ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка web-застосунку для ведення обліку водних об’єктів»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

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

_____ Богдана УСЕНКО
(підпис)

Виконав: здобувачка вищої освіти групи ПД-41

Богдана УСЕНКО

Керівник: Юрій АБРОСКІН
асистент кафедри ІПЗ

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Усенко Богдані Олексіївні

1. Тема кваліфікаційної роботи: «Розробка web-застосунку для ведення обліку водних об’єктів»
керівник кваліфікаційної роботи асистент кафедри ІПЗ Юрій АБРОСКІН,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: нормативно-правові акти, структура реєстрів сутностей, геопросторові дані та вимоги до функціоналу, технічна документація з описом бібліотек та фреймворків для реалізації застосунку.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз методів та засобів обліку водних об’єктів.
2. Проектування web-застосунку для ведення обліку водних об’єктів.
3. Розробка та опис функціонування web-застосунку для ведення обліку водних об’єктів.
4. Тестування застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Діаграма бізнес-процесу (BPMN).
6. Діаграма архітектури застосунку
7. Схема бази даних.
8. Екранні форми.
9. Апробація результатів дослідження

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02-04.03.2025	
2	Аналіз та дослідження існуючих аналогів засобів обліку водних об’єктів	05.03-13.03.2025	
3	Аналіз та вибір інструментів для розробки web-застосунку для обліку водних об’єктів	13.03-16.03.2025	
4	Проектування web- застосунку для ведення обліку водних об’єктів	16.03-21.03.2025	
5	Розробка та опис функціонування web-застосунку для ведення обліку водних об’єктів	21.03-23.04.2025	
6	Тестування застосунку	24.04-28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувачка вищої освіти _____ Богдана УСЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Юрій АБРОСКІН
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 60 стор., 1 табл., 38 рис., 20 джерел.

Мета роботи – вирішення проблеми централізованого обліку водних об’єктів за рахунок застосування web-застосунку на основі фреймворку Fastify.

Об’єкт дослідження – процес ведення обліку водних об’єктів.

Предмет дослідження – web-застосунок для ведення обліку водних об’єктів.

Короткий зміст роботи: В роботі проаналізовано методи та інструментальні засоби для ведення обліку водних об’єктів, зокрема Excel, Access, QGIS та європейську систему WISE. Визначено ключові вимоги до функціональності майбутньої системи. Розроблено архітектуру web-застосунку на базі Fastify, PostgreSQL/PostGIS та Vue.js, реалізовано модулі обліку об'єктів, користувачів, інтерактивну карту та систему контролю доступу (RBAC). Проведено тестування основних функцій і перевірено відповідність розробленого програмного забезпечення вихідним вимогам. В роботі використано фреймворк Fastify для розробки серверної частини, Vue.js для створення карти та клієнтського інтерфейсу, Handlebars для створення інтерфейсів карток об’єктів, TailwindCSS як CSS фреймворк для стилізації, maplibre як бібліотека для роботи з картою, PostgresSQL як база даних, GitLab як хмарний репозиторій коду та система контролю версій.

Сферою використання застосунку є організація обліку, моніторингу та управління інформацією про водні об’єкти та їх використання.

КЛЮЧОВІ СЛОВА: FASTIFY, ВОДНІ ОБ’ЄКТИ, ГЕОІНФОРМАЦІЙНА СИСТЕМА, БАЗА ДАНИХ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API — Application Programming Interface.

JSON — JavaScript Object Notation.

IDE — Integrated Development Environment.

VTILE — Vector Tile.

DBMS — Database Management System.

CRUD — Create, Read, Update, Delete.

REST — Representational State Transfer.

MAPBOX — Mapbox Graphics Library.

INSPIRE — Infrastructure for Spatial Information in the European Community.

BPMN — Business Process Model and Notation.

ВСТУП

Обґрунтування вибору теми та її актуальність: У контексті кліматичних змін, інтенсифікації сільського господарства та підвищеного антропогенного навантаження на природні ресурси, питання раціонального використання та контролю за станом водних об’єктів набуває особливої актуальності. В Україні, як і в багатьох країнах, спостерігається фрагментованість у веденні обліку водних ресурсів, що ускладнює контроль за їх використанням, оцінку екологічного стану та виконання міжнародних зобов’язань — зокрема, вимог Водної рамкової директиви ЄС. Відсутність єдиної, централізованої та сучасної інформаційної системи призводить до дублювання інформації, втрати актуальності даних і низької прозорості водокористування.

Створення web-застосунку для обліку водних об’єктів дозволяє вирішити ці проблеми, забезпечуючи централізовану базу даних та аналітичну підтримку управлінських рішень. Використання сучасних технологій, зокрема Fastify для серверної частини та Vue.js для клієнтського інтерфейсу, забезпечує масштабованість, продуктивність та зручність системи для різних груп користувачів – від державних установ до науковців та громадськості.

Об'єкт дослідження – процес ведення обліку водних об’єктів.

Предмет дослідження – web- застосунок для ведення обліку водних об’єктів.

Мета роботи – вирішення проблеми централізованого обліку водних об’єктів за рахунок застосування web-застосунку на основі фреймворку Fastify.

Методи дослідження – системний та структурний аналіз, методи проектування web-застосунків, принципи REST-архітектури, інструменти побудови ГІС-застосунків, методи тестування та валідації інформаційних систем.

Завдання дослідження:

1. Провести аналіз існуючих інструментальні засобів та аналогів інформаційних систем у сфері обліку водних об’єктів. На основі цього аналізу підготувати порівняльний аналіз.

2. Визначити функціональні та нефункціональні вимоги до системи з урахуванням потреб користувачів.

3. Визначитися та ознайомитися з потенційними технологіями, інструментами та бібліотеками, які можуть бути використані для реалізації web-застосунку. Обґрунтувати вибір інструментів розробки.

4. Розробити web-застосунок відповідно до визначених вимог і технологій.

5. Провести тестування системи для перевірки основної функціональності та безпеки.

6. Провести апробацію роботи.

Практичне значення роботи полягає у створенні дієвого інструменту для автоматизованого обліку водних об’єктів, що може бути використаний державними органами, водогосподарськими підприємствами, екологічними організаціями та громадськими активістами. Розроблена система сприяє покращенню якості управління водними ресурсами, забезпечує прозорість у сфері водокористування та створює базу для інтеграції з іншими державними реєстрами та міжнародними платформами моніторингу вод.

Галузь використання – водогосподарська сфера, екологічний моніторинг, державні органи управління природними ресурсами, відкриті дані.

Робота пройшла апробацію на Всеукраїнській науково-технічній конференції «Застосування програмного забезпечення в ІКТ» (24.04.2025, ДУІКТ, м. Київ) та VI Науково-технічній конференції «Сучасний стан та перспективи розвитку IoT» (15.04.2025, ДУІКТ, м. Київ). За результатами участі опубліковано тези доповідей [1] [2].

1 АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ ОБЛІКУ ВОДНИХ ОБ'ЄКТІВ

1.1 Аналіз предметної галузі

Облік водних об'єктів — це процес систематичного збирання і ведення даних про всі водойми (річки, озера, водосховища, ставки тощо) та пов'язані з ними ресурси. Актуальність цієї задачі зумовлена зростаючим навантаженням на водні ресурси та водні біоресурси. Україна належить до країн з обмеженими водними ресурсами та високим рівнем їх використання [3]. Відзначається також значне відхилення показників якості води від нормативів, що свідчить про антропогенний тиск на водойми [3]. Окрім того, надмірна експлуатація риби і незаконне рибальство призводять до виснаження водних біоресурсів. За оцінками ФАО, приблизно 20% світового вилову риби здійснюється незаконно або не обліковується [4], що підкреслює глобальну проблему і потребу в ефективному моніторингу.

Завдання створення системи обліку водних об'єктів полягає у формуванні єдиного інформаційного ресурсу про водойми країни, їхні характеристики та використання. Необхідно детально описати кожен водний об'єкт (географічне положення, площа, об'єм, глибина тощо) і пов'язану інфраструктуру (гідротехнічні споруди), а також інформацію про біоресурси (видовий склад риби, нерестовища, зимувальні ями)[5]. Нерестовища — це ділянки водойм, де відбувається нерест (розмноження) риби, які потребують особливої охорони в сезон розмноження. Зимувальні ями — глибокі ділянки річок чи водойм, де риба концентрується взимку, на цих ділянках забороняється вилов у певні періоди. Врахування таких біотопів є важливим для запобігання браконьєрству в критичні для відтворення риб періоди[5].

Особливістю обліку водних об'єктів є його міждисциплінарність та міжвідомчий характер. Вода як ресурс охоплює кілька сфер – екологію, рибне господарство, водне господарство, земельні відносини[5][6]. Відповідно, дані про

водні об’єкти наразі розпорошені між різними установами: Державним агентством водних ресурсів, Держрибагентством, органами екологічного контролю, місцевими органами влади тощо. Це ускладнює отримання цілісної картини та координацію дій. Тому виникає потреба у централізованому зборі та обробці даних про водні об’єкти, створенні єдиної бази, до якої мають доступ зацікавлені відомства[7].

Крім внутрішніх чинників, важливим стимулом є зобов’язання України щодо гармонізації з законодавством ЄС. Зокрема, Водна рамкова директива ЄС 2000/60/ЕС вимагає інтегрованого управління водними ресурсами на основі басейнового принципу[8]. Україна вже здійснила гідрографічне районування території згідно з цією директивою: у 2016 році затверджено дев’ять районів річкових басейнів як основні одиниці управління водними ресурсами [6]. Перехід до басейнового управління вимагає створення повного реєстру водних об’єктів у межах кожного басейну, з оцінкою їх стану та антропогенного навантаження. Також, у рамках Угоди про асоціацію з ЄС, Україна зобов’язана запровадити заходи проти незаконного, непідзвітного та нерегульованого вилову риби[9].

Розробка системи обліку водних об’єктів продиктована конкретними потребами різних груп користувачів: державних органів, бізнесу та громадськості. Перш за все, державні установи мають потребу в достовірних та оперативних даних для прийняття управлінських рішень. Органи водного господарства повинні відслідковувати кількісний і якісний стан водних ресурсів, режим використання водойм, дотримання лімітів водокористування[5][7]. Органи рибного господарства зацікавлені в обліку нерестовищ, зимувальних ям, контролі вилову та запобіганні незаконному рибальству тощо[5].

Підприємства та організації також є важливими користувачами системи. Орендарі водойм (рибні господарства, аквакультури, СТРГ) повинні мати змогу легко подавати звітні дані про зариблення, вилов, меліоративні роботи тощо, а також отримувати інформацію про режим спеціальних ділянок.

Громадськість та науковці — ще одна цільова аудиторія. Громадяни, особливо населення прибережних територій, рибалки-любителі, екологічні

активісти, зацікавлені в публічному доступі до інформації про водойми: межі водних об’єктів, дати дозволеного та забороненого рибальства, біоресурси водойм тощо[7], [10]. Надання цієї інформації підвищує прозорість управління і сприяє громадському контролю (відповідно до Орхуської конвенції про доступ до екологічної інформації). Наукові установи та освітні заклади можуть використовувати дані системи для досліджень кліматичних змін, екологічного стану, моделювання гідрологічного режиму тощо.

Аналіз потреб показує, що ключовими вимогами користувачів є достовірність, повнота та актуальність даних, а також зручність їх отримання. Державні органи наголошують на необхідності інтеграції системи обліку водних об’єктів з іншими державними інформаційними системами – наприклад, з земельним кадастром (щоб зв’язати водойми із земельними ділянками). Підприємства очікують, що система забезпечить електронну взаємодію з дозвільними органами (отримання дозволів, подача звітності онлайн) і сприятиме спрощенню адміністративних процедур. Таким чином, цільова аудиторія висуває вимогу зробити систему багатофункціональною, але водночас інтуїтивно зрозумілою, з належним рівнем безпеки і конфіденційності для захищених даних.

Для розв’язання задачі обліку водних об’єктів пропонується створення web-застосунку — інтегрованої інформаційно-аналітичної системи, доступної через інтернет. Сучасні інформаційно-комунікаційні технології (ІКТ) дозволяють реалізувати централізовану платформу, що об’єднає різноманітні дані та забезпечить багатокористувацький доступ.

Ведення обліку водних об'єктів в Україні на сьогодні здійснюється за допомогою різних інструментальних засобів, переважно загального призначення. Спеціалізованого програмного забезпечення, призначеного виключно для обліку водних об’єктів на рівні державного чи масового використання, наразі не розроблено. Це зумовлює фрагментацію інформації, її дублювання в різних установах, складність в оперативному оновленні та низьку інтегрованість у міжвідомчих процесах. Наявні інструментальні засоби для організації обліку представлені обмеженим набором рішень, серед яких найбільш поширеними є:

— табличні редактори (Microsoft Excel, Google Sheets), які використовуються для ручного ведення звітів та реєстрів;

— геоінформаційні системи, зокрема QGIS та ArcGIS, що дають можливість просторової прив’язки інформації, але потребують складної конфігурації та не завжди адаптовані під потреби обліку водойм;

— локальні бази даних (наприклад, на базі Microsoft Access), створені на рівні окремих організацій чи регіонів, які не мають централізованого зберігання та обмежені у можливостях масштабування.

1.2 Microsoft Excel

Microsoft Excel — це настільний табличний процесор, один з найбільш популярних та потужних у світі. Він дозволяє створювати електронні таблиці, виконувати розрахунки, аналіз даних і побудову графіків. Для обліку водних об’єктів Excel може слугувати електронним реєстром: користувачі самостійно створюють таблиці з полями, наприклад, назва водойми, тип, площа, місцезнаходження, стан тощо, і заповнюють їх даними. Приклад таблиці для ведення обліку сутності аквакультури зображено на рисунку 1.1.

З основних переваг даного засобу можна виділити наступні:

— простота початку роботи і знайомий інтерфейс – багато хто володіє базовими навичками роботи в Excel;

— програма має потужні засоби обчислення (формули, функції) та агрегування даних (зведені таблиці) [link], що дозволяє аналізувати інформацію про водні об’єкти (наприклад, підрахунок загальної площі водойм, побудова діаграм розподілу за типами тощо);

— підтримує базову перевірку даних і сортування/фільтрацію, полегшуючи впорядкування реєстру;

— інтегрується з іншими продуктами Microsoft Office, що спрощує підготовку звіт

Область	Назва району	Назва ОТГ	Код РВО згідно з Постанови 979 від 30.09.2015	Назва водного об'єкта	Кадастровий номер земельних ділянок під водним об'єктом	Категорія земель	Форма власності земельної ділянки	Розділ	Підрозділ	Назва виду цільового призначення	Місце розташування водного обєкта	Орендодавець (найменування органу виконавчої влади або органу місцевого самоврядування)	Прізвище, власне ім'я та по батькові (за наявності) для фізичних осіб та ФОП/найменування для юридичних осіб	Реєстраційний номер облікової картки платника податків для фізичних осіб та ФОП/код згідно з ЄДРПОУ юридичної особи	Місце проживання для фізичної особи та ФОП/місцезнаходження для юридичної особи (індекс, область, район, населений пункт, вулиця, номер будинку та квартири (за наявності))	Ціль використання водного об'єкта
Івано-Франківська	Верховинський р-н	Верховинська ОТГ	0901000001	Ставок № 1	2620855100:03:001:0253	землі водного фонду	комунальна	10	10.07	Для рибогосподарських потреб	смт.Верховина, Верховинська ТГ, Верховинського р-н, Івано-Франківської обл.	Верховинська ТГ	Токарчук В.І.	2035013392	78701, смт.Верховина, Верховинська ТГ, Верховинського р-н. Івано-Франківської обл.	Для рибогосподарських потреб
Івано-Франківська	Косівський р-н	Косівська ТГ	0901000002	Ставок № 1	2623682401:02:005:0334	Землі сільськогосподарського призначення	комунальна	01	0.13	Для іншого сільськогосподарського призначення	с. Вербовець, Косівська ТГ, Косівський р-н, Івано-Франківська обл.	Косівська ТГ	Рабинюк Василь Іванович	2740713699	78606, с. Старий Косів, Косівський р-н, Івано-Франківської обл.	Для іншого сільськогосподарського призначення

Рис. 1.1 Приклад ведення таблиці Аквакультури в Excel

Однак, таблиці Excel мають свої недоліки, які ускладнюють їх використання, серед них можна зазначити наступні:

— продуктивність падає з дуже великими масивами даних;

— не можлива багатокористувацька робота;

— є ризик втрати даних при відсутності резервних копій.

1.3 QGIS

QGIS (Quantum GIS) — програмне забезпечення, призначене для створення, редагування, візуалізації та аналізу просторових (географічних) даних[11]. На відміну від Excel чи Access, які оперують табличними даними, QGIS працює з картографічними шарами: кожен об’єкт має прив’язку до координат на мапі. Для обліку водних об’єктів QGIS є надзвичайно цінним, оскільки дозволяє відображати річки, озера, ставки на географічній карті, зберігаючи при цьому атрибутивні дані про них (назва, характеристики) у таблицях, пов’язаних з просторовими об’єктами. QGIS поєднує функції бази даних і карти. Приклад проекту в QGIS з відображенням водного об’єкту зображено на рисунку 1.2.

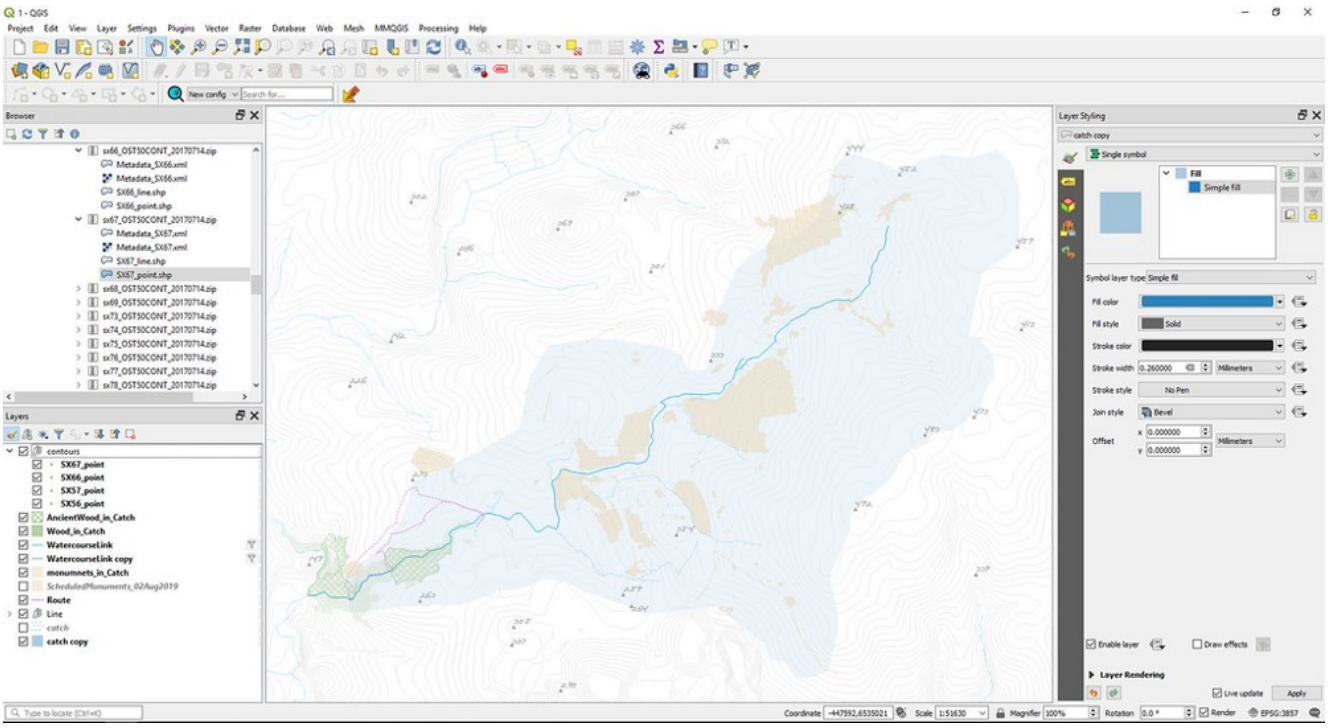


Рис. 1.2 Приклад проекту в QGIS з відображенням водних об’єктів

Також QGIS надає можливість перегляду детальної інформації про об’єкт. На рисунку 1.3 зображено інтерфейс програмного забезпечення з інформацією про об’єкт.

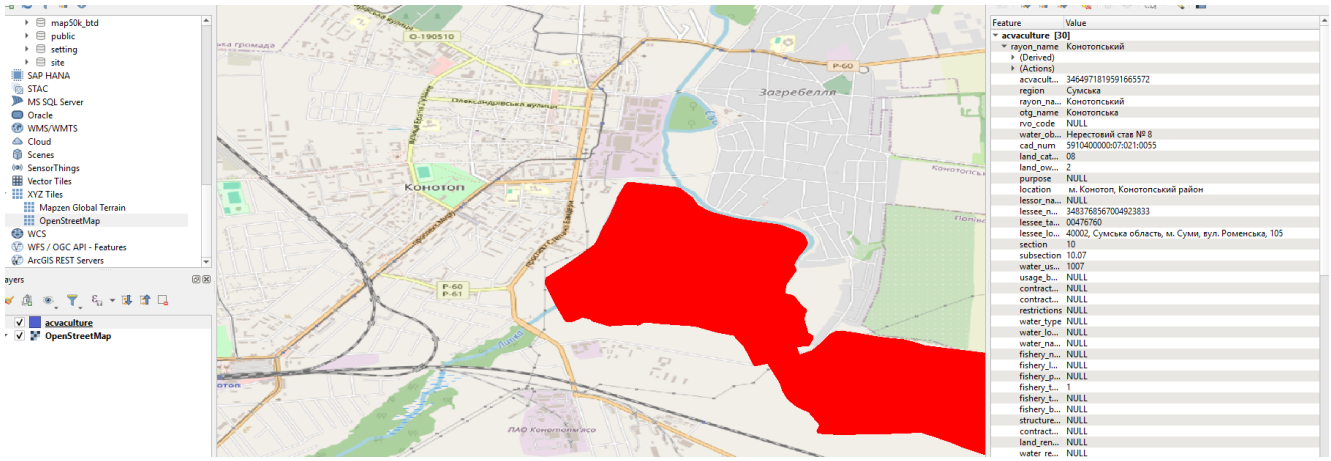


Рис. 1.3 Перегляд детальної інформації про об’єкт через QGIS

Серед переваг QGIS можна виділити наступні:

— відкритий код та безкоштовність — не потребує придбання ліцензії, що робить його доступним для державних установ, освітніх організацій та невеликих підприємств;

— широка підтримка форматів — QGIS працює з великою кількістю форматів просторових і табличних даних: SHP, GeoJSON, KML, GPKG, CSV, PostGIS тощо;

— гнучке редагування даних — дає змогу вносити зміни як у геометрію об’єктів на карті, так і в пов’язані атрибутивні таблиц;

— підтримка плагінів — доступна велика бібліотека плагінів для геоаналізу, автоматичного завантаження кадастрових даних, підключення до OpenStreetMap, побудови ізоліній, буферизації тощо;

— сумісність з PostgreSQL/PostGIS — дозволяє напряму працювати з просторовими базами даних без експорту/імпорту, підтримуючи синхронізацію та живе редагування.

— інструменти підготовки картографічної продукції — включає засоби для створення звітів, друкованих карт, легенд та схем у зручному форматі.

Серед недоліків QGIS можна виділити наступні:

— крива навчання — потребує знань ГІС, типів геоданих, CRS (систем координат), SQL-запитів для PostGIS тощо, що ускладнює вхід для користувачів без досвіду;

— потреба в ресурсах — при роботі з великими шарами, ортофотопланами чи 3D-даними можливе зниження продуктивності на слабких комп’ютерах;

— менша продуктивність у порівнянні з комерційними ГІС, порівняно з комерційними ГІС-системами на зразок ArcGIS, що мають власні оптимізовані механізми обробки;

— відсутність хмарної версії — робота з QGIS здійснюється локально, а для публікації проєктів необхідна додаткова конфігурація (QGIS Server або сторонні сервіси типу Lizmap).

