/
4. Вікі OpenStreetMaps - https://wiki.openstreetmap.org/wiki/Main_Page
5. W3Schools - https://www.w3schools.com/js/default.asp
6. Документація Google Earth - https://earth.google.com/studio/docs/
7. Опис клієнт-серверної архітектури від TutorialsPoint - https://www.tutorialspoint.com/operating-systems-client-server-communication

ДОДАТОК А

Програмний код

```
<html>
 <head>
  <link href='http://fonts.googleapis.com/css?family=Open+Sans:400,700' rel='stylesheet' type='text/css'>
 </head>
 <body style="font-family: Open Sans;">
  <div id="mapdiv" style="width: 70%; float: left"></div>
  <div style="max-width: 20%; float: left; padding-left: 20px">
   <h2>Заходи по ліквідації наслідків стихії</h2>
   <form name="purpose" id="purpose">
    <h3>Призначення</h3>
    <input type="checkbox" name="roads" checked class="trigger"><label for="roads"><img src="icon-roads.png" style="width: 28px">Дороги</label><br>
    <input type="checkbox" name="bridges" checked class="trigger"><label for="bridges"><img src="icon-bridges.png" style="width: 28px">Мости</label><br>
    <input type="checkbox" name="dam" checked class="trigger"><label for="dam"><img src="icon-dam.png" style="width: 28px">Дамби</label><br>
    <input type="checkbox" name="shoreProtection" checked class="trigger"><label for="shoreProtection"><img src="icon-shoreProtection.png" style="width: 28px">Берегові укріплення</label><br>
   </form>
   <form name="socialRisk" id="socialRisk">
    <h3>Соціальна оцінка</h3>
    <input type="checkbox" name="p1" checked class="trigger"><label for="p1">3 і більше</label><br>
    <input type="checkbox" name="p2" checked class="trigger"><label for="p2">від 2 до 3</label><br>
```

```
            <input type="checkbox" name="p3" class="trigger"><label for="p3">від 1 до 2</label><br>
            <input type="checkbox" name="p4" class="trigger"><label for="p4">до 1</label><br>
          </form>
          <form name="networth" id="networth">
            <h3>Вартість</h3>
            <input type="checkbox" name="high" checked class="trigger"><label for="high">більше 10 мільйонів гривень</label><br>
            <input type="checkbox" name="medium" checked class="trigger"><label for="medium">від 1 до 10 мільйонів гривень</label><br>
            <input type="checkbox" name="low" class="trigger"><label for="low">до 1 мільйона гривень</label><br>
          </form>
          <h3>Інше</h3>
          <form name="stations" id="stations">
            <input type="checkbox" name="stations"> <label for="stations">Метеостанції</label>
          </form>
          <button id="filter">Фільтрувати</button>
        </div>
        <script src="http://www.openlayers.org/api/OpenLayers.js"></script>
        <script src="https://cdn.rawgit.com/Keyang/node-csvtojson/d41f44aa/browser/csvtojson.min.js"></script>
        <script>

          map = new OpenLayers.Map("mapdiv");
          map.addLayer(new OpenLayers.Layer.OSM());
           var lonLat = new OpenLayers.LonLat( 24.5595, 48.6869 )
                .transform(
```

```
        new OpenLayers.Projection("EPSG:4326"), // transform from WGS 1984
        map.getProjectionObject() // to Spherical Mercator Projection
      );
var zoom=9;
map.setCenter (lonLat, zoom);


var purposes = ['bridges', 'roads', 'dam', 'shoreProtection']
var socialRisk = [
  {
    name: "p1",
    treshold: [3, +Infinity]
  },
  {
    name: "p2",
    treshold: [2, 3]
  },
  {
    name: "p3",
    treshold: [1, 2]
  },
  {
    name: "p4",
    treshold: [0, 1]
  }
];
var networthList = [
  {
    name: "high",
    treshold: [10000000, +Infinity]
```

```
  },
  {
    name: "medium",
    treshold: [1000000, 10000000]
  },
  {
    name: "low",
    treshold: [0, 1000000]
  },
];

var layers = [];

for (purpose of purposes) {
  for (risk of socialRisk) {
    for(networth of networthList) {
      var layer = new OpenLayers.Layer.Markers(purpose + risk.name +
networth.name);
      layers.push({
        layer : layer,
        purpose: purpose,
        socialRisk: risk.name,
        networth: networth.name
      });
    }
  }
}

var stationsLayer = new OpenLayers.Layer.Markers('stations');
```

```
function updateLayers() {
  enabledLayers={
    purpose: [],
    socialRisk: [],
    networth: []
  };

  for (input of document.purpose.elements) {
    if (input.checked) {
      enabledLayers.purpose.push(input.name)
