МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)
18 грудня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-наукової програми «Інформатика»
на тему: «Інформаційна технологія моніторингу за хіміко-екологічним станом поверхневих вод»
здобувача групи ІН.м-25 Козацького Богдана Ігоровича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Богдан КОЗАЦЬКИЙ
(підпис)

Керівник,
в.о. завідувача кафедри,
кандидат фізико-математичних наук Анна БАДАЛЯН _____
(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук, освітньо-наукової програми «Інформатика»
здобувача групи ІН.м-25 Козацького Богдана Ігоровича

1. Тема роботи: «Інформаційна технологія моніторингу за хіміко-екологічним станом поверхневих вод»
затверджую наказом по СумДУ від «06» грудня 2023 р. №1412-VI

2. Термін здачі здобувачем кваліфікаційної роботи до 18 грудня 2023 року

3. Вхідні дані до кваліфікаційної роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Пошук та підбір літератури по темі "Інформаційна технологія моніторингу за хіміко-екологічним станом поверхневих вод.
2) Виклад результатів огляду літератури за темою "Інформаційна технологія моніторингу за хіміко-екологічним станом поверхневих вод. 3) Вибір програмних засобів рішення задачі для розробки веб-сервісу. Виклад вибраних програмних засобів рішення поставленої задачі
4) Розробка веб-сервісу моніторингу за хіміко-екологічним станом поверхневих вод.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Пошук та підбір літератури по темі "Інформаційна технологія*		

	моніторингу за хіміко-екологічним станом поверхневих вод		
2	*Виклад результатів огляду літератури за темою "Інформаційна технологія моніторингу за хіміко-екологічним станом поверхневих вод*		
3	*Вибір програмних засобів рішення задачі для розробки веб-сервісу. Виклад вибраних програмних засобів рішення поставленої задачі*		
4	*Розробка веб-сервісу моніторингу за хіміко-екологічним станом поверхневих вод*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ Керівник _____

(підпис) (підпис)

АНОТАЦІЯ

Записка: 101 сторінка, 36 рисунків, 7 таблиць, 4 додатки, 30 джерел.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботи є актуальною, оскільки присвячена розв'язанню проблеми із отриманням інформації стосовно хіміко-екологічного стану поверхневих вод шляхом розробки відповідних методів, моделей та інформаційної технології.

Мета роботи: розробка програмної реалізації, котра дозволяє проводити моніторинг за хіміко-екологічним станом поверхневих вод.

Об'єктом дослідження є процеси оцінювання користувачами функціоналу роботи веб-сервісу за хімічним моніторингом поверхневої води.

Предметом дослідження є показники відображення результатів роботи веб-сервісу моніторингу за хімічним станом поверхневих вод.

Методи дослідження – аналіз і систематизація наукової та методичної літератури, веб-ресурсів, за допомогою яких обґрунтовано теоретичні положення з даної проблеми; верстка веб-сторінок із застосуванням технологій HTML і CSS, програмування на мові JavaScript; тестування веб-застосунків; опитування учасників в ролі потенційних користувачів у процесі дослідження значущості розробленої технології; статистична обробка результатів.

Результати. Розроблення веб-сервісу, що виконує функції моніторингу хіміко-екологічного стану поверхневих вод в режимі онлайн. Дана технологія успішно працює на ПК, android, смартфонах.

ВЕБ-СЕРВІС, JAVASCRIPT, EXPRESS JS, REACT JS, ПОВЕРХНЕВІ ВОДИ, ГДК, ВЕБ-МАПА

ЗМІСТ

СПИСОК УМОВНИХ СКОРОЧЕНЬ

API – application programming interface (інтерфейс прикладного програмування)

AJAX – asynchronous JavaScript and XML(асинхронний JavaScript та XML)

BSON – binary encoded JavaScript Object Notation

CSS – cascading style sheets

JSON – JavaScript Object Notation

HTML – hypertext markup language

HTTP – hyper text transfer protocol

HTTPS – hyper text transfer protocol security

pH – водневий показник

ГДК – гранично-допустима концентрація

ДСанПіН – Державні санітарні правила та норми

СКБД – служба керування базою даних

ХСК – хімічне споживання кисню

ВСТУП

Сьогодні спостерігається проблема в сфері екологічної безпеки та здоров'ї живих організмів, внаслідок того, що хімічний склад природних об'єктів за останні роки значно змінився в бік перевищення ГДК хімічних показників таких як нітрати, ХСК, фосфати, зокрема природних вод.

Причини перевищення ГДК у воді можуть бути різними: місцеві хімічні заводи біля водойм, несправні каналізаційні колектори, скидання свинцевих акумуляторів у водойму тощо. Дану проблематику можна спостерігати наприклад за динамікою зміни хіміко-екологічно стану водних об'єктів Сумської області за 2019 рік[1] або підрив Каховської дамби, що супроводжувало різку зміну концентрації хімічних інгредієнтів. Наслідками перевищення концентрації хімічних речовин при безпосередньому контакті із водою можуть бути летальним чи супроводжуватися різними хворобами. Всього цього можна було уникнути, якщо потрібна інформація доносилася до публіки, оскільки основний компонент будь-якої цивілізації складає суспільство, то саме вони і здатні змінити умови життя в бік їх поліпшення.

Інструментарії, котрі здатні надавати актуальну інформацію про кількісний хімічний склад природних вод для загальної публіки досить малий, оскільки більшість даних про воду відображається не в широкому колі, внаслідок того, що не усі люди знають стандарти в законодавстві ДСанПіН 2.2.4-171-10 по ГДК хімічних показників, котрі відображають інформацію у кількісній формі[2].

Щоб розробити такий інструментарій, котрий дозволяв би відстежувати звичайному користувачу динаміку зміни хіміко-екологічного стану води, треба враховувати декілька критеріїв, серед яких це вік користувачів, зручність даного інструментарію у використані та реалізація принципу доступності та відповідності інструментарію згідно із принципів дидактики та інтересів суспільства. Одним із оптимальних рішень до реалізації такого

інструментарію, є розробка веб-сервісу, котра б дозволяла відстежувати хімічний склад поверхневих вод.

У роботі використані результати опитування по роботі розробленого веб-сервісу для реальних людей із різних галузей діяльності та віку. Робота представлена в збірнику конференції факультету ЕЛІТ "FEE-2023" & "IMA-2023"

Об’єкт дослідження. Процеси оцінювання користувачами функціоналу роботи веб-сервісу за хімічним моніторингом природньої води.

Предмет дослідження. Показники відображення результатів роботи веб-сервісу моніторингу за хімічним станом природних вод.

Основні завдання роботи:

1. Для встановлення рівня актуальності теми дослідження, виконати аналіз наукових джерел і програмних засобів та розробок, що пов’язані із розробкою веб-застосунків.

2. Визначити засоби, котрі дозволяють розробити інформаційну технологію за моніторингом хімічних показників поверхневих вод.

3. Обрати програмні засоби, котрі дозволяють розробити інформаційну технологію за моніторингом хімічних показників поверхневих вод.

4. Виконати тестування та провести опитування для користувачів веб-сервісу розробленої інформаційної технології, за отриманими результатами, надати можливі рекомендації із дослідження роботи веб-сервісу.

Гіпотеза. Розробка веб-сервісу моніторингу за хіміко-екологічним станом поверхневих вод надає актуальну інформацію для суспільства різних вікових категорій за різними видами діяльності, що спрямовано на забезпечення екологічної безпеки та здоров’я живих організмів.

Наукова новизна. На відміну від існуючих аналогів інформаційних систем, описане у даній роботі програмне рішення дозволить через онлайн-мапу стежити за інформацією про хіміко-екологічний стан поверхневих вод,

коригувати та додавати значення про місця відбору проб та хімічні показники через розроблену частину сервісу для уповноваженої за це особи.

Структура. Дана робота складається зі вступу, аналізу публікацій, постановки задачі дослідження, вибір методики та інструментів для рішення поставленої проблеми, опису програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

Практичне і теоретичне значення даної роботи полягає у використанні інформаційної технології дослідження в системі моніторингу стану поверхневих вод.

Апробація результатів дослідження. Розроблений нами проєкт моніторингу за хіміко-екологічним станом поверхневих вод є стартапом під назвою Chemmagpie, котрий був представлений у фіналі XII Всеукраїнської Інноваційної екосистеми "Sikorsky Challenge Україна", а також в Сумському стартап-центрі "New Generation"[3].

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Вікові особливості людей у сприйнятті сучасних інформаційних технологій.

В різних суспільствах можуть бути спільні інтереси та цілі, але якщо брати до уваги мрії, потреби та способи досягання мети, то це вже залежить від індивідуальної думки. Особливостями кожного індивіда є характер, він складається із життєвого досвіду, темпераменту, вікових особливостей, середовища перебування.

Вік (у психології) – конкретна, відносно обмежена в часі ступінь психічного розвитку індивіда та його розвитку як особистості, яка характеризується сукупністю закономірних фізіологічних і психологічних змін, не пов'язаних з розходженням індивідуальних особливостей[4]. При цьому слід ураховувати у вікових особливостях умови, в котрих індивід зростає, так наприклад якщо людина зростала в 50-х роках XX сторіччя, то в більшості випадків в неї буде інший світогляд на відміну від індивіда, котрий зростав в епоху ренесансу інформаційно-комунікативних технологій.

В ході досліджень було виявлено, що у поведінці поколінь явно простежувалась зміна поведінки від часу, в якому вона жила. Відповідно від «епохи», в котрій народилися індивіди, вони класифікуються на[5]:

- покоління переможців (1900–1923 р.н.) – герої-примиренці, відрізняються високою працездатністю, оптимізмом і відповідальністю;
- мовчазне покоління (1923–1943 р.н.) або художники-пристосуванці – схильні поважати закон і статус людини.
- бебі-бумери (1943–1963 р.н.) або пророки-ідеалісти – цінують атмосферу стабільності та відчуття потрібності;
- покоління X (1963–1984 р.н.) – кочівники-активісти – цінують розвиток і навчання, роботі приділяють більше уваги, ніж сім’ї, спрямовані до індивідуального успіху;

- покоління Y (1984–2000 р.н.) – герої-примиренці – комунікативні, цінують вільну атмосферу, схильні до співпраці та водночас прагнуть швидких результатів
- покоління Z (народжувалось після 2000 р.) або художники-пристосуванці – це перше повністю цифрове покоління, пов'язане між собою за допомогою мережі Internet, YouTube, мобільних телефонів, SMS і MP3-плеєрів;
- покоління α(розпочало народжуватись після 2010-2020 рр.) пророки-ідеалісти – більш врівноважені, більш позитивні і менш агресивні. Це будівники ноосфери майбутнього.

Дана класифікація є умовною, наприклад індивід із роком народження 1997 може бути представником покоління Z замість Y. У той же час дана класифікація здатна надати загальну картину залежності епохи, в котрій жила людина, від її загальної картини бачення дійсності.

Слід звернути увагу на те, що представники різних поколінь можуть як взаємодіяти один із одним синхронно, або конфліктувати. Так наприклад 25 річний індивід при поясненні 75 річній людині принцип роботи андроїду, в більшості випадків 75-річним людям важко засвоювати нове, особливо тим, котрі мали у XX сторіччі інформаційний вакуум наприклад під дією пропаганди комуністичних режимів та не бажали освоювати новітні технології умовного «Заходу», загалом це тип людей, котрі не тренували і не вдосконалювали власні знання, вміння та навички впродовж власного життя чи отримували постійний стрес від взаємодії із інноваційним навколишнім середовищем чи заохочували себе шкідливим способом життя. Інший випадок людей яким за 75 років, це люди, котрі займаються собою здатні освоїти сучасні інформаційно-комунікативні технології наприклад це президент 2021-2025 років та екс віце-президент Джо Байден або представники наукових інституцій пенсійного віку.

Якщо брати до уваги взаємодію та обмін інформацією із поколіннями Z та Y, то в більшості випадків вона проходить ідеально. Оскільки більшість інтересів та ідей досягнення поставленої мети співпадають між собою. Це все відображається на доступності пояснення їм певної інформації. Таким чином наприклад, щоб розповісти з «нуля» як працює безпілотний літаючий апарат або принцип роботи веб-додатку, тобто передача суб'єктивного досвіду від покоління Z до Y є простішим ніж від Z до покоління бебі-бумерів.

1.2 Аналіз сучасних веб-сервісів, що пов'язані із мапінгом

Люди з давна користуються мапами в різних цілях: знайти дорогу додому, проаналізувати короткий маршрут, визначити ворожу локалізацію тощо. За часи ренесансу інформаційно-комунікативних технологій, вони дійшли і до розвитку концепції мапи. Поєднуючи технології глобальної мережі Internet та інші аспекти галузей, можна створити інноваційний продукт, котрий дозволить суспільству орієнтуватися, що відбувається у світі, чи є певна небезпека, де краще відпочивати тощо.

1.2.1 Додаток Liveuamap

Один із застосунків, котрі представляють собою веб-мапінг є Liveuamap. Даний сервіс, котрий дозволяє користувачам відстежувати мілітарні чи кримінальні події в різних куточках світу в режимі онлайн, базуючись на різних фактичних джерелах інформації, використовуючи бібліотеку мапінгу OpenStreetMap[6].

Технологія Liveuamap відстежує авторів цікавих та довірених джерел дописів у соціальних мережах, ідентифікуючи їхні попередні дописи, кількість дій, за якими вони стежать і застосовує методи фільтрації для отримання даних. Коли накопичення корельованих повідомлень про подію, що відбувається в певному місці, перевищує порогові значення, визначені алгоритмами, ситуація вноситься до списку для втручання людини. Принаймні два учасники Liveuamap вирішують, чи інформація про подію дійсна, чи її можна використовувати на мапі, чи потрібна додаткова перевірка. Подальша

інформація про прийняті події використовується як зворотний зв’язок для покращення системи[7]. Члени Liveuamap використовують також додаткові джерела, такі як супутникові знімки та офіційні повідомлення. Архіви доступні для відстеження еволюції певної мапи[6].

1.2.2 Додаток mine.dsns.gov.ua

Даний веб-сервіс спрямований на приблизному визначенні кількості мін в певних областях України, інформація в базу даних оновлюється щоразу, коли знаходять чи повідомляють про вибухонебезпечні предмети (похибка локалізації становить до 30 м). Цей сервіс є корисним для рятувальників та жителів, котрі проживають на деокупованих територій таких як Тростянець, Ворожба, Херсон, Берислав тощо[8].

1.2.3 Сервіс Google map

Google map – це один із популярних сервісів мапінгу, представляє собою картографічну платформу, власником на сьогодні є корпорація Google. Сервіс пропонує супутникові зображення, аерофотозйомку, карти вулиць, на 360° інтерактивні панорамні види вулиць, але не усюди, стан дорожнього руху в реальному часі та планування маршруту для подорожей пішки, на автомобілі, велосипеді, літаком (у бета-версії) та громадським транспортом. Станом на 2020 рік Google Maps щомісяця використовувало понад один мільярд людей у всьому світі[9,10].

Функції Google map:

1) Напрямки та громадський транспорт. Google map надає планувальник маршрутів, що дозволяє користувачам знаходити доступні маршрути за допомогою автомобіля, громадського транспорту, пішки чи велосипеда.
2) Умови руху. У 2007 році Google почав пропонувати дані про дорожній рух у вигляді кольорового накладення поверх доріг і автомагістралей для відображення швидкості транспортних засобів

на певних дорогах. Краудсорсинг використовується для отримання визначених GPS місцезнаходження великої кількості користувачів.

3) Перегляд вулиць. При приближені до певної точки мапи, користувач здатний розгледіти які вулиці чи вулиця є в даній координаті.

Веб-розробники здатні використовувати бібліотеки чи шари Google map за певну плату в залежності від направлення розробки та її обсягу.

1.2.4 Веб-сервіс SaveEcoBot

SaveEcoBot – перший в Україні екологічний бот для моніторингу інформації про дозвільні документи та процедури промислових та інших забруднювачів довкілля[11].

Даний сервіс має 3 онлайн мапи: мапа якості повітря, мапа радіаційного фонду, мапа пожеж та напрямку вітру.

Приклад роботи такого відображено на рисунку 1.1, котра базується на колективно створеній базі даних мап OpenStreetMap.

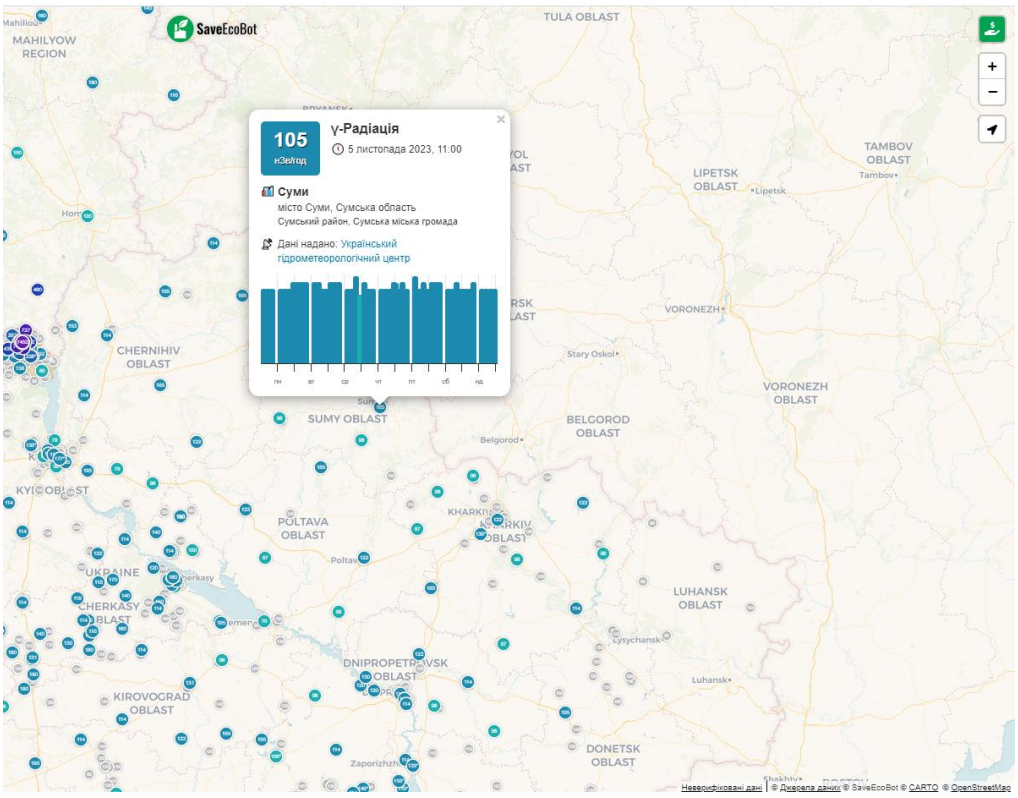

Рисунок 1.1 – приклад веб-мапи радіації проєкту SaveEcoBot на прикладі міста Суми

Відповідно до рисунку можна побачити, що дана мапа відображає показник гамма-випромінювання за часом для певного населеного пункту, в даному випадку було обрано місто Суми, при цьому користувач може змінити за бажанням інший населений пункт відображення гамма-випромінювання у формі точки. Таким чином користувач здатний бути проінформований про небезпеку радіаційного забруднення в певній точці України.

1.3 Аналіз сучасних підходів до розробки веб-сервісів

На сьогодні у зв'язку із популяризацією веб-програмування, існує різноманітний перелік підходів до створення повноцінних веб-сервісів. При цьому із розвитком та популяризацією застосунків на мобільних пристроях та iOS їх виділили в окрему гілку, оскільки як база використовується односторінкові додатки, а в традиційні версії використовуються багатосторінкові веб-сервіси.

SPA (односторінкова програма) – це реалізація веб-програми, яка завантажує лише один веб-документ, а потім оновлює основний вміст цього єдиного документа за допомогою JavaScript API, таких як XMLHttpRequest і Fetch, коли має бути показано інший вміст[12].

Таким чином, це дозволяє користувачам використовувати веб-сайти без завантаження цілих нових сторінок із сервера, що може призвести до підвищення продуктивності та більш динамічного досвіду з деякими компромісними недоліками, такими як SEO більше зусиль, котрих необхідних для підтримки стану, реалізації навігації та досягнення значущої продуктивності моніторинг. Представниками даного підходу є фреймворки React, Angular, Vue.JS.

Недоліком такого підходу є те, що не усі пошукові системи здатні запускати JavaScript, це відноситься до старих версій пошукових систем чи браузерів, наприклад Internet Explorer, що може бути пов'язано також із мірами безпеки, оскільки без оновлень така система є незахищеною та може бути вразлива до хакерських атак.

SEO (Search Engine Optimization) – це підхід до веб-маркетингу та оптимізації веб-сайту для пошукових систем з метою підвищення видимості в пошукових результатах і залучення більше відвідувачів, тобто основний акцент робиться на бізнесі в отримані матеріальних статків.

Пошукові системи сканують Internet, переходячи за посиланнями від сторінки до сторінки та індексують знайдений вміст. Під час роботи пошукової системи, вона відображає проіндексований вміст. Кроулери дотримуються правил, якщо чітко дотримуватися цих правил під час оптимізації пошукових систем для веб-сайту, автор веб-застосунку/сервісу надає сайту найкращі шанси з’явитися серед перших результатів, збільшивши трафік і, можливо, дохід (для електронної комерції).

Пошукові системи дають деякі вказівки щодо SEO але великі пошукові системи зберігають рейтинг результатів як комерційну таємницю. SEO поєднує в собі офіційні рекомендації пошукових систем, емпіричні знання та теоретичні знання з наукових статей або патентів[13].

PWA – це технологія, яка відповідає вимогам для використання веб-переглядача та нативної програми. PWA підтримує сучасний веб-сайт, який містить колекцію технологій, програмне забезпечення, а також веб-інтерфейси API, що створюють сучасні веб-програми для використання окремих нативних додатків(програмне забезпечення, яке розроблене для конкретної операційної системи або платформи)[14].

Превагами є майже миттєве, завантаження сторінок за рахунок кешування, працює в офлайн-режимі, легко оновлюється в фоні без втручання користувача. Недоліками такої системи є обмежений доступ до функцій пристрою, підвищення витрат заряду батареї на виконання JavaScript, для комерційного напрямку відсутній функціонал публікації в магазині додатків.

1.4 Огляд систем керування даними в інформаційному просторі

Дані – це та інформація, котра потрібна споживачу для задоволення власних потреб. Бази даних є важливим джерелом інформації в житті суспільства. Завдяки ним користувач здатний швидко знайти потрібну йому інформацію чи створити на основі доступної інформації щось інше чи нове, також існують сховища даних, котрі здатні великий обсяг інформації і мають можливість зберігати різні бази даних в собі[15].

В направлені критерію за реляцією баз даних виділяють реляційні та нереляційні бази даних.

SQL(реляційна база даних) – це композиція з однієї чи декількох пов’язаних з між собою відношенням таблиць із даними, зазвичай зовнішніми ключами. Система керування таким видом баз даних заснована Едгаром Коддом в 70-му році XX сторіччя. Коддівська модель керування складається з набору об’єктів (або відношень), переліку операторів для взаємодії із відношеннями та засобів підтримки цілісності даних. Майже будь-яка задача зі зберігання, читання та отримання даних може бути розв’язана за допомогою реляційних баз даних. Представниками такої моделі є MySQL, PostegreSQL, LiteSQL[16].

NoSQL(not only SQL нереляційна база даних) – це тип керування базою даних, для котрої закони реляційних таблиць не діють, вона представлена у формі об’єктів, для кожного об’єкту є властивості(ключі) із визначеним заздалегідь типами даних і дані будуть записуватися як значення в певну властивість. Представниками таких моделей є Memcache – нереляційна СКБД із типом «ключ-значення», котра використовується для тимчасового збереження інформації, що може бути отримана за відомим ключем, MongoDB – документо-орієнтована СКБД, котра здатна записувати власні дані в форматі JSON, BSON[16].

1.5 Постановка задачі

Створення інформаційної технології, котра дозволяє відстежувати хіміко-екологічний стан поверхневих вод для загальної аудиторії в формі онлайн мапи. Ця технологія повинна відображати хімічні показники поверхневих вод в залежності від місця відбору проби води для аналізу, дані повинні регулюватися в окремій адміністративній частині даної системи, уповноваженої для цього особою.

Для виконання поставленої мети реалізовані наступні завдання:

1) Відображення онлайн мапи із точками, котрі є місцями відбору проби води.
2) Для доступності та зрозумілості відображення інформації на онлайн-мапі в формі певних позначок, створення легенди веб-мапи.
3) Можливість переглядати показники води в залежності від місць відбору проби натиснувши.
4) Відображувати дані хімічних показників в словесно-цифровій формі та в формі відображення кольорів.
5) Розробити адміністративну частину такого веб-сервісу, в котрому уповноважена особа, здатна керувати даними по відображенню даних в онлайн мапі для загальної аудиторії.

За умови успішної реалізації отримаємо веб-сервіс, котрий здатний відображати для загального користування систему моніторингу хіміко-екологічного стану поверхневих вод. Очікується, що дана технологія буде працювати на ПК, android, смартфонах.

В результаті створення такої системи, провести дослідження методом опитування користувачів даного веб-сервісу так, щоб знати, чи потрібен такий додаток взагалі на основі віку користувачів і чи пов’язана їхня діяльність із природничими дисциплінами.

2 ПРОЕКТУВАННЯ РОБОТИ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ

2.1 Технології створення веб-сервісів

На сьогодні існує великий перелік мов програмування у сфері для побудови веб-сайту. В епоху буму розвитку інформаційно-комунікативних технологій, кожна мова програмування розвивається та вдосконалюється із єдиною метою – стати кращим за конкурентів.

Вибір мови програмування залежить від різних факторів:

- вік існування даної мови програмування;
- зручність написання коду на певній мові програмування;
- популярність мови програмування;
- системні вимоги для створення певного застосунку/коду за допомогою певної мови програмування;
- наявність бібліотек програмування, котрі дозволяють розширити функціонал при створені застосунків.

При цьому неважливо, яку мову веб-програмування обрав розробник, головне досягти основну мету – створити веб-застосунок, оскільки мова програмування є лише інструментарієм.

Якщо веб-проект, створений розробником, має великих обсяг байтів, доцільно використовувати фреймворки для покращення та оптимізації реалізації певного проекту. Сам по собі фреймворк є однією із функціональних можливостей для мови програмування, метою якого – поліпшення створення чогось за рахунок генерації меншого обсягу коду. Наприклад, щоб не створювати динамічні сайти з нуля, в клієнтській частині додатків, використовується React js – фреймворк мови програмування JavaScript.