1.4 WISE

Значну увагу заслуговують закордонні приклади впровадження подібних систем. Зокрема, одним із найбільш структурованих прикладів є WISE (Water Information System for Europe) — інтегрована платформа збору, обробки, управління та публічного поширення інформації про водні ресурси країн Європейського Союзу[12]. Система створена спільно Європейською Комісією, Європейським агентством з довкілля (EEA) і державами-членами ЄС для реалізації вимог Водної рамкової директиви (2000/60/ЕС) та інших європейських водних директив [8].

Основні можливості платформи WISE включають: надання доступу до баз даних за країнами та басейнами річок, інтерактивні картографічні інтерфейси, функції порівняння водної ситуації між регіонами, а також автоматизоване формування аналітичних звітів. Платформа підтримує стандарти INSPIRE для

геопросторових даних, що дозволяє забезпечити уніфікований підхід до моніторингу стану водних ресурсів у Європі [8].

До ключових переваг WISE можна віднести:

— комплексну інтеграцію даних із багатьох джерел;

— прозорий публічний доступ до екологічної інформації;

— розвинені інструменти візуалізації;

— можливість використання аналітичних модулів;

— відповідність директивам ЄС та відкритий формат даних.

Водночас, система має і певні недоліки:

— повільне оновлення інформації (інтервал 2–3 роки), що унеможливлює використання платформи для оперативного реагування;

— відсутність детальних даних на локальному рівні (наприклад, по окремих водоймах або користувачах водних ресурсів);

— відсутність українських водних об’єктів у структурі, що ускладнює інтеграцію або обмін інформацією без додаткової адаптації;

— високий поріг входу для повноцінного використання функціоналу через складну навігацію й велику кількість технічної інформації.

На рис. 1.4 зображено сторінку сайту WISE [8] з відображенням шару WFD2022 Surface water body (Поверхневі водойми).

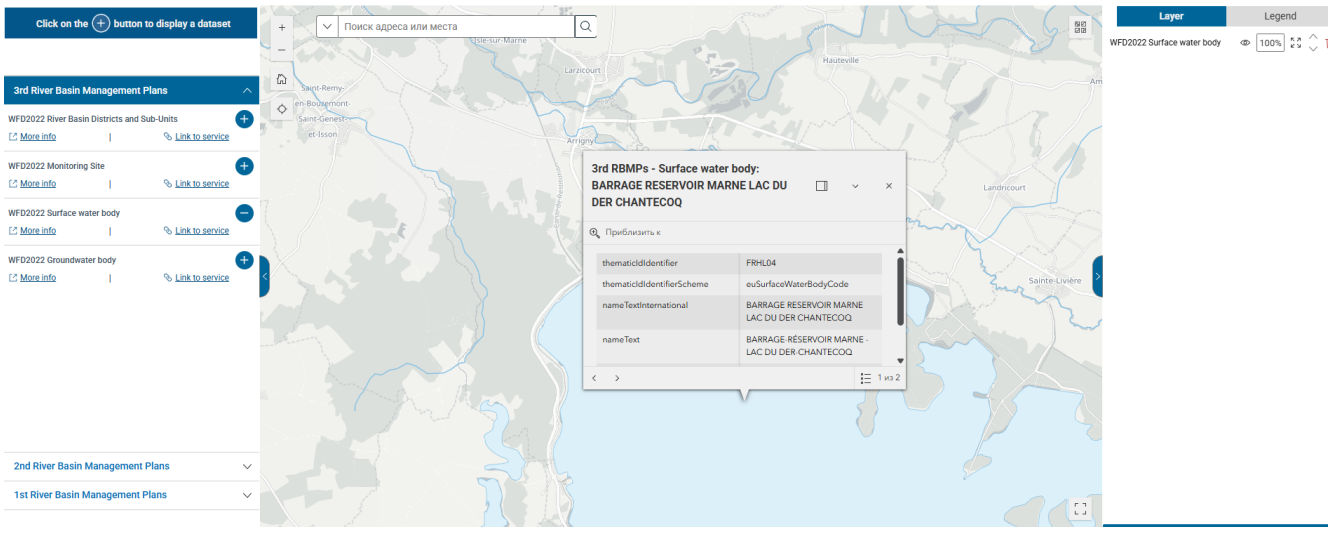


Рис. 1.4 Сторінка сайту WISE з відображенням шару WFD2022 Surface water body

На рис. 1.5 зображено сторінку з відображенням шару WFD2022 Groundwater body (Підземні води).

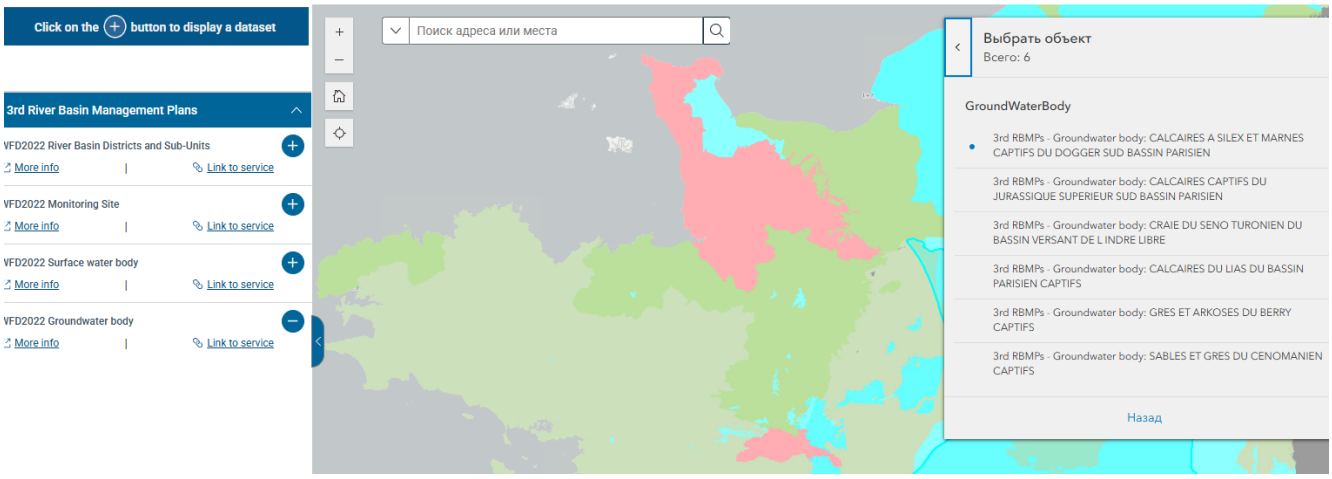


Рис. 1.5 Сторінка сайту WISE з відображенням шару WFD2022 Surface water body

Досвід впровадження платформи WISE демонструє, що ефективна система обліку водних ресурсів повинна ґрунтуватися на відкритості, геопросторовій структурі даних, автоматизації звітності та відповідності міжнародним стандартам. При розробці національної системи обліку водних об’єктів доцільно адаптувати ключові концепції WISE з урахуванням потреб локального управління, оперативності та доступності інформації для державних органів, водокористувачів та громадськості.

1.5 National Water Information System (NWIS)

National Water Information System (NWIS) [13] — це національна інформаційна система збору, зберігання та розповсюдження гідрологічних даних, створена Геологічною службою США (*United States Geological Survey*, USGS). NWIS функціонує як центральне сховище даних про водні ресурси, зібраних мережею гідрологічних станцій, лабораторій та польових вимірювань на території США. Розробка цієї системи розпочалася ще у 1970-х роках, і з того часу вона

постійно оновлюється з метою вдосконалення доступу до водної інформації як для державних органів, так і для наукової та громадської спільноти.

Основне призначення NWIS — забезпечення публічного доступу до надійних, просторово-часових даних про водні ресурси (поверхневі, підземні, якість води тощо) для підтримки моніторингу, планування, досліджень та прийняття управлінських рішень. До ключових переваг WISE можна віднести:

— великий обсяг історичних даних — NWIS зберігає дані, зібрані за понад 100 років, що дозволяє аналізувати довготривалі тренди водокористування, кліматичних змін та гідрологічних явищ;

— реальний час — значна частина станцій передає дані у режимі real-time (оновлення щогодини або частіше), що забезпечує актуальність інформації для аварійного реагування чи прогнозування повеней;

— географічне охоплення — система охоплює всю територію США та території Пуерто-Рико, Гуаму та Американських Самоа, включаючи тисячі станцій моніторингу;

— інтеграція з іншими системами — NWIS інтегрована з національними й міжнародними платформами, такими як EPA STORET, WISE, CIDA, що сприяє стандартизації дани;

— публічний API — USGS надає API для отримання даних у форматах JSON, XML або таблиць, що спрощує автоматизацію аналізу.

Водночас, система має і певні недоліки:

— фокус на США — NWIS орієнтована винятково на внутрішні потреби США, тож для глобальних задач вона потребує інтеграції з іншими системами (напр., WISE, AQUASTAT);

— складність інтерфейсу — має сучасну web-версію, але користувачі вважають її перевантаженою і важкою для навігації;

— обмежена деталізація у деяких регіонах — у малонаселених або важкодоступних регіонах (наприклад, Аляска) покриття гідрологічними станціями є менш щільни.

NWIS надає візуалізацію та завантаження таких типів даних:

— поверхневі води (Surface Water): рівень води, витрата (дебіт), температура, швидкість потоку тощо;

— підземні води (Groundwater): глибина до дзеркала води, тривалі тренди підняття/спаду рівня;

— якість води (Water Quality): хімічний склад, вміст нітратів, важких металів, pH, температура, мутність тощо;

— водокористування (Water Use): використання води в побуті, сільському господарстві, промисловост;

— кліматичні параметри: опади, температура повітря, випаровування (на окремих станціях).

Кожна станція має карту, таблицю спостережень, графіки, та можливість експорту у CSV, RDB, JSON, XML. Крім того, є інтерактивна карта з пошуком за географією або станціями. NWIS є одним з найповніших прикладів національної платформи для гідрологічного обліку, яка демонструє, як ефективно поєднувати польові вимірювання, бази даних, візуалізацію та API-доступ.

Аналіз цієї системи дозволяє зрозуміти важливість централізованого, відкритого й стандартизованого обліку водних ресурсів, що може бути взято за основу при вдосконаленні українських аналогів. На рисунку 1.6 зображено сторінку NWISE з картою шаром станцій.

На рисунку 1.7 зображено інформаційну картку об’єкту, яка доступна по кліку з картки об’єкта на карті. Картка відображає основну інформацію про об’єкт, а саме: координати, місцерозташування, глибину і тд.

Рис. 1.6 Сторінка картки порталу NWISE

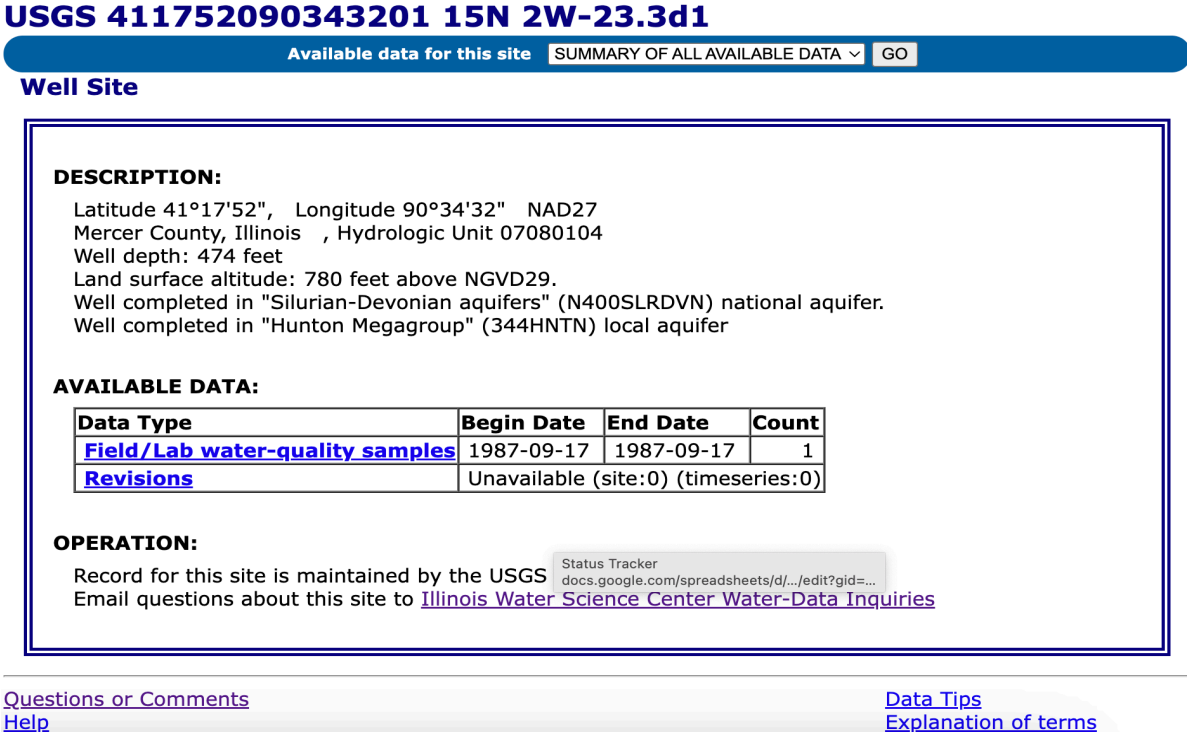


Рис. 1.7 Інформаційна картка об’єкту

1.6 Порівняльна таблиця

Таблиця 1.1

Порівняльна таблиця аналізу існуючих засобів для обліку водних об’єктів

Показник	**Excel**	**QGIS**	**WISE**
Платформи	IOS, Windows	IOS, Windows, Linux	портал
Вартість	Платний	Безкоштовний	Безкоштовно для публічного доступу
Структура даних	Плоскі таблиці	Просторові шари + атрибутивні таблиці.	Бази даних; підтримка стандарту INSPIRE
Багатокористувацька робота	-	-	+
Аналітичні можливості	+ (Потужні обчислення, функції, зведені таблиці і тд.)	+ (Просторовий аналіз (буфери, перетини), вимірювання площ/відстаней, візуалізація на карті; атрибутивні запити.)	+ (порівняння стану води за регіонами, аналіз тенденцій, інтегровані звіти)
Візуалізація даних	Таблична форма; графіки і діаграми.	Карти, шари, тематичне відображення ,прив’язка докоординат.	Інтерактивні карти, діаграми, таблиці
Простота використання	Відносно легкий старт для базових задач.	Високий поріг входження для новачка без ГІС-навичків.	Висока для спеціалістів, середня для пересічних користувачів

Продовження таблиці 1.1

Порівняльна таблиця аналізу існуючих засобів для обліку водних об’єктів

Показник	**Excel**	**QGIS**	**Access**	**WISE**
Безпека і надійність	Дані зберігаються локально; є ризик втрати при відсутності резервних копій; можна захистити файл паролем.	Дані локально або на сервері організації; контроль доступу на рівні файлу/мережі	Висока (європейські стандарти захисту даних)	
Приклади використання	Реєстри водойм у Excel на рівні громади чи області; зведені таблиці водокористування.	Геопортал Держводагентства (водний кадастр) водойм у проєктах.	Локальні бази в водогосподарських організаціях.	Оцінка виконання Водної рамкової директиви, моніторинг стану водних тіл у ЄС.

2 ПРОЕКТУВАННЯ WEB-ЗАСТОСУНКУ ДЛЯ ВЕДЕННЯ ОБЛІКУ ВОДНИХ ОБ'ЄКТІВ

2.1 Визначення вимог до web-застосунку

У рамках даної роботи розробляється web-застосунок для ведення обліку водних об'єктів на платформі Node.js із використанням фреймворку Fastify та мови JavaScript. Система має забезпечувати централізоване зберігання, редагування й візуалізацію інформації про річки, ставки, водосховища та пов'язані з ними сутності (аквакультура, СТРГ, промисел тощо). Доступ до функціоналу здійснюється через браузерний клієнт, побудований на Vue3 (Vite) із компонентами Tailwind CSS / Preline UI та інтерактивною картою MapLibre GL JS (бібліотека для створення карт), що відображає просторові дані з PostGIS.

Основний функціонал поділяється на кілька підсистем: REST API, яке надає CRUD-операції над усіма реєстрами; картографічний backend модуль (tiles); підсистему автентифікації та RBAC-керуванням ролями, а також адміністративну панель для управління користувачами й правами доступу. Для зберігання структурованої та геоінформації використовується PostgreSQL із розширенням PostGIS 3.4, що забезпечує транзакційну цілісність і підтримку GIS-запитів (ST_Intersects, ST_Within). Враховуючи очікуване навантаження на читання та фільтрацію, для кешування довідників і тайлів задіяно Redis.

Проєктування системи виконується з дотриманням принципів «чистої» (hexagonal) архітектури та шаблонів проєктування, що гарантує незалежність доменної логіки від зовнішніх інфраструктурних шарів і забезпечує подальшу масштабованість рішення.

Розробка ведеться у кросплатформеному IDE Visual Studio Code з набором розширень ESLint, Prettier.

Поставлене технічне завдання має на меті створення надійної та ефективної інформаційної системи, здатної обробляти значні обсяги просторових і атрибутивних даних у режимі реального часу, забезпечуючи мінімальні затримки,

високий рівень захисту інформації та зручний, інтуїтивно зрозумілий інтерфейс для кінцевих користувачів і адміністраторів.

2.2 Обґрунтування вибору засобів розробки

Вибір веб-платформи обґрунтований її універсальністю та зручністю розгортання. Web-застосунок доступний через браузер на різних пристроях без потреби в інсталяції спеціального ПЗ, що забезпечує крос-платформність та широке охоплення користувачів [14]. Оновлення та підтримка такого застосунку спрощуються: усі зміни вносяться на сервері, й користувачі одразу отримують оновлену версію при наступному відвідуванні сайту. Враховуючи, що система обліку водних об’єктів може мати багато користувачів, веб-підхід дозволяє забезпечити централізований доступ до єдиної бази даних у режимі реального часу. Крім того, web-застосунок легко інтегрується з картографічними сервісами та геоінформаційними системами для візуалізації водних об’єктів, що є суттєвою перевагою для даної предметної області.

До складу обраного стеку розробки входять наступні засоби:

— Fastify – серверний веб-фреймворк для Node.js, що забезпечує високу продуктивність та розширюваність [15].

— PostgreSQL – реляційна система керування базами даних з підтримкою складних транзакцій [16].

— PostGIS – розширення до PostgreSQL для роботи з геопросторовими даними [17].

— Tailwind CSS – CSS-фреймворк для швидкої адаптивної стилізації інтерфейсу.

— Preline UI – бібліотека Tailwind-компонентів для побудови UI на основі готових блоків .

— Handlebars – шаблонізатор HTML для серверної генерації інтерфейсу.

— Vue.js – JavaScript-фреймворк для створення реактивних інтерфейсів користувача[18] .

— GitLab – система керування версіями з CI/CD-підтримкою та можливістю спільної командної роботи.

— Visual Studio Code – кросплатформене інтегроване середовище розробки (IDE), оптимізоване для роботи з JavaScript, HTML, CSS, базами даних та REST API .

2.2.1 Fastify

Для реалізації бекенд-частини застосунку використано Fastify – сучасний фреймворк для Node.js, який забезпечує обробку великої кількості запитів із мінімальними затримками. Його особливості:

— підтримка асинхронної обробки HTTP-запитів;

— вбудовані інструменти для валідації вхідних даних;

— підтримка плагінної архітектури, що полегшує масштабування;

— зручна інтеграція з ORM-бібліотеками, механізмами логування та автентифікації.

Fastify дає змогу ефективно обробляти запити на отримання, фільтрацію й агрегацію даних навіть при високому навантаженні, що особливо важливо для системи з великою кількістю користувачів і складною логікою.

2.2.2 PostgreSQL та PostGIS

У якості системи керування базами даних використано PostgreSQL, яка відома своєю надійністю, транзакційною цілісністю та широкою підтримкою складних SQL-запитів. Для зберігання геоданих застосовується розширення PostGIS, яке дозволяє:

— зберігати геометрії (точки, лінії, полігони);

— виконувати просторові запити (наприклад, ST_Within, ST_Intersects, ST_Distance), може знадобитися для обробки даних та API VTILE);

— експортувати дані у форматі GeoJSON для візуалізації на карті.

Це забезпечує повну інтеграцію між просторовими об’єктами та атрибутивною інформацією, необхідною для обліку водойм, орендарів, користувачів тощо.

2.2.3 Tailwind CSS та Preline U

Для стилізації фронтенду обрано Tailwind CSS – CSS-фреймворк, що дозволяє розробляти адаптивні інтерфейси без створення окремих CSS-файлів.

Його переваги:

— швидкість верстки;

— зручність підтримки стилю;

— гнучка система класів.

Додатково використано бібліотеку Preline UI – набір готових Tailwind-компонентів (таблиці, форми, модальні вікна, вкладки), які прискорюють реалізацію інтерфейсу згідно з єдиним дизайном.

2.2.4 Handlebars

Для генерації HTML-сторінок на стороні сервера було обрано Handlebars – логічно простий і потужний шаблонізатор, який дозволяє:

— розділити структуру сторінки від логіки контролерів;

— створювати багаторазові шаблони (наприклад, для карток об’єктів, таблиць, повідомлень);

— використовувати змінні прямо в HTML-структурі, що підвищує читабельність коду.

Цей інструмент ідеально підходить для рендерингу сторінок з даними, отриманими з БД, без зайвого дублювання коду.

2.2.5 Vue.js

Клієнтська частина застосунку реалізована з використанням Vue.js – прогресивного JavaScript-фреймворку, що забезпечує:

— динамічну взаємодію з інтерфейсом (наприклад, відображення інформації при натисканні на карту);

— асинхронне завантаження даних через API;

— реактивну перевірку форм у режимі реального часу;

— гнучке компонування інтерфейсу та повторне використання елементі;

— Vue.js дозволив значно підвищити зручність користувацького досвіду у взаємодії з картою та довідниками.

2.2.6 Visual Studio Code та пакетний менеджер npm

Розробка застосунку виконувалася у середовищі Visual Studio Code — кросплатформеному IDE, що надає:

— підсвітку синтаксису для JavaScript/TypeScript, SQL, HTML / CSS;

— вбудовану інтеграцію з Git та перегляд історії комітів;

— зручний термінал, повноцінний налагоджувач (debugger) і менеджер розширень;

— підтримку інструментів форматування та статичного аналізу коду (Prettier, ESLint тощо).

Ці можливості скорочують цикл «написання → перевірка → деплой» і підвищують якість вихідного коду та документації. Разом із VS Code активно використовувався npm (Node Package Manager) — стандартний диспетчер пакетів екосистеми Node.js. Його ключова роль у проєкті:

— керування залежностями: автоматичне встановлення та оновлення сторонніх бібліотек (Fastify, Vue, Tailwind, Preline UI, Jest);

— локальна та глобальна установка CLI-інструментів (eslint, prettier, vite, docker-compose-watch) без потреби ручного завантаження;

— локфайл package-lock.json гарантує відтворюваність середовища на CI-сервері й у колег по команді;

Комбінація VS Code разом з npm забезпечила зручний робочий процес: від швидкої генерації шаблонів компонентів до автоматизованого запуску лінтерів і

тестів на кожному коміті, що позитивно позначилося на стабільності й підтримуваності розроблюваного застосунку.

