Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Розробка веб-інтерфейсу для інтерактивної візуалізації API-даних на основі Java-Script

Виконав(ла): студент(ка) 6 курсу, групи СПм-61
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)

		Коваль В. В
	(підпис)	(прізвище та ініціали)
Керівник		Цуприк Г. Б.
	(підпис)	(прізвище та ініціали)
Нормоконтроль		Стоянов Ю. М.
	(підпис)	(прізвище та ініціали)
Завідувач кафедри		Петрик М. Р.
	(підпис)	(прізвище та ініціали)
Рецензент		Никитюк В. В.
	(підпис)	(прізвище та ініціали)

Тернопіль
2024

АНОТАЦІЯ

Кваліфікаційна робота магістра містить: _76_ сторінок, _15_ рисунків, 3_ додатків, 21 джерел.

Ключові слова: API, ВІЗУАЛІЗАЦІЯ ДАНИХ, ІНТЕРАКТИВНІСТЬ, JAVASCRIPT, NEXT.JS, NODE.JS, ECHARTS, ВЕБ-ІНТЕРФЕЙС.

Мета роботи – створення веб-інтерфейсу, який дозволяє користувачам працювати з великими обсягами інформації в реальному часі, забезпечуючи персоналізовану взаємодію та зручність налаштувань візуалізації.

Методи дослідження базуються на використанні сучасних фреймворків та бібліотек JavaScript, зокрема Next.js та ECharts.

Елементи наукової новизни – створення інтерфейсу для інтерактивної взаємодії з даними в реальному часі.

Висновки та пропозиції щодо розвитку:

Розроблений веб-інтерфейс є основою для подальшого вдосконалення методів роботи з даними, зокрема:

Додавання можливості інтеграції із зовнішніми базами даних.

Розширення функціоналу для роботи з географічними даними.

Оптимізація продуктивності для роботи з надвеликими масивами інформації.

ANNOTATION

The master's thesis consists of _76_ pages, _15_ figures, _3 appendices, 21 sources.

Keywords: API, DATA VISUALIZATION, INTERACTIVITY, JAVASCRIPT, NEXT.JS, NODE.JS, ECHARTS, WEB INTERFACE.

Objective: The aim of the study is to develop a web interface that allows users to work with large volumes of information in real time, providing personalized interaction and convenient visualization customization options.

Research Methods: The study is based on the use of modern JavaScript frameworks and libraries, particularly Next.js and ECharts.

Elements of Scientific Novelty: The creation of an interface for interactive real-time data interaction.

Conclusions and Recommendations for Development:

The developed web interface serves as a foundation for further improvement of data interaction methods, including:

Adding the ability to integrate with external databases.

Expanding functionality for working with geospatial data.

Optimizing performance for handling massive datasets.

ЗМІСТ

ВСТУП

Цифрові технології стали фундаментальною складовою сучасного життя, здійснюючи значний вплив на такі сфери, як бізнес, наука, освіта, медицина та державне управління. Однією з ключових проблем сьогодення є ефективне управління великими обсягами даних, які постійно генеруються в різних галузях. Для вирішення цієї задачі використовується інтерактивна візуалізація даних, що дозволяє не лише аналізувати великі масиви інформації, а й ефективно представляти результати у зручному та зрозумілому вигляді. Веб-інтерфейси, які забезпечують динамічне відображення даних і зручну взаємодію з користувачем, є важливою складовою сучасних інформаційних систем.

Розробка таких веб-інтерфейсів потребує використання передових технологій і підходів. JavaScript, як універсальна мова програмування для веб-додатків, забезпечує високу продуктивність і гнучкість рішень, дозволяючи створювати динамічні додатки, які адаптуються до потреб користувача. Інтеграція API для отримання даних і їх подальша обробка забезпечують можливість створення інструментів для аналізу в режимі реального часу, що особливо актуально для бізнес-аналітики, моніторингу, наукових досліджень і багатьох інших сфер.

Мета роботи — створити веб-інтерфейс для інтерактивної візуалізації API-даних, який забезпечуватиме високу продуктивність, зручність використання та широкий спектр налаштувань. Основними завданнями є:

- Проведення аналізу сучасних підходів і технологій візуалізації даних у веб-середовищі.
- Розробка архітектури веб-додатка з інтеграцією API для отримання та обробки даних.
- Реалізація веб-інтерфейсу за допомогою JavaScript-бібліотек і фреймворків, таких як Next.js, Node.js, та ECharts.js.

Об’єкт дослідження — процес інтерактивної візуалізації даних, отриманих через API.

Предмет дослідження — технології та методи розробки веб-інтерфейсів для інтерактивної візуалізації даних з використанням сучасних JavaScript-інструментів.

Методи дослідження:

- Аналіз літературних джерел та існуючих рішень — для вивчення актуальних технологій та підходів у візуалізації даних.
- Проєктування — для визначення архітектури додатка, що відповідає поставленим вимогам.
- Програмування — для створення інтерактивного та функціонального веб-інтерфейсу.

Наукова новизна роботи полягає в удосконаленні підходів до інтерактивної візуалізації даних через впровадження сучасних JavaScript-технологій. Уперше було розроблено методи інтеграції персоналізованих функцій візуалізації, що дозволяють працювати з даними в реальному часі, забезпечуючи зручність і швидкість для кінцевого користувача.

Результати дослідження можуть бути використані у створенні інформаційних систем для бізнесу, науки, аналізу соціальних мереж, медицини та інших галузей, де важливу роль відіграє обробка даних. Запропоновані рішення допоможуть значно підвищити ефективність роботи з даними та забезпечити інтуїтивну взаємодію з користувачами.