Слід враховувати, що при генерації коду рівень абстракції від предметної області – чим вищий цей показник, тим менше коду потрібно відтворювати. Не менш важливим є інтеграція та рівень складності в оволодінні певною технологією в області фреймворків.

2.1.1 Технології для створення серверної частини.

Серверною(backend) частиною веб-застосунків називають частину програмного забезпечення, яка відповідає за обробку запитів від клієнтської сторони, виконання бізнес-логіки та взаємодію з базою даних. Вона забезпечує логіку, необхідну для обробки та відповіді на запити, які приходять від користувачів через інтернет[17].

Основні функції серверної частини наступні:

- сервер отримує запити від клієнтської сторони (наприклад, браузера) та обробляє їх. Це може включати отримання даних форм, обробку AJAX-запитів, HTTP-запитів тощо;
- виконання бізнес-логіки, яка визначає, як програма повинна обробляти дані та виконувати різні операції згідно з логікою додатку чи системи;
- взаємодія з базою даних для зберігання та отримання даних, необхідних для виконання запитів користувача, наприклад виконувати обробники запитів такі як GET чи POST від MongoDB чи MySQL;
- забезпечення безпеки та контролю доступу до ресурсів, включаючи автентифікацію користувачів та надання їм необхідних прав доступу;
- сервер генерує відповіді та надсилає їх клієнтській стороні. Це може бути HTML-код, JSON-об'єкти, зображення та інші ресурси.
- управління станом серверної частини, такими як сеанси користувачів, логіни, інші динамічні дані.

Серед популярних та ефективних в той же час на сьогодні інструментарієм для створення серверної частини є Express js – це фреймворк написаний на мові JavaScript. Основними його перевагами є[18]:

1) Простота та легкість використання. Має мінімальний набір необхідних функцій, що дозволяє швидко розпочати роботу та вивчити його.

2) Надає розробникам велику свободу вибору бібліотек, пакетів та архітектурних рішень. Це дозволяє створювати веб-застосунки, що відповідають конкретним вимогам проекту.
3) Має велику та активну спільноту розробників. Це означає, що ви можете легко знайти підтримку, документацію, плагіни та інші корисні ресурси.
4) Використовує потужну систему middleware, яка дозволяє розробникам впроваджувати функціональність по шарах. Це спрощує обробку різних аспектів запиту та відповіді.
5) Можна легко розширювати можливості Express.js за допомогою додаткових пакетів та middleware, які допомагають вам вирішувати конкретні завдання та вимоги.
6) Володіє досить високою швидкодією через використання Node.js, яка базується на асинхронному, подієвому програмуванні.
7) Легко інтегрується з іншими бібліотеками та сервісами, що робить його ефективним рішенням для розробки різноманітних веб-застосунків.

З огляду на популярність використання впровадження бібліотек та популярності фреймворку Express js, було обрано його в якості інструментарію обробки запитів для бази даних для веб-сервісу для моніторингу хіміко-екологічного стану природних вод.

2.1.2 Технології для реалізації клієнтської частини

Клієнтська частина(frontend) – це та частина веб-застосунку або програмного забезпечення, яку користувач бачить та із якою він взаємодіє безпосередньо. Це візуальна та інтерактивна сторона додатку, що відображається у веб-браузері або на іншому пристрої[19].

Основні завдання та аспекти клієнтської частини включають[20]:

1) Створення естетичного та зручного для користувача вигляду інтерфейсу за допомогою HTML, CSS та інших технологій.

2) Додавання різноманітних елементів, таких як кнопки, форми, меню та інші, які роблять веб-застосунок інтерактивним та відповідальним на користувацькі дії.
3) Реалізація логіки, яка відбувається безпосередньо на браузері або на пристрої користувача. Це може включати в себе валідацію даних, обробку подій, маніпуляції із DOM та інші клієнтські операції.
4) Взаємодія з серверною частиною за допомогою AJAX-запитів або інших методів, щоб отримувати та відправляти дані.
5) Забезпечення, щоб веб-застосунок виглядав і працював ефективно на різних пристроях та роздільній здатності екрану.
6) Валідація введених даних, захист від атак та забезпечення безпеки користувачів.
7) Керування станом веб-застосунку та відстеження змін, які відбуваються в інтерфейсі.

Одним із таких фреймворків, котрі дозволять реалізувати «обличчя» нашого веб-сервісу, було обрано фреймворк React. Особливості React полягає у постійні підтримці фреймворку з боку спільноти в формі різних бібліотек і оновлень до них, простий поділ JavaScript частини та HTML частини в одному файлі за необхідності. Як наслідок, значна кількість «front-end» або «full-stack» розробників реалізують свою роботу в формі через даний фреймворк[21].

2.2 Проектування інформаційної моделі веб-сервісу

IDEF (Integrated Definition) – це графічна методологія моделювання процесів, яка використовується для впровадження систем та програмного забезпечення. Ці методи використовуються для функціонального моделювання даних, симуляції, об’єктно-орієнтованого аналізу та отримання знань. Дана методологія моделювання процесу застосовується переважно для розробки програмного забезпечення[22].

IDEF відноситься до сімейства мов моделювання, яке включає 16 різних методів. Ці методи моделювання процесу охоплюють широкий спектр

використання, і кожен метод охоплює певний тип даних. Нами було обрано моделювання за IDEF0, вона використовується для моделювання функціональної діяльності системи або процесу основна мета її – це покращення спілкування та розуміння елементу чи елементів всередині та між організаціями, залученими до розробки складних систем.

Для візуалізації через IDEF0 систему для веб-сервісу за динамікою зміни хіміко-екологічного стану поверхневих вод, нами було використано один із інструментів Microsoft – Visio. Дана програма надає різноманітні шаблони для різних типів діаграм, включаючи діаграми IDEF0. Користувачі можуть використовувати ці шаблони для створення діаграм IDEF0 для моделювання функцій і дій системи або процесу в формі контекстної діаграми[23].

Контекстна діаграма, що описує принцип роботи веб-сервісу за моніторингом хіміко-екологічного стану поверхневих вод в узагальненому вигляді, зображена на рис 2.1. Принцип роботи наступний: є база даних, котра має дані, введені уповноваженим за це користувачем через спеціальний розроблений ресурс, ці дані виводяться до загального аудиторії у формі дисплею-мапи, а веб-мапа відображає дані для загальної аудиторії.

Використовуючи Microsoft Visio, було виконане структурно-функціональне моделювання бізнес-процесів у нотації IDEF0. Визначено, що ресурсами даного процесу є дані, введені адміністратором веб-сервісу, якщо регіон(країна, область) розширюється в цьому випадку додаються дані від IT-спеціаліста. Користувачами даних є усі учасники веб-сервісу за динамікою зміни хіміко-екологічного стану поверхневих вод. Критерії ГДК в користувацькій частині сервісу регламентуються згідно ДСанПіН 2.2.4-171-10 «Вода питна»[2].

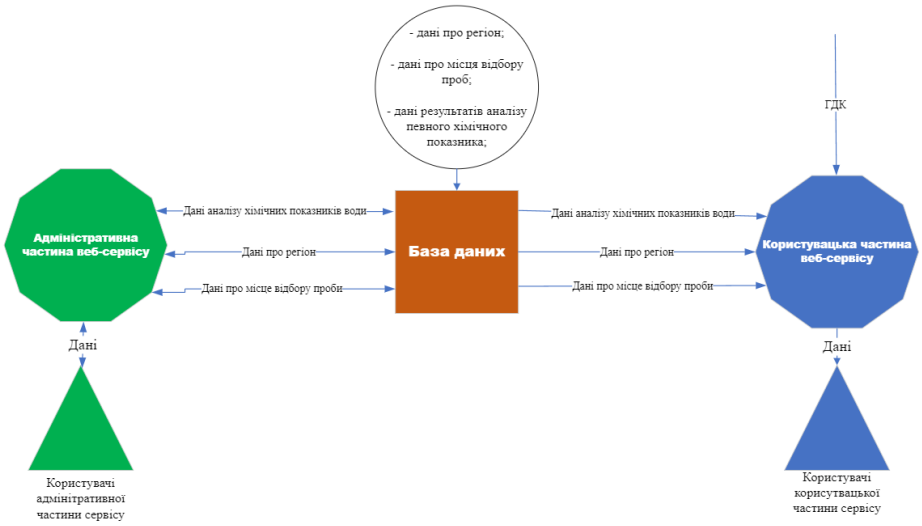

Рисунок 2.1 – Контекстна діаграма процесу «Принцип роботи веб-сервісу за хіміко-екологічним станом поверхневих вод» у нотації IDEF0 через інструмент Visio

Для більш детального опису того, що відбувається в основній діяльності, знадобилося зобразити схему із 5 блоків:

- зелений трикутник – користувачі адміністративної частини сервісу;
- зелений десятикутник – адміністративна частина веб-сервісу;
- коричневий квадрат – база даних веб-сервісу;
- синій десятикутник – користувацька частина веб-сервісу;
- синій трикутник – користувачі публічної частини сервісу, що представляє собою мапу із даними.

Решта зі схем стрілки та білий круг із чорним контуром позначають вхід-вихід інформації даних в залежності від направленостей стрілок. Відповідно до контекстної діаграми принцип додатку наступний: сервіс складається із адміністративної та користувацької частини, поєднаних між собою базою даних, адміністративна частина сервісу здатна вводити-виводити дані, користувацька частина – виводить дані, таким чином дана технологія функціонувати в глобальній мережі Internet.

2.3 Проектування моделі бази даних

База даних, котра була як основа для зберігання інформації про місцевість, результати аналізу проб води, була обрана MongoDB. Оскільки ми працюємо над веб-сервісом і потрібно, щоб дані зберігалися у хмарі, для цього було використано сервіс MongoDB Atlas. Його переваги: безпека даних, чудово синхронізується із такими фреймворками як Express js, швидко виводить та обробляє дані.

На відміну від реляційної моделі, представниками якої є мова баз даних MySQL, MongoDB представляє дані у формі колекції, котра в подальшій обробці здатна візуалізувати дані для розробника в форматі JSON чи BSON файлах. Хоч тут нема базису поєднання таблиць даних у формі зовнішніх ключів, розробник здатний за бажанням створити зовнішні ключі в ручному режимі з метою поєднання певних залежностей, наприклад, якщо в одній колекції знаходяться загальні дані такі як країна та регіон, то на основі цієї інформації, ми можемо створити спільний для двох колекцій зовнішній ключ, котрий буде називатися наприклад region, поєднати із другою колекцією, в котрій містяться конкретні адреса клієнтів, завдяки цьому, в другій колекції ми можемо не писати в якій країні абонент знаходиться, достатньо знати регіон(область) де він проживає.

Для нашого веб-сервісу було створено 3 колекції, кожна із яких поєднана створеним нами зовнішніми ключами. Колекції зберігаються в хмарному сервісі баз даних – MongoDB Atlas, відображені на рис. 2.2.

Рисунок 2.2 – Колекції даних хмарного сервісу MongoDB Atlas для веб-сервісу стеження зміни хіміко-екологічного стану поверхневих вод

Колекції, котрі були створені для нашого веб-сервісу моніторингу хіміко-екологічного стану поверхневих вод – це places, sampling_places, chemical_indexes.

Колекція «sampling_places» відповідає за місця відбору проб за певними координатами, необхідна для визначення місця розташування ознаки забруднення поверхневої води. Дані коригуються користувачами адміністративної частини веб-сервісу(див. табл. 2.1).

Таблиця 2.1 – Опис властивостей ключів колекції «sampling_places»

Назва ключа	**Опис**	**Роль ключа**
_id	Ідентифікатор місць за певною країною та певним регіоном	PK
region	Назва регіону	FK
name_place	Назва місця відбору проби	FK
type_water_object	Тип водного об’єкта	
name_water_object	Назва водно об’єкту	
latitude	Координата широти відбору проби	
longitude	Координата широти відбору проби	
comment	Коментарі для створеного запису	

Колекція «places», відображає країну та регіон/область, виконує роль набору даних, в якій міститься загальна доступна інформація з інших джерел про країну та регіони, котрі знаходяться в ній, деталі описані в табл. 2.2.

Таблиця 2.2 – Опис властивостей ключів колекції «places»

Назва ключа	**Опис**	**Роль ключа**
_id	Ідентифікатор місць за певною країною та певним регіоном	PK
country	Назва країни	
region	Назва регіону	FK

Колекція «chemical_indexes» відповідає за результати даних певного хімічного показника за певним водним об’єктом. Дані регулюються користувачами адміністративної частини веб-сервісу(див. табл. 2.3).

Таблиця 2.3 – Опис властивостей ключів колекції «chemical_indexes»

Назва ключа	**Опис**	**Роль ключа**
_id	Ідентифікатор результату аналізу проби води	PK
name_place	Назва місця відбору проби	FK
chemical_index	Назва хімічного показника, що був досліджений в результаті аналізу	
result_chemical_index	Результат аналізу хімічного показника води	
date_analysis	Дата аналізу проби води	
comment	Коментарі для створеного запису	

Для встановлення залежностей між зовнішніми ключами колекцій було використано фізичну ER-діаграму було використано онлайн інструмент Lucidchart. Даний інструмент дозволяє створювати діаграми, включаючи ER-діаграми. Lucidchart забезпечує простий та інтуїтивно зрозумілий інтерфейс

для моделювання баз даних, який може бути корисним для розробників та архітекторів даних при проектуванні та аналізі структури даних у веб-додатках. ER-діаграма для веб-сервісу за динамікою-зміни хіміко-екологічного стану поверхневих вод зображена на рис.2.3.

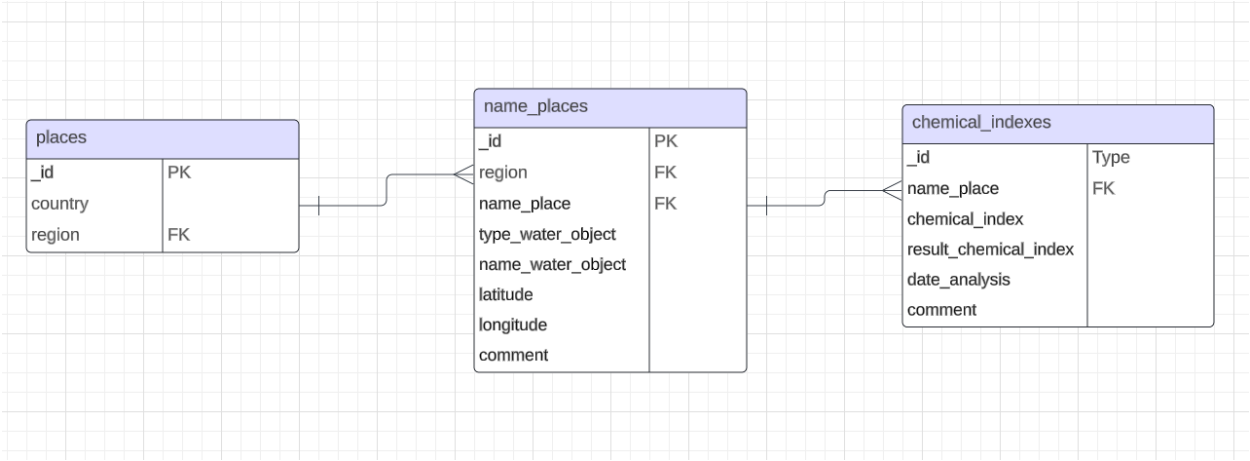

Рисунок 2.3 – Фізична ER-діаграма колекції даних веб-сервісу

3 ІНОФРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ТЕХНОЛОГІЇ

3.1 Опис реалізації технології за хіміко-екологічним станом поверхневих вод

Для реалізації технології за моніторингом хіміко-екологічного стану поверхневих вод, було розроблено Chemmagpie – це стартап-проєкт, який має дві частини – клієнтську та серверну, котрі поєднані із нереляційною базою даних MongoDB в формі хмарної платформи MongoDB Atlas(див. розділ 2.3). Серверна частина працює через фреймворк Express js, клієнтська частина реалізована React js фреймворком, щоб підтримувати взаємозв'язок між клієнтською та серверною частиною, в клієнтській частині доданий проксі сервер із запущеної в режимі онлайн серверної частини.

Для достовірності функціонування проекту Chemmagpie, було запущено хостинг через сервіс AWS EC2 із основним доменом chemmagpie.com. Використання хостингу дозволило зробити дослідження користувачів даного веб-сервісу стосовно необхідності розробленої нами технології для певних цілей: рибництво, рибальство, водний туризм, екологічна безпека водних ресурсів тощо.

3.2 Клієнтські частини

Наш веб-сервіс має дві клієнтські частини – онлайн-мапа із точками відбору та результатами хімічного аналізу проби, друга – адміністративна частина сервісу, котра відповідає за обробку та введення даних точки відбору проби води, результати аналізу проби води. Розділення веб-сервісу на дві клієнтські частини, дозволило нам зменшити час на розробку нашого на веб-сервісу.

Щоб створити наш React проєкт, було використана командну консоль:

```
npx create-react-app «Назва нашого реакт проекту»
```

Після встановлення проєкту було проведене відповідно до якої клієнтської частини редагування файлів та папок, котрі саме потрібні для

нашого проєкту, після виконали завантаження потрібних нам бібліотек, наприклад якщо потрібно створити веб-мапу, то потрібна команда:

```
npm i react-leaflet leaflet
```

Для запуску наших проєктів використана команда:

```
npm start
```

Відповідно до певної клієнтської частини було розроблено програмний код.

3.2.1 Онлайн-мапа відображення результатів хімічних показників та точок відбору проби поверхневих вод

Інтерфейс веб-мапи, що зображений на рис. 3.1, відображає точки місця відбору проби. При натисканні на точку відображається інформація про неї: назва водного об’єкту, назва місця пробовідбору водного об’єкту, тип водного об’єкту, таблиця хімічних показників, результати до них та дата останнього аналізу води. Результати зображені у двох формах: строково-цифровій та кольоровій. З метою того, щоб загальній аудиторії було зрозуміло, чи можна користуватися водою з даного місця відбору проби чи ні, було впроваджено кольорові індикатори, якщо певний хімічний показник за результатами хімічного аналізу перевищує норму ГДК – це червоний індикатор, якщо не виходить за межі ГДК – зелений. На додаток до мапи було впроваджено легенду, при натисканні на неї користувач може подивитися загальну інформацію про мапу.

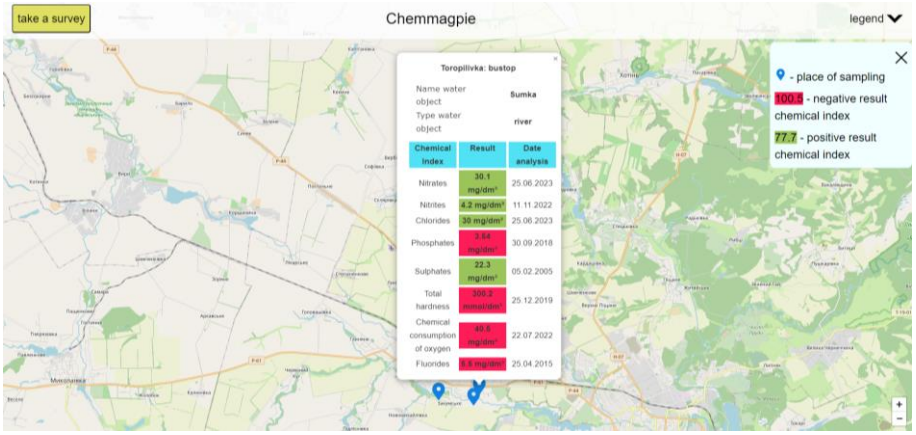

Рисунок 3.1 – інтерфейс онлайн мапи веб-сервісу Chemamgpie

Фреймворк, котрий буде підтримувати виконання логіки на клієнтській час-тині веб-додатку, має таку структуру, що відображена на рис.3.2.

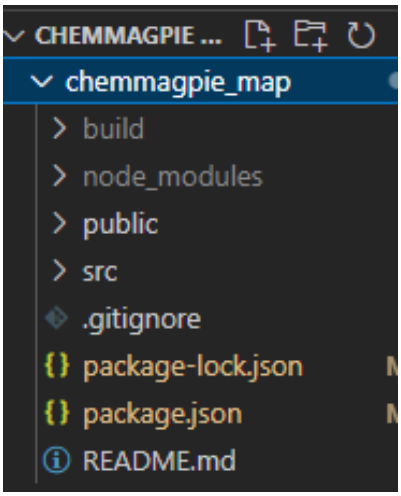

Рисунок 3.2 — Структура файлів та папок клієнтської частини веб-мапи фреймворку React js «chemmagpie_map»

Папка «build» містить скомпільовану версію проєкту, котра готова для розгортання на сервері, фактично дана папка є «обличчям» для відображення усього виконаного, шляхом кодування, проєкту. Для компіляції папки «build» використовували команду:

```
npm run build
```

В цій папці міститься фавікон, HTML файли, CSS файли, JavaScript файли.

Папка node_modules – це стандартна папка, яка містить усі залежності (пакети) нашого проекту Node.js. Папку встановлювали за допомогою пакетного менеджера, через команду:

```
npm install
```

Ця папка зазвичай містить бібліотеки, фреймворки та інші пакети, які використовуються в проекті. Вона може бути доволі об'ємною, оскільки містить велику кількість файлів, пов'язаних з кожним пакетом.

Папка «public» містить статичні ресурси, які будуть доступні безпосередньо в браузері без обробки або змін від React або його бібліотек.

Файл package-lock.json та package.json містять в собі загальні налаштування проекту, завантажені бібліотеки для роботи, певні залежності для запуску проекту.

Папка "src" є стандартною папкою для розміщення вихідного коду веб-додатку. Ця папка містить основний код нашого проєкту React, включаючи компоненти, стилі, зображення та інші файли, які ви створює автор для розробки проєкту. В контексті розробки саме src є найголовнішою папкою.

Для того, щоб розібрати принцип роботи веб-мапи клієнтської частини веб-сервісу розглянемо ємність папки «src», що зображена на рис. 3.3

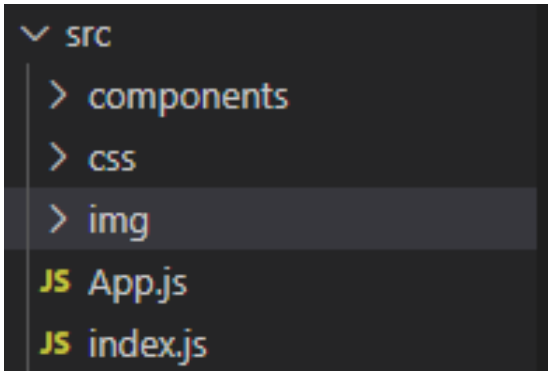

Рисунок 3.3 – Структура файлів та папок в «src» папці клієнтської частини веб-мапи сервісу

Папка «component» містить LeafletMap.js файл, котрий є основним при розробці веб-мапи.

«CSS» папка містить стилі котрі використовувалися при створені веб-мапи, зокрема стилі для оптимізації веб-елементів для мобільних пристроїв та ПК із застосуванням grid, flex макетів, а також запитів @media, наприклад запит утримування розміру шрифту на 18px для розмірів екранів по ширині до 1024:

```
@media screen and (max-width:1024px){
  body{
    font-size: 18px;
  }
}
```

Папка «img» використовує файли зображень, котрі висвітлюються для веб-мапи. В основному використання зображень було спрямоване на функціонал як кнопки керування та відображень міток на мапі.

App.js – це основний файл компонента, де визначається структура веб-мапи. Він містить основний код React для компонентів, логіки відображення, обробники подій та зв'язки з іншими компонентами використовується як основа написаного розробником коду це має наступний вигляд:

```
import React from 'react';

import LeafletMap from './components/LeafletMap'
class App extends React.Component {
   constructor(props){
      super(props)
      this.state = {
      }
   }
   render(){
      return (
      <div>
      <LeafletMap />
      </div>
   )
}
}
export default App;
```

Як можна побачити, на основі зображеного коду, присутній імпортований файл із папки «componenets», в якому зберігається файл, що відображає основну частину веб-мапи для відображення хімічних показників води.

index.js – це файл, що відповідає за завантаження нашого React застосунку в HTML-документ. Він ініціалізує нашу React частину веб-мапи додатку, трансформує це в DOM (Document Object Model) та встановлює точку входу для роботи веб-мапи:

```
import React from 'react';
import * as ReactDOMClient from "react-dom/client"
import App from "./App"
import './css/main.css'
const root = ReactDOMClient.createRoot(document.getElementById("root"));
root.render(<App />);
```

Стосовно того, як функціонує веб-мапа із хімічними показниками та точками відбору проб води, то це файл LeafletMap.js в папці «components». Даний файл містить у собі створений нами клас LeafletMap. Бібліотеки, котрі ми використали для написання коду в фреймворку React:

leaflet – це відкрита бібліотека для роботи з інтерактивними мапами. У нашому випадку, використовується react-leaflet, яка є обгорткою над бібліотекою Leaflet для використання її в додатках React. На основі даної бібліотеки, були використані наступні компоненти: MapContainer, TileLayer, Marker, Popup для роботи з мапою в React[24].

Icon з бібліотеки leaflet – використовується для створення та налаштування позначень, що використовуються на мапі.