2.2.7 GitLab

Для зберігання коду, відстеження змін і автоматичного розгортання використано GitLab. Його можливості включають:

— контроль версій через Git;

— створення merge-запитів;

— вбудовану систему CI/CD для автоматизації тестування та деплою;

— керування гілками, правами доступу, історією змін.

Завдяки GitLab забезпечується безпечна колективна розробка з можливістю швидкого повернення до стабільних версій у разі помилок.

2.3 Моделювання вимог до програмного забезпечення

Діаграма варіантів використання — це спосіб, який графічно ілюструє взаємодію зовнішніх суб'єктів (акторів) із системою через сценарії. Стандарт UML розглядає таку діаграму як форму «фіксації функціональності системи, доступної для зовнішнього спостереження» [19]. Основні цілі побудови діаграми:

— сформувати та структуризувати інформацію до єдиного цілісного набору вимог, що описує для "що конкретно" прикладники на різних рівнях споживання системи;

— спільна мова для комунікацій – аналітики безпосередньо з клієнтом отримали можливість вести діалог однією мовою з розробниками, що усі одразу зрозуміють;

Під кожен такий варіант використання існує можливість деталізації в текстовий сценарій, activity чи sequence-діаграму. На рисунку 2.1 наведено use-case-діаграму для розроблюваного застосунку. Вона демонструє, як дві ключові категорії користувачів взаємодіють із системою:

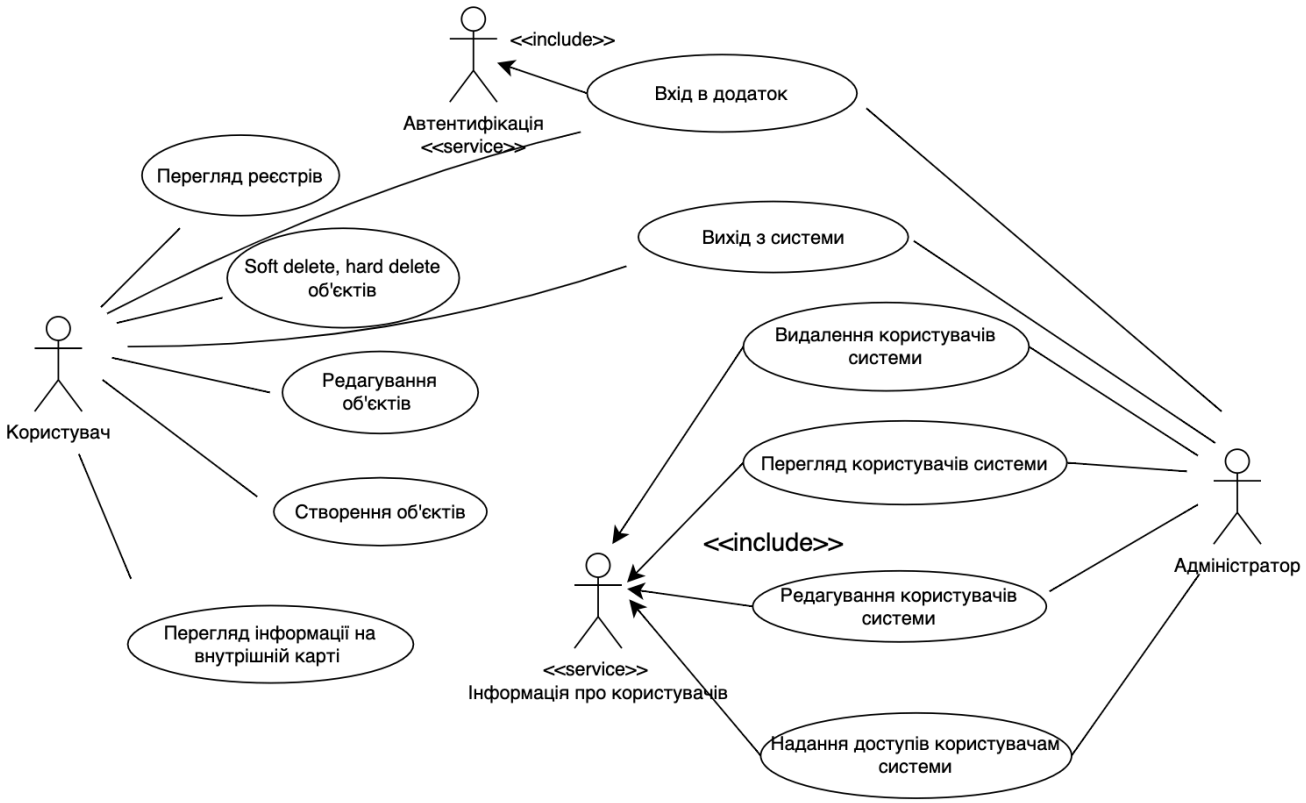


Рис. 2.1 Діаграма варіантів використання

У діаграмі варіантів використання виділено два основних актори системи — Користувач і Адміністратор**.**

Користувач (працівник територіального підрозділу, який має обмежений, рольовий доступ) має можливість:

— створювати нові записи водних об’єктів;

— редагувати інформацію в межах наданих прав;

— виконувати м’яке видалення (переведення об’єкта в архів);

— повне видалення об’єкта — для налаштованих для цього реєстрів;

— переглядати електронну карту з відображенням об’єктів;

— переглядати списки та картки об’єктів у довідниках.

Адміністратор (технічний адміністратор або співробітник профільного держоргану, відповідальний за підтримку системи) успадковує всі можливості звичайного користувача й, додатково, має повноваження:

— керувати обліковими записами користувачів (створення, редагування, блокування);

— призначати ролі та налаштовувати права доступу відповідно до моделі RBAC.

Функціональні вимоги — це опис того, які саме дії повинна виконувати система та які послуги надавати користувачам. Вони формулюють конкретні функції й сценарії (створити запис, відобразити карту), що безпосередньо реалізуються у вигляді інтерфейсів, алгоритмів чи API-ендпоінтів.

Нефункціональні вимоги — це характеристики та обмеження, як система має виконувати ці дії: продуктивність, масштабованість, надійність, безпека, юзабіліті, сумісність, обмеження на апаратні чи мережеві ресурси тощо. Вони не додають нових функцій, але задають якісні показники, яким повинна відповідати реалізація.

Функціональні вимоги

Користувач:

— авторизація;

— розлологуванн;

— перегляд об’єктів реєстрів;

— фільтрація об’єктів в реєстрах;

— пошук об’єктів в реєстрах;

— редагування об’єктів;

— видалення та м’яке видалення об’єктів;

— створення об’єктів;

— перегляд об’єктів на карті.

Адміністратор:

— управління користувачами;

— надання доступів користувачам;

— налаштування повноважень на редагування об’єктів.

Нефункціональні вимоги

— зберігання паролів у зашифрованому вигляді;

— компоненти інтерфейсу мають бути повторно використовуваними;

— база даних повинна бути нормалізованою;

— використання реляційної бази даних PostgreSQL;

— web-застосунок має бути сумісний з останніми версіями браузерів (Chrome, Firefox, Safari та Edge).

2.4 BPMN діаграма

BPMN (Business Process Model and Notation) — це стандарт графічного моделювання бізнес-процесів, затверджений OMG. Діаграма BPMN описує послідовність дій, подій, умовних розгалужень і взаємодію учасників (людей та IT-систем) у межах одного процесу. Вона читається практично «мовою блок-схеми», але має формальну нотацію, яку можуть однаково інтерпретувати аналітик, розробник і замовник. На рис. 2.2 представлено діаграму бізнес-процесу (BPMN), яка ілюструє сценарій перевірки прав доступу користувача у системі обліку водних об’єктів. Сценарій охоплює основні етапи автентифікації, авторизації, перевірки ролей та виконання сценарію внесення даних.

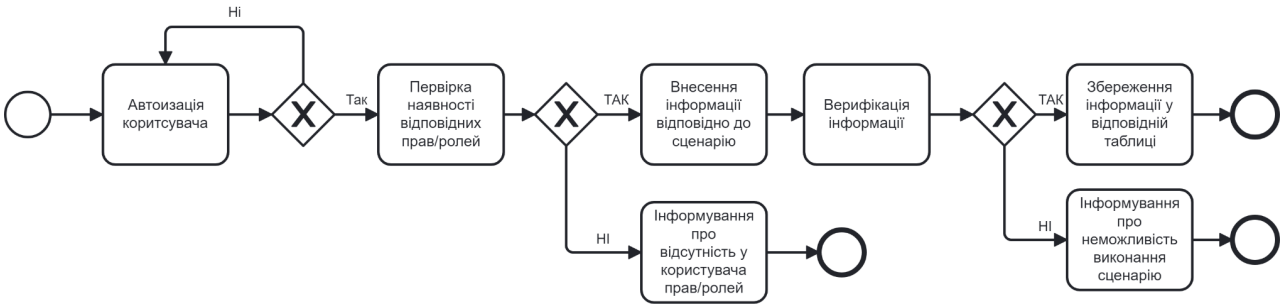


Рис. 2.2 Діаграма бізнес-процесу (BPMN)

Опис етапів процесу:

1. Автентифікація користувача — перший крок, у якому перевіряється, чи має користувач дійсні облікові дані.

2. Перевірка наявності прав/ролей — визначається, чи має автентифікований користувач достатні повноваження для виконання дій.

3. Внесення даних згідно з обраним сценарієм — у випадку наявності прав, користувач може ініціювати зміну або створення запису.

4. Верифікація інформації — перед збереженням відбувається перевірка коректності введених даних.

5. Результат**:**

— у випадку успішної верифікації — дані зберігаються в базі;

— у разі невдачі — користувач отримує повідомлення про неможливість виконання сценарію.

Використання BPMN дозволяє формалізувати поведінку системи та визначити точки прийняття рішень, що особливо актуально при реалізації контролю доступу відповідно до моделі Role-Based Access Control (RBAC)[19].

2.5 Моделювання архітектури системи

Моделювання архітектури системи є ключовим етапом у процесі розробки програмного забезпечення, що дозволяє заздалегідь спроєктувати структуру та взаємодію її основних компонентів. Архітектура визначає, як функціональні модулі системи поєднуються між собою, як вони обмінюються даними, які технології застосовуються для їх реалізації та як забезпечується масштабованість, безпека й підтримуваність системи.

Правильно спроєктована архітектура не лише визначає технічні основи функціонування системи, а й забезпечує адаптивність до змін вимог, зручність подальшого розвитку та інтеграції з іншими сеірвісами або базами даних. Вона виступає як каркас, що забезпечує ефективну організацію коду, повторне використання компонентів, мінімізацію залежностей між модулями та полегшує тестування і відлагодження.

Архітектура застосунку обліку водних об’єктів включає кілька основних компонентів: клієнтську частину (інтерфейс адміністратора та користувача), серверну частину, що реалізована на базі Fastify (Node.js), базу даних PostgreSQL з розширенням PostGIS, модуль генерації картографічних тайлів, а також сторонні

npm-пакети, які містять спільні інтерфейси форм, парсери таблиць та повторно використовувані UI-компоненти.

Серверна частина (Fastify): є основою API-запитів. Вона обробляє запити з клієнтської частини, виконує перевірку прав доступу, взаємодіє з базою даних, здійснює генерацію та видачу тайлів.

База даних (PostgreSQL + PostGIS): використовується для зберігання детальної інформації про водні об’єкти, користувачів, паспорти, договори оренди, геометричні дані (полігони, точки) та зв'язки між сутностями. Постійно оновлювана структура таблиць і міжтабличні зв'язки дозволяють масштабувати систему та підтримувати її цілісність.

Картографічний сервіс: api vtile та інші апі для карти, які відповідають за видачу картографічних шарів у форматі PNG-тайлів та GeoJSON, які інтегруються у клієнтський інтерфейс, відображення інформації про об’єкти на карті. Він використовує MapLibre GL JS для візуалізації шарів на карті.

Пакетні модулі npm: у проєкті використовуються декілька npm-бібліотек: одна містить спільні інтерфейси та схеми об’єктів, друга — CRUD для базових сутностей, третя — UI-компоненти (таблиці, форми, модальні вікна). Це забезпечує повторне використання коду та уніфікацію логіки в різних модулях.

Безпека та доступ: система підтримує автентифікацію користувачів, зберігання паролів у зашифрованому вигляді, параметризовані запити для захисту від SQL-ін’єкцій та детальне керування повноваженнями через ролі та доступи.

Рисунок 2.3 ілюструє загальну архітектуру системи: взаємодію між клієнтом, Fastify-сервером, базою даних та картографічним модулем, що разом забезпечують повноцінну функціональність застосунку для ведення обліку водних об’єктів.

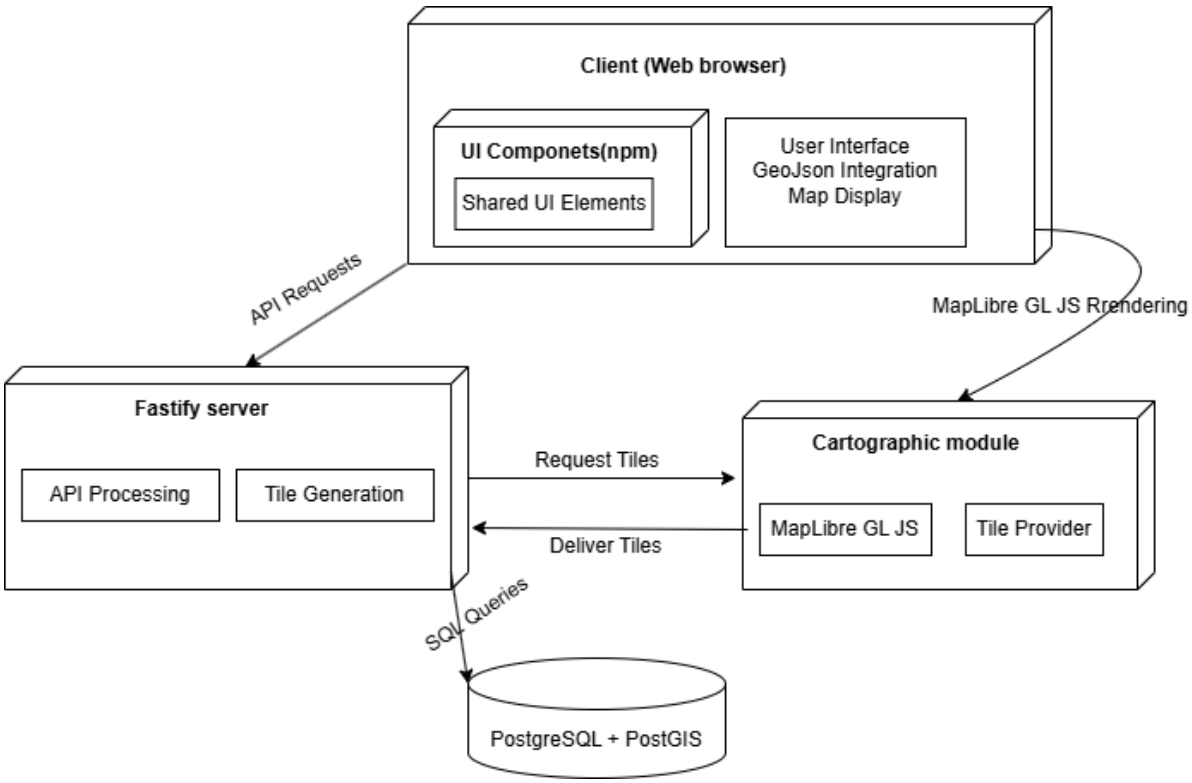


Рис.2.3 Загальна архітектура системи

2.7 Файлова архітектура

Правильно спроєктована структура каталогів і файлів — одна з базових передумов якісної розробки. Від того, як саме структуровано вихідний код, ресурси, тести та конфігураційні артефакти, залежить:

— читаність: новий розробник одразу розуміє, де шукати модель, контролер або міграцію;

— підтримка: упорядковані шари спрощують рефакторинг і локалізацію помилок;

— масштабованість: проєкт безболісно розширюється модулями й пакунками, не перетворюючись на «монолітичний хаос».

Файлова архітектура виконує роль «карти» застосунку: вона задає логічні кордони між доменною логікою, інфраструктурним кодом та презентаційним шаром, роблячи систему прозорою й передбачуваною для всієї команди. У контексті розробки застосунку для обліку водних об’єктів на основі Fastify, Vue,

PostgresSQL, файлова архітектура відображає, як різні компоненти системи розподілені та взаємодіють між собою. Файлова архітектура застосунку відображена на рисунку 2.4.

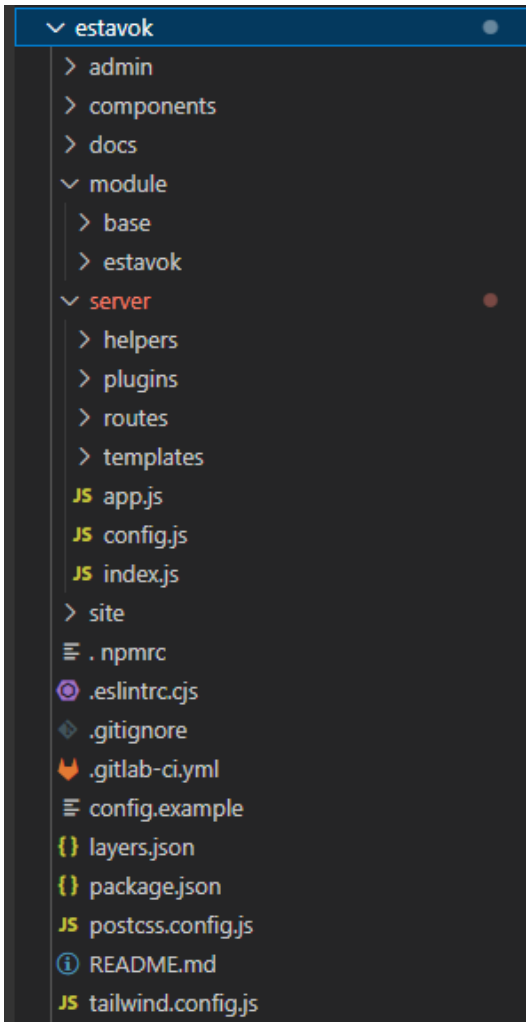


Рис. 2.4 Файлова архітектура застосунку

Файлова архітектура проєкту має критичне значення для ефективної розробки, масштабування та довготривалого супроводу програмного забезпечення. Невпорядкована або погано спроєктована структура файлів може призвести до помилок при розробці, складності у тестуванні та проблем із впровадженням нових функціональностей. Натомість логічно структурована архітектура дозволяє швидко орієнтуватися в коді, забезпечує узгодженість між модулями, полегшує впровадження стандартів розробки та підвищує якість програмного продукту в цілому.

3 РОЗРОБКА ТА ОПИС ФУНКЦІОНУВАННЯ WEB-ЗАСТОСУНКУ ДЛЯ ОБЛІКУ ВОДНИХ ОБ'ЄКТІВ

3.1 Реалізація електронної карти водних об'єктів

Електронна карта водних об'єктів є ключовим компонентом інформаційної системи для обліку рибогосподарських водних об'єктів. Вона дозволяє користувачам в інтерактивному режимі переглядати просторову інформацію про водойми, а також отримувати супутні атрибутивні дані про їх характеристики, форму власності, призначення, обмеження використання тощо. Метою реалізації цієї карти є забезпечення візуалізації просторових шарів, інтерактивного пошуку та фільтрації об'єктів, а також зручної навігації та доступу до деталізованих карток кожного об'єкта. Такий підхід дозволяє підвищити прозорість управління водними ресурсами, підтримує прийняття рішень органами державної влади та полегшує моніторинг у сфері рибного господарства.

Інтерфейс електронної карти реалізовано за допомогою бібліотеки MapLibre GL JS, що забезпечує роботу з векторною плиткою Mapbox у форматі MVT. Вся логіка відображення карти зосереджена у компоненті vs-map.vue, який відповідає за ініціалізацію карти, керування шарами, обробку кліків по об'єктах та показ карток. Зовнішній вигляд застосунку відповідає сучасним вимогам до UX/UI, а саме компоненти згруповані за функціональністю та мають логічне розташування:

— Ліва панель: виводить перелік геопросторових шарів (аквакультура, промисел, любительське рибальство тощо) з можливістю вмикання/вимикання, фільтрації та регулювання прозорості (vs-map-layers.vue).

— Центральна область: безпосередньо карта з базовим шаром (наприклад, OpenStreetMap, Google Satellite).

— Права панель: інформаційна картка про об'єкт (vs-map-card.vue), що з'являється при натисканні на елемент карти.

— Пошук: реалізований через компонент vs-map-search.vue, що дозволяє здійснювати пошук по координатах, АТУ або поточному місцезнаходженню користувача.

На рис 3.2 зображено сторінку карти з зазначеними вище компонентами.

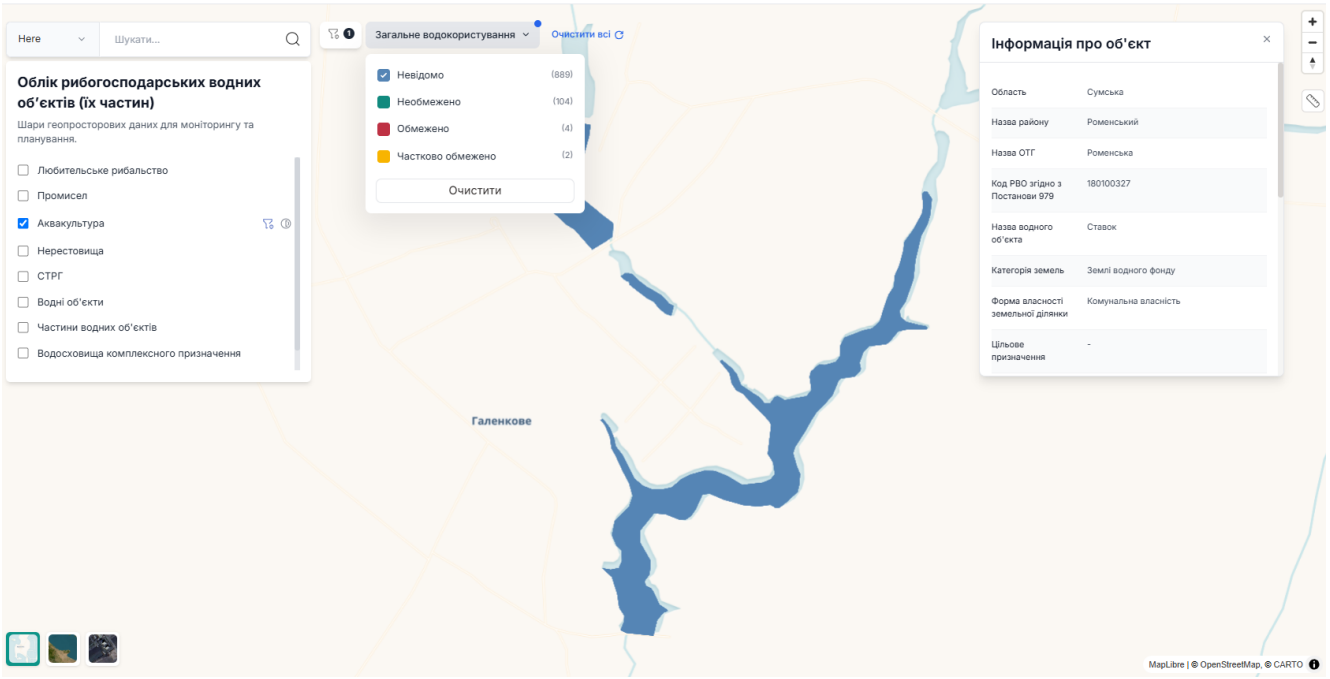


Рис. 3.2 Сторінка карти

На рис 3.1 зображено файлову структуру компоненти карти

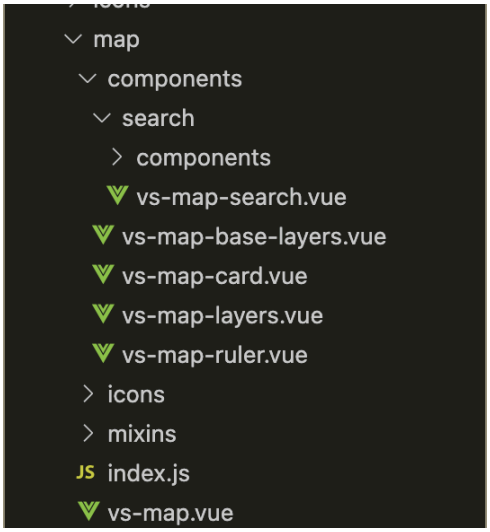


Рис. 3.1 Файлова структура компоненти карти

Функціональні компоненти карти:

Базові підложки (vs-map-base-layers.vue) — забезпечує перемикання між кількома картографічними підложками (OpenStreetMap, ESRI, Google Satellite). Користувач обирає карту, на фоні якої відображатимуться всі інші шари.