    }
  }
  for (input of document.socialRisk.elements) {
    if (input.checked) {
      enabledLayers.socialRisk.push(input.name)
    }
  }
  for (input of document.networth.elements) {
    if (input.checked) {
      enabledLayers.networth.push(input.name)
    }
  }

  for (i = 0; i < map.layers.length; i++) {
    if (map.layers[i].name === "OpenStreetMap") {
      continue;
    }
    map.removeLayer(map.layers[i]);
  }
```

```
for (i = 0; i < map.layers.length; i++) {
  if (map.layers[i].name === "OpenStreetMap") {
    continue;
  }
  map.removeLayer(map.layers[i]);
}

for (i = 0; i < map.layers.length; i++) {
  if (map.layers[i].name === "OpenStreetMap") {
    continue;
  }
  map.removeLayer(map.layers[i]);
}
for (i = 0; i < map.layers.length; i++) {
  if (map.layers[i].name === "OpenStreetMap") {
    continue;
  }
  map.removeLayer(map.layers[i]);
}

for (i = 0; i < map.layers.length; i++) {
  if (map.layers[i].name === "OpenStreetMap") {
    continue;
  }
  map.removeLayer(map.layers[i]);
}

for (layer of layers) {
```

```
            if (enabledLayers.purpose.includes(layer.purpose) &&
enabledLayers.socialRisk.includes(layer.socialRisk) &&
enabledLayers.networth.includes(layer.networth))  {
              map.addLayer(layer.layer)
             }
           }

           if (document.stations.elements[0].checked) {
            map.addLayer(stationsLayer);
           }
          }

          updateLayers();

          document.getElementById('filter').addEventListener('click', updateLayers);

          async function markerProcessions(item){
            var markerCoordinates = new OpenLayers.LonLat( item.Longotude,
item.Latitude )
              .transform(
               new OpenLayers.Projection("EPSG:4326"), // transform from WGS 1984
               map.getProjectionObject() // to Spherical Mercator Projection
              );

            var size = new OpenLayers.Size(28,28);
            var offset = new OpenLayers.Pixel(-(size.w/2), -size.h);

            var iconSuffix;

            if(item['Пріоритет'] === 'true'){
```

```
            iconSuffix = `${item.purpose}-p`;
          } else {
            iconSuffix = item.purpose;
          }
          var icon = new OpenLayers.Icon(`icon-${iconSuffix}.png`, size, offset);

          var marker  = new OpenLayers.Marker(markerCoordinates, icon);

          var purpose = item.purpose;
          var risk = function(riskValue){
            for (risk of socialRisk) {
              if (riskValue >= risk.treshold[0] &&  riskValue <= risk.treshold[1]) {
                return risk.name
              }
            }
          }(parseFloat(item['Загальна  бальна оцінка']));

          var networth = function(networthValue){
            for (networth of networthList) {
              if (networthValue >= networth.treshold[0] &&  networthValue <=
networth.treshold[1]) {
                return networth.name
              }
            }
          }(item['Орієнтовна вартість робіт на відновлення об’єкту (тис']['грн'][')']
* 1000);

          for (layer of layers) {
            if (layer.purpose === purpose && layer.socialRisk === risk &&
layer.networth === networth) {
```

```
      layer.layer.addMarker(marker)
    }
  }


  var id  = "chicken" + Math.random() * 10000
  marker.events.register('click', this, function(event){

    object = item['Назва'];
    var images = []
    if (object !== '') {
      images  = item.images
      images = images.filter(image => {
        return image.includes(object)
      })
      images = images.map(image => {
        return image.replace(`./${object}`, '')
      })
      images = images.filter(image => image !== "")
      images = images.map(image => {
        return `photos/${object}` + image;
      })
    }
    var block='';

    for (image of images) {
      block+=`<a target="_blank" href="${image}"><img style="width: 400px"
src="${image}" /></a><br/><br/>`
    }
    var popup = new OpenLayers.Popup.FramedCloud(
```

```
                    id,
                    event.object.lonlat,
                    new OpenLayers.Size(300,300),
                    `<h3 style='max-width: 300px'>${item["Назва об'єкту"]}</h3>
<p>Вартість робіт по відновленню: ${(item['Орієнтовна вартість робіт на відновлення об’єкту (тис']['грн'][')'] * 1000).toLocaleString()} грн. </p> <p>