Для використання веб-мапи Chemmagpie, ми визначилися із полями стану, котрі будуть присутні в класі LafletMap, на початку вони усі порожні чи мають значення false, в подальшому можуть використовуються для динамічного оновлення стану інформації нашої веб-мапи, наприклад:

```
this.state = {
        sampling_places: [],
        clickMarker: '',
        showLegend: false
}
```

Для відображення даних у веб-мапі та елементів в ній, було використано бібліотеку axios, котра дозволяє це реалізувати, параметри налаштувань мапи було використано в частині render коду React:

```
<MapContainer center={[50.440636, 30.462625]} zoom={6} zoomControl={false}>
    <TileLayer attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
      url="https://tile.openstreetmap.org/{z}/{x}/{y}.png"
      />
      <ZoomControl position="bottomright" className="zoom-control"/>
```

Для завантаження інформації з бази даних MongoDB у веб-мапу, було використано бібліотеку axios, базуючись на проксі сервері запущеної серверної частини нашого додатку, наприклад для колекції sampling_places використано наступний код:

```
fetchSamplingPlaces = async () => {
 try{
  const response = await axios.get('/api/sampling_places');
  const sampling_places = response.data;
  this.setState({ sampling_places: sampling_places }, () => {
   console.log(this.state.sampling_places);
  });
 }catch(err){
  console.log('Not coonect with database', err);
 }
}
```

Оскільки в нашій базі даних використовуються дата формату DD.MM.YYYY, то було прийнято рішення записувати та оформляти дані дати у форматі типу даних String. Щоб в подальшому програма здатна була зчитувати інформацію як дата, було створено функцію перетворення строкових даних в дату, котра зрозуміла машині, з метою подальшого користування математичною логікою:

```
 parseDate = (dateStr) => {
  const parts = dateStr.split('.');
  const day = parseInt(parts[0], 10);
  const month = parseInt(parts[1], 10) - 1;
  const year = parseInt(parts[2], 10);
  return new Date(year, month, day);
 };
```

Більшість користувачів нашого сервісу, можуть бути не хіміками-аналітиками, з метою відображення зрозумілих для них даних у формі кольорових індикаторів, було розроблено функцію, котра дозволяє порівнювати отримані із бази даних значення результатів аналізу води за певним хімічним показником із ГДК. ГДК дані були узяті із ДСанПіН 2.2.4-171-10 «Вода питна»[2]:

```
const MPC = {
 nitrate: 50,
 nitrite: 0.5,
 chloride: 250,
 phosphate: 3.5,
 sulphate: 250,
 TH: 7,
 COC: 5,
 fluoride: 0.7
}
```

Щоб відображувати лише останній аналіз результату певного хімічного показника води для веб-мапи за датою аналізу і одночасно розпізнавало, для якого саме показника було проведено аналіз, було використано булінівську операцію на прикладі аналізу нітрат-іону:

```
this.setState({dataNitrate: ''});
this.state.chemical_indexes.map(e => {if(e.name_place === place.name_place){
  if(e.chemical_index === 'NO3-' && (date_max_nitrate === '' || date_max_nitrate <
this.parseDate(e.date_analysis))) {
    this.setState({dataNitrate: e});
    date_max_nitrate = this.parseDate(e.date_analysis)
  }}
```

Для реалізації обирання системою порівнянь який колір відображати за результатом аналізу, було використано в частині візуалізації веб-сторінки React js наступну операцію із залученням тернарних операторів JavaScript:

```
{dataNitrate.result_chemical_index > MPC.nitrate ?
        <div className='redundantCPM'>{dataNitrate.result_chemical_index} mg/dm³</div>
       : dataNitrate.result_chemical_index <= MPC.nitrate ?
        <div className='normalCPM'>{dataNitrate.result_chemical_index} mg/dm³</div>
       : <div>-</div>
       }
       <div>{dataNitrate.date_analysis}</div>
```

Якщо поточне значення перевищує норми ГДК використовуються стиль червоного кольору, якщо результат зворотній – зелений стиль інформаційного поля. Інформація про структуру коду онлайн-мапи наведена в додатку А.

3.2.2 Адміністративна частина веб-сервісу Chemmagpie

Час від часу дані потрібно оновлювати, не будь-ким, а персоналом, котрий має обізнаність в цьому, якщо розробити окрему клієнтську частину для введення даних, проблема буде вирішена. Розроблений нами інтерфейс клієнтської адміністративної частини веб-сервісу має вигляд, що відображений на рис. 3.4:

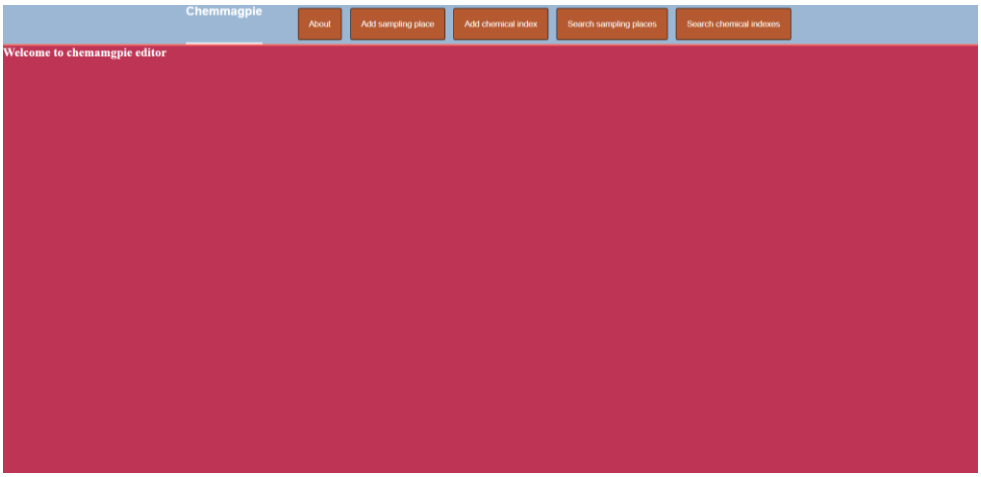

Рисунок 3.4 – Вступний інтерфейс адміністративної частини сервісу Chemmagpie

Із даного інтерфейсу можна побачити, що в нас є загальна частина та навігаційна. Загальна частина відображена у задньому фоні червоного кольору, відповідає за генерацію на сторінці основного контенту, котрий потрібен адміністративному веб-сервісу. Навігаційна частина позначена сіро-блакитним кольором, при натисканні на одну із коричневих кнопок відбувається перехід на інший функціонал адміністративної частини Chemmapie сервісу. Наприклад, при натисканні на елемент «Add sampling place», відбувається перехід до секції додання місця проби, що відображена на рисунку 3.5.

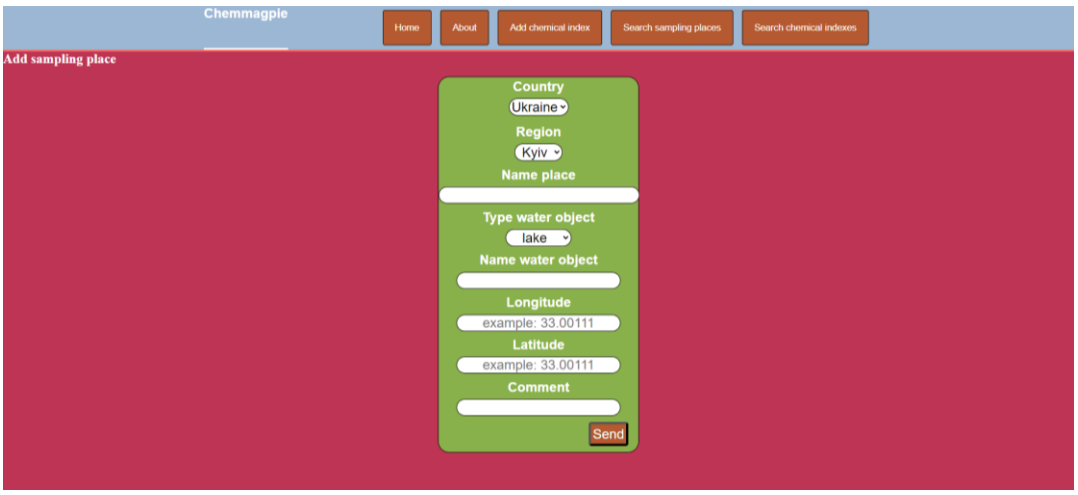

Рисунок 3.5 –інтерфейс додання даних про місце відбору проби води адміністративної частини сервісу Chemmagpie

Фреймворк, що буде підтримувати виконання логіки на клієнтській частині веб-сервісу, має загальну структуру, аналогічну до розробленої веб-мапи, що зображена на рис. 3.2, назва папки проекту має назву «manage_chemmagpie». Для того, щоб розібрати принцип роботи адміністративної частини веб-сервісу розглянемо ємність папки «src», що зображена на рис. 3.6:

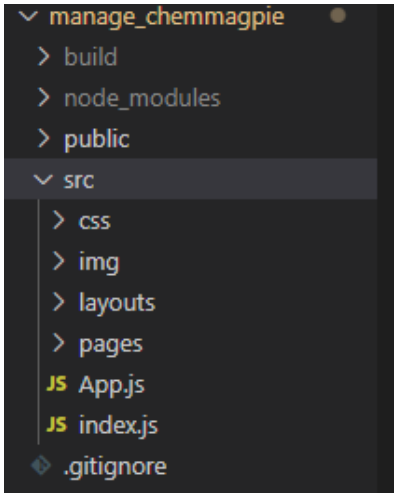

Рисунок 3.6 – Структура файлів та папок в «src» папці клієнтської адміністративної частини веб-сервісу Chemmagpie

Важливим для розробки нашого веб-сервісу є файл App.js та папки layouts та pages. Інші папки та файли були майже аналогічні до структури файлів та папок в «src» в підрозділі 3.2.1.

Папка layouts використовується для встановлення назначених нами кнопок для маршрутів URL. Папка містить файл RootLayout.js, в котрому прописаний блок навігації, щоб відображати кнопки управління перемикання на різні шляхи нашої адміністративної частини сервісу.

Файл App.js містить в собі наступну структуру коду:

```
import { NavLink, Outlet } from 'react-router-dom';
const RootLayout = () => {
  return (
    <div className='root-layout'>
      <header>
```

```
        <nav>
        <h1>Chemmagpie</h1>
        <div className='grid-link'>
          <NavLink to='/' className='nav-link'>Home</NavLink>
          <NavLink to='about' className='nav-link'>About</NavLink>
          <NavLink to='addSamplingPlace' className='nav-link'>Add sampling place</NavLink>
          <NavLink to='addChemicalIndex' className='nav-link'>Add chemical index</NavLink>
          <NavLink to='searchSamplingPlaces' className='nav-link'>Search sampling places</NavLink>
          <NavLink to='searchChemicalIndexes' className='nav-link'>Search chemical indexes</NavLink>
        </div>
        </nav>
      </header>
      <main>
        <Outlet />
      </main>
    </div>


  )
}

export default RootLayout;
```

Зазвичай React js проекту дозволяє відображати лише одну сторінку в додатку, при цьому фреймворк в динамічному стані здатний приховувати-відображати потрібні користувачу елементи сторінки. Базуючись на цьому, ми використали спеціальну бібліотеку react-router-dom, котра дозволила нам створити залежність відображення певної сторінки від введеного в URL шляху. Використали в дану бібліотеку в корінному файлі App.js, через тег Route:

```
import {BrowserRouter, Routes, Route, Link, NavLink, createRoutesFromElements, createBrowserRouter,
RouterProvider} from 'react-router-dom';
import RootLayout from './layouts/RootLayout';


import Home from './pages/Home';
import About from './pages/About';

import AddSamplingPlaces from './pages/AddSamplingPlaces';
import AddChemicalIndex from './pages/AddChemicalIndex';
import SearchSamplingPlaces from './pages/SearchSamplingPlaces';
import EditSamplingPlace from './pages/EditSamplingPlace';
import SearchChemicalIndexes from './pages/SearchChemicalIndexes';
import EditChemicalIndex from './pages/EditChemicalIndex';
```

```
const router = createBrowserRouter(
 createRoutesFromElements(
   <Route path="/" element={<RootLayout />}>
     <Route index element={<Home />}/>
     <Route path="addSamplingPlace" element={<AddSamplingPlaces />}/>
     <Route path="addChemicalIndex" element={<AddChemicalIndex/>} />
     <Route path="searchSamplingPlaces" element={<SearchSamplingPlaces/>} />
     <Route path="searchChemicalIndexes" element={<SearchChemicalIndexes/>} />
     <Route path="editSamplingPlace/:id" element={<EditSamplingPlace/>} />
     <Route path="editChemicalIndex/:id" element={<EditChemicalIndex/>} />
     <Route path='about' element={<About />}/>
   </Route>
 )
)

function App(){
 return(
   <RouterProvider router={router} />
 )
}

export default App;
```

Крім того, в наведеному вище коді, для створення кнопки динамічного керування маршрутами, було створено папку в App.js RootLayout на початку усіх маршрутів. Таким чином ми здатні відображати усі потрібні нам кнопки керування. В тегу Route є атрибути path – це назва нашого шляху в URL, element – назва вмісту файлу яке повинно відображатися під даним шляхом. Елементи інтерфейсу містяться в папці «pages».

Відповідно від кількості елементів в тегу Route файлу, було створено рівну кількість файлів із вмістом елементів котрі розроблені для інтерфейсу адміністративної частини веб-сервісу Chemmagpie, інформація про них відображена в табл. 3.1:

Таблиця 3.1 – опис елементів інтерфейсу адміністративної частини веб-сервісу Chemmagpie

Назва елементу	**Опис елементу**	**Шлях URL**
Home	Вступна сторінка	/
About	Додаткові дані(автор розробки)	/about
AddSamplingPlaces	Додання даних до бази даних стосовно місця відбору проби	/addSamplingPlace
AddChemicalIndex	Додання даних результату хімічного аналізу за певним хімічним показником	/addChemicalIndex
SearchSamplingPlaces	Пошук даних стосовно місця відбору проби води	/searchChemicalIndexes
EditSamplingPlace	Редагування даних стосовно місця відбору проби	/editSamplingPlace/:id
searchChemicalIndexes	Пошук даних результатів аналізу проби води	/searchChemicalIndexes
EditChemicalIndex	Редагування даних результату хімічного аналізу за певним хімічним показником	/editChemicalIndex/:id

Для кожного елементу інтерфейсу адміністративної частини веб-сервісу Chemmagpie операцію завантаження даних для веб-сторінки через серверну частину проекту, використовували бібліотеку axios. Для того, щоб теги select відображались в динамічному стані, наприклад при зміні країни з Польщі на Україну змінювалися під це регіони замість Warshawa був Kyiv, було створено спеціальні операції із JavaScript подібні до AddSamplingPlaces елементу:

```
const { places, selectedCountry} = this.state;
    const filteredPlaces = places.filter(place => place.country === selectedCountry);
    const uniqueCountries = [...new Set(places.map(place => place.country))];
```

Де places – масив об’єктів бази даних «places», selectedCountry – обрана країна через тег Select. Щоб це візуалізувати на сторінці, було використано цикли, тернарні оператори, що характерні для React js фреймворку:

```
        <h1>Country</h1>
        <select id="country" name="country" onChange={this.handleCountryChange}
defaultValue={this.state.selectedCountry}>
         {uniqueCountries.map(country => (
          <option key={country} value={country}>{country}</option>
         ))}
        </select>
         <form ref={this.formRef} onSubmit={this.handleSubmit}>
         <h1>Region</h1>
         <select id="region" name='region'>
            {filteredPlaces.map(place => (
               <option key={place._id} value={place.region}>{place.region}</option>
            ))}
         </select>
         {region && <div>{region}</div>}
```

Для адміністративного інтерфейсу з метою коректного додання чи зміни даних, було використано обробники помилок на клієнтській стороні, наприклад якщо не заповнене поле або невірна введена дата аналізу щоб дані не записувалися, а лише операція зупинялася та відображала певне записане повідомлення, як на рис. 3.7.

Рисунок 3.7 – реалізація обробника помилок на прикладі елементу AddSamplingPlaces адміністративного інтерфейсу Chemmagpie

Наш обробник помилок працює за рахунок принципу змінної з елементами, котру ми назвали errors, при умові знаходження помилки в цю змінну додаються елементи із текстом повідомлення про помилки. При закінчені знаходження помилок, якщо хоч однин елемент, в ході обробників присутній у змінній errors, то функція повертає errors замість виконання редагування запису в базу даних, при цьому повідомлення про помилки відображаються в render частині.

```
const { region, name_place, type_water_object, name_water_object, longitude, latitude, comment } =
event.target;
const errors = {};
if (!region.value) {
      errors.region = "fill in region field";
    }
```

Окрім такого обробника помилок, присутній обробник помилок, що пов’язує серверну частину із нашою клієнтською через операцію try-catch, якщо відбувається виявлення помилки на серверній стороні, наприклад дані не записалися через вимкнення серверної сторони, то операція зміни даних в базі даних також не відбувається. Код елементів інтерфейсу адміністративної частини веб-сервісу відображений в додатку Б. В результаті запуску

клієнтської частини в хостинг AWS EC2 та використання розширення для мобільних пристроїв через браузер Google Chrome інструментом Toggle Device Toolbar, було отримано успішний результат роботи нашої клієнтської частини на ПК, андроїдах та смартфонах.

3.2 Серверна частина

Серверна частина нашого веб-сервісу відповідає за реалізацію завантаження та обробку даних із колекцій бази даних MongoDB Atlas, а також REST API частини. Для реалізації серверної частини було використано Express js фреймворк JavaScript мови програмування[25,26]. Дана частина веб-сервісу Chemmagpie використовується в якості проксі серверу для відображення певних даних з бази даних.

Для початку роботи над проектом була використані команди:

```
npm init -y
npm install express
```

Далі були створені потрібні файли, спершу створений корінний файл з якого починається запуск усієї серверної частини, назвали його app.js:

```
async function startServer() {
  try {
    const connection = await connectToDatabase();
    if (!connection) {
      console.log("Unable to connect to MongoDB. Exiting...");
      process.exit(1);
    }
    await app.listen(PORT);
    console.log(`Server has been started ${PORT}`);
  } catch (err) {
    console.log(`Error starting server: ${err}`);
    process.exit(1);
  }
}
```

Далі створили моделі баз даних, котрі використовувалися в для обробки даних через клієнтські частини сервісу, використовуючи встановлену бібліотеку mongoose[27], наприклад для схеми колекції chemical_indexes код наступний:

```
const Chemical_Index_Schema = new mongoose.Schema({
  name_place: {
```

```
    type: String,
    required: true,
    ref: 'sampling_place',
  },
  chemical_index: {
    type: String,
    required: true,
  },
  result_chemical_index: {
    type: String,
    required: true,
  },
  date_analysis: {
    type: String,
    required: true,
  },
  comment: {
    type: String
  }
})
```

Аналогічно було розроблено і для колекцій places та sampling_places, що працюють від хмарної бази даних MongoDB Atlas. Щоб підключитися до нашої хмарної бази даних було використано наступний код в файлі app.js:

```
async function connectToDatabase() {
  try {
    const connection = await mongoose.connect(process.env.MONGO_DATABASE_URL, {
      useNewUrlParser: true,
      useUnifiedTopology: true
    });
    console.log("Database MongoDB connect successful");
    return connection;
  } catch (err) {
    console.log(`Something wrong with MongoDB ${err}`);
    return null;
  }
}
```

Далі була розроблена частина REST API(контролер), щоб керувати записами бази даних, вона створена в окремому файлі, при цьому реалізується вона через app.js наступним кодом:

```
//Routes
app.use('/', indexRouter);
```

Кожний маршрут в якості проксі серверу мав власний шлях в URL. Шляхи та наших методи HTTP протоколів, після реалізації в AWS EC2 це методи HTTPS протоколів, описані в табл. 3.2.

Таблиця 3.2 – опис маршрутів серверної частини веб-сервісу Chemmagpie

Назва шляху URL	**Тип методу HTTP/HTTP протоколу**	**Опис шляху**
/api/places	GET	Відображає усі дані колекції places
/api/sampling_places	GET	Відображає усі дані колекції sampling_places
/api/chemical-indexes	GET	Відображає усі дані колекції sampling_places
/api/edit-sampling-places/:id	GET	Відображає обрані дані за параметром _id колекції sampling_places
/api/edit-chemical-indexes/:id	GET	Відображає обрані дані за параметром _id колекції chemical_indexes
/api/add-sampling-place	POST	Зберігає дані в колекцію sampling_places

Продовження табл. 3.2

Назва шляху URL	**Тип методу HTTP/HTTP протоколу**	**Опис шляху**
/api/add-chemical-index	POST	Зберігає дані в колекцію chemical_indexes
/api/edit-sampling-places/:id/update	PUT	Змінює дані за певним _id в колекції sampling_places
/api/edit-chemical-indexes/:id/update	PUT	Змінює дані за певним _id в колекції chemical_indexes

Наші контролери містять в собі обробники помилок на стороні серверу, це необхідно для збільшення рівня захисту при зміні даних в базу даних. Наприклад для обробника помилок при доданні даних в колекцію sampling_places було використано наступний код:

```
router.post('/api/add-sampling-place', async (req, res) => {
  try {
    // Отримання даних з форми
    const { region, name_place, type_water_object, name_water_object, longitude, latitude, comment }
= req.body;
    const errors = [];


     if(!region || !name_place || !type_water_object || !name_water_object || !longitude || !latitude){
       errors.push('Require always field except country and commentar must be field');
     }

     const SamplingplacesCount = await Sampling_Place.find({
       $or: [
         { longitude: longitude, latitude: latitude},
         { name_place: name_place }
       ]
     }).count();
```

```
    if(SamplingplacesCount > 0){
      errors.push('Detected duplicate data!');
    }

    const regexLongitude= /^(-)?(([0-8]?[0-9])(\.\d{1,6})?)$|90$/
    const regexLatitude= /^(-)?((1?[0-7]?[0-9])(\.\d{1,6})?)$|180$/

    if(!regexLongitude.test(longitude) || !regexLatitude.test(latitude)){
      errors.push({ status: 400, message: `Incorrect enter data coordinate ${longitude} or ${latitude}` });
    }


    if (errors.length > 0) {
      return res.status(400).json({ errors });
    }


   // Створення нового зразка місця
   const newSamplingPlace = new Sampling_Place({
    region,
    name_place,
    type_water_object,
    name_water_object,
    longitude,
    latitude,
    comment
   });

   // Збереження зразка місця до бази даних
   const savedSamplingPlace = await newSamplingPlace.save();

   // Відправка відповіді про успішне збереження
   res.status(200).json({ success: true, message: 'Sampling place saved successfully!' });
  } catch (error) {
   // Обробка помилок
   console.error('Error saving sampling place:', error);
   res.status(500).json({ success: false, message: 'Failed to save sampling place.', error: error });
  }
});
```

Даний принцип роботи подібний до обробника помилок в клієнтській частині описаний в підрозділі 3.1, оскільки використовується змінна errors, в неї додається певний елемент помилки, якщо елементів більше нуля то дані не зберігаються, а також виконується функція try-catch в основному для запобігання та швидкого виправлення проблем із підключенням бази даних MongoDB Atlas із нашим проєктом[28–30].

3.3 Тестування веб-сервісу

Оскільки даний веб-сервіс розрахований на користувачів глобальної мережі Internet, було визначено проводити тести в клієнтськиз частинах сервісу в формі інтструменту Selenium IDE.

Проведення тестування з використанням Selenium IDE є важливим для клієнтської частини:

1) Автоматизація тестів: Selenium IDE надає можливість записувати та відтворювати тести для веб-додатків. Це дозволяє автоматизувати тестовий процес, зменшуючи кількість ручних операцій і забезпечуючи більш ефективне та консистентне тестування клієнтської частини.

2) Швидкість та ефективність: автоматизоване тестування за допомогою Selenium IDE може швидко виконувати однотипні операції та перевіряти різні сценарії взаємодії з клієнтською частиною. Дозволяє виявляти проблеми швидше та здійснювати тестування при змінах у коді.

3) Перевірка функціональності: Selenium IDE дозволяє перевіряти функціональність клієнтської частини, виконуючи різні дії, такі як натискання кнопок, заповнення форм, наведення курсору тощо. Дозволяє переконатися, що користувацький інтерфейс працює правильно та відповідає очікуванням.

4) Визначення регресійних помилок: при внесенні змін у клієнтську частину додатка можуть виникати нові помилки або порушення функціональності. Selenium IDE дозволяє створювати та виконувати тести регресії, які перевіряють, чи не вплинули зміни на існуючий функціонал.

5) Спрощення процесу тестування: Selenium IDE надає інтерактивне середовище для створення та редагування тестів, що полегшує процес тестування, навіть для тих, хто не має глибокого досвіду у програмуванні.

6) За допомогою цього розробник здатний тестувати власні веб-додатки при увімкнених портах на браузерах Firefox, Google Chrome, Microsoft Edge, Safari.

7) Сам код даних тестів зберігається у форматі side, щоб у майбутньому можна було скористатися даними тестами в розширені Selenium IDE.

Виконаних результатів тестування клієнтських частин сервісу відображені в додатку В:

Не менш важливим є тестування коду серверної частини. Основні причини тестування веб-сервісів на серверній стороні:

1) Надійність: серверна частина веб-додатку відповідає за обробку запитів від клієнтської частини і повернення відповідей. Дозволяє впевнитися, що серверна частина працює надійно, без збоїв та помилок.
2) Безпека: захист даних та серверної інфраструктури важливий для запобігання несанкціонованому доступу та атакам. Тестування може допомогти виявити слабкі місця в безпеці та уникнути можливих порушень безпеки.
3) Швидкодія: ефективна робота серверної частини допомагає забезпечити високу швидкодію веб-додатку. Дозволяє виявляти та виправляти проблеми, які можуть впливати на продуктивність.
4) Сумісність: залежно від того, як розроблена клієнтська частина (наприклад, веб-браузер, мобільний додаток), сервер повинен надавати дані та послуги у форматі, зрозумілому для клієнта. Тестування дозволяє переконатися, що серверна частина взаємодіє з різними клієнтами належним чином.
5) Масштабованість: підвищення навантаження може призвести до проблем з масштабованістю серверної частини. Може допомогти виявити, як технологія веде себе при великому обсязі запитів та чи виконується вона ефективно.

Таким чином, тестування серверної частини веб-додатку є важливим етапом розробки, спрямованим на забезпечення якості та ефективності системи в цілому.

Для тестування серверної частини додатку, в нашому випадку це Express js, ми використали тестування через програму Postman – це програма для тестування та розробки API (інтерфейсів програмування застосунків). Для серверної частини веб-додатків, особливо тих, що використовують REST. Оскільки під REST є певні маршрути в URL серверної частини через які відбувається запис, редагування чи візуалізація даних.

Для веб-сервісу «Chemamgpie» було протестовано серверну частину коду фреймворку express.js фреймворку файлу routes.js, в якому зберігаються основні маршрути URL на такі методи: GET, POST та PUT. Перед тим як тестувати наші URL маршрути, було запущено основний URL адреса htttp://localhost:3000 щоб це можна було виконувати в обмеженому режимі – на конкретному обладнанні.