Шари даних (vs-map-layers.vue):

— виведення переліку доступних шарів (активуються через API /api/gis-layer-list);

— увімкнення/вимкнення шару;

— підвантаження фільтрів (/api/gis-filters/:layer) та їх застосування;

— регулювання прозорості шару;

— збереження обраних шарів у URL для відновлення стану при оновленні сторінки.

Інформаційна картка об'єкта (vs-map-card.vue) — при кліку на об'єкт, карта визначає його ідентифікатор та викликає API /api/gis-forma, що повертає HTML-представлення картки. В картці вказані всі ключові атрибути (область, ОТГ, назва, форма власності тощо), а також є кнопка переходу до розширеної картки об'єкта.

Компонента пошуку (vs-map-search.vue):

— пошук за адресою через HERE API;

— пошук за координатами (формат lat,lng);

— пошук за адміністративно-територіальними одиницями;

— центрування карти на знайденій геометрії з додаванням маркеру.

На рисунку 3.2 зображено пошук за Адміністративно-територіальним устроєм.

На рисунку 3.3 зображено інформаційну картку об’єкту з можливості переходу до картки в реєстрі з повною інформацією.

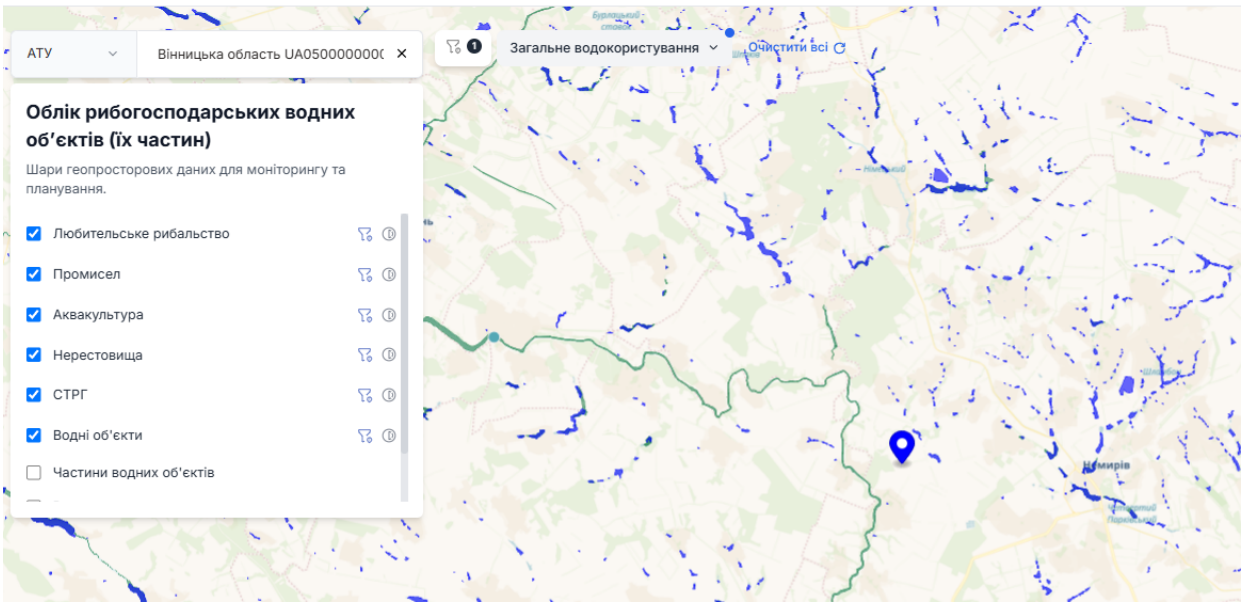


Рис. 3.2 Пошук за АТУ

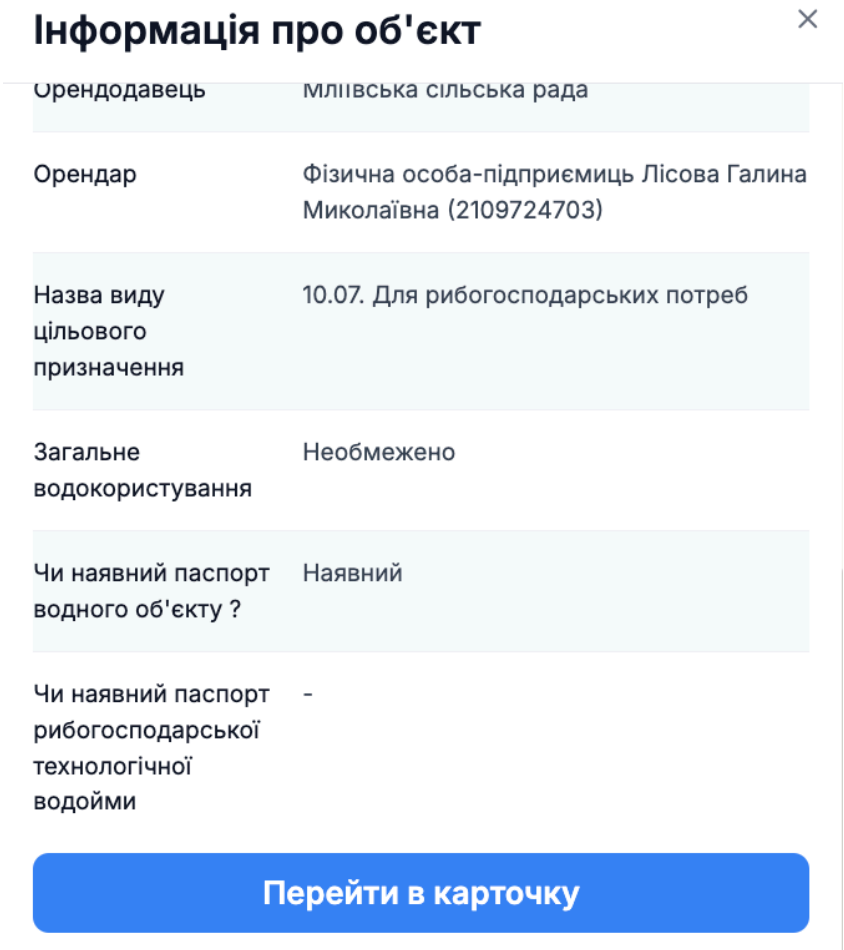


Рис. 3.3 Інформаційна картка об’єкту

Карта взаємодіє з серверною частиною, реалізованою на Fastify через наступні маршрути:

1. /api/vtile/:layer/:lang/:z/:x/:y.vmt — цей маршрут відповідає за генерацію векторної плитки MVT (Mapbox Vector Tile) для заданого шару. На основі зуму (z), координат плитки (x, y) та налаштувань шару (filter, pointZoom, clusterZoom) формується SQL-запит з ST_AsMVTGeom, який трансформує геометрії до EPSG:3857. Обробка запиту відбувається через бібліотеку @mapbox/sphericalmercator.

2. /api/gis-filters/:layer — повертає структуру фільтрів для обраного шару згідно з описом у .yml файлі шару (наприклад, estavok.acvaculture.yml). Запит автоматично виконує group by по зазначених полях, а також додає текстові підписи з довідників.

3. /api/gis-format?table=[table]&id=[id] — повертає HTML-представлення інформації про об'єкт. Динамічно підставляє заголовки колонок, додає класифікатори, формує структуру відповідно до формату, заданого у файлі шару (тип картки, перелік колонок, значення).

4. /api/gis-layer-list — формує перелік усіх доступних шарів, їх географічні межі, кількість записів та URL для плиток. Інформація береться з шаблонів yml, які описують тип геометрії, джерело даних (table_name) і стиль відображення. Кожен шар карти описано у відповідному .yml файлі, ці файли дозволяють динамічно підключати нові шари без потреби змінювати фронтенд-логіку. Структуру yml файлу зображено на рисунку 3.4.

На рис 3.5 діаграма демонструє шлях обробки просторових запитів: від взаємодії користувача з веб-картою до формування геопросторових тайлів сервером на основі даних у PostGIS.

```yaml
title: Частини водних об'єктів
table_name: estavok.water_object_parts

style:
  type: polygon
  color: green
  width: 1
  border: 1
  clusterZoom: 14
  opacity: 0.6
  stroke: "#eee"

setting:
  popup: [part_name]
  title: {
    part_name: Назва частини водного об'єкту,
    description: Опис
  }
  card_type: list
  card_list: [ part_name,description ]
  info: [part_name, description]
```

Рис. 3.4 Структура yml файлу для налаштування шару

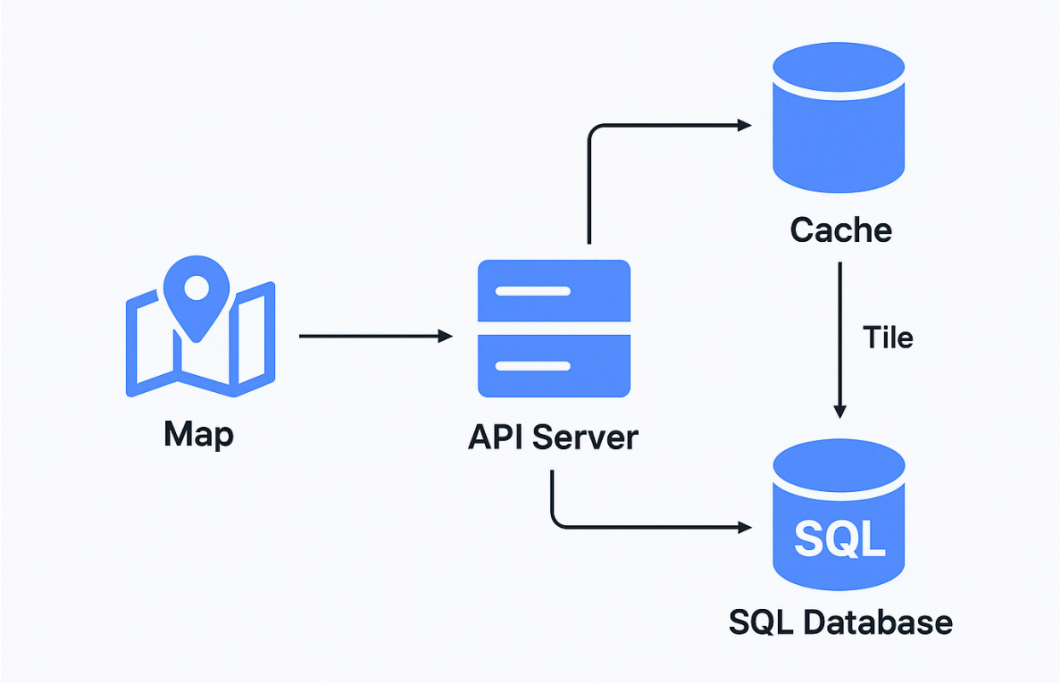


Рис. 3.5 Архітектура обробки запитів у геоінформаційній системі

Електронна карта водних об’єктів реалізована як сучасний інтерактивний веб-інструмент, який дозволяє в реальному часі переглядати, шукати та фільтрувати інформацію про водойми на основі просторових та атрибутивних даних. Гнучка архітектура, заснована на API, YAML-конфігурації шарів та компонентному підході на Vue 3, забезпечує масштабованість та можливість швидкої адаптації системи під нові вимоги.

3.2 Розробка бази даних

У якості системи управління базами даних для реалізації застосунку обліку водних об’єктів обрано PostgreSQL — надійну, високопродуктивну, масштабовану СУБД з відкритим вихідним кодом. Її популярність серед розробників геоінформаційних систем пояснюється розширеною підтримкою типів даних, транзакційністю, розвиненою системою індексів і можливістю вертикального та горизонтального масштабування.

Схема admin забезпечує повноцінну систему контролю доступу, реалізовану на принципах рольової моделі (RBAC). Це дозволяє задавати як типові групи користувачів, так і гнучко конфігурувати права окремих осіб. Завдяки модульній структурі routes і menu, реалізовано можливість динамічного управління інтерфейсами без жорсткого закодування їх у програмний код. Це суттєво підвищує розширюваність застосунку.

RBAC (Role-Based Access Control) — це модель керування доступом до ресурсів інформаційної системи, у якій дозволи на виконання дій не призначаються безпосередньо користувачам, а замість цього надаються ролям, а користувачі вже призначаються до цих ролей. Кожна роль відповідає певній групі дозволів, які можуть охоплювати доступ до модулів, інтерфейсів або окремих функцій системи [1]. У 2020 році було запропоновано нову структуру для реалізації динамічного RBAC у децентралізованих додатках (dApps), яка повністю відокремлює контроль доступу від основної логіки застосунку, забезпечуючи гнучкість та масштабованість [1].

Серед переваг даної моделі можна виділити:

— масштабованість — дозволяє централізовано керувати доступом великої кількості користувачів через ролі, а не вручну задавати права кожному;

— гнучкість підтримка — одночасного призначення кількох ролей одному користувачу (user_roles), з можливістю встановлення строку дії та набору дозволених дій;

— безпека — легко реалізується принцип найменших привілеїв: користувачі мають лише ті права, які необхідні їм для роботи;

— аудит та контроль — можна легко відстежити, хто і з якими правами має доступ до критичних модулів або даних;

Дана схема складається з наступних таблиць:

— users — користувачі системи;

— roles — групи користувачів;

— user_roles — прив’язка користувачів до ролей;

— routes — інтерфейси (модулі) системи;

— role_access — доступ ролей до інтерфейсів;

— menu — пункти меню.

ERD діаграма схеми admin зображена на рисунку 3.6.

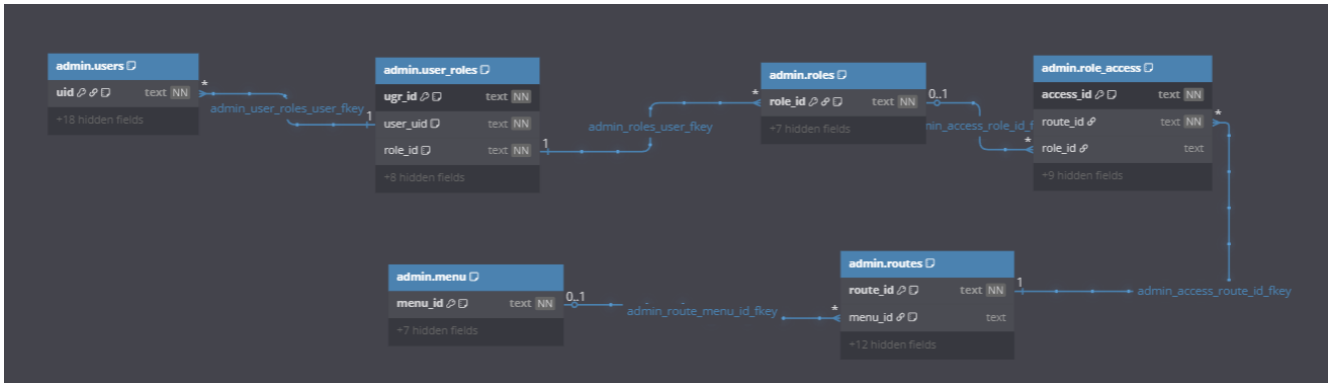


Рис. 3.6 ERD діаграма схеми admin з відображенням зв’язків

Крім схеми admin реалізована схема estavok, яка містить всі основні сутності застосунку та зв’язки між ними, які зображені на ERD діаграмі на рисунку 3.7.

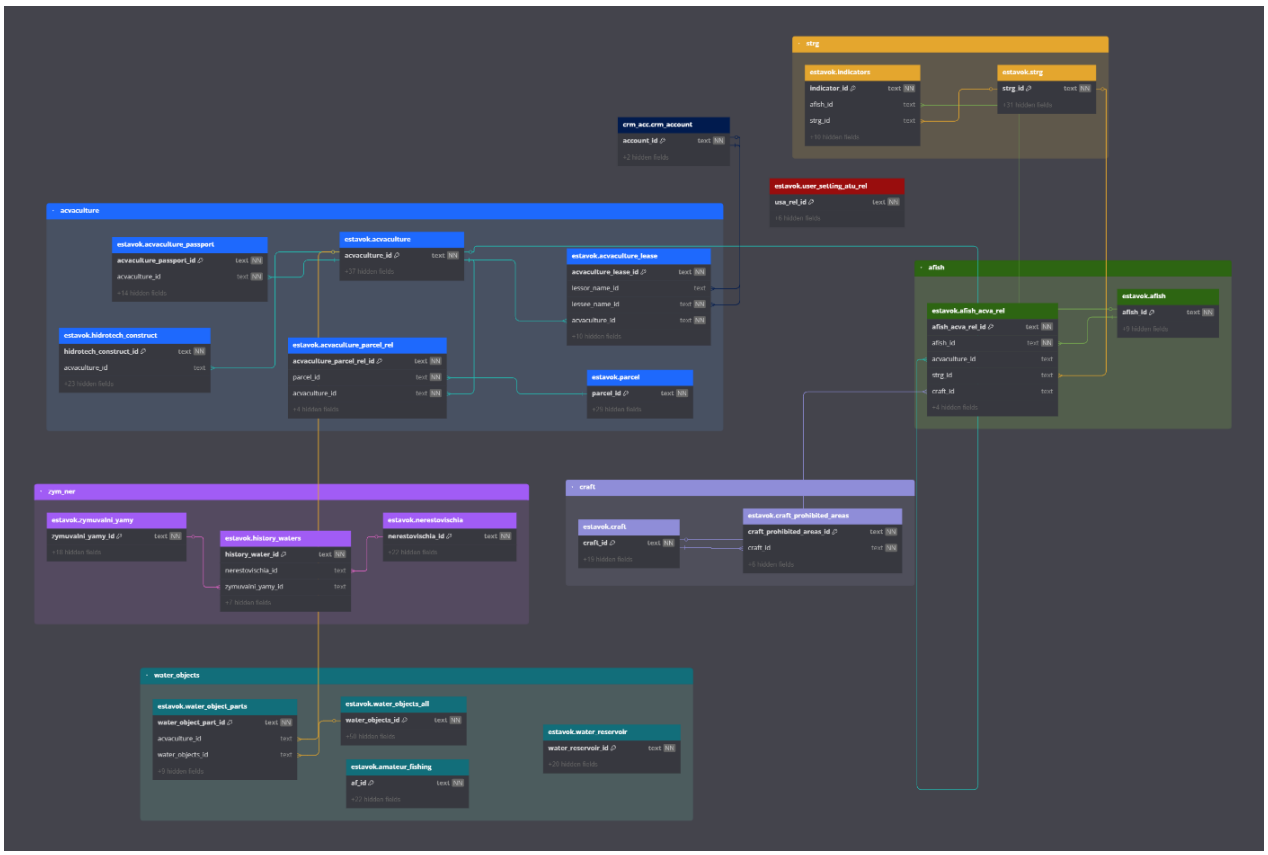


Рис. 3.7 ERD діаграма основних сутностей застосунку разом зі зв’язками між ними

Дана схема охоплює ключові аспекти обліку водних об’єктів в Україні: аквакультуру, оренду, земельні ділянки під водними об’єктами, рибогосподарські об’єкти, види біоресурсів, водосховища просторові компоненти та ін. База побудована з урахуванням нормалізації даних, просторової прив’язки (через тип geometry) та міжтабличних зв'язків.

Нормалізація бази даних — це процес організації її структури з метою мінімізації дублювання даних та забезпечення логічної цілісності. У реалізованій базі нормалізація дозволила розділити дані за логічними сутностями: наприклад, окремо виділені таблиці для договорів оренди (acvaculture_lease), паспортів водних об’єктів (acvaculture_passport), видів риб (afish) тощо. Це дозволяє уникнути надлишкових записів і спростити оновлення, зберігаючи узгодженість інформації.У результаті застосування нормальних форм:

— 1НФ: всі таблиці мають атомарні поля, тобто дані подані у вигляді елементарних значень без повторюваних груп;

— 2НФ: таблиці не мають часткових залежностей, тобто кожна неключова колонка залежить від усього первинного ключа;

— 3НФ: усі атрибути залежать тільки від первинного ключа, що мінімізує транзитивні залежності.

Завдяки нормалізації, база є масштабованою, легко супроводжуваною та захищеною від помилок при оновленні даних, що критично важливо в контексті ведення державного обліку водних ресурсів. Основну схему бази даних водних об’єктів, можна розділити на певні групи, перелічені нижче:

Група «Аквакультура»

— acvaculture — основна таблиця, що містить інформацію про водні об’єкти, які використовуються для аквакультури (назва, розташування, площа, цільове призначення, статуси, паспорти, геометрія);

— acvaculture_lease — зв’язок об’єкта аквакультури з договорами оренди (орендодавець, орендар, розміри орендної плати, терміни);

— acvaculture_parcel_rel — зв’язок аквакультури із земельними ділянками;

— acvaculture_passport — дані з паспорта рибогосподарської технологічної водойми ;

— hidrotech_construct — гідротехнічні споруди, пов’язані з об’єктом аквакультури.

Група «Види риб (1А)»

— afish — класифікатор видів риб (українська та латинська назви, ISSCAAP-коди);

— afish_acva_rel — зв’язок риби з аквакультурою, промислом, СТРГ.

Група "СТРГ (Спеціальне Товарне Рибне Господарство)»

— strg — таблиця з інформацією про режими використання водойм у межах СТРГ: наукове обґрунтування, органи узгодження, розміри водойм, показник;

− indicators — обсяги вилову/вселення риби (факт/план по роках), зв’язок із видом риби та СТРГ.

Група «Промисел»

— craft — таблиця з описом промислового використання водойм: правила, обмеження, кількість суден, дозволів;

— craft_prohibited_areas — просторові межі заборонених для промислу зон.

Група «Водні об’єкти»

— water_objects_all — найбільший шар, повний список водних об’єктів із атрибутами (площа, назва, суббасейн, категорія, координати тощо);

— water_object_parts — частини водних об’єктів (використовуються в аквакультурі);

— water_reservoir — водосховища з обліковими показниками (об’єми, координати, призначення);

— amateur_fishing — зони любительського рибальства (географічна прив’язка, періоди заборони).

Група «Зимувальні ями та Нерестовища»

— nerestovischia — просторові об’єкти, які мають сезонні обмеження для рибальства;

— zymuvalni_yamy — зимувальні ями з описом характеристик і координатами;

— history_waters — історія змін зимувальних ям і нерестовищ по роках;

Додаткові таблиці:

— parcel — земельні ділянки, пов’язані з водними об’єктами або спорудами;

— user_setting_atu_rel — повноваження (зв’язок користувачів із територіями, до яких вони мають доступ).

Зв’язки між таблицями

— основні зв’язки будуються за зовнішніми ключами (acvaculture_id, afish_id, strg_id;

— просторові зв’язки реалізуються через колонки типу geometry;

— додаткові індекси (btree, gin, gist) використовуються для оптимізації пошуку, просторових операцій, повнотекстового пошук.