Соціальна оцінка: ${item['Загальна  бальна оцінка']} </p>  <p style='max-width: 300px'>${item["Обгрунтування щодо потреби у виконанні робіт"]}</p>${block}`,
                    null,
                    true);
        map.addPopup(popup);

        OpenLayers.Event.stop(event);
      });
    }

  (async function(){

    roadsCsv = await (await fetch('roads.csv')).text();
    roads = await csv().fromString(roadsCsv);

    bridgesCsv = await (await fetch('bridges.csv')).text();
    bridges = await csv().fromString(bridgesCsv);

    damCsv = await (await fetch('dam.csv')).text();
    dam = await csv().fromString(damCsv);

    shoreProtectionCsv = await (await fetch('shoreProtection.csv')).text();
    shoreProtection = await csv().fromString(shoreProtectionCsv);
```

```
var result = await fetch('photos/files.txt')
imagesText = await result.text();
images  = imagesText.split('\n');


roads.map(item => {
  item.purpose = "roads"
  item.images = images;
  return item;
})

bridges.map(item => {
  item.purpose = "bridges";
  item.images = images;
  return item;
})

dam.map(item => {
  item.purpose = "dam";
  item.images = images;
  return item;
});

shoreProtection.map(item => {
  item.purpose = "shoreProtection";
  item.images = images;
  return item;
})

roads.forEach(markerProcessions);
```

```
            bridges.forEach(markerProcessions);
            dam.forEach(markerProcessions);
            shoreProtection.forEach(markerProcessions);



            var stationsText = await (await fetch('stations.csv')).text();
            var stationsByLine = stationsText.split('\n');
            var stationsCount = stationsByLine.length / 15;

            var stationsDataSplitted = [];

            for (var i = 0; i < stationsCount; i++) {
              stationsDataSplitted[i] = stationsByLine.slice((0 + (i * 15)), (15 + (i * 15)));
            }



            for (stationData of stationsDataSplitted) {
              generalInfo = await csv({noheader: true, output:
'csv'}).fromString(stationData[0]);
              yearlyData  = await csv({noheader: true, output:
'csv'}).fromString(stationData.slice(1, 15).join('\n'));
              generalStruct = {
                name: generalInfo[0][1],
                latitude: parseFloat(generalInfo[0][2].replace('Довгота - ', '')),
                longotude: parseFloat(generalInfo[0][3].replace('Широта - ', ''))
              }

              var cords = new OpenLayers.LonLat(generalStruct.longotude,
generalStruct.latitude)
                .transform(
```

```
            new OpenLayers.Projection("EPSG:4326"), // transform from WGS 1984
            map.getProjectionObject() // to Spherical Mercator Projection
          );

          var size = new OpenLayers.Size(28,28);
          var offset = new OpenLayers.Pixel(-(size.w/2), -size.h);
          var icon = new OpenLayers.Icon(`icon-stations.png`, size, offset);

          var marker  = new OpenLayers.Marker(cords, icon);
          stationsLayer.addMarker(marker);

          headerCells = yearlyData[0]
            .map(cellContent => {
              return `<th style="border: 1px solid black; padding:
3px">${cellContent}</th>`
            })
            .join('');
          headerRaw = `<tr>${headerCells}</tr>`;

          data = yearlyData
          .slice(1)
          .map(rawArray => {
            rawContent = rawArray
              .map(cellContent => {
                return `<td style="border: 1px solid black; padding:
3px">${cellContent}</td>`
              })
              .join('');
            return `<tr>${rawContent}</tr>`
          })
```

```
            .join('');

            addStationPopup(marker, cords, generalStruct, headerRaw, data);
          }

        })();

        function addStationPopup(marker, cords, generalStruct, headerRaw, data) {
          marker.events.register('click', this, function(event){
            var popup = new OpenLayers.Popup.FramedCloud(
                    `station-${Math.random() * 100000}`,
                    cords,
                    new OpenLayers.Size(500,500),
                    `<h3>${generalStruct.name}</h3><table style="border-collapse:
collapse;"><caption>Статистика</caption>${headerRaw}${data}</table>`,
                    null,
                    true);
            map.addPopup(popup);
          })
        }

      </script>
    </body></html>
```