Загалом по відношенню до контролера проекту Chemmagpie було протестовано наступне:

GET:

1) відображення даних колекції chemical_indexes що відповідає за результати хімічного аналізу поверхневої води;
2) відображення даних колекції sampling_places, що відповідає за координати відбору проби для хімічного аналізу поверхневої води;
3) відображення даних колекції places, що відповідає за дані країн регіонів/областей бази відбору проби.

Для POST методу:

1) перевірка на обробку помилок додання даних;
2) перевірка на додання даних в колекції chemical_indexes та sampling_places.

Для PUT методу:

1) перевірка на обробки помилок редагування даних(дати, на числовий/строковий формат, пусті поля даних);
2) перевірка на редагування даних в колекціях chemical_indexes та sampling_places.

Результати тестування серверної частини веб-сервісу Chemmagpie наведені в додатку Г.

3.4 Оцінка якості роботи технології веб-сервісу за динамікою зміни хіміко-екологічного стану поверхневих вод

Для тестування значущості веб-сервісу для користувачів веб-застосунків, ми використали метод опитування серед різної категорії людей по нашому веб-сервісу. Для цього було створено хостинг із нашим розробленим проєктом через сервіс AWS EC2. Після запуску проєкту в хостинг, ми створили URL-посилання на запущений веб-застосунок різним категоріям людей, вони ознайомилися із нашим сервісом та пройшли опитування за іншим посиланням, що представляє Google Form із переліком складених нами питань. Опитування було анонімне, було залучено до цього дослідження 11 респондентів.

В нашій гугл формі опитування складалося із наступних пунктів:

1. Вкажіть ваш віковий інтервал
2. Ваша діяльність пов'язана із природничим науками?
3. Чи зрозумілий принцип роботи веб-застосунку?
4. На вашу думку застосунок здатний впливати на безпеку поверхневих вод?
5. Застосунок здатний забезпечити вплив на збереження здоров'я?
6. Ваші пропозиції до даного застосунку за бажанням(опціонально)

На основі отриманих результатів, була проведена статична обробка результатів опитування, котрі відображені в формі діаграм. Обробка статистичних результатів для побудови потрібних нам діаграм була виконана

через мову Python із використанням бібліотек numpy, pandas, seaborn та matplotlib в середовищі Jupiter. Обрання саме таким способом побудови графіків зумовлено тим, що дані бібліотеки мають великий перелік конфігурацій, котрі можна налаштувати під різні потреби.

Згідно із результатів рис. 3.8 були залучені люди віком від 30-45 та 18-30 років. Таким чином, в опитувані були залучені, відповідно користувалися додатком, представники поколінь Z та Y.

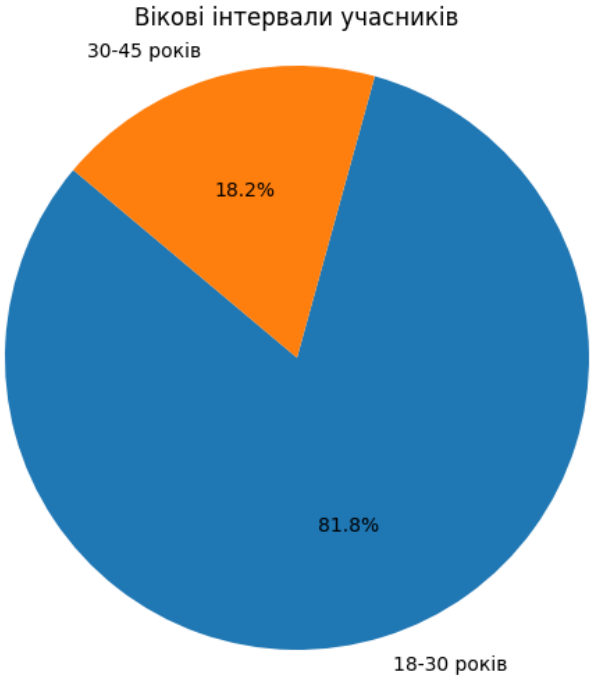

Рисунок 3.8 – Діаграма результатів відповідей на перше питання опитування по веб-сервісу Chemmagpie

За результатами опитування другого питання, що відображене на рис. 3.9, більша частка респондентів не має напрям діяльності із природничими науками, що дозволило нам отримати результати по веб-сервісу для цільової аудиторії, котрою є звичайні користувачі, при цьому люди із напрямом діяльності теж важливі, з огляду на те, що додаток пов'язаний із природничим дисциплінами, їхня думка по відношенню до цього є важливою в рамках

коректної розробки інтерфейсу сервісу із відображенням даних хімічних показників води в онлайн-мапі.

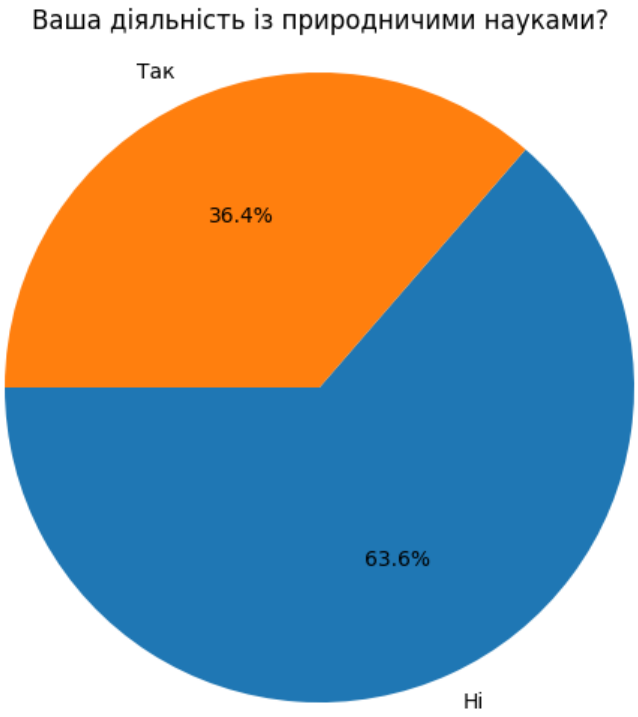

Рисунок 3.9 –Діаграма результатів відповідей на друге питання опитування по веб-сервісу Chemmagpie

За результатами 3 питання, результат його відображений на рис. 3.10, що майже усі респонденти розуміють принцип роботи веб-застосунку, 9,1% склав негативний результат. Щоб перевірити чи не пов’язаний негативний результат із незрозумілістю роботи старшим поколінням, ми використали групування за віком та на основі цього розробили дві порівнюючи кругові діаграми, що відображені на рис. 3.11. Результати показали, що не було зрозуміло лише певним користувачам, котрі мали вік від 18 до 30 років. Це свідчить про те, що використання нашої технології для поколінь Z та Y є в основному зрозумілим.

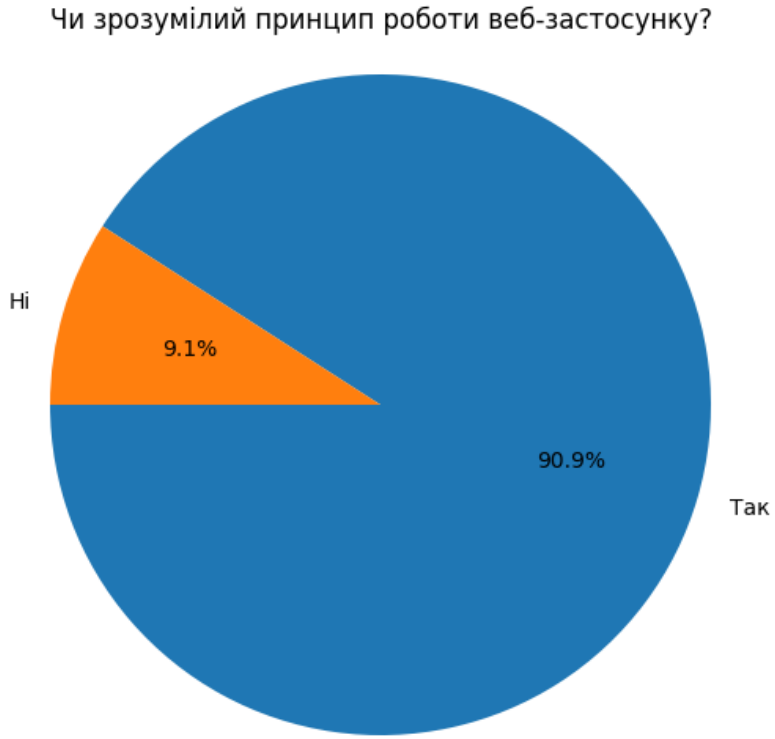

Рисунок 3.10 – Діаграма результатів відповідей на третє питання опитування по веб-сервісу Chemmagpie

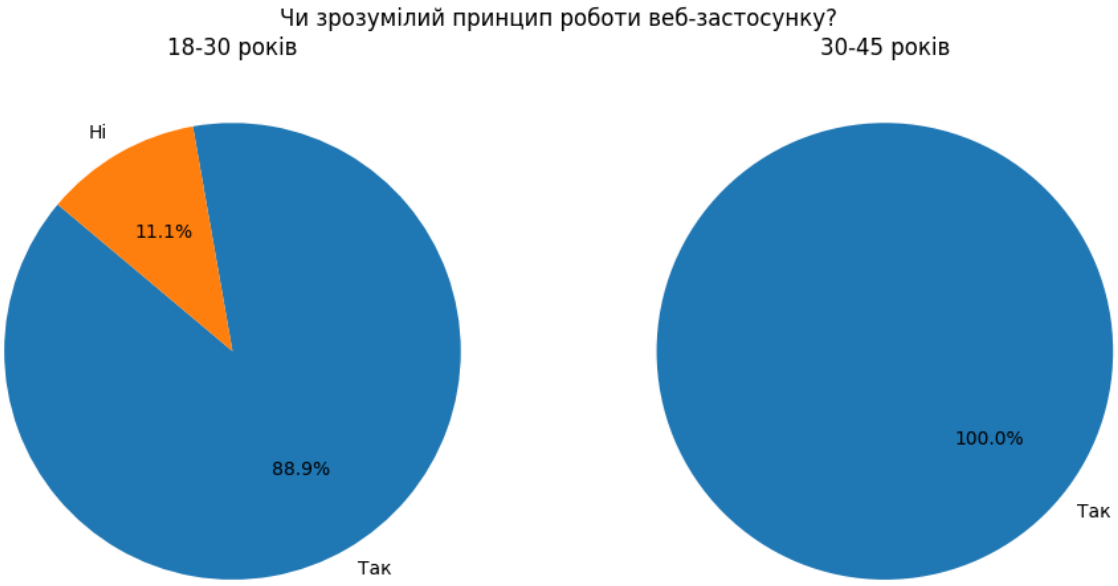

Рисунок 3.11 – Порівняльні діаграми результатів відповідей на третє питання опитування по веб-сервісу Chemmagpie, попередньо відфільтрованих за віком

За результатами діаграми відповідей на четверте питання, котре полягає в тому, що даний додаток є доцільний для екологічної безпеки поверхневих вод, майже 3 з 4 відповіли, що так. Це значить, що розроблена нами технологія, здатна прямими чи опосередкованим чином впливати на покращення безпеки поверхневих вод по відношенню до антропогенних факторів, джерелами котрих є людське суспільство. Результат відображений у формі діаграми на рис. 3.12.

Рисунок 3.12 –Діаграма результатів відповідей на четверте питання опитування по веб-сервісу Chemmagpie

За результатами діаграми відповідей на п’яте питання, котре полягає в тому, що даний додаток здатний вплинути на збереження здоров’я, 90,9% серед усіх респондентів відповіли, що так. Таким чином для більшості, розроблена нами технологія здатна надавати позитивний вплив на живі організми, шляхом надання доступу до інформації про хімічний стан поверхневих вод. Результати зображені в діаграмі на рис. 3.13.

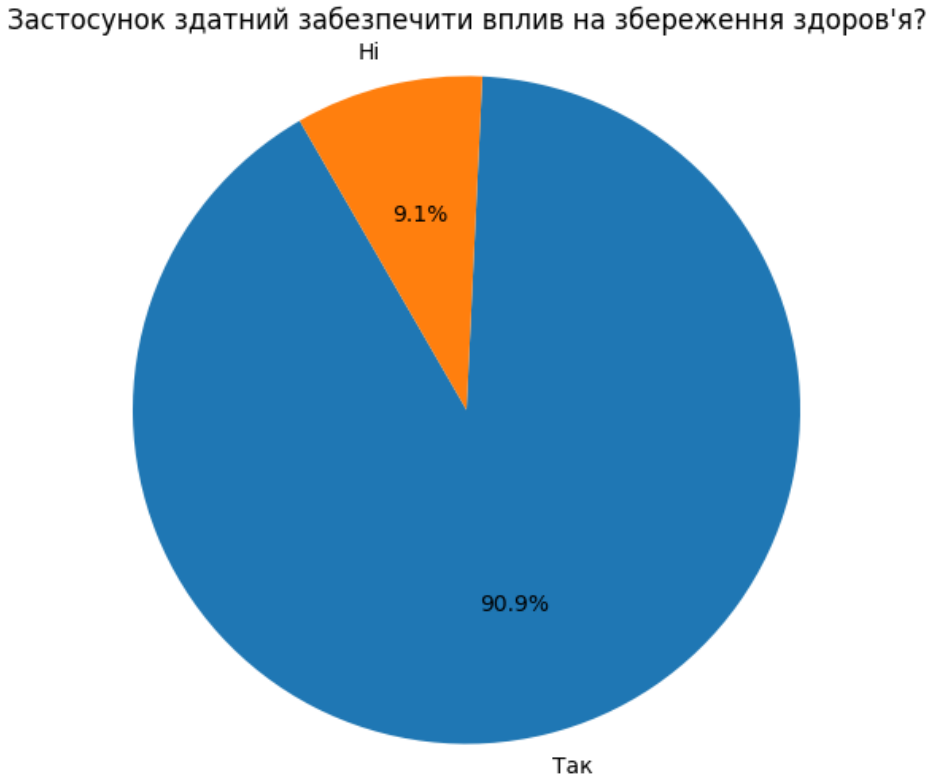

Рисунок 3.13 –Діаграма результатів відповідей на четверте питання опитування по веб-сервісу Chemmagpie

Стосовно шостого питання, котре є опціональне, надати побажання та рекомендації по нашій технології, переважно це позитивні відгуку, а також є рекомендація додання інформації результатів аналізів по хімічним показникам за минулі терміни. Даний результат опитування зображений на рис. 3.14.

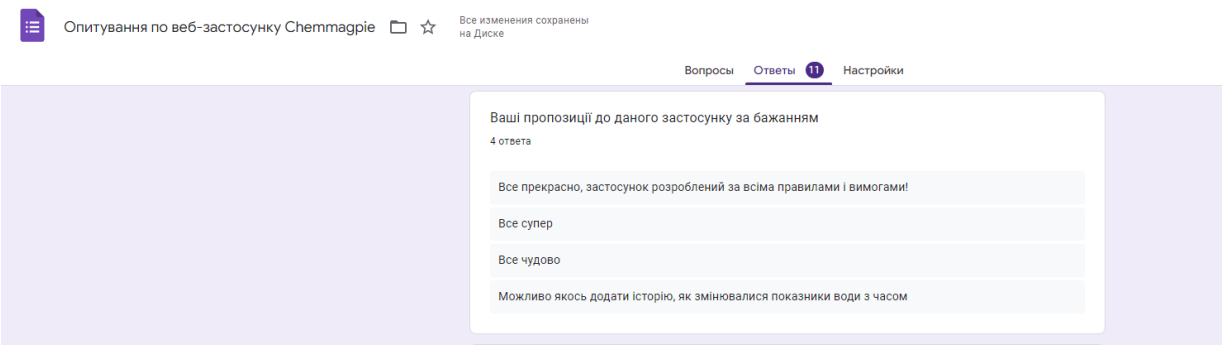

Рисунок 3.14 – результати з опитування на шосте питання по веб-сервісу Chemmagpie

ВИСНОВКИ

1. За результатами огляду джерел інформації за темою дослідження встановлено, що розробка інформаційної технології моніторингу за хіміко-екологічним станом поверхневих вод є актуальною. Сучасні розробки відображення певної інформації у формі веб-мапи користуються популярністю серед веб-застосунків.

2. З оглядом на специфіку розробки інформаційної технології, була узята нами нереляційна модель бази даних, в котрій було створено 3 колекції, котрих було достатньо для створення інформаційної складової для технології стеження динаміки зміни хіміко-екологічного стану поверхневих вод. Мова програмування JavaScript дозволяє розробити серверну та клієнтську частини нашої інформаційної технології.

3. Використання таких програмних засобів як Express js, React js, а також бази даних MongoDB дозволило розробити інформаційну технологію моніторингу за динамікою зміни хіміко-екологічно стану поверхневих вод. Клієнтська та серверна сторони веб-сервісу були перевірені шляхом тестування в локальному середовищі, а також в умові запуску хостингу через технологію AWS EC2.

4. За результатами проведеного дослідження методом опитування серед респондентів, а також тестування серверної та клієнтської частини веб-сервісу встановлено, що даний сервіс виконує заплановані нами функції та є актуальним серед користувачів поколінь Z та Y, вони розуміють як працює розроблена нами інформаційна технологія. Переважна більшість респондентів підтвердила, що дана технологія є корисна для поліпшення здоров’я людини та екологічної безпеки поверхневих вод. Дану інформаційну технологію можна використовувати для надання суспільству інформації про актуальний стан хімічних показників поверхневих вод в режимі онлайн.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Доповідь про стан навколишнього середовища у Сумській області у 2019 році. Суми. Сумська ОДА. 2020. 234 с. Sumy, 2020. 234 p.
2. Про затвердження Державних санітарних норм та правил “Гігієнічні вимоги до води питної, призначеної для споживання людиною” (ДСанПіН 2.2.4-171-10): Наказ від Міністерства Охорони Здоров’я України від 01 липня 2010 р. № 341: станом на 18.02.2022 р. URL: https://zakon.rada.gov.ua/laws/show/z0452-10#Tex (Дата звернення: 05.05.2020).
3. Project Chemmagpie startup-center Sumy State University [Electronic resource]. URL: https://startup.sumdu.edu.ua/ssu_portfolio/chemmagpie (accessed: 15.11.2023).
4. ТРОФАЇЛА Н. ДИДАКТИЧНА ГРА ЯК ЗАСІБ ОЗНАЙОМЛЕННЯ ДІТЕЙ ІЗ ПРЕДМЕТНИМ ДОВКІЛЛЯМ // Acta Paedagogica Volynienses. 2021. № 4.
5. Чичинська О.В. Психологічні особливості представників z покоління // Актуальні проблеми психології в закладах освіти. 2019. Vol. 9.
6. Liveuamap: About liveuamap.com. Retrieved 30 March 2022.
7. Skinner, Barnaby (2022-03-23). “«Ich verfolge in Echtzeit, wie mein Apartment zerstört wird» – wie zwei Ukrainer minuziös den Krieg in ihrer Heimat aufzeichnen” [’I’m following live coverage of the destruction of my apartment’ - how two Ukrainians are recording the details of the war in their homeland]. Neue Zürcher Zeitung (in German). Archived from the original on 2022-06-08. Retrieved 2022-03-26.
8. Інтерактивна мапа територій, які потенційно можуть бути забруднені вибухонебезпечними предметами . [Електронний ресурс] – Режим доступу до ресурсу: https://mine.dsns.gov.ua/.
9. Google Maps Metrics and Infographics - Google Maps for iPhone". sites.google.com. Archived from the original on March 21, 2022. Retrieved April 1, 2021.
10. Mehta H., Kanani P., Lande P. Google Maps // Int J Comput Appl. 2019. Vol. 178, № 8.

11. SaveEcoBot [Electronic resource]. URL: https://www.saveecobot.com/features/pollution-complaints/ (accessed: 06.11.2023).

12. SPA (Single-page application) [Electronic resource]. URL: https://developer.mozilla.org/en-US/docs/Glossary/SPA (accessed: 06.11.2023).

13. SEO(Search Engine Optimization) [Electronic resource]. URL: https://developer.mozilla.org/en-US/docs/Glossary/SEO (accessed: 06.11.2023).

14. Nurwanto N. Penerapan Progressive Web Application (PWA) pada E-Commerce // Techno.Com. 2019. Vol. 18, № 3.

15. Дмитришин Б.В. Сутність концепції сховища даних. Сучасні проблеми економічної теорії, маркетингу та моделювання соціально-економічних систем: матеріали V Всеукраїнської наук.-практ. інтернет конференції, м. Кропивницький, 28-29 квітня 2021 року / Центральноукраїнський національний технічний університет, Кропивницький, 2021. С. 107-110.

16. Khan W. et al. SQL and NoSQL Database Software Architecture Performance Analysis and Assessments—A Systematic Literature Review // Big Data and Cognitive Computing. 2023. Vol. 7, № 2.

17. Techopedia. What is a Web-Based Application? - Definition from Techopedia // Techopedia. 2021.

18. Varinia Azkarin, Rangga Gelar Guntara, Oding Herdiana. Development of a REST API for Human Resource Information System for Employee Referral Management Domain Using the Express JS Framework and Node.js // Journal of Scientific Research, Education, and Technology (JSRET). 2023. Vol. 2, № 3.

19. Hafez A.I. et al. Client Applications and Server-Side Docker for Management of RNASeq and/or VariantSeq Workflows and Pipelines of the GPRO Suite // Genes (Basel). 2023. Vol. 14, № 2.

20. Kulawiak M. Client-side versus server-side geographic data processing performance comparison: Data and code // Data Brief. 2019. Vol. 26.

21. Source F.O. React JS // Meta Platforms. 2022.

22. What is IDEF - Definition, Methods, and Benefits [Electronic resource]. URL: https://www.edrawsoft.com/what-is-idef.html#:~:text=IDEF%20(Integrated%20Definition)%20is%20a,oriented

%20analysis%2C%20and%20knowledge%20acquisition. (accessed: 13.11.2023).

23. Wahyu Setia Bintara. VISIO // Dianisa.Com. 2020.

24. React Leaflet [Electronic resource]. URL: https://react-leaflet.js.org/ (accessed: 16.11.2023).

25. Mulana L. et al. Analisis Perbandingan Kinerja Framework Codeigniter Dengan Express.Js Pada Server RESTful Api // Jurnal Ilmiah Wahana Pendidikan. 2022. Vol. 8, № 16.

26. Adam Huda Nugraha. APLIKASI MONITORING DAN PELAPORAN KASUS COVID 19 DALAM LINGKUP RT BERBASIS WEBSITE MENGGUNAKAN EXPRESS JS & MONGODB // Jurnal Ilmiah Teknik. 2022. Vol. 1, № 2.

27. Raju S., Soundararajan S., Loganathan V. MERN Stack Web Application // R.S.C.B. 2021. Vol. 25.

28. Krishna V.V., Gopinath G. Test Automation of Web Application Login Page by Using Selenium Ide in a Web Browser // Webology. 2021. Vol. 18, № Special Issue.

29. Jaya T.S. Selenium IDE // Tutorials Point (I) Pvt. Ltd. 2019.

30. Leotta M., Molinari A., Ricca F. Assessor: a PO-Based WebDriver Test Suites Generator from Selenium IDE Recordings // Proceedings - 2022 IEEE 15th International Conference on Software Testing, Verification and Validation, ICST 2022. 2022.