Можна зробити висновок, що реалізована база даних демонструє високий рівень структурованості, модульності та орієнтації на просторову взаємодію між об’єктами. Це дозволяє ефективно підтримувати функції обліку, контролю, візуалізації й аналітики водних об’єктів, зокрема в системах типу веб-реєстрів або ГІС-додатків.

3.3 Забезпечення безпеки web-застосунку

Забезпечення безпеки є одним з найважливіших аспектів розробки застосунку, особливо коли йдеться про зберігання персональних даних користувачів, обмеження доступу до функціональних модулів системи та захист від несанкціонованого доступу. У web-застосунку реалізовано комплексний підхід до захисту інформації, який охоплює як серверну частину Fastify, так і рівень бази даних PostgreSQL.

Для захисту облікових записів користувачів у базі даних використовується тригерна функція insert_update_user_before(), яка автоматично виконується при вставці або оновленні запису у таблиці admin.users. Основна логіка функції полягає в тому, що пароль користувача:

− спочатку проходить перевірку на мінімальну довжину (8 символів);

− потім до нього додається згенерована «сіль» (salt) — випадкове значення, унікальне для кожного користувача;

− далі пароль хешується у кілька ітерацій за допомогою алгоритму MD5 (10 циклів), а потім додатково обробляється через внутрішню функцію crypt().

Цей підхід дозволяє зберігати лише хешований пароль у базі даних, що унеможливлює його пряме відновлення навіть у випадку витоку. Використання хешування із сіллю відповідає загальноприйнятим практикам безпечного зберігання паролів, рекомендованим OWASP [20].

Функція також дозволяє уникати дублювання логіки на рівні серверного коду, перекладаючи відповідальність за хешування на рівень СУБД PostgreSQL. Крім серверного хешування паролів у базі даних, реалізовано також механізм перевірки автентичності на рівні бекенду. Цей механізм базується на відповідності введених користувачем облікових даних з тими, що вже збережені у базі.

На стороні бекенду реалізована функція verifyPassword, яка приймає параметри username і password та виконує такі кроки:

1. Валідація параметрів — перевіряється наявність і непорожність введеного логіну та пароля;
2. Пошук користувача у таблиці admin.users за логіном, email або телефоном, з умовою що користувач активований (enabled = true);
3. Повторне хешування пароля — використовується той самий алгоритм, що й у тригері в базі даних;
4. Порівняння хешів — якщо отриманий результат збігається з паролем у базі, користувач автентифікований.

Цей підхід забезпечує:

— відповідність між бекендом і тригерною логікою БД;

— збереження принципу zero-knowledge (сервер ніколи не зберігає або передає відкриті паролі);

— можливість масштабування або заміни механізму перевірки без змін у базі даних.

Таким чином, логіка автентифікації побудована на двох рівнях:

— серверному — верифікація за допомогою функції verifyPassword;

— базі даних — шифрування та збереження пароля через тригер insert_update_user_before().

Це забезпечує послідовність, цілісність та безпечність автентифікації у застосунку. На рисунку 3.8 зображено програмний код функції перевірки паролю користувача при логіні.

```
if (!username || username === '') return { message: 'not enough params: username' };
if (!password || password === '') return { message: 'not enough params: password' };

const query = 'select * from admin.users where $1 in (login,email,phone) and enabled limit 1';
const json = await pg.query(query, [username]).then((res) => res.rows[0]);

if (!json) return { message: 'user not found' };

let hash = '';

for (let i = 0; i < 10; i += 1) {
  hash = md5(password + hash + json.salt);
}

hash = crypt(hash, json.salt);
```

Рис.3.8 Програмний код функції перевірки паролю при логіні

На рисунку 3.9 зображено екранну форму логіну.

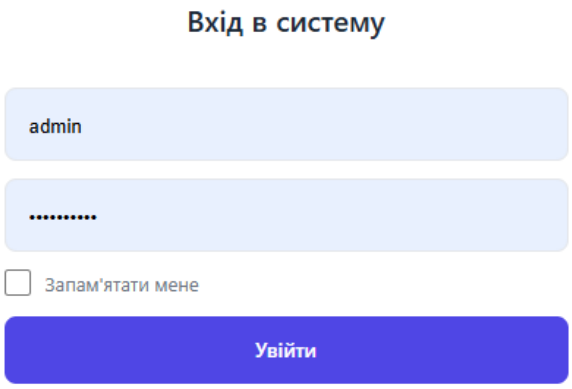


Рис.3.9 Екранна форма логіну

Крім шифрування паролю у застосунку, важливо захистити саму взаємодію з базою даних та мінімізувати можливості sql-ін’єкцій — для цього застосовуються параметризовані запити (prepared statements).

SQL-ін’єкція — це клас атак, за якого зловмисник підставляє у вхідні поля (форми, URL-параметри, cookie) фрагменти SQL-коду; якщо ці дані без перевірки вклеюються у текст запиту, код виконується в СУБД із правами застосунку. Наслідки — несанкціоноване читання, модифікація або видалення даних, витік облікових записів, ескалація привілеїв.

У Fastify (Node.js) параметризація реалізується через механізми драйверів — наприклад, метод query(text, values) бібліотеки pg. Таким чином, розробник пише так, як зображено на рисунку 3.10, а не формує рядок шляхом конкатенації змінних. На рівні PostgreSQL підготовлені запити кешуються у планувальнику, що додатково підвищує швидкодію під час повторних викликів. Аналогічно, при використанні ORM-рівня (Knex, Prisma, Objection) параметризація застосовується автоматично.

```
const json = await pg.query(query, [username]).then((res) => res.rows[0]);
```

Рис. 3.10 Приклад параметризованого запиту

Для браузерної частини (Vue.js) критично ніколи не пересилати «сирий» SQL із клієнта — усі запити проходять через валідований REST API контролер, який викликає підготовлені оператори в бекенді. Таким чином досягається потрійний захист:

1. Валідація схеми (JSON-schema) на вході у Fastify-роуті.
2. Параметризований SQL усередині контролера.
3. Обмежені права базового користувача БД (REVOKE/GRANT лише на потрібні таблиці).

Завдяки поєднанню шифрування паролів і параметризованих запитів web-застосунок отримує надійний фундамент інформаційної безпеки: дані ідентифікації захищені у сховищі, а канал взаємодії з БД — від атак на рівні запитів.

3.4 Налаштування інтерфейсів карток об’єктів

Для реалізації гнучкої та масштабованої структури інтерфейсів використовується шаблонізатор Handlebars, а також низка кастомних хелперів, що дозволяють керувати логікою відображення, перевіркою повноважень на редагування у користувача, наприклад:

— contentList (отримання даних за запитом з бд з картки);

— descriptionList (виведення інформації у форматі блоку);

— tableList (виведення інформації у форматі таблиці);

— ifCond, ifCondOr, ifCond (задання умов на виведення даних).

Структура відображення карток визначається у файлі конфігурації index.yml, що описує компонування панелей (розділів), які відображаються у вигляді вертикальних вкладок або просто картки розділеної на колонки по блоках. Кожна вкладка відповідає за окрему частину даних об’єкта: основну інформацію, земельні ділянки, оренду, гідротехнічні споруди тощо. Для кожної вкладки створено відповідний .hbs-файл, наприклад main_info.hbs, який відповідає за відображення загальної інформації про об’єкт. На рис. 3.11 представлено приклад відображення картки водного об’єкта з реалізованими вкладками та функціональністю динамічного керування вмістом залежно від прав доступу.

```yaml
component: default
isHeaderEditButton: false

panels:
  - type: vertical-tabs
    items:
      - name: main_info
        title: Основна інформація
      - name: parcel
        title: Земельні ділянки
      - name: lease
        title: Інформація про оренду
      - name: hidrotech_construct
        title: Гідротехнічні споруди
      - name: afish
        title: Види водних біоресурсів
      - name: water_objects_parts
        title: Частини водного об'єкту
      - name: tech_project
        title: Паспорти рибогосподарської технологічної водойм

```

Рис. 3.11 Приклад конфігурації картки об’єкту

На рисунку 3.12 зображений інтерфейс картки об’єкту, блок основної інформації.

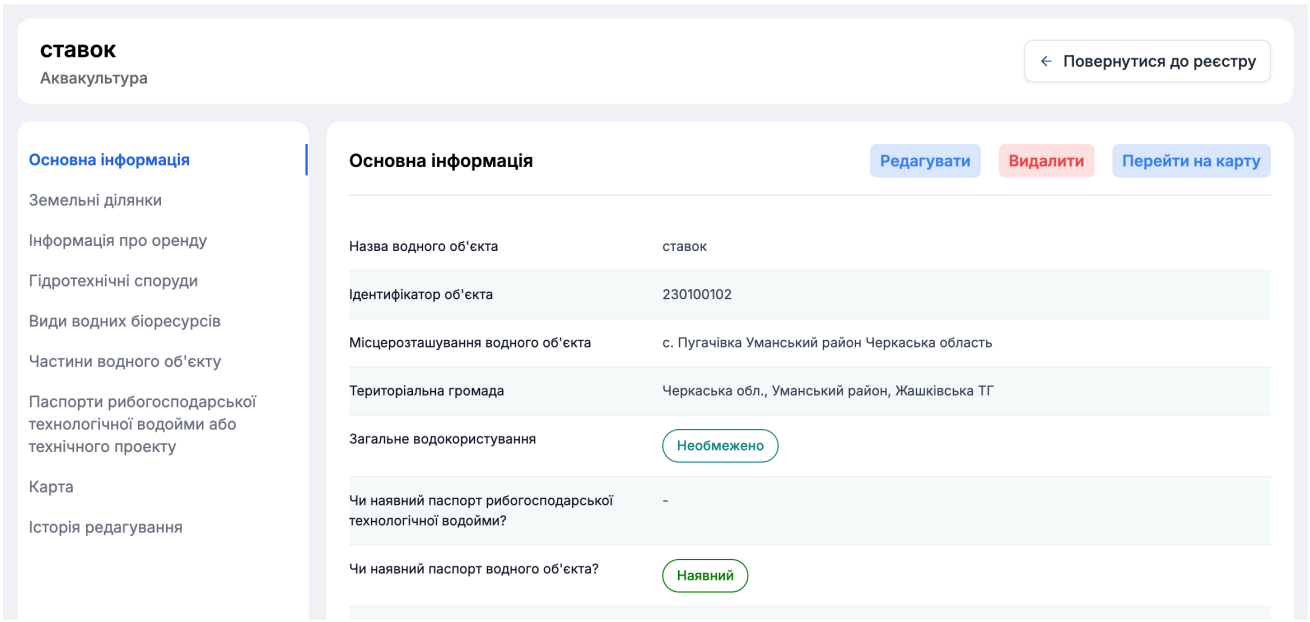


Рис. 3.12 Блок основної інформації картки об’єкту

З даної картки ми маємо можливість редагувати, видалити об’єкт (при умові наявності повноважень) та перейти на карту, де буде відображений даний об’єкт з його інформаційною карткою. В даній картці використовуються кастомні хелпери для виведення інформації (descriptionList), кнопок з діями (button). Також для видалення реалізована окрема апі, яка м’яко видаляє об’єкт, лише призначає йому конкретний статус, але не видаляє з таблиці повністю. Для видалення та редагування об’єктів областей, необхідно мати повноваження на цю область, які налаштовуються в окремому реєстрі. В картці відбувається запит на отримання областей, до яких користувач має доступ і далі перевірка на відповідність отриманої області користувача та самої області об’єкта. При наявності конкретних повноважень, користувач буде мати можливість видаляти та редагувати об’єкт. Реалізація функціоналу повноважень на редагування зображений на рисунку 3.13.

```
{{#contentList table="(select '{{region_atu}}'= any(array_agg(atu_id)) as is_atu   from estavok.user_setting_atu_rel where user_uid='{{user.uid}}')"}}
    {{#ifCondOr rows.[0].is_atu '==' true ../user.user_type '==' 'admin'}}
        {{{button this token=(token table='estavok.acvaculture' form='estavok.acvaculture.form' id=@root.id) title="Редагувати" edit=1}}}
        <button onclick="window.v3plugin.$api({ api: '/api/obj-status/estavok.acvaculture/{{@root.id}}', method:'get',confirm: { title:'Підтвердити операцію',
        text: 'Ви впевнені що хочете вилучити запис?', cancel: 'Скасувати', confirm : 'Виконати'} })" class="px-2 py-1 inline-flex border-solid justify-center it
        Видалити
        </button>
    {{/ifCondOr}}
{{/contentList}}
```

Рис. 3.13 Реалізація функціоналу повноважень на видалення та редагування об’єктів

На рисунку 3.14 зображений блок пов’язаної інформації, а саме види водних біоресурсів, який виводиться у картку окремим табом за допомогою хелперу tableList. З даної картки ми також маємо можливість редагувати та видаляти пов’язані об’єкти. Також є можливість додавання – нові об’єкти будуть мати зв’язок з поточним об’єктом та відображатися одразу в таблиці.

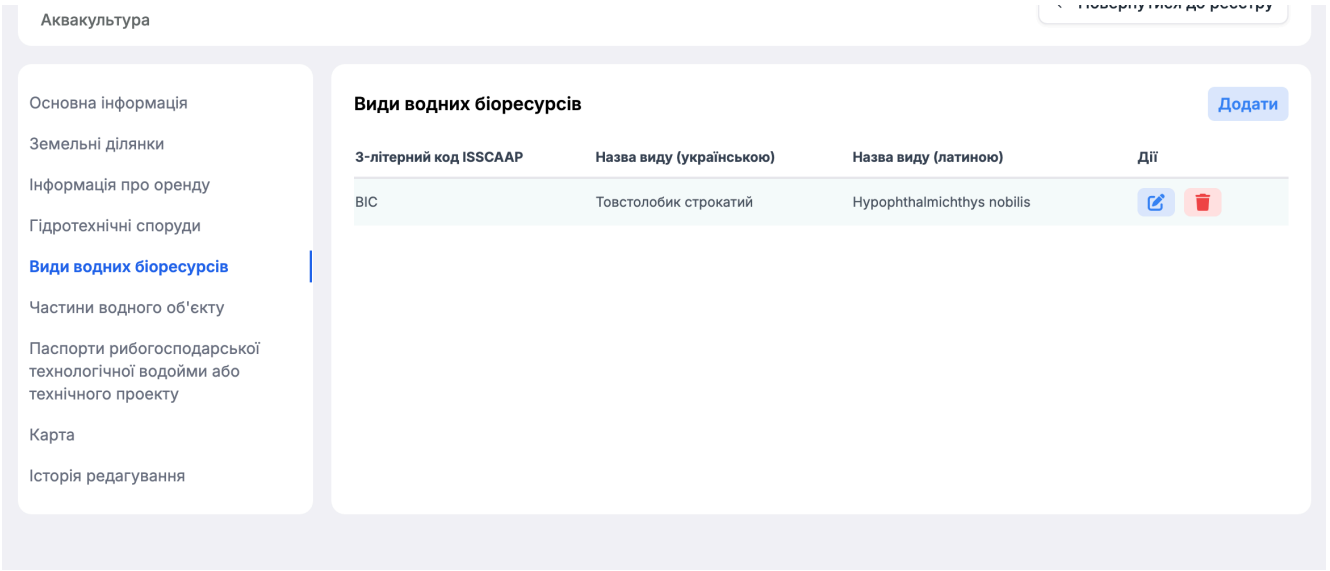


Рис. 3.14 Таб «види водних біоресурсів» картки об’єкту Аквакультури

У системі кастомні Handlebars-хелпери, такі як descriptionList та tableList, відіграють ключову роль у формуванні інтерфейсів карток об’єктів. Їх використання дозволяє:

— автоматизація виводу даних: вони динамічно формують структуру виводу (опис чи таблицю) на основі конфігурації columns, що значно зменшує кількість шаблонного коду;

— гнучкість: підтримка шаблонів ({{handlebars}}) у назвах колонок або ключах дозволяє вставляти динамічні значення без ручового дублювання логіки;

— уніфікованість стилю: вся візуальна частина списків даних (таблиць, описів) має єдиний дизайн, що підтримується централізовано;

— підтримка порожніх станів: якщо дані відсутні, хелпери автоматично виводять заздалегідь визначений блок “Інформація відсутня”;

— безпечна обробка: значення true/false або порожніх даних обробляються коректно та конвертуються у локалізовані варіанти ("Так", "Ні", "-").

descriptionList призначений для відображення переліку властивостей у вигляді двоколонкового блоку (label + value).

tableList призначений для генерації табличного списку з можливістю відображення дій (Редагувати, Видалити), інтегрованих із системою ролей та токенами доступу. Обидва хелпери працюють із конфігураційним файлом .yml, що дозволяє легко змінювати структуру інтерфейсів без втручання в код шаблонів. На рисунку 3.15 зображено приклад конфігурації картки об’єкту.

```
{{{tableList
    rows
    form="estavok.hidrotech_construct_acva.form"
    id='hidrotech_construct_id'
    table='estavok.hidrotech_construct'
    uid=../user.uid
    nodata="<div class='bg-gray-200 text-center p-6 rounded-xl'><h3 class='text-lg fo
    comma=";"
    columns="Назва, {{name}},
    Характеристика, {{characteristic}},
    Стан,{{select state data='estavok.hidrotech_state'}},
    Форма власності,{{select ownership_form data='ownership_code'}},
    Балансоутримувач,{{select account_id data='account_id'}}
    "
}}}
```

Рис. 3.15 Приклад конфігурації картки об’єкту за допомогою хелперу tableList

На рисунку 3.16 зображено програмну реалізацію хелперу descriptionList.

```
export default async function descriptionList(data, opt) {
    const { hash } = opt;

    if (hash.nodata && !data) {
        const noDataText = typeof hash.nodata == 'string' ? hash.nodata : '<div class="bg-gray-200 text-center p-6 rounded-xl">
        return noDataText
    }
    if (!hash.columns) return 'columns empty'
    const keys = hash.columns.split(hash.divider || ',').map(el => hash.comma ? el.trim().replace(new RegExp(hash.comma || '#',

    const result = [];

    for (let i = 0; i < keys.length; i += 2) {
        const name = keys[i];
        const nameHBS = name.includes('{{') ? await handlebars.compile(name)({ ...data, hash }) : false;

        if (!nameHBS && name.includes('{{')) continue;

        const key = keys[i + 1];

        const d1 = await format(data[key], key, data) || '-';

        result.push(`<div class="grid grid-cols-1 gap-1 py-3 sm:grid-cols-3 sm:gap-4 even:bg-gray-50 text-[12px]">
            <dt class="text-gray-900">${nameHBS || name}</dt>
            <dd class="text-gray-700 sm:col-span-2">${d1}</dd>
            </div>
        `);

    }
    return '<dl class=" divide-y divide-gray-100 py-[5px]">' + result.join('') + '</dl>';
}
```

Рис. 3.16 Програмна реалізація хелперу descriptionList

4 ОГЛЯД РОБОТИ WEB-ЗАСТОСУНКУ ТА ТЕСТУВАННЯ АПІ

4.1 Огляд основних функцій застосунку

На рисунку 4.1 зображено реєстр «Аквакультура», який містить визначений перелік колонок та фільтрів, які налаштовуються в json-шаблонах. Це робить систему легко розширюваною.

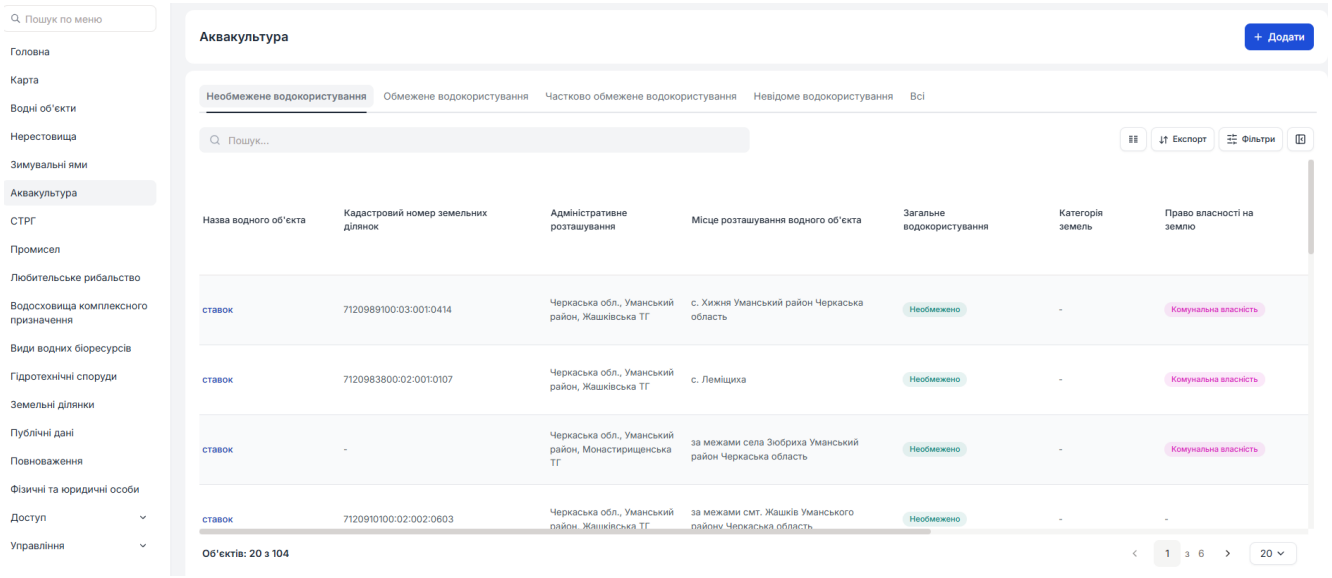


Рис. 4.1 Таблиця реєстру «Аквакультура»

Також у кожної таблиці реєстру є можливість фільтрувати дані по визначеним колонкам, здійснювати пошук по назві, обирати колонки в таблиці, які хочемо відобразити в таблиці, а які приховати. На рисунку 4.2 зображено фільтри реєстру «Аквакультура».

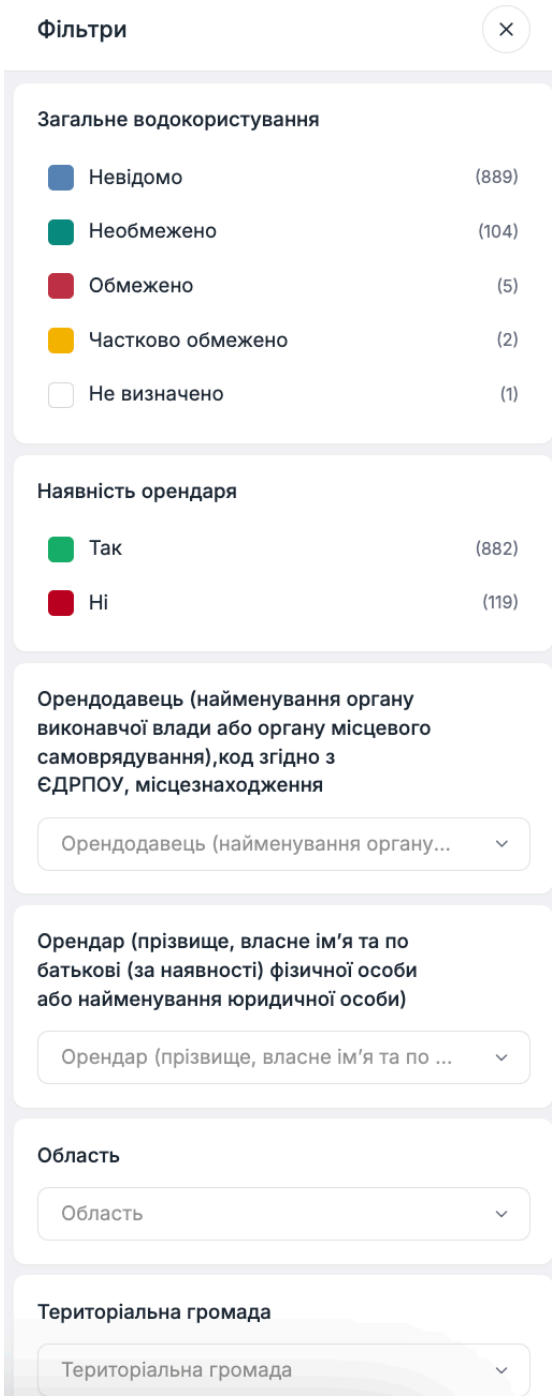


Рис. 4.2 Фільтри реєстру «Аквакультура»

На рисунку 4.3 зображено форму створення об’єкту. Форма містить основні поля, такі як загальне водокористування, назва водного об’єкту, батьківський водний об’єкт та додаткові блоки (пов’язані таблиці), в які ми маємо можливість додати записи при створенні основного об’єкту.