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Дипломна робота
на ступінь вищої освіти бакалавр

на тему: «Розробка ПЗ клієнтської частини для інтерактивної карти візуалізації протипаводкових заходів Івано-Франківської області»

Виконав: студент 4 курсу, групи КНД–42
спеціальності
6.050102 Комп'ютерні науки
Саміляк Іван Миколайович

Керівник Іщеряков Сергій Михайлович

Мета роботи:

розробка багатофункціональної інтерактивної карти

Об'єкт дослідження:

інтерактивна карта протипаводкових заходів

Предмет дослідження:

веб-додаток для відображення карти та позначок на ній

Актуальність роботи

- Потреба регіональних служб та органів в зручному інструменті консолідації географічних та структурованих даних

- Відсутність альтернативних існуючих рішень

Практична цінність

Щорічні паводки потребують глибокого та швидкого аналізу великих масивів даних різними людьми для різноманітних цілей.

Роззосередженість даних по документальній базі сповільнює цей процес.

Задіяні служби сформували свої потреби в загальне бачення майбутнього продукту

Google Earth

Прототип системи був побудований на базі Google Earth.

В процесі експлуатації були виявлені проблеми з читабельністю даних, простотою використання та актуалізації інформації про проекти.

Технічне завдання

В процесі обговорення було сформоване технічне завдання.

Воно описує вимоги до дизайну, формат даних про кожен об’єкт та можливості їх фільтрації на карті

Архітектура

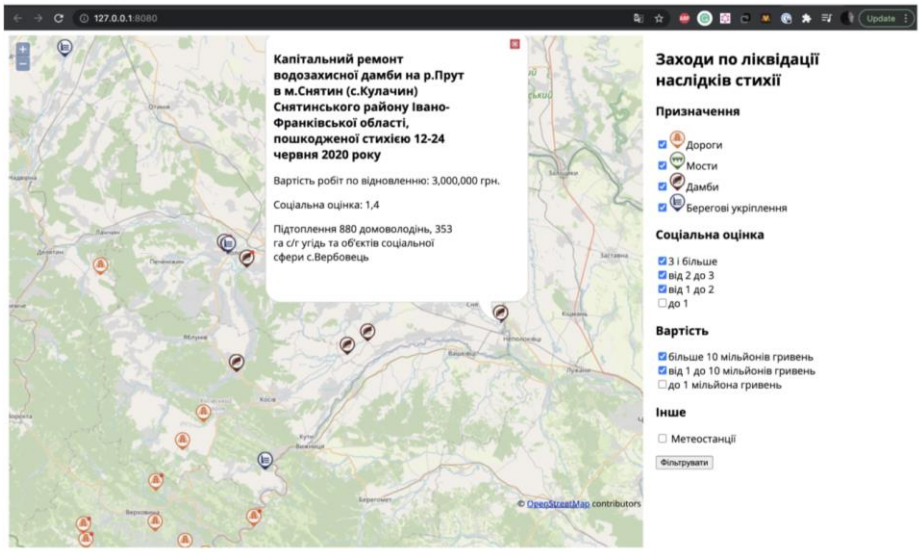
На етапі проектування:

- визначено технології для реалізації
- визначено формат даних
- визначено структуру даних та медіафайлів
- встановлено принципи розробки фронтенду

Реалізація

- за обмежений час був представлений повністю функціональний прототип
- реалізація відповідала встановленим вимогам
- проект успішно пройшов тестування кінцевими користувачами

Демонстрація



Висновки

- Проект пройшов повний цикл розробки: від постановки задачі до прийняття результату користувачем
- Поставленні в науковій роботі задачі виконані в повній мірі
- Проект знайшов використання за своїм призначенням в Івано-Франківській ОДА