ДОДАТКИ

Додаток А. Лістинг інтерфейсу мапи веб-сервісу за моніторингом хіміко-екологічного стану поверхневих вод

```
import React from 'react';
import "leaflet/dist/leaflet.css";
import { MapContainer, TileLayer, Marker, Popup, ZoomControl } from 'react-leaflet';
import { Icon } from "leaflet";
import axios from 'axios'

//Images
import downChevronImage from '../img/down-chevron.png';
import placeholder from '../img/placeholder.png';
import close from '../img/close.png'

const MPC = {
  nitrate: 50,
  nitrite: 0.5,
  chloride: 250,
  phosphate: 3.5,
  sulphate: 250,
  TH: 7,
  COC: 5,
  fluoride: 0.7
}

class LeafletMap extends React.Component {
  constructor(props){
    super(props);
    this.state = {
      sampling_places: [],
      chemical_indexes: [],
      clickMarker: '',
      showLegend: false,

      dataNitrate: '',
      dataNitrite: '',
      dataChloride: '',
      dataPhosphate: '',
      dataSulphate: '',
      dataTH: '',
      dataCOC: '',
      dataFluoride: ''
    }
  }

  parseDate = (dateStr) => {
    const parts = dateStr.split('.');
    const day = parseInt(parts[0], 10);
    const month = parseInt(parts[1], 10) - 1; // Місяці в JavaScript починаються з 0 (січень - 0, лютий - 1 і т.д.)
```

```
    const year = parseInt(parts[2], 10);
    return new Date(year, month, day);
  };

customIcon = new Icon(
  {
    iconUrl: require('../img/placeholder.png'),
    iconSize: [38, 38]
  }
);


componentDidMount(){
  this.fetchSamplingPlaces();
  this.fetchChemicalIndexes();
}


fetchSamplingPlaces = async () => {
  try{
    const response = await axios.get('/api/sampling_places');
    const sampling_places = response.data;
    this.setState({ sampling_places: sampling_places }, () => {
      console.log(this.state.sampling_places); // Викликатимеся після успішного оновлення стану
    });
  }catch(err){
    console.log('Not coonect with database', err);
  }
}


fetchChemicalIndexes = async () => {
  try{
    const response = await axios.get('/api/chemical-indexes');
    const chemical_indexes = response.data;
    this.setState({chemical_indexes: chemical_indexes}, () => {
      console.log(this.state.chemical_indexes);
    })
  }catch(err){
    console.log('Not connect with database', err)
  }
}


changeChemicalIndexes = (place) => {
  this.setState({dataNitrate: '', dataNitrite: '', dataChloride: '', dataPhosphate: '', dataSulphate: '',
dataCOC: '', dataTH: '', dataFluoride: ''});
  let date_max_nitrate = '';
  let date_max_nitrite = '';
  let date_max_chloride = '';
  let date_max_phosphate = '';
  let date_max_sulphate = '';
```

```
  let date_max_COC = '';
  let date_max_TH = '';
  let date_max_fluoride = '';
  this.state.chemical_indexes.map(e => {
    if(e.name_place === place.name_place){
    if(e.chemical_index === 'NO3-' && (date_max_nitrate === '' || date_max_nitrate <
this.parseDate(e.date_analysis))) {
      this.setState({dataNitrate: e});
      date_max_nitrate = this.parseDate(e.date_analysis)
    }
    else if(e.chemical_index === 'NO2-' && (date_max_nitrite === '' || date_max_nitrite <
this.parseDate(e.date_analysis))){
      this.setState({dataNitrite: e});
      date_max_nitrite = this.parseDate(e.date_analysis);
    }
    else if(e.chemical_index === 'Cl-' && (date_max_chloride === '' || date_max_chloride <
this.parseDate(e.date_analysis))){
      this.setState({dataChloride: e});
      date_max_chloride = this.parseDate(e.date_analysis);
    }
    else if(e.chemical_index === 'PO43-' && (date_max_phosphate === '' || date_max_phosphate <
this.parseDate(e.date_analysis))){
      this.setState({dataPhosphate: e});
      date_max_phosphate = this.parseDate(e.date_analysis);
    }
    else if(e.chemical_index === 'SO42-' && (date_max_COC === '' || date_max_sulphate <
this.parseDate(e.date_analysis))){
      this.setState({dataSulphate: e});
      date_max_sulphate = this.parseDate(e.date_analysis);
    }
    else if(e.chemical_index === 'COC' && (date_max_sulphate === '' || date_max_COC <
this.parseDate(e.date_analysis))){
      this.setState({dataCOC: e});
      date_max_COC = this.parseDate(e.date_analysis);
    }
    else if(e.chemical_index === 'TH' && (date_max_TH === '' || date_max_TH <
this.parseDate(e.date_analysis))){
      this.setState({dataTH: e});
      date_max_TH = this.parseDate(e.date_analysis);
    }
    else if(e.chemical_index === 'F-' && (date_max_fluoride === '' || date_max_TH <
this.parseDate(e.date_analysis))){
      this.setState({dataFluoride: e});
      date_max_fluoride = this.parseDate(e.date_analysis);
    }
    }
  })
}
handleLegend = () => {
  if(this.state.showLegend === false) this.setState({showLegend: true})
  else this.setState({showLegend: false})
}
```

Додаток Б Лістинг адміністративної частини веб-сервісу за моніторингом хіміко-екологічного стану поверхневих вод

```
import React from 'react'
import axios from 'axios'
```

Лістинг AddSamplingPlaces

```
class AddSamplingPlaces extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      places: [],
      sampling_places: [],
      selectedCountry: "Ukraine",
      successMessage: "",
      errorMessages: {},
  }
  //for reset
  this.formRef = React.createRef();
 }

  componentDidMount() {
    this.fetchPlaces();
    this.fetchSamplingPlaces();
  }


  fetchPlaces = async () => {
    try {
      const response = await axios.get('/api/places');
      const places = response.data;
      this.setState({ places });
    } catch (err) {
      console.log('Error fetching "places":', err);
    }
  }

  fetchSamplingPlaces = async () => {
   try {
    const response = await axios.get('/api/sampling_places');
    const sampling_places = response.data;
    this.setState({ sampling_places });
   } catch (err) {
    console.log('Error fetching "sampling_places":', err);
   }
  }


  handleCountryChange = (event) => {
    const selectedCountry = event.target.value;
```

```
    this.setState({ selectedCountry });
  }

  handleSubmit = async (event) => {
    event.preventDefault();

    // Перевірка на наявність пропущених рядків
    const { region, name_place, type_water_object, name_water_object, longitude, latitude, comment }
= event.target;
    const { sampling_places } = this.state;
    const errors = {};


    if (!region.value) {
      errors.region = "fill in region field";
    }

    if(!name_place.value){
      errors.name_place = "fill in name place field";
    } else if(name_place.value){
      const existingPlaceWithName = sampling_places.find(place => place.name_place ===
name_place.value);
      if (existingPlaceWithName) {
        errors.name_place = "This name place is already taken"
      }
    }


    if(!type_water_object.value){
      errors.type_water_object = "fill in type water object field";
    }
    if(!name_water_object.value){
      errors.name_water_object = "fill in name water object field";
    }

    const regexLongitude = /^(-)?(([0-8]?[0-9])(\.\d{1,6})?)$|90$/
    const regexLatitude = /^(-)?((1?[0-7]?[0-9])(\.\d{1,6})?)$|180$/

    if(!longitude.value){
      errors.longitude = "fill coordinate x field";
    }else if(!regexLongitude.test(longitude.value)){
      errors.longitude = "error format, need form -90 to 90, characters after dote 6";
    }

    if(!latitude.value){
      errors.latitude ="fill coordinate y field";
    }else if(!regexLatitude.test(latitude.value)){
      errors.latitude = "error format, need form -180 to 180, characters after dote 6";
    }

    const existingPlaceWithCoordinates = sampling_places.find(place => place.longitude ===
longitude.value && place.latitude === latitude.value);
```

```
    if (existingPlaceWithCoordinates) {
      errors.coordinates_match =  "These coordinates are already taken";
    }

    if (Object.keys(errors).length > 0) {
      // Відобразити повідомлення про помилки
      this.setState({ successMessage: "", errorMessages: errors });
      return;
    }

  try {
    const response = await axios.post('/api/add-sampling-place', {
      region: region.value,
      name_place: name_place.value,
      type_water_object: type_water_object.value,
      name_water_object: name_water_object.value,
      longitude: longitude.value,
      latitude: latitude.value,
      comment: comment.value
    });

    // Відобразити повідомлення про успішне збереження
    this.setState({ successMessage: response.data.message, errorMessages: {} });
    // Reset the form
    this.formRef.current.reset();
  } catch (error) {
    // Відобразити повідомлення про помилку
    this.setState({ successMessage: "", errorMessages: { general: "Failed to save sampling place." } });
  }
}
```

Лістинг AddChemicalIndex

```
function EditChemicalIndex(){
    const navigate = useNavigate();
    const {id} = useParams();

    const [places, setPlaces] = useState([]);
    const [sampling_places, setSamplingPlaces] = useState([]);
    const [chemical_indexes, setChemicalIndexes] = useState([]);
    //Для виключення за _id у випадку редагування збоїв перевірки унікальності за тим же
_id(name_place, date_analysis, latitude)
    const [excludeChemicalIndexes, setExcludeChemicalIndexes] = useState([]);
    const [chemicalIndex, setChemicalIndex] = useState({
        name_place: '',
        chemical_index: '',
        result_chemical_index: '',
        date_analysis: '',
        comment: ''
    });
```

```
//select
const [selectCountry, setSelectCountry] = useState('');
const [selectRegion, setSelectRegion] = useState('');
const [selectNamePlace, setSelectNamePlace] = useState('');

const [selectChemicalIndex, setSelectChemicalIndex] = useState({});

//unique
const [uniqCountries, setUniqCountries] = useState([]);
const [uniqRegions, setUniqRegions] = useState([]);
const [uniqNamePlace, setUniqNamePlace] = useState([]);

const [errors, setErrors] = useState([]);

const indicators = [{
  name: 'nitrates',
  abbr: 'NO3-',
  real: 'NO3-',
  metters: 'mg/dm3',
},
{
  name: 'chlorides',
  abbr:'Cl-',
  real: 'Cl-',
  metters: 'mg/dm3',
},
{
  name: 'nitrites',
  abbr:'NO2-',
  real: 'NO2-',
  metters: 'mg/dm3',
},
{
  name: 'phosphates',
  abbr:'PO43-',
  real:'PO43-',
  metters: 'mg/dm3',
},
{
  name: 'sulphates',
  abbr:'SO42-',
  real: 'SO42-',
  metters: 'mg/dm3',
},
{
  name: 'total hardness',
  abbr:'TH',
  real: 'TH',
  metters: 'mmol/dm3',
},
{
  name: 'chemical oxygen consumption',
```

```
    abbr:'COC',
    real: 'COC',
    metters: 'mg/dm³',
  },
  {
    name: 'fluorides',
    abbr:'F-',
    real:'F⁻',
    metters: 'mg/dm³',
  }]

  useEffect(() => {
    fetchDownload();
  }, [])

  const fetchDownload = async () => {
    // places
    const responsePlaces = await axios.get('/api/places');
    const placesData = responsePlaces.data;
    setPlaces(placesData);


    //sampling_places
    const responseSamplingPlaces = await axios.get('/api/sampling_places');
    const samplingPlacesData = responseSamplingPlaces.data;
    setSamplingPlaces(samplingPlacesData);

    //chemical_index
    const responseChemicalIndex = await axios.get(`/api/edit-chemical-indexes/${id}`);
    const chemicalIndexData = responseChemicalIndex.data;
    setChemicalIndex(chemicalIndexData);

    //chemical_indexes
    const responseChemicalIndexes = await axios.get('/api/chemical-indexes');
    const chemicalIndexesData = responseChemicalIndexes.data;
    setChemicalIndexes(chemicalIndexesData)

    const excludeChemicalIndexes = chemicalIndexesData.filter(place => place._id !== id);
    setExcludeChemicalIndexes(excludeChemicalIndexes);

    fetchPlacesbySamplingPlacesbyChemicalIndexes(placesData, samplingPlacesData,
chemicalIndexesData, chemicalIndexData);
  }

  const fetchPlacesbySamplingPlacesbyChemicalIndexes = async (places, sampling_places,
chemical_indexes, chemical_index) => {
    //Для select
    const selectNamePlace = chemical_index.name_place;
    setSelectNamePlace(selectNamePlace);
    console.log('selectNamePlace', selectNamePlace)
```

```
      const filterNamePlaceByNamePlaceFirst = sampling_places.filter(place =>
chemical_index.name_place.includes(place.name_place));
      const uniqRegionFirst = [...new Set(filterNamePlaceByNamePlaceFirst.map(place =>
place.region))].sort((a, b) => a.localeCompare(b));
      const selectRegion = uniqRegionFirst[0];
      setSelectRegion(selectRegion);


      const filterRegionByRegionFirst = places.filter(place => uniqRegionFirst.includes(place.region));
      const uniqCountryFirst = [...new Set(filterRegionByRegionFirst.map(place => place.country))].sort((a,
b) => a.localeCompare(b));
      const selectCountry = uniqCountryFirst[0];
      setSelectCountry(selectCountry);

      //Для списку option


      const uniqCountrySecond = [...new Set(places.map(place => place.country))].sort((a, b) =>
a.localeCompare(b));
      setUniqCountries(uniqCountrySecond);

      const filterRegionByCountry = places.filter(place => selectCountry === place.country);
      const uniqRegionsPlaces = [...new Set(filterRegionByCountry.map(place => place.region))].sort((a, b)
=> a.localeCompare(b));

      const filterRegionByRegionSecond = sampling_places.filter(place =>
uniqRegionsPlaces.includes(place.region));
      console.log('filterRegionByRegionSecond', filterRegionByRegionSecond)
      const uniqRegionSecond = [...new Set(filterRegionByRegionSecond.map(place =>
place.region))].sort((a, b) => a.localeCompare(b));
      setUniqRegions(uniqRegionSecond);

      const filterRegionByNamePlace = sampling_places.filter(place => selectRegion === place.region);
      const uniqNamePlaceSP = [...new Set(filterRegionByNamePlace.map(place => place.name_place))];

      const filterNamePlaceByNamePlace = chemical_indexes.filter(place =>
uniqNamePlaceSP.includes(place.name_place));
      const uniqNamePlaceSecond = [...new Set(filterNamePlaceByNamePlace.map(place =>
place.name_place))].sort((a, b) => a.localeCompare(b));
      setUniqNamePlace(uniqNamePlaceSecond);
      console.log('uniqNamePlaceSecond', uniqNamePlaceSecond)


      //Окремо для chemical index
      const selectChemicalIndex = indicators.find(element => element.abbr ===
chemical_index.chemical_index);
      setSelectChemicalIndex(selectChemicalIndex);
      console.log('selectChemicalIndex', selectChemicalIndex)


    }
```

```
  const handleChangeCountry = (event) => {
    const selectCountry = event.target.value;
    setSelectCountry(selectCountry);

    const filterRegionByCountry = places.filter(place => selectCountry === place.country);
    const uniqRegionsPlaces = [...new Set(filterRegionByCountry.map(place => place.region))].sort((a, b)
=> a.localeCompare(b));

    const filterRegionByRegion = sampling_places.filter(place =>
uniqRegionsPlaces.includes(place.region));
    const uniqRegionsSP = [...new Set(filterRegionByRegion.map(place => place.region))].sort((a, b) =>
a.localeCompare(b));
    setUniqRegions(uniqRegionsSP);
    setSelectRegion(uniqRegionsSP[0]);

    const filterRegionbyNamePlace = filterRegionByRegion.filter(place => uniqRegionsSP[0] ===
place.region);
    const uniqNamePlaceSP = [...new Set(filterRegionbyNamePlace.map(place =>
place.name_place))].sort((a, b) => a.localeCompare(b));

    const filterNamePlaceBynamePlace = chemical_indexes.filter(place =>
uniqNamePlaceSP.includes(place.name_place));
    const uniqNamePlaceSecond = [...new Set(filterNamePlaceBynamePlace.map(place =>
place.name_place))].sort((a, b) => a.localeCompare(b));

    setUniqNamePlace(uniqNamePlaceSecond);
    setSelectNamePlace(uniqNamePlaceSecond[0]);
    setChemicalIndex({...chemicalIndex, name_place: uniqNamePlaceSecond[0]})
  }

  const handleChangeRegion = (event) => {
    const selectRegion = event.target.value;
    setSelectRegion(selectRegion);

    const filterRegionbyNamePlace = sampling_places.filter(place => selectRegion === place.region);

    const uniqNamePlaceSP = [...new Set(filterRegionbyNamePlace.map(place =>
place.name_place))].sort((a, b) => a.localeCompare(b));
    const filterNamePlaceBynamePlace =  chemical_indexes.filter(place =>
uniqNamePlaceSP.includes(place.name_place))

    const uniqNamePlaceCI = [...new Set(filterNamePlaceBynamePlace.map(place =>
place.name_place))].sort((a, b) => a.localeCompare(b));

    setUniqNamePlace(uniqNamePlaceCI);
    setSelectNamePlace(uniqNamePlaceCI[0]);
    setChemicalIndex({...chemicalIndex, name_place: uniqNamePlaceCI[0]})
  }


  const changeNamePlace = (event) => {
    const selectNamePlace = event.target.value;
```

```
    setSelectNamePlace(selectNamePlace);
    setChemicalIndex({...chemicalIndex, name_place: event.target.value})
  }

  const changeChemicalIndex = (event) => {
   const selectedValue = event.target.value;
   const selectedChemicalIndex = indicators.find((indicator) => indicator.abbr === selectedValue);
   setSelectChemicalIndex(selectedChemicalIndex); // Оновлення selectChemicalIndex
   setChemicalIndex({
    ...chemicalIndex,
    chemical_index: selectedValue, // Оновлення хімічного показника в вашому стані
   });
  }

  const handleSubmit = async (event) => {
    event.preventDefault();
    const { name_place, chemical_index, result_chemical_index, date_analysis } = event.target;

    const errors = [];
    if(!name_place.value){
      errors.name_place = 'Fill field "name place"';
    }
    if(!chemical_index.value){
      errors.chemical_index = 'Fill field "chemical index"'
    }
    if(!result_chemical_index.value){
      errors.result_chemical_index = 'Fill field "result chemical index"';
    }

    const regexResultAnalysis = /^(\d+(\.\d{1,4})?)?$/;

    if(!regexResultAnalysis.test(result_chemical_index.value)){
      errors.result_chemical_index = 'Is not correct result analysis, must be for example 3, 30.1, 30.11,
30.123 or 30.2233 '
    }

    const dateRegex = /^(\d{2})\.(\d{2})\.(\d{4})$/;
    const matches = date_analysis.value.match(dateRegex);

    if (matches && matches.length === 4) {
     if(matches) {
      const day = parseInt(matches[1], 10);
      const month = parseInt(matches[2], 10);
      const year = parseInt(matches[3], 10);
      // Перевірка валідності дати
      if (month >= 1 && month <= 12) {
       const maxDaysInMonth = new Date(year, month, 0).getDate();
       if (day >= 1 && day <= maxDaysInMonth) {
        // Перевірка на високосний рік
        const isLeapYear = (year % 4 === 0 && year % 100 !== 0) || (year % 400 === 0);
        if (month === 2 && day > 29 && !isLeapYear) {
```

```
            errors.date_analysis = 'Invalid date: February cannot have more than 29 days in a non-leap
year.';
           } else {
            // Перевірка на дату, яка не перевищує сьогоднішню дату
            const currentDate = new Date();
            currentDate.setHours(0, 0, 0, 0); // Встановити час на початок дня
            const enteredDate = new Date(year, month - 1, day); // month - 1, оскільки місяці в Date
починаються з 0
            if (enteredDate > currentDate) {
              errors.date_analysis = 'Invalid date: The date cannot be in the future.';
            }
           }
          } else {
            errors.date_analysis = 'Invalid date: The day exceeds the maximum number of days in the
month.';
          }
         } else {
           errors.date_analysis = 'Invalid date: The month must be in the range 1 to 12.';
         }
        }
      } else {
        errors.date_analysis = 'Invalid format date or empty field';
      }

      if (Object.keys(errors).length > 0) {
        // Відобразити повідомлення про помилки
        setErrors(errors);
        return;
      }
      try{
        const response = await axios.put(`/api/edit-chemical-indexes/${id}/update`, chemicalIndex);
        const updateChemicalIndex = response.data;
        console.log('Data updated:', updateChemicalIndex);
        navigate('/searchChemicalIndexes');
      }catch(err){
        console.log('Something wrong upon change data chemical_indexes:', err)
      }
   }
```

Лістинг SearchSamplingPlaces

```
class SearchSamplingPlaces extends React.Component{
   constructor(props){
      super(props);
      this.state = {
         selectedCountry: 'Ukraine',
         selectedRegion: '',
         selectedNameWaterObject: '',

         regime: false,
```

```
        places: [],
        sampling_places: [],

        filterRegionsByCountry:[],
        filterRegionByRegion: [],
        filterNWObyRegion: [],
        filterNWO:[],

        uniqCountry: [],
        uniqRegionPlace: [],
        uniqRegionSampling: [],
        uniqNWO: [],
      }
  }
  componentDidMount(){
    this.fetchPlaces();
    this.fetchSamplingPlaces();
  }

  fetchPlaces = async () => {
    try {
      const response = await axios.get('/api/places');
      const places = response.data;
      const uniqCountry = [...new Set(places.map(place => place.country))].sort((a, b) =>
a.localeCompare(b));
      const filterRegionsByCountry = places.filter(place => this.state.selectedCountry === "" ?
place.country === uniqCountry[0] : place.country === this.state.selectedCountry);
      const uniqRegionPlace = [...new Set(filterRegionsByCountry.map(place => place.region))].sort((a,b)
=> a.localeCompare(b));
      console.log('filterRegions', filterRegionsByCountry)
      this.setState({ places: places, filterRegionsByCountry: filterRegionsByCountry, uniqCountry:
uniqCountry, uniqRegionPlace: uniqRegionPlace});
      if(!this.state.selectedCountry){
        const selectedCountry = uniqCountry[0]
        this.setState({ selectedCountry: selectedCountry})
      }
    } catch (error) {
      console.error('Error fetching places:', error);
    }
  };

  fetchSamplingPlaces = async () => {
    try {
        const response = await axios.get('/api/sampling_places');
        const sampling_places = response.data;

        //Filter and connect for another two collections for concret country another word
        const filterRegionByRegion = sampling_places.filter(place =>
this.state.uniqRegionPlace.includes(place.region));
        const uniqRegionSampling = [...new
Set(filterRegionByRegion.map(place=>place.region))].sort((a,b) => a.localeCompare(b));
```

```
          //Name water object
          const filterNWObyRegion = filterRegionByRegion.filter(place => place.region ===
uniqRegionSampling[0]);

          const uniqNWO = [...new Set(filterNWObyRegion.map(place =>
place.name_water_object))].sort((a, b) => a.localeCompare(b));

          const filterNWO = filterNWObyRegion.filter(place =>
uniqNWO[0].includes(place.name_water_object))
          this.setState({sampling_places,
            uniqRegionSampling: uniqRegionSampling, uniqNWO: uniqNWO,
            filterRegionByRegion: filterRegionByRegion, filterNWO: filterNWO, filterNWObyRegion:
filterNWObyRegion,
          })
        } catch (error) {
          console.error('Error fetching sampling places:', error);
        }
      };

      searchData = () => {
        this.setState({regime: true});
      }

      backSearch = () => {
        this.setState({regime: false});
      }

      handleCountry = async(event) => {
        const selectedCountry = event.target.value;
        this.setState({
        selectedCountry: selectedCountry,
        selectedRegion: '',
        selectedNameWaterObject: '',})

        const places = this.state.places;

        //Part country
        const filterRegionsByCountry = places.filter(place => place.country === selectedCountry);
        const uniqRegionPlace = [...new Set(filterRegionsByCountry.map(place =>
place.region))].sort((a,b) => a.localeCompare(b));

        //Part region
        const filterRegionByRegion = this.state.sampling_places.filter(place =>
uniqRegionPlace.includes(place.region));
        const uniqRegionSampling = [...new
Set(filterRegionByRegion.map(place=>place.region))].sort((a,b) => a.localeCompare(b));

        //Part name water object
        const filterNWObyRegion = filterRegionByRegion.filter(place => place.region ===
uniqRegionSampling[0]);
```

```
        const uniqNWO = [...new Set(filterNWObyRegion.map(place =>
place.name_water_object))].sort((a, b) => a.localeCompare(b));
        const filterNWO = filterNWObyRegion.filter(place =>
uniqNWO[0].includes(place.name_water_object));

        this.setState({
        uniqRegionPlace, uniqRegionSampling, uniqNWO,
        filterRegionsByCountry, filterRegionByRegion, filterNWObyRegion, filterNWO
        })
      }



        handleRegion = async(event) => {
          const selectedRegion = event.target.value;
          this.setState({
            selectedRegion: selectedRegion,
            selectedNameWaterObject: '',
          })

          //Part region
          const filterRegionByRegion = this.state.sampling_places.filter(place =>
this.state.uniqRegionPlace.includes(place.region));
          const uniqRegionSampling = [...new
Set(filterRegionByRegion.map(place=>place.region))].sort((a,b) => a.localeCompare(b));
          //Part name water object
          const filterNWObyRegion = filterRegionByRegion.filter(place => place.region ===
selectedRegion);
          const uniqNWO = [...new Set(filterNWObyRegion.map(place =>
place.name_water_object))].sort((a, b) => a.localeCompare(b));
          const filterNWO = filterNWObyRegion.filter(place =>
uniqNWO[0].includes(place.name_water_object));

          this.setState({
            uniqRegionSampling, uniqNWO,
            filterRegionByRegion, filterNWObyRegion, filterNWO
            })
        }



        handleNameWaterObject = async(event) => {
          const selectedNameWaterObject = event.target.value;
          this.setState({selectedNameWaterObject})
          //Part name water object
          const filterNWO = this.state.filterNWObyRegion.filter(place => place.name_water_object ===
selectedNameWaterObject);

          console.log("selectedNameWaterObject:", selectedNameWaterObject)

          this.setState({
            filterNWO
```

```
        })
      }
```