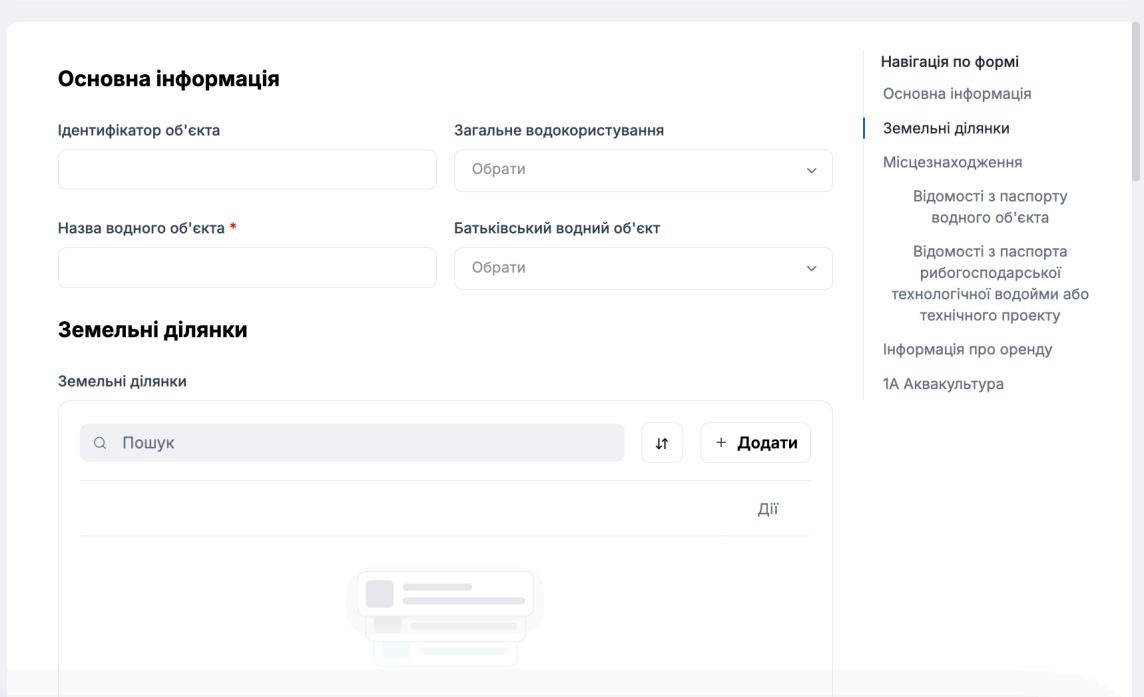


Рис. 4.3 Форма створення об’єкту «Аквакультура»

На рисунку 4.4 зображено поле форми для внесення геометрії.

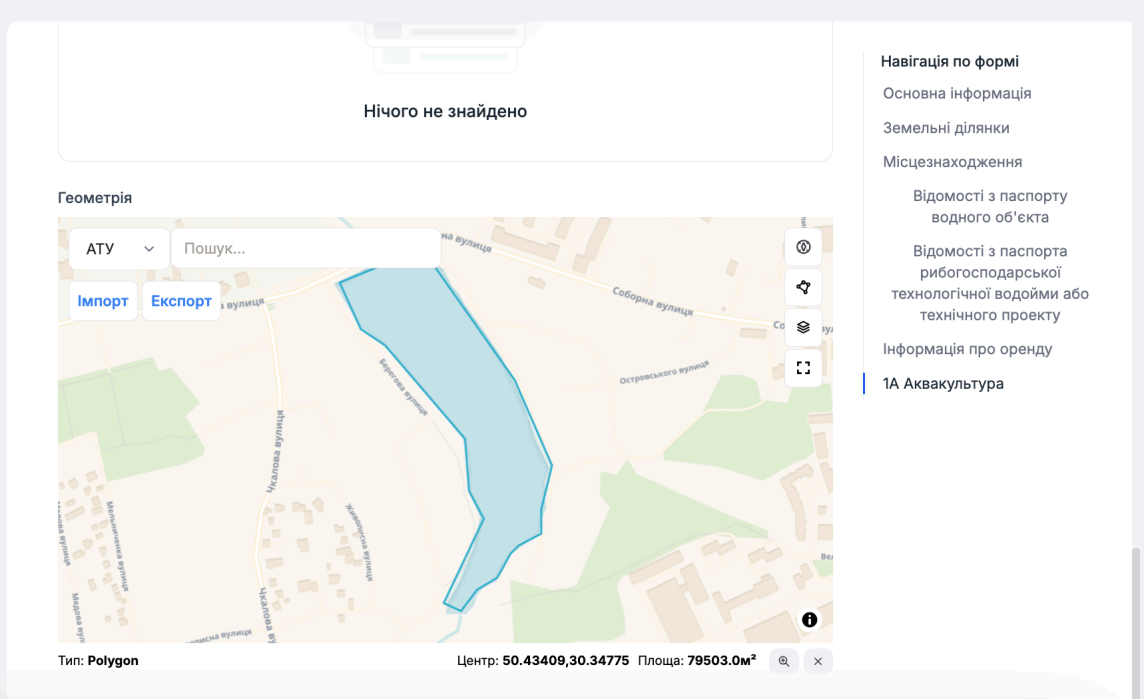


Рис. 4.4 Поле форми для внесення геометрії об’єкту

Після створення об’єкту маємо можливість перейти до картки об’єкту. Основна інформація об’єкту зображена на рисунку 4.5. З картки ми маємо можливість редагування, видалення об’єкту та переходу до карти.

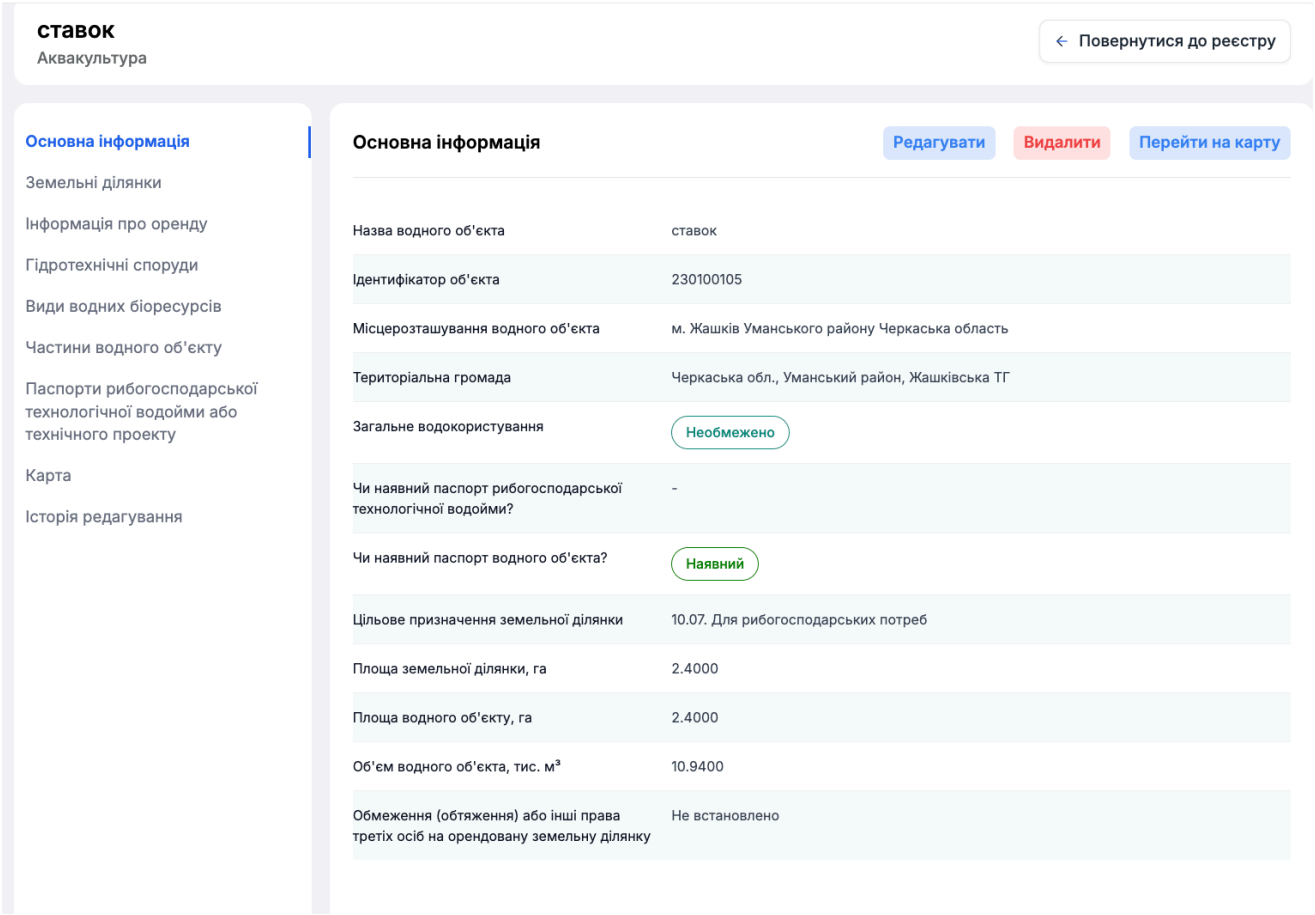


Рис. 4.5 Основна інформація об’єкту відображена у картці

На рисунку 4.6 зображено таб з додатковою інформацією про об’єкт, а саме інформацію з таблиці «Види водних об’єктів» аквакультури. З цього табц ми також маємо можливість редагувати та видаляти дані.

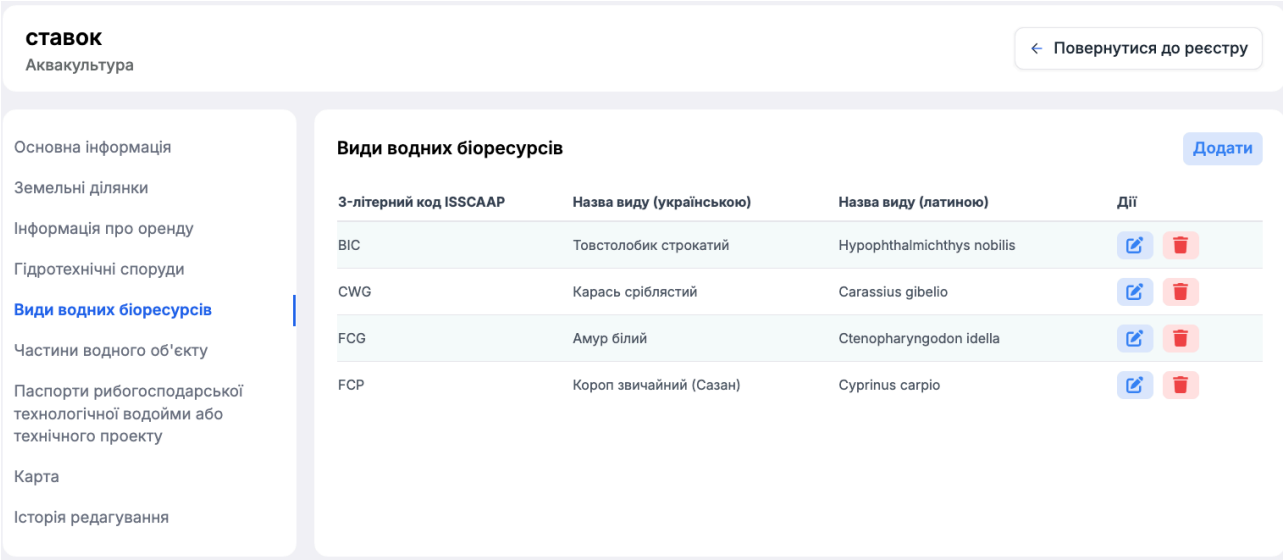


Рис. 4.6 Таб з інформацією про «Види водних об’єктів»

Також з картки об’єкту ми маємо можливість перегляду геометрії об’єкту. Віджет з відображенням геометрії об’єкту зображений на рисунку 4.7.

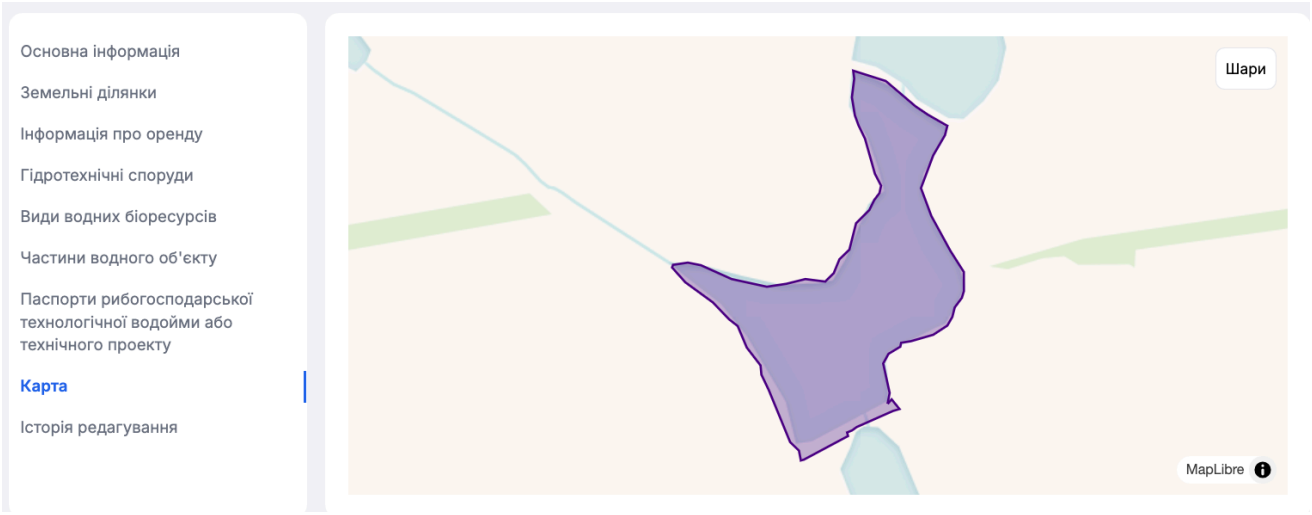


Рис. 4.7 Віджет з відображенням геометрії об’єкту.

З картки ми також маємо можливість переходу до карти, з зумом до самого об’єкту та відкриттям інформаційної картки по цьому об’єкту. На рисунку 4.8 зображена карта з відображенням інформаційної картки об’єкта, можливістю перемикання шарів, пошуку та відкриттям інших шарів.

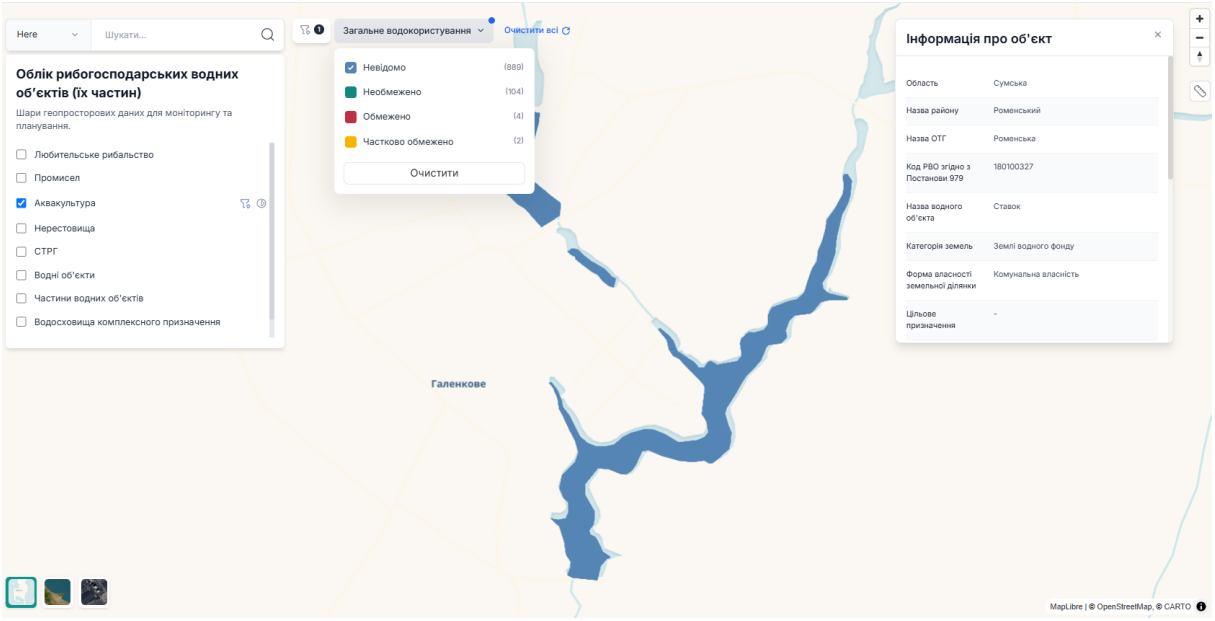


Рис. 4.8 Відображення інформаційної картки об’єкту при переході з картки

Користувач з роллю адміністратора має можливість створювати користувачів та визначати їм права. На рисунку 4.9 зображений інтерфейс налаштування прав для групи доступу.

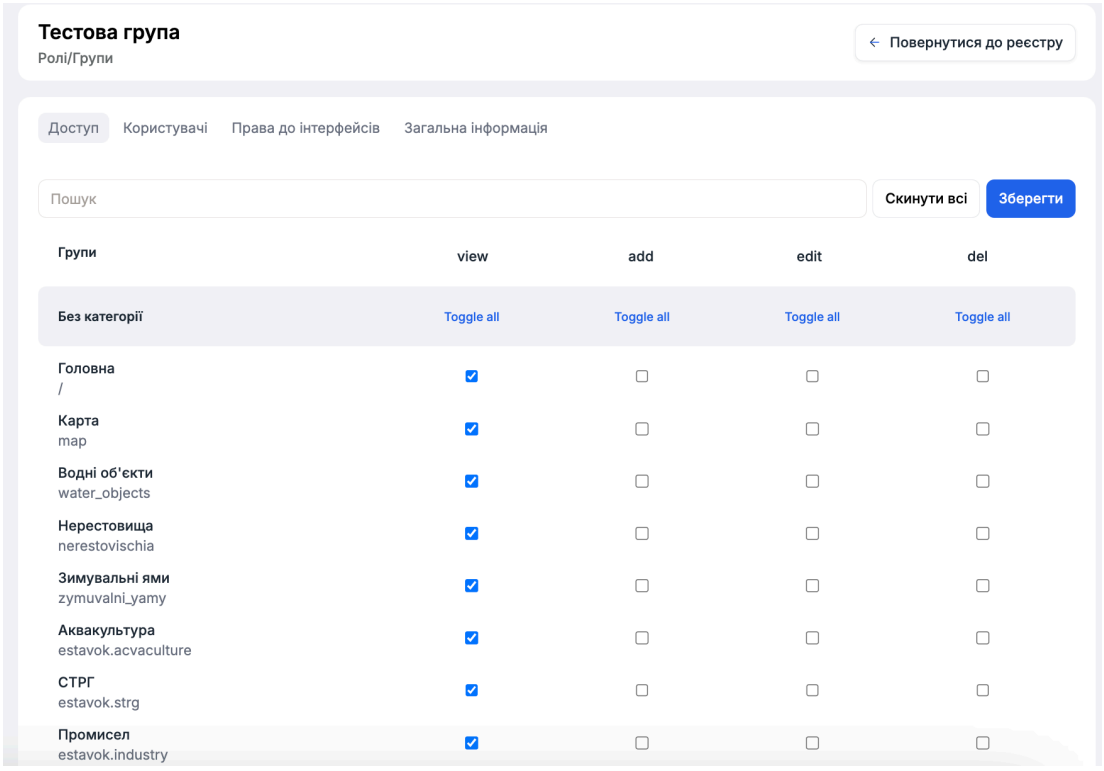


Рис. 4.9 Інтерфейс налаштування групи доступу.

На рисунку 4.9 зображена форма створення користувача.

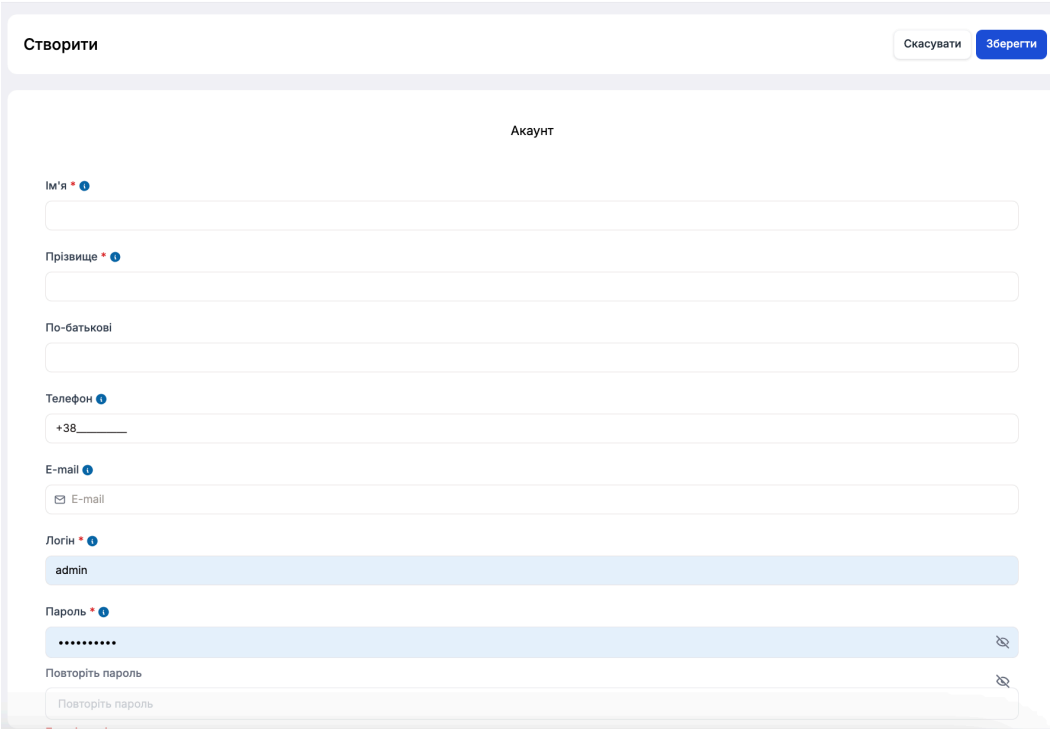


Рис. 4.9 Форма створення користувача

4.2 Тестування апі

Юніт-тести (або модульні тести) — це автоматизовані перевірки, які тестують окремі частини програмного коду (так звані юніти) ізольовано від інших частин системи. Юніт-тести є фундаментом тестування програмного забезпечення. Вони забезпечують стабільність, передбачуваність та довіру до поведінки коду, особливо в проєктах із динамічним розвитком. В розроблюваному web-застосунку використовується вбудований модуль node:test для написання тестів.

На рисунку 4.10 зображено приклад тестів для хелперу ifCond, який використовується для динамічного виводу інформації в шаблонах карток.