Лістинг SearchChemicalIndexes

```
class SearchChemicalIndexes extends React.Component{
  constructor(props){
    super(props);
    this.state = {

      //databases
      places: [],
      samplingPlaces: [],
      chemical_indexes: [],

      //selected on panel managment
      selectedCountry: 'Ukraine',
      selectedRegion: '',
      selectedNamePlace: '',
      selectedDateFrom: '',
      selectedDateTo: '',

      //filters for database
      filterRegionsByCountry: [],
      filterRegionByRegion: [],
      filterRegionSamplingbyNamePlace: [],
      filterNamePlacesByRegion: [],
      filterNamePlaceIndex: [],
      filterNamePlacebyNamePlace: [],
      //За ним відбувається пошук потрібних даних
      filterSearch: [],


      //uniq value form database usses for filter
      uniqCountry: [],
      uniqRegionPlace: [],
      uniqRegionSampling: [],
      uniqNamePlaceSampling: [],
      uniqNamePlaceIndex: [],

      regime: false,

      errorMessageDate: '',
    }
  }

  async componentDidMount(){
    await this.fetchPlaces();
    await this.fetchSamplingPlaces();
```

```
    await this.fetchChemicalIndexes();
  }


  //Part for fetch
  fetchPlaces = async () => {
    try{
      const response = await axios.get('/api/places');
      const places = response.data;
      const uniqCountry = [...new Set(places.map(place => place.country))].sort((a, b) =>
a.localeCompare(b));
      const filterRegionsByCountry = places.filter(place => this.state.selectedCountry === "" ?
place.country === uniqCountry[0] : place.country === this.state.selectedCountry);
      const uniqRegionPlace = [...new Set(filterRegionsByCountry.map(place =>
place.region))].sort((a,b) => a.localeCompare(b));
      this.setState({ places: places, filterRegionsByCountry: filterRegionsByCountry,
        uniqCountry: uniqCountry, uniqRegionPlace: uniqRegionPlace});
      if(!this.state.selectedCountry){
        const selectedCountry = uniqCountry[0]
        this.setState({ selectedCountry: selectedCountry})
      }
    }catch(err){
      console.log('Error fetching data from places', err)
    }
  }

  fetchSamplingPlaces = async () => {
    const response = await axios.get('/api/sampling_places');
    const sampling_places = response.data;

    //Зв'язати зовнішні ключі між places та sampling_places колекціями
    const filterRegionByRegion = sampling_places.filter(place =>
this.state.uniqRegionPlace.includes(place.region));
    const uniqRegionSampling = [...new Set(filterRegionByRegion.map(place=>place.region))].sort((a,b)
=> a.localeCompare(b));

    const filterRegionSamplingbyNamePlace = filterRegionByRegion.filter(place => place.region ===
uniqRegionSampling[0])
    const uniqNamePlaceSampling = [...new
Set(filterRegionSamplingbyNamePlace.map(place=>place.name_place))].sort((a, b) =>
a.localeCompare(b));
    this.setState({sampling_places: sampling_places,
          uniqRegionSampling: uniqRegionSampling, uniqNamePlaceSampling:
uniqNamePlaceSampling,
          filterRegionByRegion: filterRegionByRegion, filterRegionSamplingbyNamePlace:
filterRegionSamplingbyNamePlace
    })
    console.log('uniqNamePlaceSampling', uniqNamePlaceSampling);
    if(!this.state.selectedRegion){
      const selectedRegion = uniqRegionSampling[0]
      this.setState({ selectedRegion: selectedRegion})
    }
```

```
  }

  fetchChemicalIndexes = async () => {
    const response = await axios.get('/api/chemical-indexes');
    const chemical_indexes = response.data;
    const uniqNamePlaceSampling = this.state.uniqNamePlaceSampling
    console.log(uniqNamePlaceSampling)

    //Зв'язати зовнішні ключі між sampling_places та chemical_indexes колекціями
    const filterNamePlacebyNamePlace = chemical_indexes.filter(place =>
uniqNamePlaceSampling.includes(place.name_place));
    const uniqNamePlaceIndex = [...new
Set(filterNamePlacebyNamePlace.map(place=>place.name_place))].sort((a, b) => a.localeCompare(b));
    console.log('uniqNamePlaceIndex', uniqNamePlaceIndex);
    console.log('uniqNamePlaceIndex:', uniqNamePlaceIndex)
    const filterSearch = filterNamePlacebyNamePlace.filter(place => place.name_place ===
uniqNamePlaceIndex[0]);

    this.setState({chemical_indexes: chemical_indexes,
      filterNamePlacebyNamePlace: filterNamePlacebyNamePlace,
      filterSearch: filterSearch,
      uniqNamePlaceIndex: uniqNamePlaceIndex
    });
  }

  //Part for handle
  handleCountry= (event) => {
    try{
      const selectedCountry = event.target.value;

      this.setState({
        selectedCountry,
        selectedRegion: '',
        selectedNamePlace: '',
        selectedDateFrom: '',
        selectedDateTo: '',
      })

      const {places, sampling_places, chemical_indexes} = this.state;

      const filterRegionsByCountry = places.filter(place => place.country === selectedCountry);
      const uniqRegionPlace = [...new Set(filterRegionsByCountry.map(place =>
place.region))].sort((a,b) => a.localeCompare(b));

      //Зв'язати зовнішні ключі між places та sampling_places колекціями
      const filterRegionByRegion = sampling_places.filter(place =>
uniqRegionPlace.includes(place.region));
      const uniqRegionSampling = [...new
Set(filterRegionByRegion.map(place=>place.region))].sort((a,b) => a.localeCompare(b));
      console.log('uniqRegionSampling:', uniqRegionSampling)
```

```
        const filterRegionSamplingbyNamePlace = filterRegionByRegion.filter(place => place.region ===
uniqRegionSampling[0]);
        const uniqNamePlaceSampling = [...new
Set(filterRegionSamplingbyNamePlace.map(place=>place.name_place))].sort((a, b) =>
a.localeCompare(b));

        //Зв'язати зовнішні ключі між sampling_places та chemical_indexes колекціями
        const filterNamePlacebyNamePlace = chemical_indexes.filter(place =>
uniqNamePlaceSampling.includes(place.name_place));
        const uniqNamePlaceIndex = [...new
Set(filterNamePlacebyNamePlace.map(place=>place.name_place))].sort((a, b) => a.localeCompare(b));

        const filterSearch = filterNamePlacebyNamePlace.filter(place => place.name_place ===
uniqNamePlaceIndex[0]);

        this.setState({filterRegionsByCountry: filterRegionsByCountry, filterRegionByRegion:
filterRegionByRegion,
          filterRegionSamplingbyNamePlace: filterRegionSamplingbyNamePlace,
filterNamePlacebyNamePlace: filterNamePlacebyNamePlace,
          filterSearch: filterSearch,

          uniqRegionPlace: uniqRegionPlace, uniqRegionPlace: uniqRegionPlace,
          uniqRegionSampling: uniqRegionSampling,
          uniqNamePlaceSampling: uniqNamePlaceSampling, uniqNamePlaceIndex: uniqNamePlaceIndex
        })
      }catch(err){
        console.log('Error handle change country:', err)
      }
    }

    handleRegion = (event) => {
      try{
        const {filterRegionByRegion, chemical_indexes} = this.state;

        const selectedRegion = event.target.value;
        this.setState({
          selectedRegion,
          selectedNamePlace: '',
          selectedDateFrom: '',
          selectedDateTo: '',
        })

        const filterRegionSamplingbyNamePlace = filterRegionByRegion.filter(place => place.region ===
selectedRegion);
        const uniqNamePlaceSampling = [...new
Set(filterRegionSamplingbyNamePlace.map(place=>place.name_place))].sort((a, b) =>
a.localeCompare(b));

        //Зв'язати зовнішні ключі між sampling_places та chemical_indexes колекціями
        const filterNamePlacebyNamePlace = chemical_indexes.filter(place =>
uniqNamePlaceSampling.includes(place.name_place));
```

```
        const uniqNamePlaceIndex = [...new
Set(filterNamePlacebyNamePlace.map(place=>place.name_place))].sort((a, b) => a.localeCompare(b));

        const filterSearch = filterNamePlacebyNamePlace.filter(place => place.name_place ===
uniqNamePlaceIndex[0]);

        this.setState({filterRegionSamplingbyNamePlace: filterRegionSamplingbyNamePlace,
filterNamePlacebyNamePlace: filterNamePlacebyNamePlace,
          filterSearch: filterSearch,
          uniqNamePlaceSampling: uniqNamePlaceSampling, uniqNamePlaceIndex:
uniqNamePlaceIndex,
        });

      }catch(err){
        console.log('Error handle change region:', err)
      }
    }

    handleNamePalce = (event) => {
      try{
        const selectedNamePlace = event.target.value;
        this.setState({ selectedNamePlace: selectedNamePlace,
          selectedDateFrom: '',
          selectedDateTo: '',
        })
        const uniqNamePlaceIndex = [...new
Set(this.state.filterNamePlacebyNamePlace.map(place=>place.name_place))].sort((a, b) =>
a.localeCompare(b));
        const filterSearch = this.state.filterNamePlacebyNamePlace.filter(place => place.name_place ===
selectedNamePlace);
        this.setState({ uniqNamePlaceIndex: uniqNamePlaceIndex,
          filterSearch: filterSearch})
      }catch(err){
        console.log('Error handle change name place:', err)
      }
    }


    handleDateFrom = (event) => {
      try {
        const selectedDateFrom = event.target.value;
        this.setState({ selectedDateFrom: selectedDateFrom });
      } catch (err) {
        console.log("Error change date from", err);
      }
    }

    handleDateTo = (event) => {
      try {
        const selectedDateTo = event.target.value;
        this.setState({ selectedDateTo: selectedDateTo });
      } catch (err) {
```

```
        console.log("Error change date to", err);
      }
    }

    handleDate = () => {
      try {
        const { selectedDateFrom, selectedDateTo } = this.state;
        const oldFilterSearch = this.state.filterNamePlacebyNamePlace.filter((place) =>
          this.state.selectedNamePlace === ""
            ? place.name_place === this.state.uniqNamePlaceIndex[0]
            : place.name_place === this.state.selectedNamePlace
        );

        const regexDate = /^(\d{2})\.(\d{2})\.(\d{4})$/;

        const newFilterSearch = oldFilterSearch.filter((place) => {
          const transformDate_Analysis = place.date_analysis.match(regexDate);
          console.log('transformDate_Analysis:', transformDate_Analysis)
          const date_analysisDay = parseInt(transformDate_Analysis[1], 10);
          const date_analysisMonth = parseInt(transformDate_Analysis[2], 10);
          const date_analysisYear = parseInt(transformDate_Analysis[3], 10);

          let transformDateFrom;
          let fromDay;
          let fromMonth;
          let fromYear;

          let transformDateTo;
          let toDay;
          let toMonth;
          let toYear;

          if (regexDate.test(selectedDateFrom) || regexDate.test(selectedDateTo)) {
            if (regexDate.test(selectedDateFrom)) {
              transformDateFrom = selectedDateFrom.match(regexDate);
              fromDay = parseInt(transformDateFrom[1], 10);
              fromMonth = parseInt(transformDateFrom[2], 10);
              fromYear = parseInt(transformDateFrom[3], 10);
            }

            if (regexDate.test(selectedDateTo)) {
              transformDateTo = selectedDateTo.match(regexDate);
              toDay = parseInt(transformDateTo[1], 10);
              toMonth = parseInt(transformDateTo[2], 10);
              toYear = parseInt(transformDateTo[3], 10);
            }

            if (regexDate.test(selectedDateFrom) && regexDate.test(selectedDateTo)) {
              if (fromYear <= date_analysisYear && date_analysisYear <= toYear) {
                if (fromYear < date_analysisYear || (fromYear === date_analysisYear && fromMonth <=
date_analysisMonth && date_analysisMonth <= toMonth)) {
```

```
                        if (fromYear < date_analysisYear || (fromYear === date_analysisYear && fromMonth
=== date_analysisMonth && fromDay <= date_analysisDay && date_analysisDay <= toDay)) {
                            return true;
                        }
                    }
                }
            } else if (regexDate.test(selectedDateFrom) && selectedDateTo === "") {
                if (fromYear <= date_analysisYear) {
                    if (fromYear < date_analysisYear || (fromYear === date_analysisYear && fromMonth <=
date_analysisMonth)) {
                        if (fromYear < date_analysisYear || (fromYear === date_analysisYear && fromMonth
=== date_analysisMonth && fromDay <= date_analysisDay)) {
                            return true;
                        }
                    }
                }
            } else if (selectedDateFrom === "" && regexDate.test(selectedDateTo)) {
                if (date_analysisYear <= toYear) {
                    if (date_analysisYear < toYear || (date_analysisYear === toYear && date_analysisMonth
<= toMonth)) {
                        if (date_analysisYear < toYear || (date_analysisYear === toYear &&
date_analysisMonth === toMonth && date_analysisDay <= toDay)) {
                            return true;
                        }
                    }
                }
            } else if (selectedDateFrom === "" && selectedDateTo === "") {
                return true;
            }
            return false;
        });
        console.log('newFilterSearch:', newFilterSearch)
        this.setState({ filterSearch: newFilterSearch });
    } catch (err) {
        console.log('Error handle change date analysis:', err);
    }
  };

  handleRegimeON = () => {
    this.setState({regime: true})
  }
  handleRegimeOFF = () => {
    this.setState({regime: false})
  }
```

Лістинг EditSamplingPlace

```
//Update
import React, { useState, useEffect } from 'react';
import axios from 'axios';
```

```
import { useParams, useNavigate} from 'react-router-dom';



function EditSamplingPlace(){
   const navigate = useNavigate();


   const {id} = useParams();
   const [places, setPlaces] = useState([]);
   const [sampling_places, setSamplingPlaces] = useState([]);
   //Для виключення за _id у випадку редагування збоїв перевірки унікальності за тим же
_id(name_place, longitude, latitude)
   const [excludeSamplingPlaces, setExcludeSamplingPlaces] = useState([])

   const [sampling_place, setSamplingPlace] = useState({
      region: '',
      name_place: '',
      type_water_object: '',
      name_water_object: '',
      longitude: '',
      latitude: '',
      comment: '',
   });

   const [selectCountry, setSelectCountry] = useState('');
   const [selectRegion, setSelectRegion] = useState('');
   const [selectTypeWaterObject, setSelectTypeWaterObject] = useState('');

   const [errors, setErrorMessages] = useState([]);


   //Робимо фільтри та уніки для таких властивостей як region і country за зворотньою сумісністю по
зовнішніх ключах

   const [uniqCountry, setUniqCountry] = useState([]);
   const [uniqRegion, setUniqRegion] = useState([]);

   useEffect(() => {
      fetchDownloadElements();
   }, []);


   const fetchDownloadElements = async () => {
     try {
       //places aprt
         const responsePlaces = await axios.get('/api/places');
         const placesData = responsePlaces.data;
         setPlaces(placesData);


       //sampling_places part
```

```
        const responseSamplingPlaces = await axios.get('/api/sampling_places');
        const samplingPlacesData = responseSamplingPlaces.data;
        const excludeSamplingPlaces = samplingPlacesData.filter(place => place._id !== id)
        setExcludeSamplingPlaces(excludeSamplingPlaces);
        setSamplingPlaces(samplingPlacesData);

       //sampling_place part
        const responseSamplingPlace = await axios.get(`/api/edit-sampling-places/${id}`);
        const samplingPlaceData = responseSamplingPlace.data;
        setSamplingPlace(samplingPlaceData);

       //activation this is all in state across method
       fetchPlacesbySamplingPlaces(placesData, samplingPlacesData, samplingPlaceData,
excludeSamplingPlaces);
     } catch (err) {
        console.log('Error fetching data', err);
     }
  };

   const fetchPlacesbySamplingPlaces = async (places, sampling_places, sampling_place,
exclude_sampling_places) => {
     try{
       //зворотна сумісність
       const filterRegionByRegionFirst = places.filter(place =>
sampling_place.region.includes(place.region))

       console.log('filterRegionByRegionFirst:', filterRegionByRegionFirst)
       const uniqCountryInv = [...new Set(filterRegionByRegionFirst.map(place => place.country))].sort((a,
b) => a.localeCompare(b));
       const selectCountry = uniqCountryInv[0]; // саме це значення ми використаємо для
відображення при завантажені сторінки із початковими даними
       setSelectCountry(selectCountry);

       //Old School
       const uniqCountryList = [...new Set(places.map(place => place.country))].sort((a, b) =>
a.localeCompare(b));
       setUniqCountry(uniqCountryList);

       const fitlerRegionsByCountry = places.filter(place => selectCountry === place.country);
       const uniqPlacesRegionList = [...new Set(fitlerRegionsByCountry.map(place =>
place.region))].sort((a, b) => a.localeCompare(b));

       const filterRegionByRegionSecond = sampling_places.filter(place =>
uniqPlacesRegionList.includes(place.region));
       const uniqSPRegionList = [...new Set(filterRegionByRegionSecond.map(place =>
place.region))].sort((a, b) => a.localeCompare(b));

       setSelectRegion(sampling_place.region);
       setUniqRegion(uniqSPRegionList);

     }catch(err){
       console.log('Not completed connect database Places with sampling_places', err);
```

```
    }
  }


  const handleChangeCountry = (event) => {
    const selectCountry = event.target.value;
    setSelectCountry(selectCountry)
    setSelectRegion('')

    const fitlerRegionsByCountry = places.filter(place => selectCountry === place.country);
    const uniqPlacesRegionList = [...new Set(fitlerRegionsByCountry.map(place => place.region))].sort((a,
b) => a.localeCompare(b));

    const filterRegionByRegionSecond = sampling_places.filter(place =>
uniqPlacesRegionList.includes(place.region));
    const uniqSPRegionList = [...new Set(filterRegionByRegionSecond.map(place =>
place.region))].sort((a, b) => a.localeCompare(b));
    setUniqRegion(uniqSPRegionList);
  }


  const handleChangeRegion = (event) => {
    const selectRegion = event.target.value;
    setSelectRegion(selectRegion);

    setSamplingPlace({
      ...sampling_place,
      region: selectRegion,
    });

    // const filterRegionByRegionSecond = sampling_places.filter(place => place.region ===
selectRegion);
    // const uniqSPRegionList = [... new Set(filterRegionByRegionSecond.map(place =>
place.region))].sort((a, b) => a.localeCompare(b));
    // setUniqRegion(uniqSPRegionList);

  }

  const handleChangeTypeWaterObject = (event) => {
    const selectTypeWaterObject = event.target.value;
    setSelectTypeWaterObject(selectTypeWaterObject);
    setSamplingPlace({
      ...sampling_place,
      type_water_object: selectTypeWaterObject,
    });
  }



  const handleSubmit = async (event) => {
    event.preventDefault();
```

```
    const { region, name_place, type_water_object, name_water_object, longitude, latitude } =
event.target;
    const errors = [];

    if (!region.value) {
      errors.region = "fill in region field";
     }

     if(!name_place.value){
      errors.name_place = "fill in name place field";
     }

     if(!type_water_object.value){
      errors.type_water_object = "fill in type water object field";
     }
     if(!name_water_object.value){
      errors.name_water_object = "fill in name water object field";
     }

     const regexLongitude= /^(-)?(([0-8]?[0-9])(\.\d{1,6})?)$|90$/
     const regexLatitude= /^(-)?((1?[0-7]?[0-9])(\.\d{1,6})?)$|180$/



     if(!longitude.value|| !regexLongitude.test(longitude.value)){
      if(!longitude.value){
       errors.longitude = "fill coordinate x field";
      }
      else{
       errors.longitude = "error format, need form -90 to 90, characters after dote max 6";
      }
     }

     if(!latitude.value || !regexLatitude.test(latitude.value)){
      if(!latitude.value){
       errors.latitude ="fill coordinate y field";
      }
      else{
       errors.latitude ="error format, need form -180 to 180, characters after dote max 6";
      }
     }

     const existingPlaceWithName = excludeSamplingPlaces.find(place => place.name_place ===
name_place.value);
     if (existingPlaceWithName) {
      errors.place_match = "This name place is already taken"
     }

     const existingPlaceWithCoordinates = excludeSamplingPlaces.find(place => place.longitude ===
longitude.value && place.latitude === latitude.value);
     if (existingPlaceWithCoordinates) {
      errors.coordinates_match =  "These coordinates are already taken";
```

```
      }

      if (Object.keys(errors).length > 0) {
        setErrorMessages(errors);
        return;
      }
      try {
        // Використовуйте тут ваш серверний URL для оновлення даних
        const response = await axios.put(`/api/edit-sampling-places/${id}/update`, sampling_place);
        const updatedPlace = response.data;
        console.log('Data updated:', updatedPlace);
        navigate('/searchSamplingPlaces');
      } catch (error) {
        console.error('Error updating data:', error);

      }
    };
```

Лістинг EditChemicalIndex

```
function EditChemicalIndex(){
    const navigate = useNavigate();
    const {id} = useParams();

    const [places, setPlaces] = useState([]);
    const [sampling_places, setSamplingPlaces] = useState([]);
    const [chemical_indexes, setChemicalIndexes] = useState([]);
    //Для виключення за _id у випадку редагування збоїв перевірки унікальності за тим же
_id(name_place, date_analysis, latitude)
    const [excludeChemicalIndexes, setExcludeChemicalIndexes] = useState([]);
    const [chemicalIndex, setChemicalIndex] = useState({
        name_place: '',
        chemical_index: '',
        result_chemical_index: '',
        date_analysis: '',
        comment: ''
    });

    //select
    const [selectCountry, setSelectCountry] = useState('');
    const [selectRegion, setSelectRegion] = useState('');
    const [selectNamePlace, setSelectNamePlace] = useState('');

    const [selectChemicalIndex, setSelectChemicalIndex] = useState({});

    //unique
    const [uniqCountries, setUniqCountries] = useState([]);
    const [uniqRegions, setUniqRegions] = useState([]);
    const [uniqNamePlace, setUniqNamePlace] = useState([]);

    const [errors, setErrors] = useState([]);
```

```
const indicators = [{
  name: 'nitrates',
  abbr: 'NO3-',
  real: 'NO3-',
  metters: 'mg/dm³',
},
{
  name: 'chlorides',
  abbr:'Cl-',
  real: 'Cl-',
  metters: 'mg/dm³',
},
{
  name: 'nitrites',
  abbr:'NO2-',
  real: 'NO2-',
  metters: 'mg/dm³',
},
{
  name: 'phosphates',
  abbr:'PO43-',
  real:'PO43-',
  metters: 'mg/dm³',
},
{
  name: 'sulphates',
  abbr:'SO42-',
  real: 'SO42-',
  metters: 'mg/dm³',
},
{
  name: 'total hardness',
  abbr:'TH',
  real: 'TH',
  metters: 'mmol/dm³',
},
{
  name: 'chemical oxygen consumption',
  abbr:'COC',
  real: 'COC',
  metters: 'mg/dm³',
},
{
  name: 'fluorides',
  abbr:'F-',
  real:'F-',
  metters: 'mg/dm³',
}]

useEffect(() => {
   fetchDownload();
```

```
  }, [])

  const fetchDownload = async () => {
    // places
    const responsePlaces = await axios.get('/api/places');
    const placesData = responsePlaces.data;
    setPlaces(placesData);


    //sampling_places
    const responseSamplingPlaces = await axios.get('/api/sampling_places');
    const samplingPlacesData = responseSamplingPlaces.data;
    setSamplingPlaces(samplingPlacesData);

    //chemical_index
    const responseChemicalIndex = await axios.get(`/api/edit-chemical-indexes/${id}`);
    const chemicalIndexData = responseChemicalIndex.data;
    setChemicalIndex(chemicalIndexData);

    //chemical_indexes
    const responseChemicalIndexes = await axios.get('/api/chemical-indexes');
    const chemicalIndexesData = responseChemicalIndexes.data;
    setChemicalIndexes(chemicalIndexesData)

    const excludeChemicalIndexes = chemicalIndexesData.filter(place => place._id !== id);
    setExcludeChemicalIndexes(excludeChemicalIndexes);

    fetchPlacesbySamplingPlacesbyChemicalIndexes(placesData, samplingPlacesData,
chemicalIndexesData, chemicalIndexData);
  }

  const fetchPlacesbySamplingPlacesbyChemicalIndexes = async (places, sampling_places,
chemical_indexes, chemical_index) => {
    //Для select
    const selectNamePlace = chemical_index.name_place;
    setSelectNamePlace(selectNamePlace);
    console.log('selectNamePlace', selectNamePlace)

    const filterNamePlaceByNamePlaceFirst = sampling_places.filter(place =>
chemical_index.name_place.includes(place.name_place));
    const uniqRegionFirst = [...new Set(filterNamePlaceByNamePlaceFirst.map(place =>
place.region))].sort((a, b) => a.localeCompare(b));
    const selectRegion = uniqRegionFirst[0];
    setSelectRegion(selectRegion);


    const filterRegionByRegionFirst = places.filter(place => uniqRegionFirst.includes(place.region));
    const uniqCountryFirst = [...new Set(filterRegionByRegionFirst.map(place => place.country))].sort((a,
b) => a.localeCompare(b));
    const selectCountry = uniqCountryFirst[0];
    setSelectCountry(selectCountry);
```

```
      //Для списку option


      const uniqCountrySecond = [...new Set(places.map(place => place.country))].sort((a, b) =>
a.localeCompare(b));
      setUniqCountries(uniqCountrySecond);

      const filterRegionByCountry = places.filter(place => selectCountry === place.country);
      const uniqRegionsPlaces = [...new Set(filterRegionByCountry.map(place => place.region))].sort((a, b)
=> a.localeCompare(b));

      const filterRegionByRegionSecond = sampling_places.filter(place =>
uniqRegionsPlaces.includes(place.region));
      console.log('filterRegionByRegionSecond', filterRegionByRegionSecond)
      const uniqRegionSecond = [...new Set(filterRegionByRegionSecond.map(place =>
place.region))].sort((a, b) => a.localeCompare(b));
      setUniqRegions(uniqRegionSecond);

      const filterRegionByNamePlace = sampling_places.filter(place => selectRegion === place.region);
      const uniqNamePlaceSP = [...new Set(filterRegionByNamePlace.map(place => place.name_place))];

      const filterNamePlaceByNamePlace = chemical_indexes.filter(place =>
uniqNamePlaceSP.includes(place.name_place));
      const uniqNamePlaceSecond = [...new Set(filterNamePlaceByNamePlace.map(place =>
place.name_place))].sort((a, b) => a.localeCompare(b));
      setUniqNamePlace(uniqNamePlaceSecond);
      console.log('uniqNamePlaceSecond', uniqNamePlaceSecond)


      //Окремо для chemical index
      const selectChemicalIndex = indicators.find(element => element.abbr ===
chemical_index.chemical_index);
      setSelectChemicalIndex(selectChemicalIndex);
      console.log('selectChemicalIndex', selectChemicalIndex)


   }
   const handleChangeCountry = (event) => {
      const selectCountry = event.target.value;
      setSelectCountry(selectCountry);

      const filterRegionByCountry = places.filter(place => selectCountry === place.country);
      const uniqRegionsPlaces = [...new Set(filterRegionByCountry.map(place => place.region))].sort((a, b)
=> a.localeCompare(b));

      const filterRegionByRegion = sampling_places.filter(place =>
uniqRegionsPlaces.includes(place.region));
      const uniqRegionsSP = [...new Set(filterRegionByRegion.map(place => place.region))].sort((a, b) =>
a.localeCompare(b));
      setUniqRegions(uniqRegionsSP);
      setSelectRegion(uniqRegionsSP[0]);
```

```
      const filterRegionbyNamePlace = filterRegionByRegion.filter(place => uniqRegionsSP[0] ===
place.region);
      const uniqNamePlaceSP = [...new Set(filterRegionbyNamePlace.map(place =>
place.name_place))].sort((a, b) => a.localeCompare(b));

      const filterNamePlaceBynamePlace = chemical_indexes.filter(place =>
uniqNamePlaceSP.includes(place.name_place));
      const uniqNamePlaceSecond = [...new Set(filterNamePlaceBynamePlace.map(place =>
place.name_place))].sort((a, b) => a.localeCompare(b));

      setUniqNamePlace(uniqNamePlaceSecond);
      setSelectNamePlace(uniqNamePlaceSecond[0]);
      setChemicalIndex({...chemicalIndex, name_place: uniqNamePlaceSecond[0]})
    }

    const handleChangeRegion = (event) => {
      const selectRegion = event.target.value;
      setSelectRegion(selectRegion);

      const filterRegionbyNamePlace = sampling_places.filter(place => selectRegion === place.region);

      const uniqNamePlaceSP = [...new Set(filterRegionbyNamePlace.map(place =>
place.name_place))].sort((a, b) => a.localeCompare(b));
      const filterNamePlaceBynamePlace =  chemical_indexes.filter(place =>
uniqNamePlaceSP.includes(place.name_place))

      const uniqNamePlaceCI = [...new Set(filterNamePlaceBynamePlace.map(place =>
place.name_place))].sort((a, b) => a.localeCompare(b));

      setUniqNamePlace(uniqNamePlaceCI);
      setSelectNamePlace(uniqNamePlaceCI[0]);
      setChemicalIndex({...chemicalIndex, name_place: uniqNamePlaceCI[0]})
    }


    const changeNamePlace = (event) => {
      const selectNamePlace = event.target.value;
      setSelectNamePlace(selectNamePlace);
      setChemicalIndex({...chemicalIndex, name_place: event.target.value})
    }

    const changeChemicalIndex = (event) => {
      const selectedValue = event.target.value;
      const selectedChemicalIndex = indicators.find((indicator) => indicator.abbr === selectedValue);
      setSelectChemicalIndex(selectedChemicalIndex); // Оновлення selectChemicalIndex
      setChemicalIndex({
        ...chemicalIndex,
        chemical_index: selectedValue, // Оновлення хімічного показника в вашому стані
      });
    }

    const handleSubmit = async (event) => {
```

```
    event.preventDefault();
    const { name_place, chemical_index, result_chemical_index, date_analysis } = event.target;

    const errors = [];
    if(!name_place.value){
      errors.name_place = 'Fill field "name place"';
    }
    if(!chemical_index.value){
      errors.chemical_index = 'Fill field "chemical index"'
    }
    if(!result_chemical_index.value){
      errors.result_chemical_index = 'Fill field "result chemical index"';
    }

    const regexResultAnalysis = /^(\d+(\.\d{1,4})?)?$/;

    if(!regexResultAnalysis.test(result_chemical_index.value)){
      errors.result_chemical_index = 'Is not correct result analysis, must be for example 3, 30.1, 30.11,
30.123 or 30.2233 '
    }

    const dateRegex = /^(\d{2})\.(\d{2})\.(\d{4})$/;
    const matches = date_analysis.value.match(dateRegex);

    if (matches && matches.length === 4) {
     if(matches) {
      const day = parseInt(matches[1], 10);
      const month = parseInt(matches[2], 10);
      const year = parseInt(matches[3], 10);
      // Перевірка валідності дати
      if (month >= 1 && month <= 12) {
       const maxDaysInMonth = new Date(year, month, 0).getDate();
       if (day >= 1 && day <= maxDaysInMonth) {
        // Перевірка на високосний рік
        const isLeapYear = (year % 4 === 0 && year % 100 !== 0) || (year % 400 === 0);
        if (month === 2 && day > 29 && !isLeapYear) {
          errors.date_analysis = 'Invalid date: February cannot have more than 29 days in a non-leap
year.';
        } else {
          // Перевірка на дату, яка не перевищує сьогоднішню дату
          const currentDate = new Date();
          currentDate.setHours(0, 0, 0, 0); // Встановити час на початок дня
          const enteredDate = new Date(year, month - 1, day); // month - 1, оскільки місяці в Date
починаються з 0
          if (enteredDate > currentDate) {
            errors.date_analysis = 'Invalid date: The date cannot be in the future.';
          }
        }
       } else {
        errors.date_analysis = 'Invalid date: The day exceeds the maximum number of days in the
month.';
       }
```

```
        } else {
          errors.date_analysis = 'Invalid date: The month must be in the range 1 to 12.';
        }
       }
     } else {
       errors.date_analysis = 'Invalid format date or empty field';
     }

     if (Object.keys(errors).length > 0) {
        // Відобразити повідомлення про помилки
        setErrors(errors);
        return;
     }
     try{
        const response = await axios.put(`/api/edit-chemical-indexes/${id}/update`, chemicalIndex);
        const updateChemicalIndex = response.data;
        console.log('Data updated:', updateChemicalIndex);
        navigate('/searchChemicalIndexes');
     }catch(err){
        console.log('Something wrong upon change data chemical_indexes:', err)
     }
  }
```