.

```
test('ifCond helper', async (t) => {
  await t.test('should return true block for equality (==)', () => {
    const template = Handlebars.compile('{{#ifCond 5 "==" 5}}Equal{{^}}Not Equal{{/ifCond}}');
    const result = template({});
    assert.equal(result, 'Equal');
  });

  await t.test('should return false block for inequality (!=)', () => {
    const template = Handlebars.compile('{{#ifCond 5 "!=" 10}}Not Equal{{^}}Equal{{/ifCond}}');
    const result = template({});
    assert.equal(result, 'Not Equal');
  });

  await t.test('should handle strict equality (===)', () => {
    const template = Handlebars.compile('{{#ifCond "5" "===" "5"}}Strictly Equal{{^}}Not Strictly Equal{{/ifCond}}');
    const result = template({});
    assert.equal(result, 'Strictly Equal');
  });
```

Рис. 4.9 Тести для хелперу ifCond

На рисунку 4.11 зображено результати виконання для хелперу ifCond.

```
▶ ifCond helper
  ✓ should return true block for equality (==) (6.1145ms)
  ✓ should return false block for inequality (!=) (0.8346ms)
  ✓ should handle strict equality (===) (0.7846ms)
  ✓ should handle strict inequality (!==) (0.7356ms)
  ✓ should handle "in" operator for strings (0.9983ms)
  ✓ should handle "not in" operator for strings (0.6818ms)
  ✓ should handle ">" operator (0.5784ms)
  ✓ should handle "<" operator (0.7295ms)
  ✓ should handle "&&" operator (0.7625ms)
```

Рис. 4.11 Результат виконання тестів для хелперу ifCond

ВИСНОВКИ

В результаті виконання дипломної роботи розроблено web-застосунок для ведення обліку водних об’єктів, який базується на використанні Fastify, NodeJS, PostgresSQL та VueJS. Актуальність цієї роботи полягає у відсутності єдиної електронної системи для обліку водних об’єктів в Україні та необхідністю створення її у зв’язку з зростаючим навантаженням на водні ресурси.

1. Проведено аналіз предметної галузі обліку водних об’єктів. Ключові проблеми, які можна виділити, пов’язані з відсутністю централізованої системи для обліку водних об’єктів. Це призводить до фрагментації даних між різними установами, ускладнення процесу прийняття управлінських рішень, недостатнього контролю за використанням водних ресурсів та утруднення моніторингу екологічного стану водойм. Відсутність єдиної бази даних також ускладнює виконання міжнародних зобов'язань України щодо управління водними ресурсами відповідно до вимог законодавства Європейського Союзу.

2. Розглянуто програмне забезпечення, яке може бути використано для обліку водних об’єктів та аналізу пов'язаної інформації. Було досліджено існуючі інструментальні засоби, зокрема системи на базі електронних таблиць (Google Sheets), ГІС-проєкти (QGIS) та європейську систему WISE.

3. Сформульовано вимоги до розробки, включно з функціональними (керування шарами, пошук, фільтрація, перегляд інформації про об’єкти, адміністрування користувачів) та нефункціональними (кросбраузерність, безпека, розширюваність).

4. Спроєктовано архітектуру системи, яка включає клієнтську частину (Vue + TailwindCSS), серверну частину (Fastify), базу даних (PostgreSQL/PostGIS), картографічний модуль (MapLibreGL), REST API для взаємодії між компонентами. Також розроблено erd-діаграму бази даних, діаграму варіантів використання та діаграму архітектури системи.

5. Розроблено web-застосунок з функціоналом роботи з реєстрами, створення, редагування, видалення та перегляду об’єктів, геопросторового пошуку, фільтрів для об'єктів на карті, відображення спливаючих карток об'єктів, підключення шарів через vtile.

6. Здійснено юніт-тестування API та окремих функціональних модулів (обробка геоданих, застосування фільтрів, CRUD операції), що дозволило виявити та усунути критичні помилки на ранніх етапах.

Робота пройшла апробацію на наукових конференціях, за її результатами апробації опубліковано тези доповідей:

1. Усенко Б.О., Аброскін Ю.Ю. Система обліку водних об’єктів як інструмент для збору, обробки та аналізу даних, необхідних для прийняття управлінських рішень. *Науково-технічна конференція «Сучасний стан та перспективи розвитку IoT»*. 15 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. (подано до друку)

2. Усенко Б.О., Аброскін Ю.Ю. PostGIS як інструмент просторового аналізу даних у геоінформаційних системах. *Всеукраїнська науко-вотехнічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях»*. 24 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С.460 – 463.

ПЕРЕЛІК ПОСИЛАНЬ

1. Усенко Б.О., Аброскін Ю.Ю. Система обліку водних об’єктів як інструмент для збору, обробки та аналізу даних, необхідних для прийняття управлінських рішень. *Науково-технічна конференція «Сучасний стан та перспективи розвитку IoT»*. 15 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. (подано до друку)

2. Усенко Б.О., Аброскін Ю.Ю. PostGIS як інструмент просторового аналізу даних у геоінформаційних системах. *Всеукраїнська науко-вотехнічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях»*. 24 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025. С.460 – 463.

3. Кравченко М. В., Волошкина, О. С., & Василенко, Л. О. (2021). Застосування методу зворотного осмосу для доочистки питної води. *Екологічна безпека та природокористування*. С.32. URL: https://doi.org/10.32347/2411-4049.2021.4.32-45

4. Fao / black sea illegal fishing. URL: https://media.un.org/unifeed/en/asset/d216/d2168222 (дата звернення: 24.04.2025).

5. Про рибне господарство, промислове рибальство та охорону водних біоресурсів: Закон України від 08.07.2011 № 3677-VI: станом на 15.11.2024 р. URL: https://zakon.rada.gov.ua/laws/main/3677-17 (дата звернення: 27.04.2025).

6. Про державний земельний кадастр : Закон України від 07.07.2011 № 3613-VI: станом на 15.11.2024 р. URL: https://zakon.rada.gov.ua/laws/main/3613-17 (дата звернення: 27.04.2025).

7. Про доступ до публічної інформації: Закон України від 13.01.2011 № 2939-VI: станом на 08.10.2023 р. URL: https://zakon.rada.gov.ua/laws/main/2939-17 (дата звернення: 29.04.2025).

8. Директива 2000/60/ЄС Європейського Парламенту та Ради від 23 жовтня 2000 р., що встановлює рамки діяльності Співтовариства у сфері водної політики (Водна рамкова директива: станом на 20.11.2014 р. URL: https://zakon.rada.gov.ua/laws/main/994_962 (дата звернення: 29.04.2025).

9. Council of the European Union. Council Regulation (EC) No 1224/2009 of 20 November 2009 establishing a Community control system for ensuring compliance with the rules of the common fisheries policy. *Official Journal of the European Union*. 2009. P.1-50.URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32009R1224 (дата звернення: 30.04.2025).

10. Конвенція про доступ до інформації, участь громадськості в процесі прийняття рішень та доступ до правосуддя з питань, що стосуються довкілля (Орхуська конвенція): Конвенція; ООН від 25.06.1998: станом на 27.05.2005 р. URL: https://zakon.rada.gov.ua/laws/main/994_015 (дата звернення: 30.04.2025).

11. QGIS – A Free and Open Source Geographic Information System [Електронний ресурс]. – Режим доступу: https://qgis.org/ (дата звернення: 30.04.2025).

12. WISE Freshwater – Water Information System for Europe [Електронний ресурс]. – Режим доступу: https://water.europa.eu/freshwater (дата звернення: 01.05.2025).

13. U.S. Geological Survey. National Water Information System (NWIS) [Електронний ресурс]. – Режим доступу: https://waterdata.usgs.gov/nwis (дата звернення: 02.05.2025).

14. Mozilla Developer Network (MDN). Progressive web apps – advantages [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps (дата звернення: 03.05.2025).

15. Fastify [Електронний ресурс]. – Режим доступу: https://www.fastify.io/ (дата звернення: 03.05.2025).

16. PostgreSQL – The world's most advanced open source database [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/ (дата звернення: 03.05.2025).

17. PostGIS – Spatial and Geographic objects for PostgreSQL [Електронний ресурс]. – Режим доступу: https://postgis.net/ (дата звернення: 03.05.2025).

18. Vue.js – The Progressive JavaScript Framework [Електронний ресурс]. – Режим доступу: https://vuejs.org/ (дата звернення: 03.05.2025).

19. OMG. UML 2.5 Specification. – Object Management Group, 2015. Режим доступу: https://www.omg.org/spec/UML/2.5/ (дата звернення: 03.05.2025).

20. OWASP Foundation. Password Storage Cheat Sheet [Електронний ресурс]. – 2023. – Режим доступу: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (дата звернення: 30.04.2025).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка web-застосунку для ведення обліку водних об’єктів

Виконала студентка 4 курсу
групи ПД-41
Усенко Богдана Олексіївна
Керівник роботи
асистент кафедри ІПЗ, Аброскін Юрій Юрійович

Київ – 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** - централізований облік водних об’єктів за рахунок застосування web-застосунку на основі фреймворку Fastify.
- **Об’єкт дослідження -** процес ведення обліку водних об’єктів.
- **Предмет дослідження -** web-застосунок для ведення обліку водних об’єктів.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз літературних джерел з проблематикою обліку водних об’єктів в країні.
2. Здійснити огляд та аналіз існуючих інструментальних засобів та додатків, що використовуються для ведення обліку водних об’єктів.
3. Провести огляд засобів для розробки програмного забезпечення web-застосунку для ведення обліку водних об’єктів.
4. Сформувати вимоги до розробки web-застосунку.
5. Спроектувати архітектуру web-застосунку для ведення обліку водних об’єктів.
6. Розробити web-застосунок для ведення обліку водних об’єктів.
7. Провести тестування web-застосунку для ведення обліку водних об’єктів.

3

АНАЛІЗ АНАЛОГІВ

Показник	Excel	QGIS	WISE	розроблений застосунок
Платформи	IOS, Windows	IOS, Windows, Linux	Веб-застосунок	Веб-застосунок
Багатокористувацька робота	-	-	+	+
Аналітичні можливості	+	+	+	+
Безпека і надійність	Є ризик втрати при відсутності резервних копій	Контроль доступу на рівні файлу/мережі	Висока	Висока (хешування паролів, захист від SQL-ін’єкції)
Простота використання	+	-	-	+
Відображення геоданих	-	+	+	+

4

ВИМОГИ ДО ДОДАТКУ

Функціональні вимоги:

1. Авторизація та розлогування.
2. Створення, редагування, перегляд, видалення об’єктів.
3. Можливість фільтрації та пошуку об’єктів в реєстрах.
4. Можливість перегляду об’єктів на карті.
5. Можливість управління користувачами(створення, редагування, видалення, перегляд).
6. Можливість надання доступів користувачам.
7. Можливість налаштування повноважень на редагування об’єктів.

Нефункціональні вимоги:

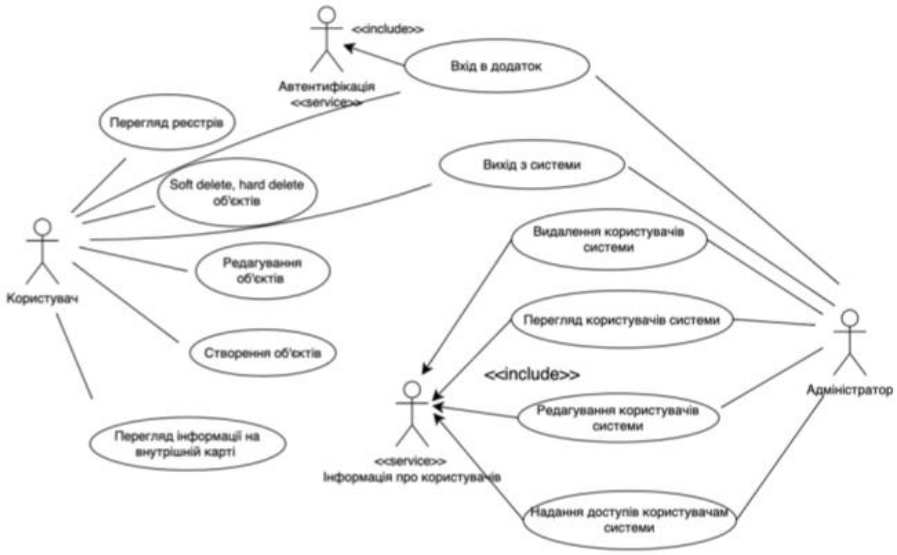
1. Зберігання паролів у зашифрованому вигляді.
2. Компоненти інтерфейсу мають бути повторно використовуваними.
3. База даних повинна бути нормалізованою.
4. Web-застосунок має бути сумісний з останніми версіями браузерів (Chrome, Firefox, Safari та Edge).
5. Використання реляційної бази даних PostgreSQL.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

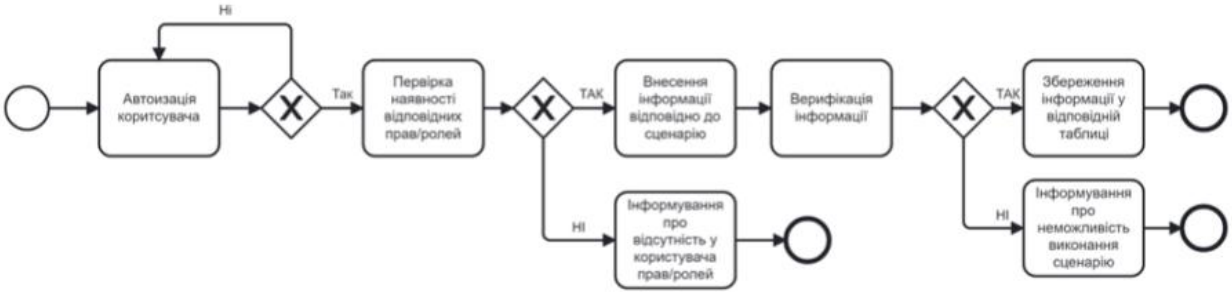
6

Діаграма варіантів використання



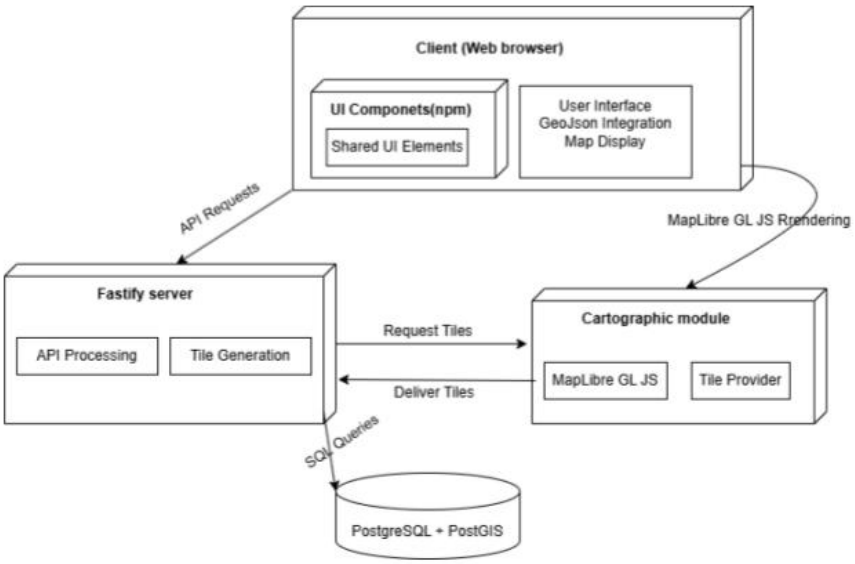
7

Діаграма бізнес-процесу (BPMN)



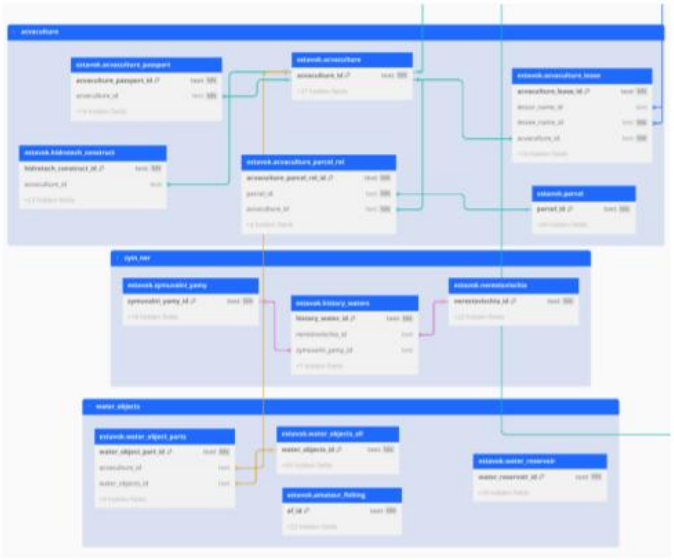
8

Діаграма архітектури додатку



9

Схема бази даних

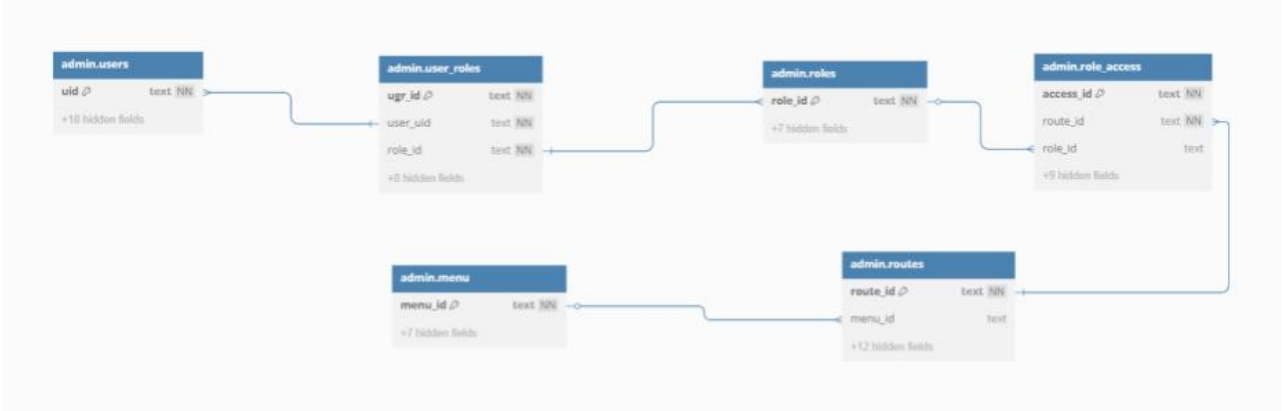


10

Схема бази даних(продовження)

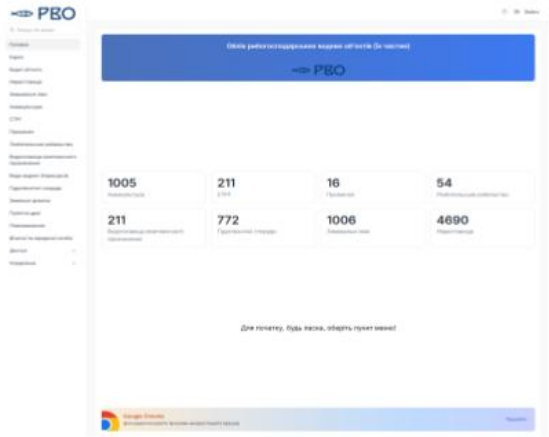
11

Схема бази даних (схема управління користувачами)

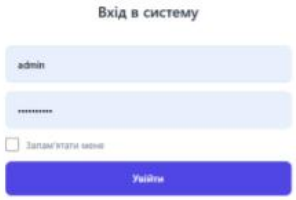


12

ЕКРАННІ ФОРМИ



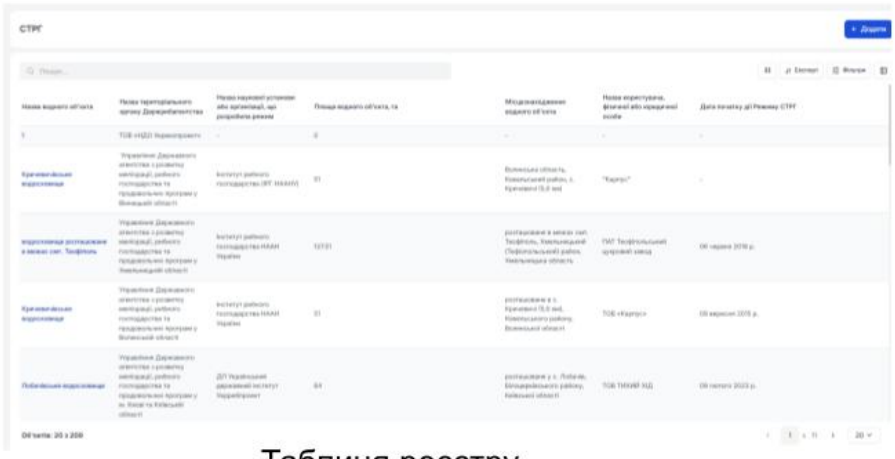
Головний екран додатку



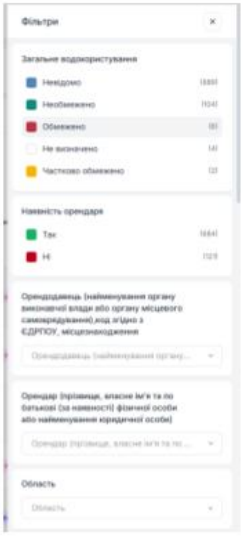
Форма авторизації

13

ЕКРАННІ ФОРМИ



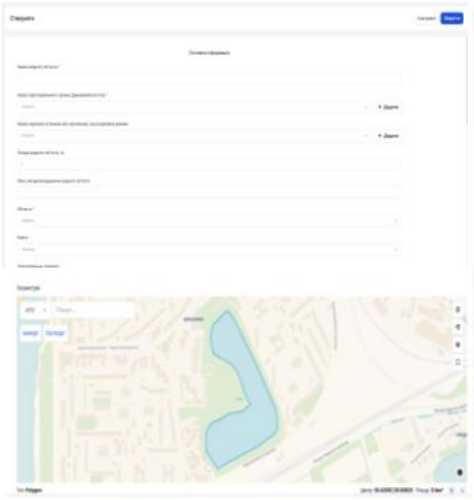
Таблиця реєстру



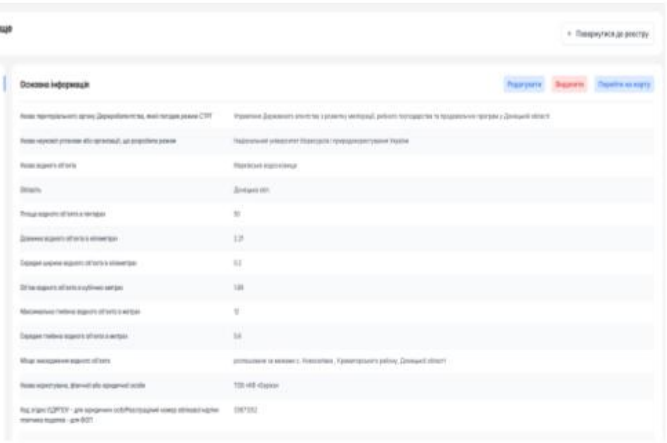
Фільтри таблиці

14

ЕКРАННІ ФОРМИ



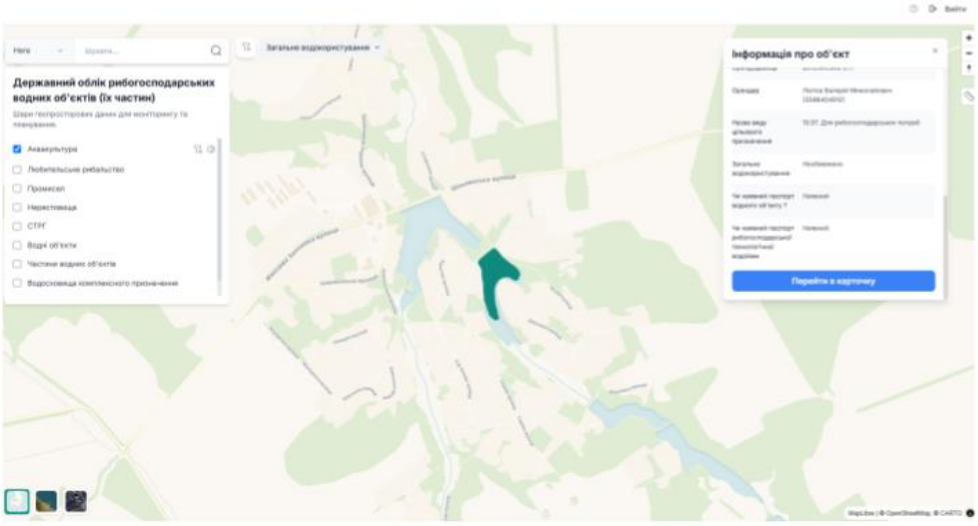
Форма створення об’єкту



Інформаційна картка об’єкту

15

ЕКРАННІ ФОРМИ



Карта з відображенням інформаційної картки об’єкта

16

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Усенко Б.О., Аброскін Ю.Ю. Система обліку водних об’єктів як інструмент для збору, обробки та аналізу даних, необхідних для прийняття управлінських рішень. Науково-технічна конференція «Сучасний стан та перспективи розвитку IoT». 15 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025, С. 460 - 463.
2. Усенко Б.О., Аброскін Ю.Ю. PostGIS як інструмент просторового аналізу даних у геоінформаційних системах. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24 квітня 2025 р., Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ, 2025, (подано до друку).

ВИСНОВКИ

1. Проведено аналіз літературних джерел з проблематикою обліку водних об’єктів в країні.
2. Розглянуто програмне забезпечення, яке може бути використано для обліку водних об’єктів та аналізу пов'язаної інформації. Було досліджено існуючі інструментальні засоби, зокрема системи на базі електронних таблиць (Google Sheets), ГІС-проєкти (QGIS) та європейську систему WISE.
3. За результатами порівняння аналогів сформульовано вимоги до розробки web-застосунку, включно з функціональними (керування шарами, пошук, фільтрація, перегляд інформації про об’єкти, адміністрування користувачів) та нефункціональними (кросбраузерність, безпека, розширюваність).
4. Спроєктовано архітектуру системи, яка включає клієнтську частину (Vue + TailwindCSS), серверну частину (Fastify), базу даних (PostgreSQL/PostGIS), картографічний модуль (MapLibreGL), REST API для взаємодії між компонентами.
5. Проаналізовано засоби розробки web-застосунку та обгрунтовано їх вибір.
6. Розроблено web-застосунок з функціоналом роботи з реєстрами, створення, редагування, видалення та перегляду об’єктів, геопросторового пошуку, фільтрів для об'єктів на карті, відображення спливаючих карток об'єктів, підключення шарів через vtile.
7. Здійснено юніт-тестування API та окремих функціональних модулів (обробка геоданих, застосування фільтрів, CRUD операції), що дозволило виявити та усунути критичні помилки на ранніх етапах.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
import './tailwind.js';
declare const window: any;

import { createApp } from 'vue';
import v3admin from '@opengis/admin/dist/admin.js';
import v3core from '@opengis/v3-core';
import v3filter from '@opengis/v3-filter';
import componentsApp from '../../components/index.js';

import '@opengis/v3-core/dist/style.css';
import '@opengis/admin/dist/style.css';

import App from './App.vue';
const app = createApp(App);

const settings = {
  titlePrefix: 'державний облік рибогосподарських
водних об’єктів (їх частин)',
  title:  ‘Облік рибогосподарських водних об’єктів (їх
частин)',
  logo: '/assets/logo.svg',
  favicon: '/assets/logo.svg',
  sidebarDark: false,
  sidebarColor: 'white',
  header: 'vs-header'
};

window.tailwind = {
  config: {
    darkMode: 'class'
  }
};

async function installApp() {
  v3core.install(app, { createApp });
  v3filter.install(app);
  //gis.install(app);
  await v3admin.install(app, { componentsApp, settings
});
  app.mount('#app');
}
installApp();
```

```
import gisLayerList from './controllers/gis-layer-list.js';
import gisFilters from './controllers/gisFilters.js';
import gisFormat from './controllers/gisFormat.js';
import baseLayers from './controllers/baseLayers.js';

import vtile from './controllers/vtile.js';
import layerColumn from './controllers/layerColumn.js';
export default async function route(app) {
  app.route({
    method: 'GET',
    url: '/gis-layer-list',
    handler: gisLayerList,
  });
  app.route({
```

```
'use strict'

import Fastify from 'fastify';
import closeWithGrace from 'close-with-grace'

import appService from './app.js'
import config from './config.js'

import { logger } from '@opengis/fastify-table/utils.js';

const app = Fastify({ logger })

app.register(appService)
const isProduction = process.env.NODE_ENV ===
'production';

if (isProduction) {
  const closeListeners = closeWithGrace({ delay:
process.env.FASTIFY_CLOSE_GRACE_DELAY || 500 },
async function ({ signal, err, manual }) {
    if (err) {
      app.log.error(err)
    }
    await app.close()
  });

  app.addHook('onClose', async (instance, done) => {
    closeListeners.uninstall()
    done()
  });
}
process.env.PORT = process.env.PORT || config.port ||
3000;
// Start listening.
app.listen({ host: '0.0.0.0', port: process.env.PORT }, (err)
=> {
  if (err) {
    app.log.error(err)
    process.exit(1)
  }
})

import { readFile } from 'fs/promises';
import fs from 'fs';
const config = fs.existsSync('config.json') ?
JSON.parse(await readFile('config.json')) : {};
const data = await pg.query(`SELECT (select
st_extent(geom)::box2d from ${table_name}) as geom,
(SELECT reltuples::bigint FROM pg_class WHERE oid =
to_regclass('${table_name}')) as count`).then(el =>
el.rows[0]);
  const bounds = data.geom ? data.geom.replace(/[A-
Z\)\(]+/g, '').split(/[ ,]+/).map(el => el - 0) : null
  return { bounds, count: data.count };

}

const time = Date.now();
```

```
    method: 'GET',
    url: '/gis-format-admin',
    handler: gisFormat,
  });
  app.route({
    method: 'GET',
    url: '/gis-filters/:layer',
    handler: gisFilters,
  });
  app.route({
    method: 'GET',
    url: '/vtile-admin/:layer/:lang/:z/:y/:x',
    handler: vtile,
  });
  app.route({
    method: 'GET',
    url: '/gis-layer/:id/:column',
    handler: layerColumn,
  });
  app.route({
    method: 'GET',
    url: '/base-layers',
    handler: baseLayers,
  });
}


import { getTemplatePath, getTemplate, pgClients }
from "@opengis/fastify-table/utils.js";

const pg = pgClients.client;
const data = await getTemplatePath('layer');

async function bboxTable(table_name) {
  const [ns, table] = table_name.split('.');
  // ST_EstimatedExtent('${ns}','${table}', 'geom') as
bounds


from ${table} group by unnest(${column})::text`)
      : await pg.query(`select ${column}::text as id,
count(*) from ${table} group by ${column}::text`);


    const cls = await getSelect(el.data, pg);

    const ids = rows.filter(el => el.id).map(el1 => el1.id);

    const clsData = await getSelectVal({ pg, values: ids,
name: el.data });
    console.log(clsData)
    const options = rows.map(cel => {
      const data = cls?.arr?.find(c => c.id === cel.id) || {
text: clsData[cel?.id] };
      return { ...cel, ...data };
    });
    Object.assign(el, { options });
  }));
  return filters
}
import { getTemplate, metaFormat } from
"@opengis/fastify-table/utils.js"

const layers = await Promise.all(data.map(async el =>
getTemplate('layer', el[0]).then(async d => ({
  id: el[0],
  service: 'vtile',
  url: `/api/vtile-admin/${el[0]}/ua/{z}/{x}/{y}.vmt`,
  ...await bboxTable(d.table_name),
  ...d
}))));
console.log('time:', Date.now() - time)

export default () => layers

import { getTemplate, getSelectVal, getSelect } from
"@opengis/fastify-table/utils.js"

export default async function gisFilters({ params, pg }) {

  const layer = await getTemplate('layer', params.layer)
  if (!layer) return { status: 404, message: 'not found' }
  const filters = layer.filter_list;
  if (!filters?.filter) return { status: 404, message: 'filters
empty' }
  const table = layer.table_name || layer.table;


  const { fields: columns } = await pg.query(`select * from
${table} limit 0`);

  await Promise.all(filters.filter((el) => el.data || el.type ===
'Check').map(async (el) => {
    const column = el.id || el.name;
    const { dataTypeID = 25 } = columns.find((col) =>
col.name === column) || {};

    const { rows } =
pg.pgType?.[dataTypeID]?.includes?.('[]')
      ? await pg.query(`select unnest(${column})::text as id,
count(*)

  await metaFormat({ rows, cls: layer.setting?.cls, sufix:
false });

  const columns = all.filter(el => type === 'list' ?
list.includes(el.name) : true)
    .map(el => `${el.title || el.name}| ${el.name}`).join('|');
  const html = await descriptionList(rows[0], { hash: {
columns, divider: '|' } });

  return { time: Date.now() - time, id: rows?.[0]?.id, rows,
html, columns: all, };
}

  component: default
isHeaderEditButton: false

panels:
  - type: vertical-tabs
    items:
      - name: main_info
        title: Основна інформація
      - name: parcel
        title: Земельні ділянки
      - name: lease
```

```
import descriptionList from
"../../../helpers/list/descriptionList.js";

export default async function gisFormat({ query, pg }) {
  const time = Date.now();
  const { table, id } = query;
  const layer = await getTemplate('layer', table);
  if (!layer) return { message: 'invalid dataset:' + table,
status: 404 };
  const layerTable = layer.table_name || layer.table;
  if (!layerTable) {
    return { message: 'invalid dataset: empty params
table', status: 400 };
  }

  const pk = pg.pk?.[layerTable];
  // data
  const { rows = [] } = pk ? await pg.query(`select *,
${pk}, geom::json from ${layerTable} where
${pk}=$1`, [id]) || {} : {};
  if (pk) rows.forEach((row) => Object.assign(row, { id:
row?.[pk] }));


  // title & columns

{{{button this token=(token table='estavok.acvaculture'
form='estavok.acvaculture.form' id=@root.id)
title="Редагувати" edit=1}}}
                <button onclick="window.v3plugin.$api({
api: '/api/obj-status/estavok.acvaculture/{{@root.id}}',
method:'get',confirm: { title:'Підтвердити операцію',
text: 'Ви впевнені що хочете вилучити запис?', cancel:
'Скасувати', confirm : 'Виконати'} })" class="px-2 py-1
inline-flex border-solid justify-center items-center gap-2
rounded-md font-semibold focus:outline-none text-sm
transition-all border border-transparent hover:text-white
ring-offset-white bg-red-100 text-red-500 hover:bg-red-
500 focus:ring-red-500">
                  Видалити
                  </button>
              {{/ifCondOr}}
          {{/contentList}}

          {{#if geom}}
             <button class="px-2 py-1 inline-flex border-
solid justify-center items-center gap-2 rounded-md font-
semibold focus:outline-none text-sm transition-all border
border-transparent hover:text-white ring-offset-white bg-
blue-100 text-blue-500 hover:bg-blue-500 focus:ring-
blue-500">
              <a
href="/map?x={{geom_point.[0]}}&y={{geom_point.[1
]}}&z=13&layers=estavok.acvaculture&selectedId={{
@root.id}}&selectedFeatureId=76&table=acvaculture">
Перейти на карту</a></button>
          {{/if}}
          </div>
    </div>
    </div>
       <br>
       {{{descriptionList
          this
          title: Інформація про оренду
       - name: hidrotech_construct
          title: Гідротехнічні споруди
       - name: afish
          title: Види водних біоресурсів
       - name: water_objects_parts
          title: Частини водного об'єкту
       - name: tech_project
          title: Паспорти рибогосподарської технологічної
водойми або технічного проекту
       - component: vs-widget-map
          title: Карта

<div>
    <div class="flex justify-between items-center border-b
pb-[15px]">
       <div class="font-semibold text-[16px] dark:text-
white">Основна інформація</div>
          <div class="flex gap-4">
          {{#contentList table="(select '{{region_atu}}'=
any(array_agg(atu_id)) as is_atu   from
estavok.user_setting_atu_rel where
user_uid='{{user.uid}}')"}}
              {{#ifCondOr rows.[0].is_atu '==' true
../user.user_type '==' 'admin'}}

{{#if parent_id}}Батьківський водний об'єкт{{/if}},<a
class='text-blue-500'
href='/card/estavok.water_objects_all.table/{{parent_id}}'>{
{select parent_id data='estavok.parent_id'}}</a>,
          Чи наявний паспорт рибогосподарської
технологічної водойми?,{{{badge is_tech_passport
data='estavok.is_passport'}}},
          Чи наявний паспорт водного об'єкта?,{{{badge
is_water_passport data='estavok.is_passport'}}},
          Цільове призначення земельної ділянки,{{select
water_usage_purpose data='estavok.purpose_code'}},
          Площа земельної ділянки; га,{{land_area}},
          Площа водного об'єкту; га, {{water_area}},
          Об'єм водного об'єкта; тис.
м³,{{water_volume_npr}},
          Обмеження (обтяження) або інші права третіх
осіб на орендовану земельну ділянку,{{restrictions}}
          "
       }}}

       <br>

       {{#ifCond is_water_passport '==' '1'}}
       <br>
       <div class="flex justify-between items-center border-b
pb-[15px]">
          <div class="font-semibold text-[16px] dark:text-
white">Відомості з паспорта водного об'єкта</div>
       </div>

       {{{descriptionList
          this
          comma=";"
          columns="Тип водного об'єкту,{{select water_type
data='water_type'}},
          Назва, {{water_name}},
```

```
        comma=";"
        columns="Назва водного об'єкта,water_object_name,
        Ідентифікатор об'єкта,rvo_code,
        Місцерозташування водного об'єкта,{{location}},
        {{#if community_atu}}Територіальна громада{{/if}},{{select community_atu data='edrato.atu_communities'}},
        Загальне водокористування,{{{badge general_use data='estavok.general_use'}}},
        Місцезнаходження водного об’єкта,{{water_location}},
        Об’єм при нормальному підпірному рівні; тис. м³,{{water_volume}},
        Скан-копія паспорту водного об'єкта,{{#if water_passport_file}}{{{buttonFilePreview water_passport_file}}}{{/if}}
        "
      }}}
    {{/ifCond}}
</div>


</div>
<div>
  {{#contentList table="(select '{{region_atu}}'= any(array_agg(atu_id)) as is_atu  from estavok.user_setting_atu_rel where user_uid='{{user.uid}}')"}}
  <div class="flex justify-between items-center mb-[15px]">
    <div class="font-semibold text-[16px]">Оренда</div>
    <div class="flex gap-4">
      {{#ifCondOr rows.[0].is_atu '==' true ../user.user_type '==' 'admin'}}
        {{{button this token=(token table='estavok.acvaculture_lease' obj=(concat 'acvaculture_id=' @root.id) form="estavok.acvaculture_lease.form" ) add=1}}}
      {{/ifCondOr}}
    </div>
  </div>
  <div class="overflow-x-auto">
  {{#contentList table="(select * from estavok.acvaculture_lease where acvaculture_id='{{id}}')"}}
  {{#ifCondOr ../rows.[0].is_atu '==' true ../user.user_type '==' 'admin'}}
    {{{tableList
      rows
      form="estavok.acvaculture_lease.form"
      id='acvaculture_lease_id'
      table='estavok.acvaculture_lease'
      uid=../user.uid
      nodata="<div class='bg-gray-200 text-center p-6 rounded-xl'><h3 class='text-lg font-semibold'>Інформація відсутня</h3></div>"
      comma=";"
      columns="
      Орендар,<a class='text-blue-500' href='/card/account_light.table/{{lessee_name_id}}'>{{select lessee_name_id data='account_id'}}</a>,
      Орендодавець,<a class='text-blue-500' href='/card/account_light.table/{{lessor_name_id}}'>{{select lessor_name_id data='account_id'}}</a>,
      Номер договору, {{contract_number}},
      Строк дії договору, {{contract_duration}},
      Дата укладання договору, {{formatDate contract_date}},

      Підстава для користування, {{select usage_basis data='estavok.usage_basis'}},
      Розмір орендної плати за користування земельною ділянкою; грн, {{land_rent_amount}},
      Розмір орендної плати за користування водним об'єктом; грн, {{water_rent_amount}}
      "
    }}}
  {{^}}
    {{{tableList
      rows
      form="estavok.acvaculture_lease.form"
      id='acvaculture_lease_id'
      table='estavok.acvaculture_lease'
      noactions='1'
      uid=../user.uid
      nodata="<div class='bg-gray-200 text-center p-6 rounded-xl'><h3 class='text-lg font-semibold'>Інформація відсутня</h3></div>"
      comma=";"
      columns="Орендар,{{select lessee_name_id data='account_id'}},
      Орендодавець,{{select lessor_name_id data='account_id'}},
      Номер договору, {{contract_number}},
      Строк дії договору, {{contract_duration}},
      Дата укладання договору, {{formatDate contract_date}},
      Підстава для користування, {{select usage_basis data='estavok.usage_basis'}},
      Розмір орендної плати за користування земельною ділянкою; грн, {{land_rent_amount}},
      Розмір орендної плати за користування водним об'єктом; грн, {{water_rent_amount}}"
    }}}
  {{/ifCondOr}}
  {{/contentList}}
  {{/contentList}}
  </div>
</div>
<div>
  <div class="flex justify-between items-center pb-[15px]">
    <div class="font-semibold text-[16px]">Показники</div>
  </div>
  <h4 class="text-center font-bold">Обсяги вилову</h4>
  <div class="flex gap-4 mb-5">
  {{{button this token=(token table='estavok.indicators' obj=(concat 'strg_id=' @root.id '#indicator_type=2')

{{/with}}
        {{^}}
```

```
                <td>-</td>
                <td>-</td>
                <td>-</td>
                {{/if}}
                {{/each}}
              </tr>
              {{/each}}
            </tbody>
          </table>
    {{/with}}
    {{^}}
    <div class='mb-5 mt-5 bg-gray-200 text-center p-6
rounded-xl'><h3 class='text-lg font-semibold
'>Показники відсутні</h3></div>
    {{/if}}
    {{/contentList}}
    <br>
    <h4 class="text-center font-bold">Обсяги
вселення</h4>
    <div class="flex gap-4 mb-5">
    {{{button this token=(token table='estavok.indicators'
obj=(concat 'strg_id=' @root.id '#indicator_type=1')
form="estavok.strg_indicator.form" ) add=1}}}
    </div>
    {{#contentList table="(select
estavok.crosstab_strg_indicators('{{@root.id}}',1) )"}}
      {{#if rows.[0].crosstab_strg_indicators.data}}
      {{#with rows.[0].crosstab_strg_indicators}}
        <table class="min-w-full divide-y divide-gray-
200">
          <thead>
            <tr class="border-t border-gray-200">
              <th rowspan="2">Вид водних
біоресурсів (українська назва)</th>
              <th rowspan="2">Вид водних
біоресурсів (латинська назва)</th>
              {{#each year_list}}
              <th colspan="3">{{this}}</th>
              {{/each}}
            </tr>
            <tr class="border-t border-gray-200">
              {{#each year_list}}
              <th>план</th>
              <th>факт</th>
              <th>дії</th>
              {{/each}}
<template>
  <div class="vs-map-portal relative">
    <div
      id="maplibre-map"
      ref="maplibreMap"
      :style="{ height: 'calc(100vh - 60px)', width: '100%'
}"
    ></div>
    <MapSearch :map />
    <MapLayers :map :layers v-if="layers &&
mapLoaded" />
    <MapBaseLayers :map />
    <MapRuler :map @toggleMeasureState="measure =
$event" />
    <MapCard
      v-if="features?.length"
      :map
```

```
            </td>

            {{#each data as | _obj |}}
            <tr class="border-t border-gray-200">
              <td>{{#if afish_id}}{{select afish_id
data="afish_id_ukr"}}{{^}}Інші
                  види{{/if}}</td>
              <td>{{select afish_id
data="afish_id_lat"}}</td>
              {{#each ../year_list}}
              {{#if (lookup _obj this)}}
              {{#with (lookup _obj this)}}
              <td>{{this.plan}}</td>
              <td>{{this.fact}}</td>
              <td>
                {{{button this token=(token
table='estavok.indicators' form='estavok.strg_indicator.form'
id=this.indicator_id) edit=1 icon=1}}}
                {{{button this token=(token
table='estavok.indicators' id=this.indicator_id) del=1
icon=1}}}
              </td>
              {{/with}}
              {{^}}
              <td>-</td>
              <td>-</td>
              <td>-</td>
              {{/if}}
              {{/each}}
            </tr>
            {{/each}}
          </tbody>
        </table>
      {{/with}}
      {{^}}
      <div class='mt-5 bg-gray-200 text-center p-6 rounded-
xl'><h3 class='text-lg font-semibold '>Показники
відсутні</h3></div>
      {{/if}}
      {{/contentList}}
</div>
            </tr>
          </thead>
          <tbody>
rounded-lg border shadow-lg overflow-hidden flex flex-col"
 >
  <div class="flex items-start justify-between border-b p-4
pb-1">
    <div class="text-xl font-semibold text-gray-900 mb-2">
      Інформація про об'єкт
    </div>
    <closeIcon
      @click="$emit('close')"
      class="cursor-pointer text-gray-500 hover:text-gray-
700"
    />
  </div>

  <div
    class="text-gray-800 p-4 overflow-y-auto h-full flex-1
break-words overflow-x-hidden [&::-webkit-scrollbar]:h-2
[&::-webkit-scrollbar]:w-2 [&::-webkit-scrollbar-
```

```
      :html="features[activeFeatureIndex]?.html"
      :id="features[activeFeatureIndex]?.id"
      :table="features[activeFeatureIndex]?.table"
      :active-index="activeFeatureIndex"
      :total-features="features.length"
      @close="closeCard"
      @next-feature="nextFeature"
      @prev-feature="prevFeature"
    />
  </div>
</template>

<script>
import axios from "axios";
import * as turf from "@turf/turf";
import maplibregl from "maplibre-gl";
import "maplibre-gl/dist/maplibre-gl.css";

import MapLayers from "./components/vs-map-
layers.vue";
import MapBaseLayers from "./components/vs-map-
base-layers.vue";
import MapRuler from "./components/vs-map-
ruler.vue";
import layerMixin from "./mixins/layerMixins.js";
import MapSearch from "./components/search/vs-map-
search.vue";
import MapCard from "./components/vs-map-card.vue";
import config from "../../config.json";
<template>
  <div
    class="absolute top-6 right-16 w-[430px] max-h-
[500px] bg-white
thumb]:rounded-full [&::-webkit-scrollbar-track]:bg-stone-
100 [&::-webkit-scrollbar-thumb]:bg-stone-300"
    >
      <div v-html="html" class="break-words"></div>
      <a
        target="_blank"
        :href="`/card/${table}.table/${id}?tab=main_info`"
        v-if="table !== 'water_parts'"
        class="w-full flex items-center justify-center bg-blue-
500 text-white font-semibold py-2 px-4 rounded-lg
hover:bg-blue-600 focus:outline-none focus:ring-2
focus:ring-blue-400"
      >
        Перейти в карточку
      </a>
    </div>
  </div>
</template>

<script>
import closeIcon from "../icons/close.vue";
export default {
  props: ["html", "table", "id"],
  components: { closeIcon },
};
</script>
```