Додаток В. Результати тестування роботи клієнтської частини веб-сервісу Chemmagpie через Selenium IDE

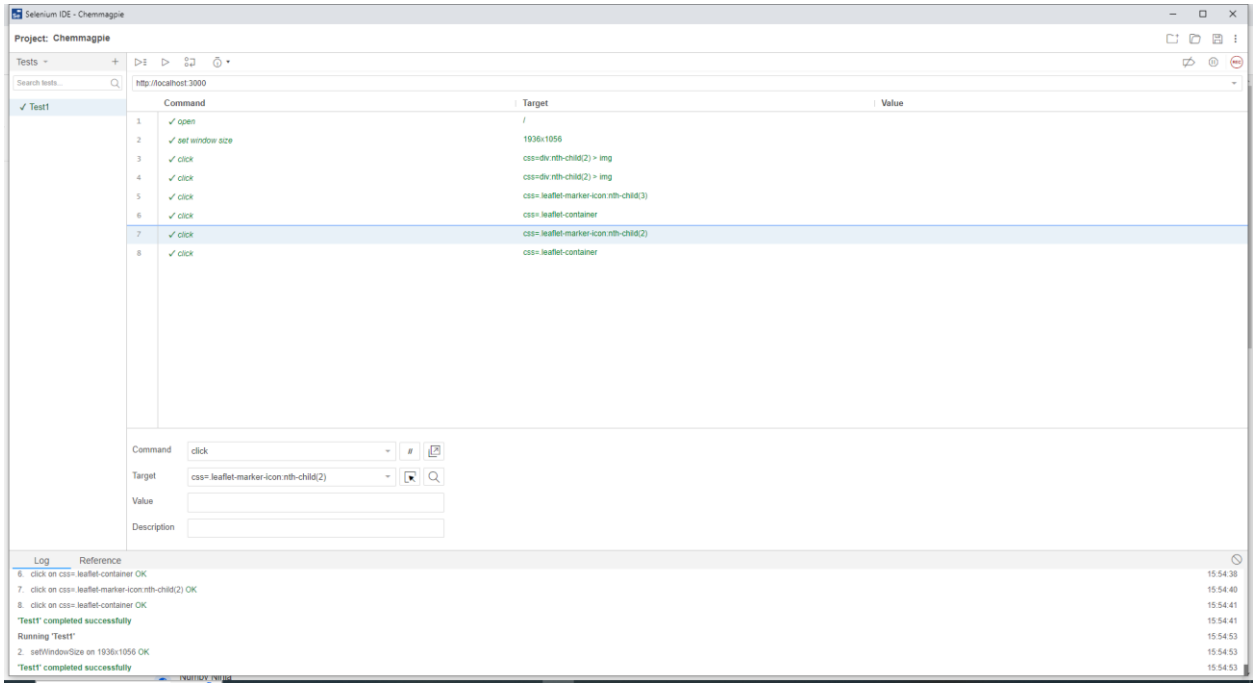

Рисунок В3.1 – Результати тестів роботи користувацького інтерфейсу веб-мапи

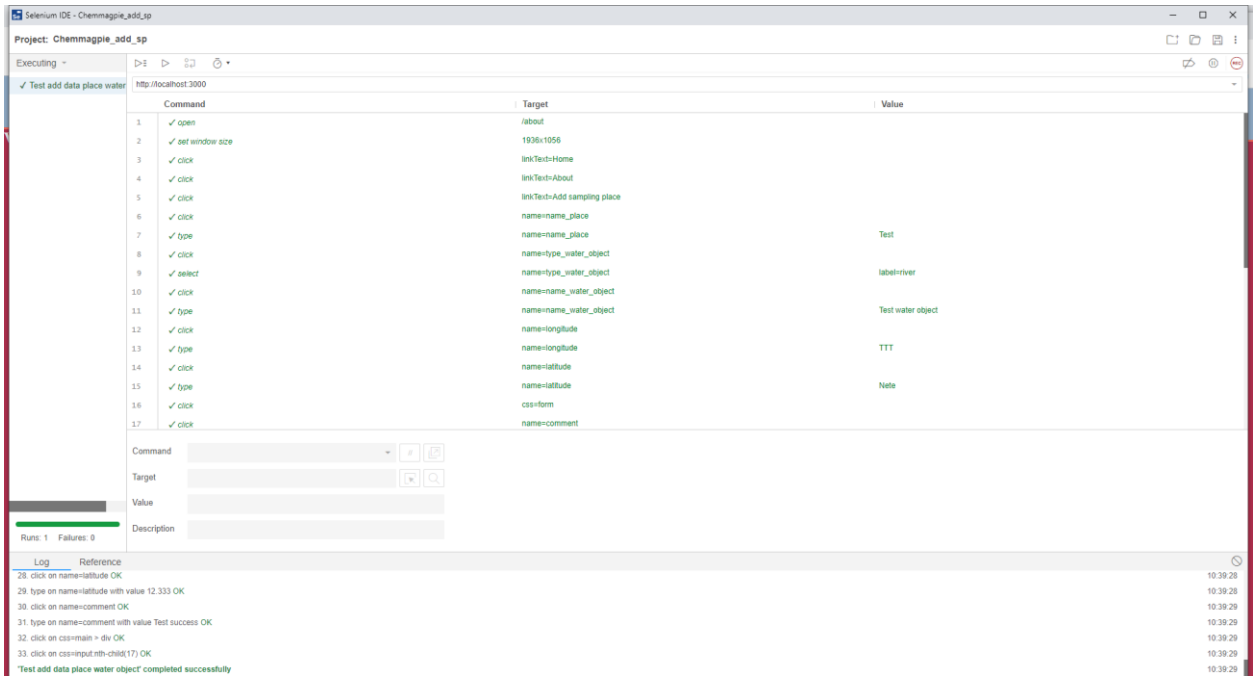

Рисунок В3.2 – Результати тестів інтерфейсу додання даних про відбір проби води в колекцію sampling_places

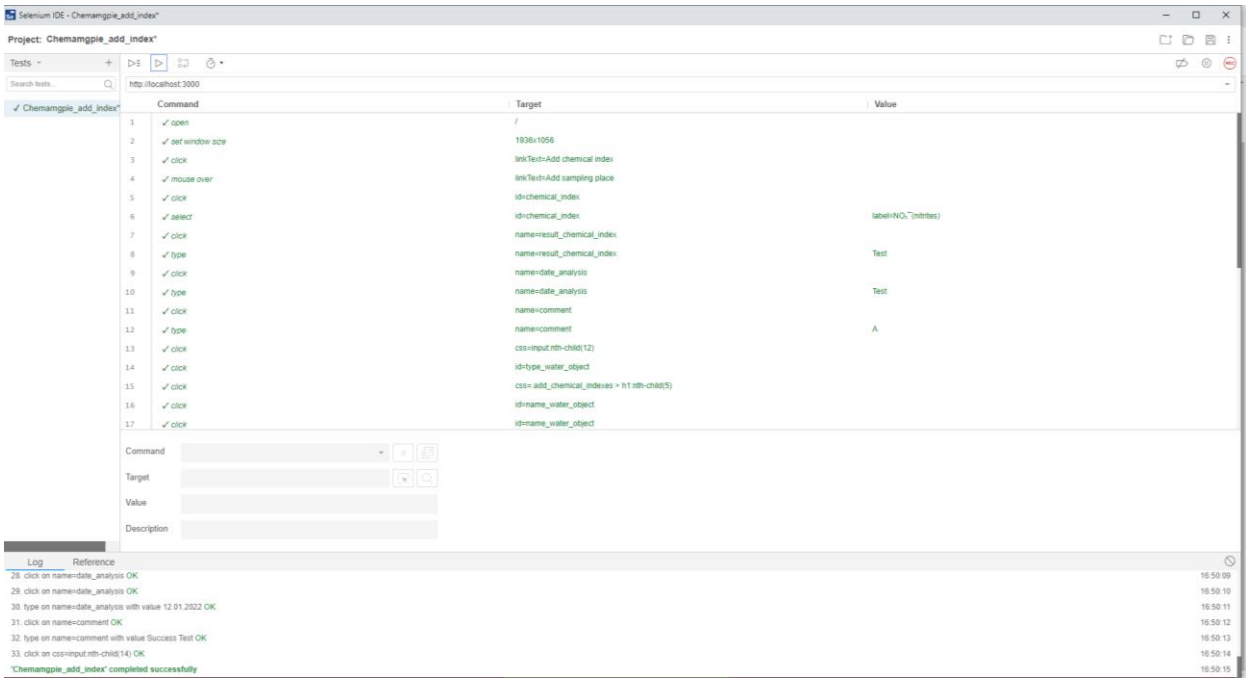

Рисунок В3.3 – Результати тестів інтерфейсу додання хімічного показника в колекцію chemical_indexes

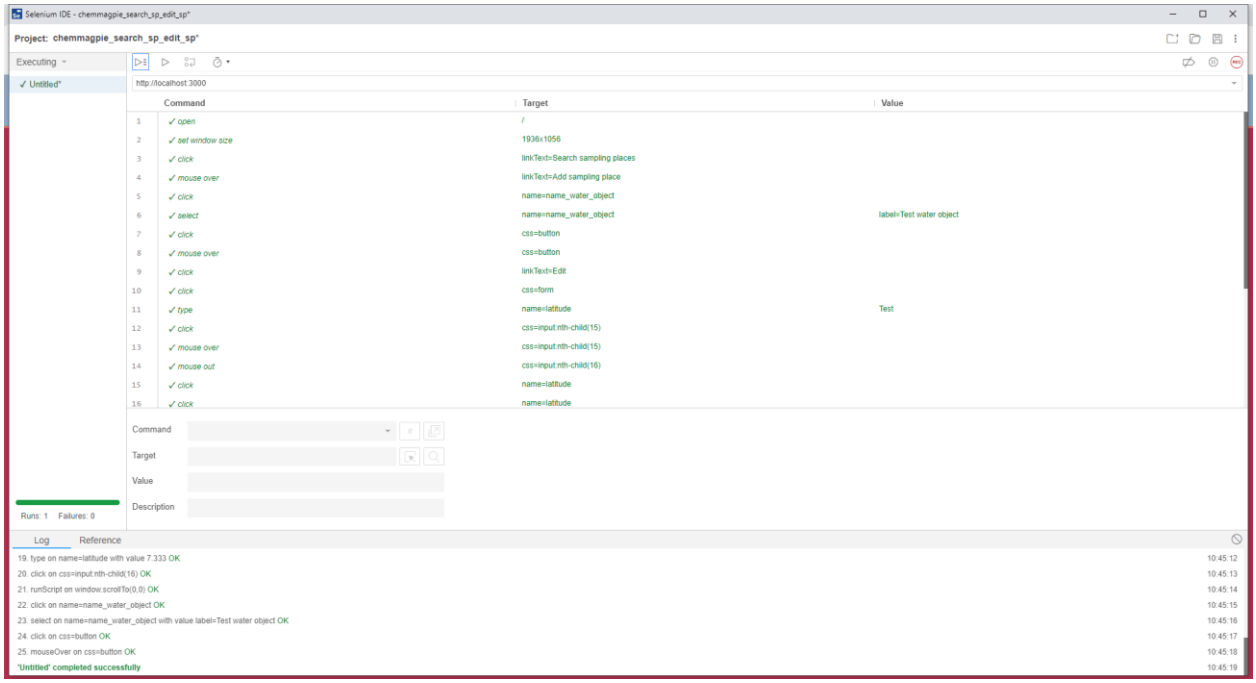

Рисунок В3.4 – Результати тестів інтерфейсу пошуку даних та їх редагування для колекції sampling_places

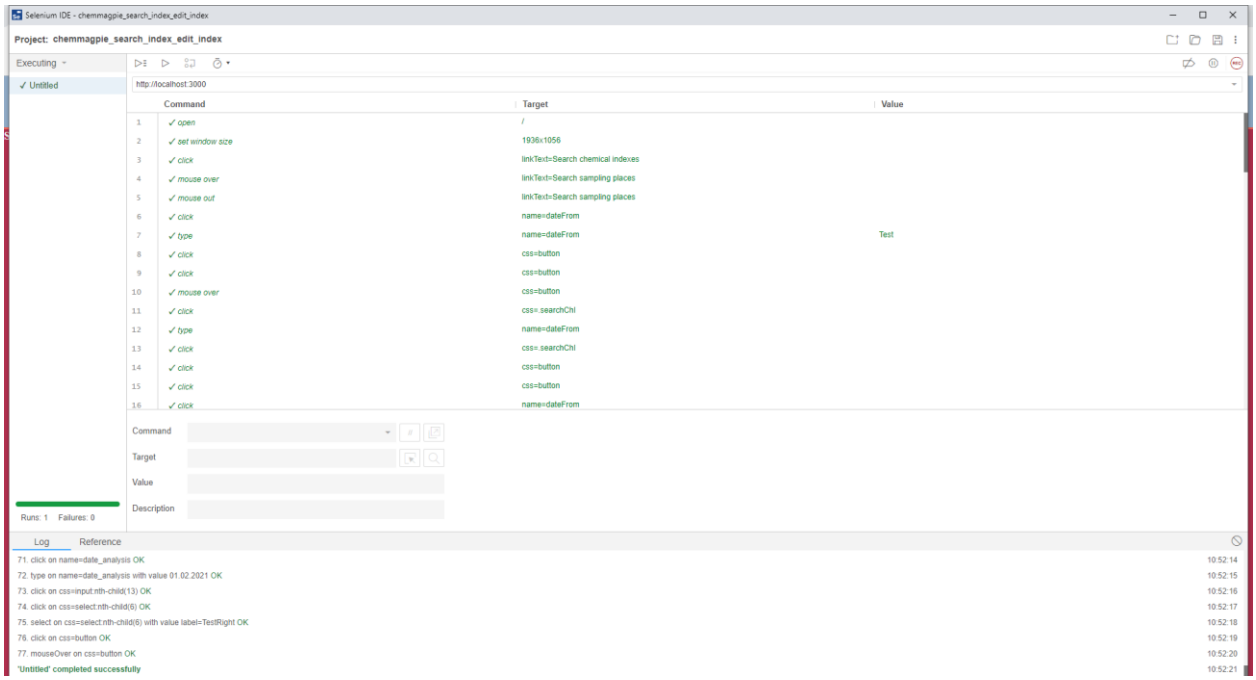

Рисунок В3.5 – Результати тестів інтерфейсу пошуку даних та їх редагування для колекції chemical_indexes

Додаток Г. Результати тестування роботи серверної частини веб-сервісу Chemmagpie

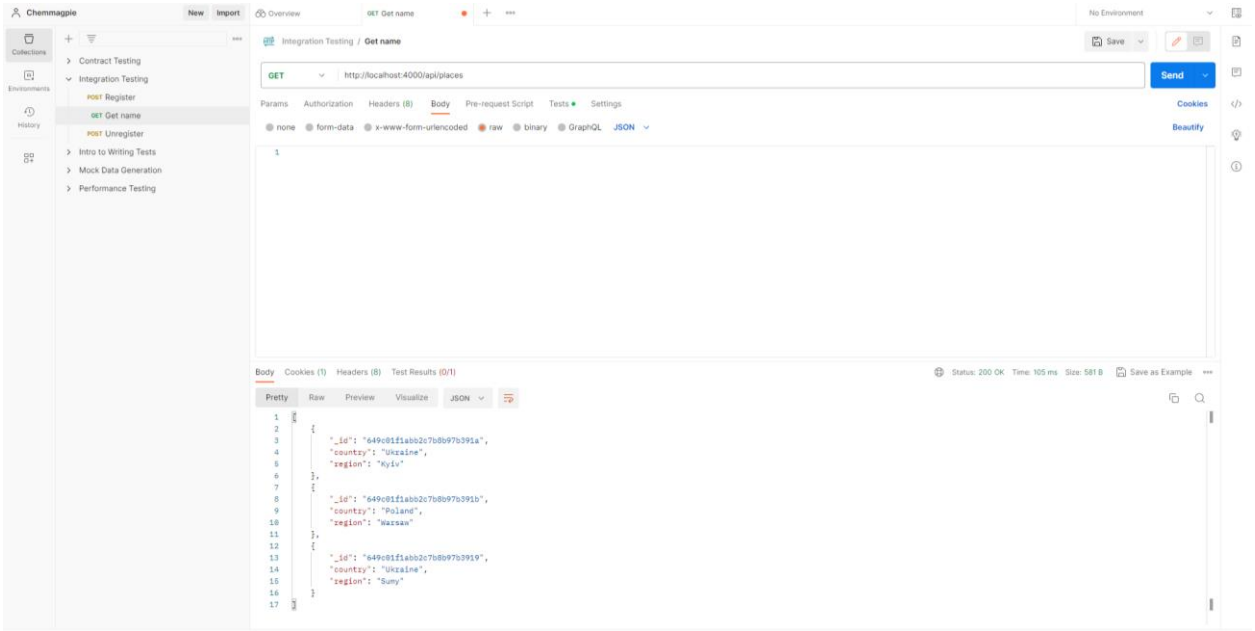

Рисунок Г3.1 – Результати тестування виведення GET методом даних колекції places

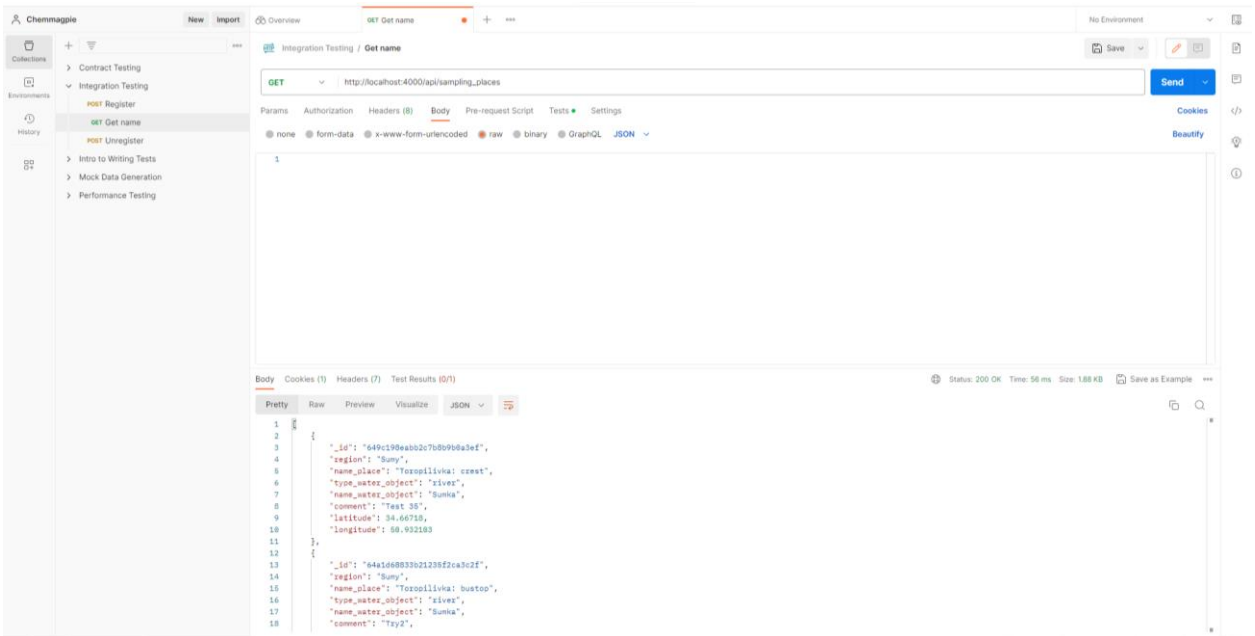

Рисунок Г3.2 – Результати тестування виведення GET-методом даних колекції sampling_places

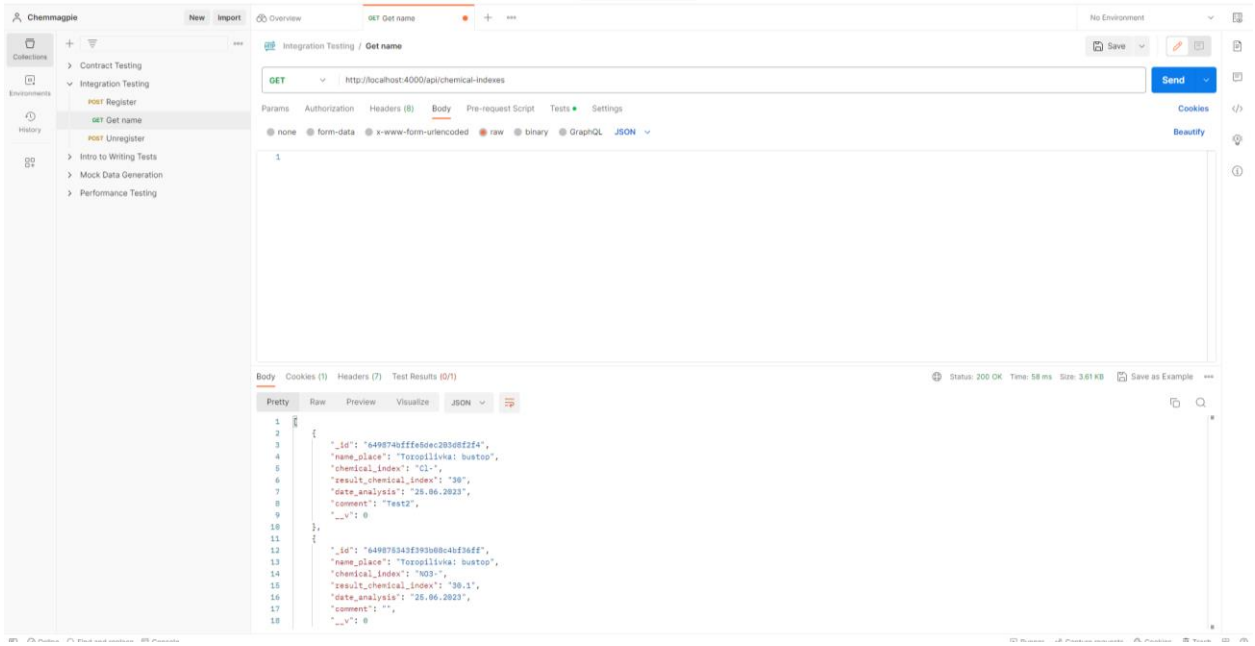

Рисунок Г3.3 – Результати тестування виведення GET-методом даних колекції chemical_indexes

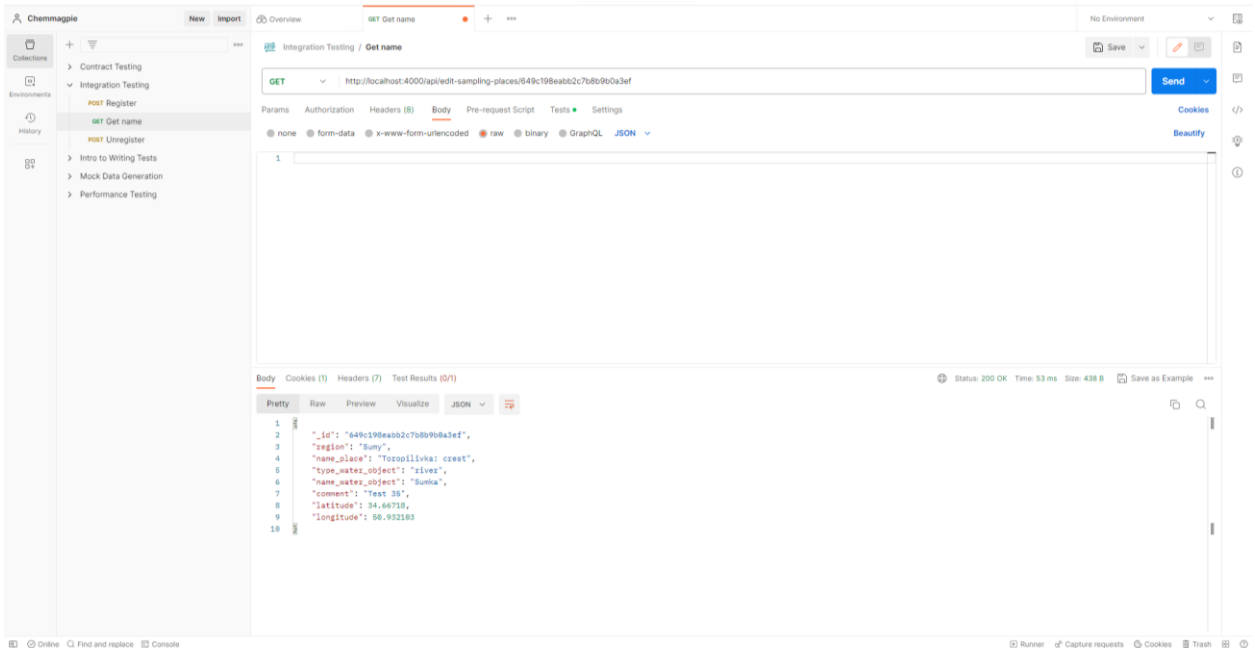

Рисунок Г3.4 – Результати тестування виведення GET-методом за _id конкретного елементу із колекції sampling_places

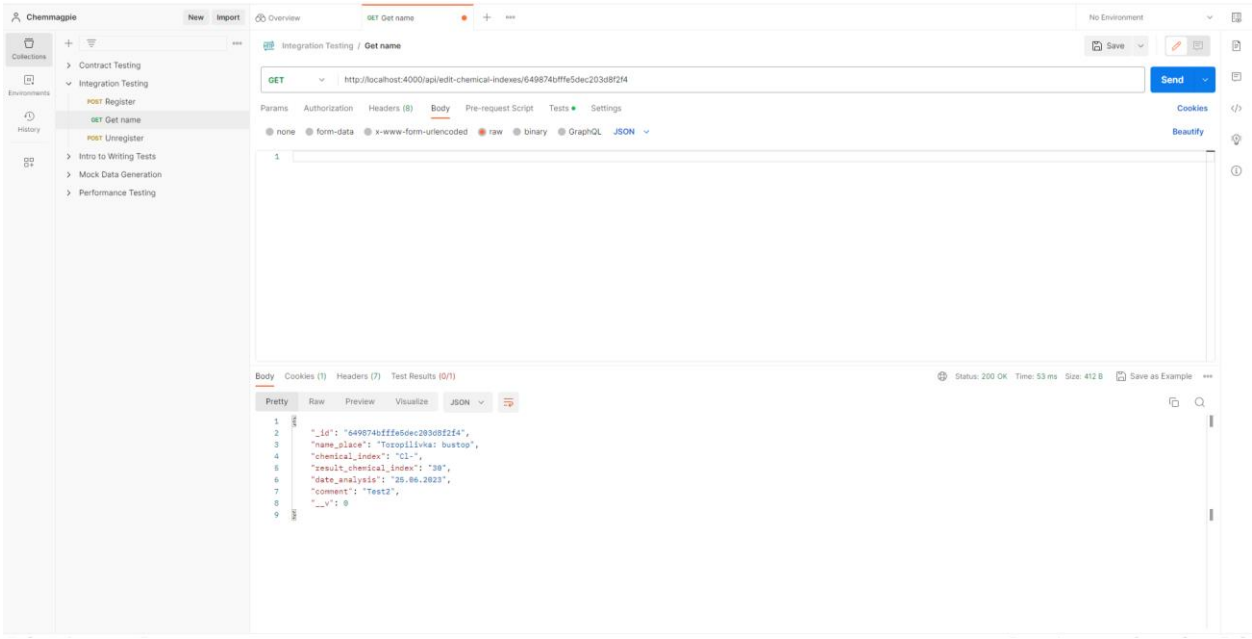

Рисунок Г3.5 – Результати тестування виведення GET-методом результату за _id конкретного елементу із колекції chemical_indexes

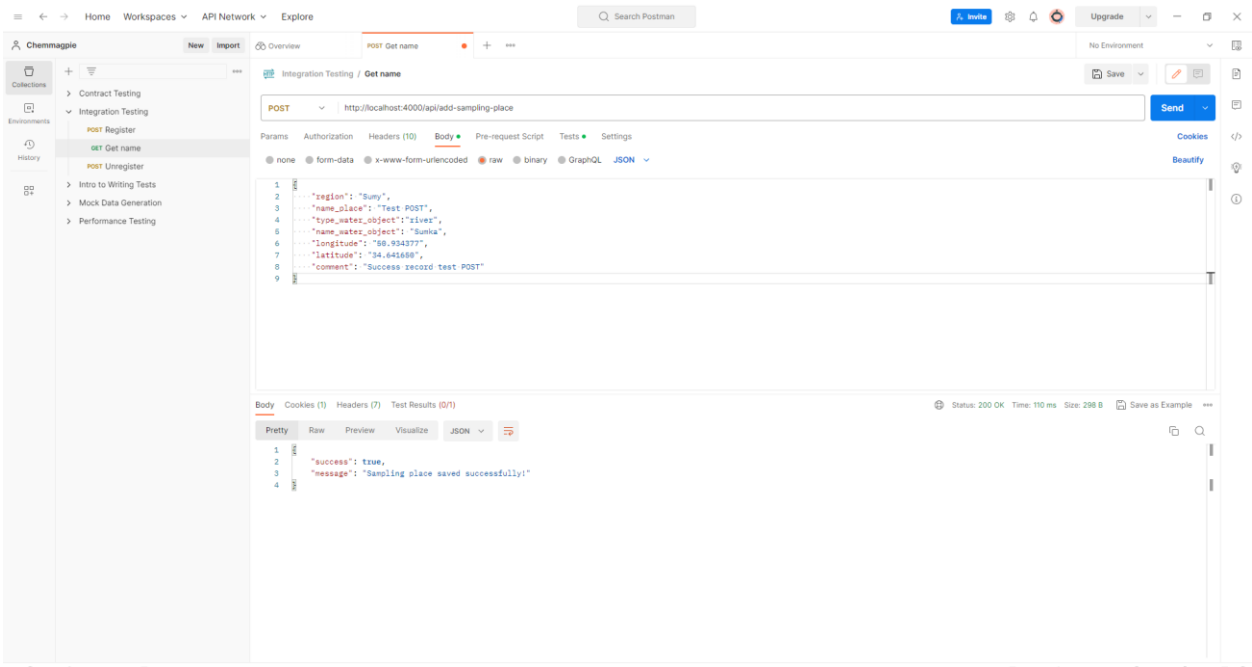

Рисунок Г3.6 – Результати тестування додання даних POST-методом до колекції sampling_places

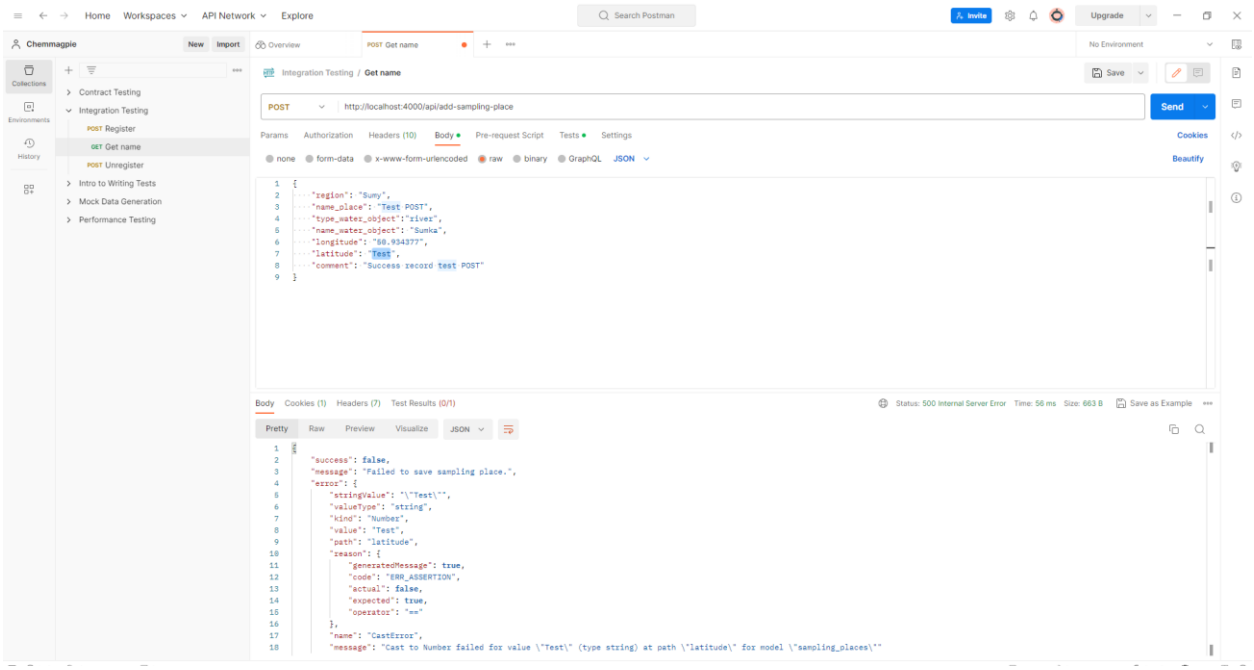

Рисунок Г3.7 – Результати тестування перевірки роботи обробника помилок POST-методом при доданні даних до колекції sampling_places

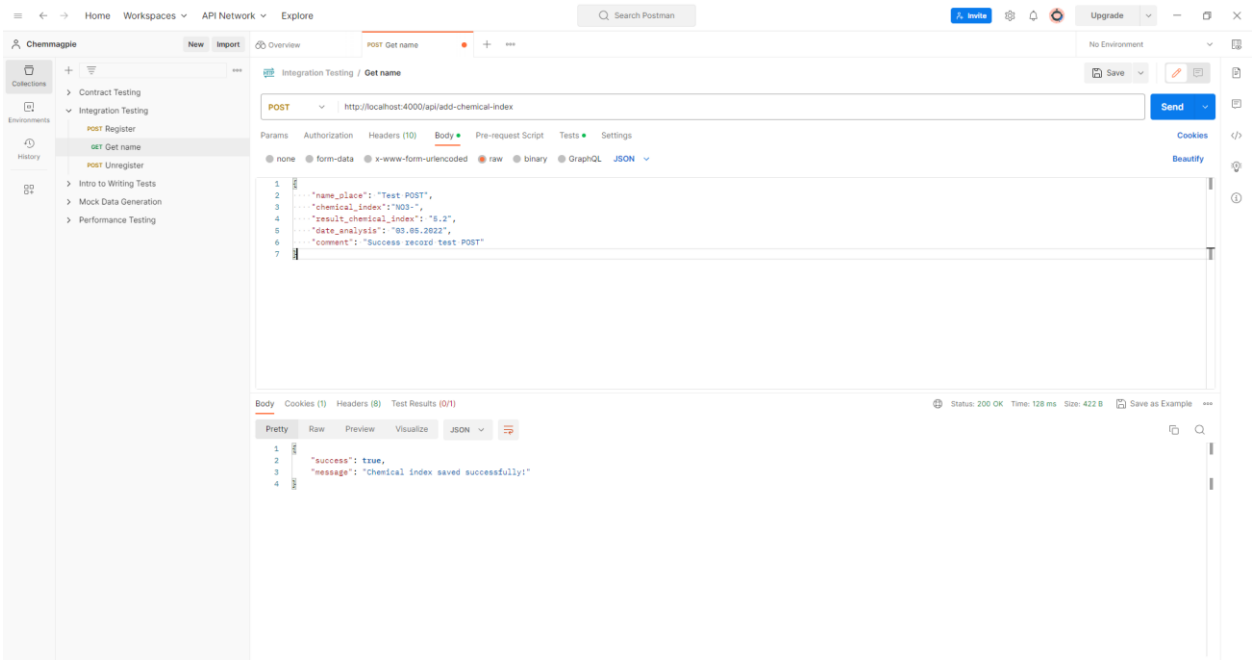

Рисунок Г3.8 – Результати тестування додання даних POST-методом до колекції chemical_indexes при успішному записі

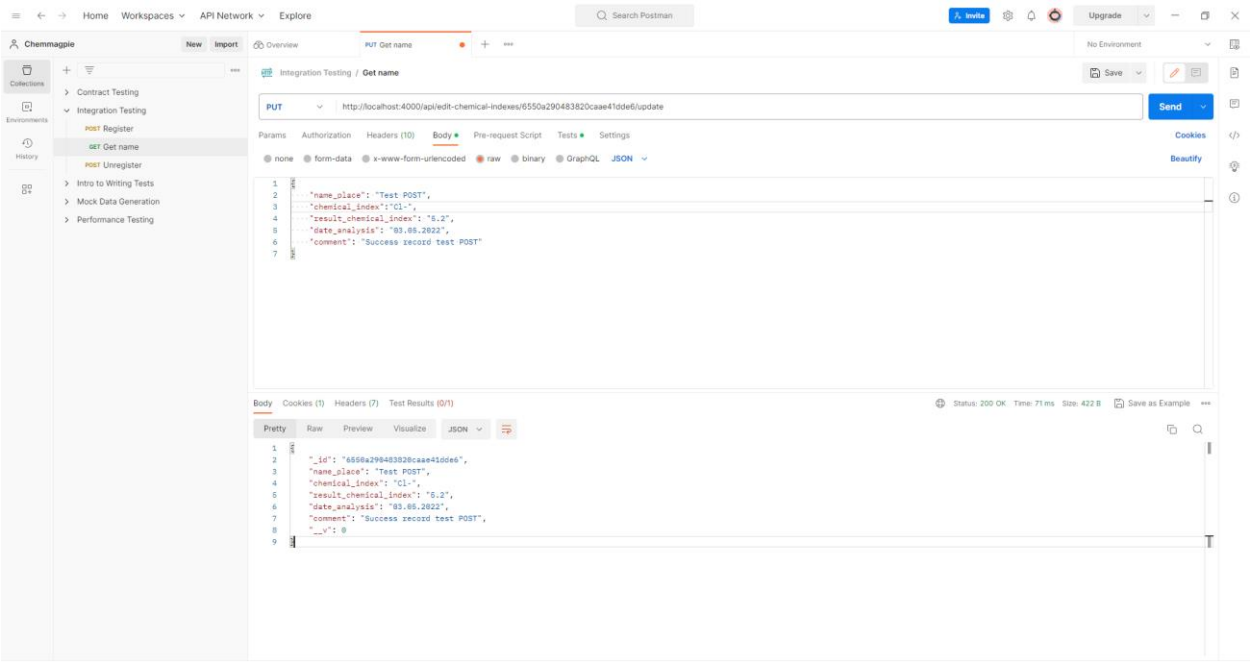

Рисунок Г3.9 – Результати тестування пошуку даних колекції chemical_indexes методом GET при успішному записі

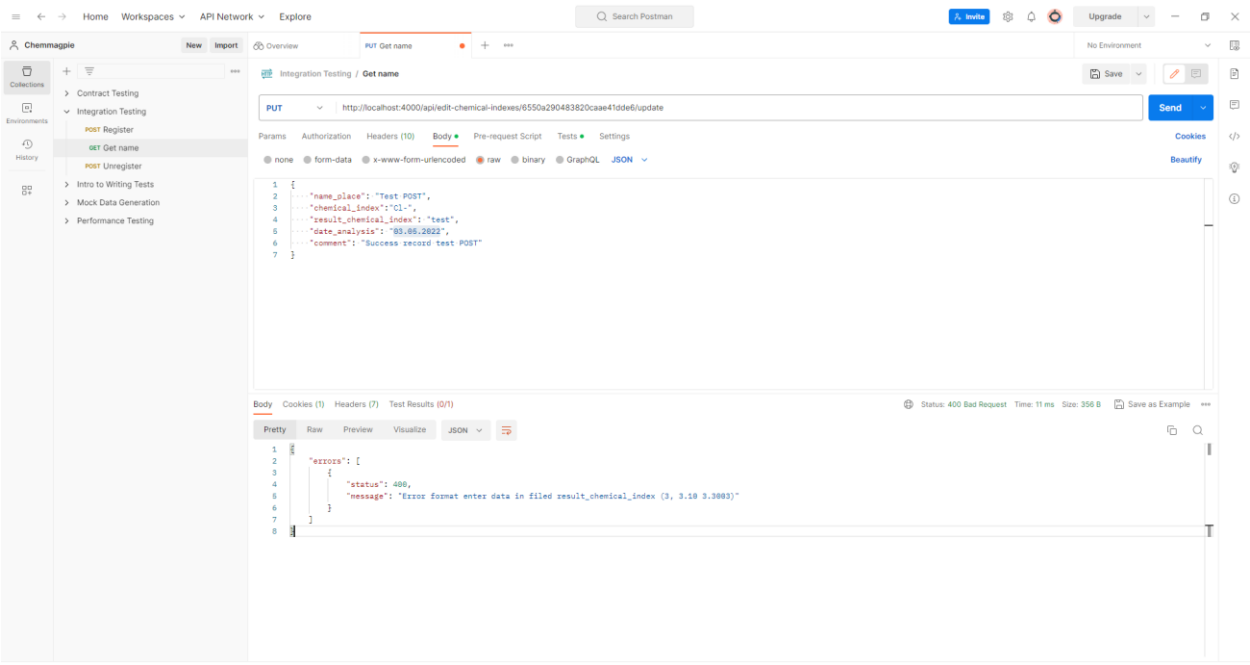

Рисунок Г3.10 – Результати тестування пошуку даних колекції chemical_indexes методом GET при обробці помилок

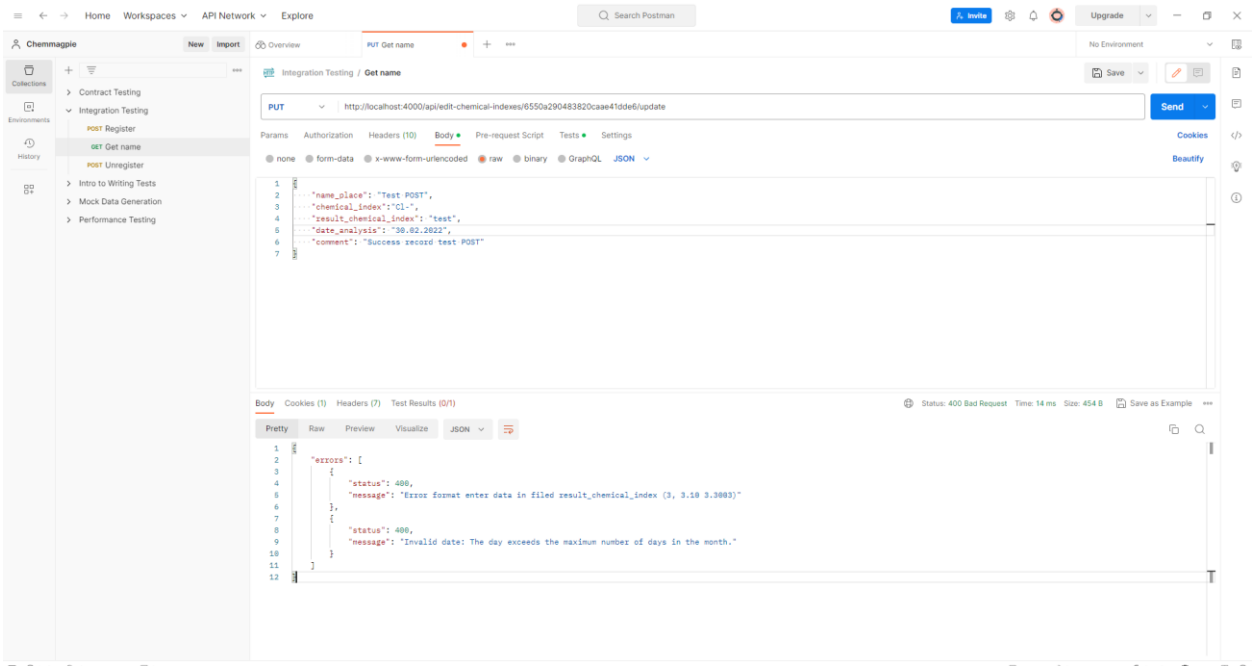

Рисунок Г3.11 – Результати тестування редагування даних колекції chemical_indexes методом PUT при обробці помилок

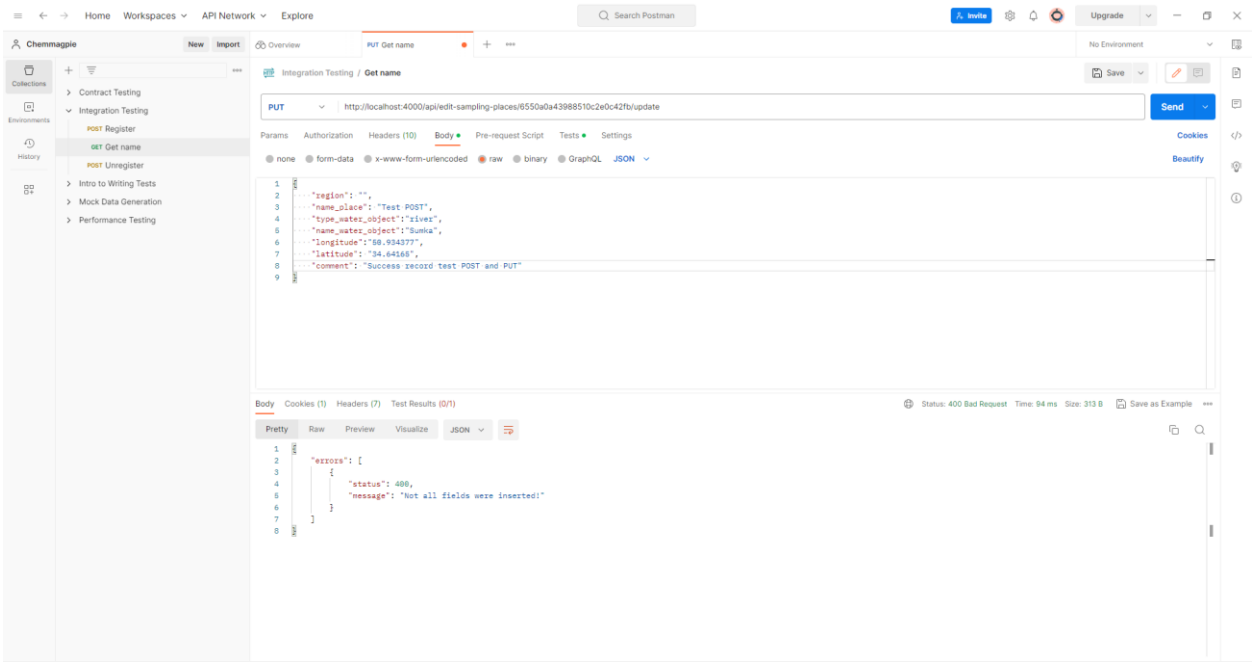

Рисунок Г3.12 – Результати тестування редагування даних колекції sampling_places методом PUT при обробці помилок

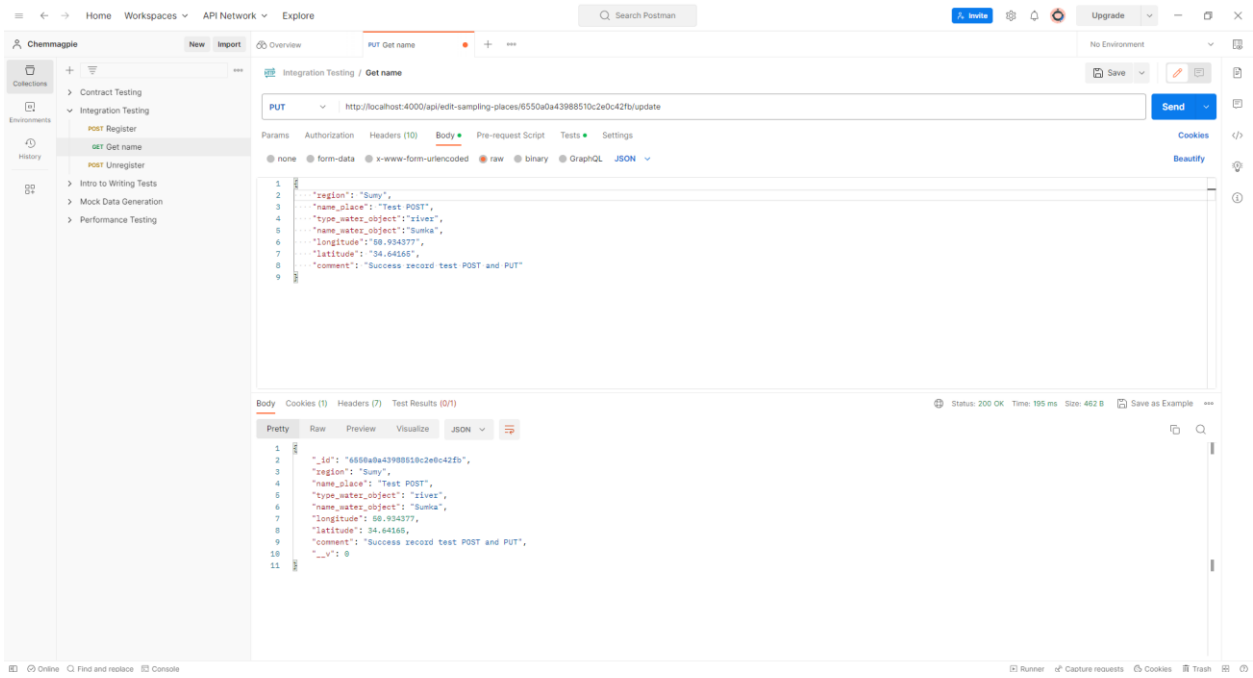

Рисунок Г3.13 – Результати тестування редагування даних колекції sampling_places методом PUT при успішному записі