Для реалізації було використано сучасні технології, такі як Next.js для розробки клієнтської частини, Node.js для обробки серверної логіки, та ECharts.js для інтерактивної візуалізації даних.

1 ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ВЕБ-ІНТЕРФЕЙСІВ

В результаті обговорення я обрав собі тему, яку і погодив мій керівник, що стосується розробки веб-інтерфейсу для інтерактивної візуалізації API-даних на основі JavaScript [1-2]. Для кращого розуміння теми потрібно глибше ознайомитись з основами цієї мови програмування та її супутніми бібліотеками.

1.1 Історія JavaScript та веб-інтерфейсів

1.1.1 Початок еволюції JavaScript

JavaScript був створений у 1995 році Бренданом Айхом, який працював у компанії Netscape Communications. Спочатку мова програмування була розроблена для використання у браузері Netscape Navigator, щоб надати можливість веб-сторінкам бути більш динамічними і інтерактивними. До цього часу веб-сторінки були статичними і не дозволяли користувачам взаємодіяти з ними у реальному часі. Технології HTML та CSS тільки забезпечували структуру і стиль, але не могли змінювати контент сторінки без її перезавантаження.

JavaScript спочатку називався "LiveScript", однак під тиском маркетингової стратегії він був перейменований на JavaScript, що мало на меті привернути увагу до популярної на той час мови програмування Java. Незважаючи на схожість в іменах, JavaScript і Java є абсолютно різними мовами програмування з різними синтаксисами і підходами до розробки [3].

Протягом перших років свого існування JavaScript здебільшого використовувався для простих інтерактивних елементів на веб-сторінках, таких як валідація форм або зміна стилю елементів за допомогою подій (наприклад, натискання кнопки). Це дозволило зробити веб-сторінки більш зручними та

привабливими для користувачів, але не призводило до значних змін в архітектурі веб-додатків.

1.1.2 Революція AJAX

Великою віхою в еволюції JavaScript стала поява AJAX (Asynchronous JavaScript and XML) в середині 2000-х років. AJAX дозволяв здійснювати асинхронні запити до сервера, не перезавантажуючи сторінку. Це значно покращило користувацький досвід, оскільки зміни на сторінці можна було здійснювати миттєво, не чекаючи на завантаження всього контенту знову.

Одним із перших масштабних прикладів використання AJAX був Google Maps, який дозволив користувачам взаємодіяти з картою динамічно, без необхідності оновлення сторінки. Це дозволило змінити підхід до створення веб-додатків, де динамічність і інтерактивність стали основними характеристиками.

Технологія AJAX також стала основою для подальшого розвитку концепції Single Page Application (SPA), у якій веб-додаток завантажує лише один HTML документ і динамічно оновлює зміст сторінки в залежності від взаємодії з користувачем, не перезавантажуючи всю сторінку [4].

1.1.3 Розвиток фреймворків і бібліотек

У середині 2000-х років почали з'являтися перші популярні JavaScript фреймворки та бібліотеки, які значно спростили розробку веб-додатків. Одним з таких перших фреймворків був jQuery, який надавав зручний API для маніпулювання елементами DOM (Document Object Model), обробки подій, а також

для роботи з AJAX-запитами. Це дозволило розробникам писати менше коду, а також полегшило крос-браузерну сумісність.

У 2010 році був випущений AngularJS, що став першим популярним інструментом для створення додатків з архітектурою однієї сторінки (SPA). AngularJS дозволяв розробникам створювати складні веб-додатки, використовуючи концепцію двосторонньої прив'язки даних і MVC (Model-View-Controller) архітектури. Це значно спростило керування станом додатка та підвищило зручність його тестування.

У 2013 році був випущений React.js — бібліотека для побудови інтерфейсів користувача, розроблена компанією Facebook. React став основним інструментом для створення компонентних інтерфейсів. Однією з ключових переваг React є використання Virtual DOM, що дає змогу оновлювати лише ті частини інтерфейсу, які зазнали змін, без необхідності перезавантаження всієї сторінки [5].

Інші бібліотеки, такі як Vue.js та Svelte, також отримали значну популярність завдяки своїй легкості у використанні та гнучкості. Всі ці інструменти стали основою для розробки складних інтерактивних веб-додатків, включаючи візуалізацію даних і інтерактивні елементи.

1.1.4 Розвиток інтерактивних веб-інтерфейсів

З появою фреймворків, таких як React, Angular і Vue, а також інструментів для візуалізації даних, наприклад, D3.js та EChart.js, інтерактивні веб-інтерфейси перетворилися на ключовий елемент сучасних веб-додатків. Завдяки цим технологіям веб-розробники отримали можливість створювати динамічні інтерфейси, які дозволяють взаємодіяти з користувачем у режимі реального часу.

Сучасні веб-додатки, що включають візуалізацію великих обсягів даних, можуть бути інтерактивними та адаптивними, змінюючи свій вигляд і поведінку в залежності від дій користувача. Наприклад, інтерактивні графіки та карти

дозволяють користувачам взаємодіяти з даними, змінювати фільтри або вибирати категорії для більш детального перегляду.

Веб-інтерфейси також стали більш доступними завдяки використанню CSS3 для анімацій і трансформацій, а також новим можливостям HTML5, які дозволяють інтегрувати мультимедійні елементи, такі як відео та аудіо, без необхідності використання сторонніх плагінів.

1.2 Відомості про API-дані та інтерактивну візуалізацію

1.2.1 Роль API в сучасному веб-розробці

API (Application Programming Interface) є важливою складовою сучасних веб-додатків. Вони забезпечують можливість взаємодії між різними програмними системами, дозволяючи одній програмі запитувати дані або функціональність у іншої. У контексті веб-розробки API зазвичай використовуються для отримання даних з серверів, інтеграції сторонніх сервісів, або взаємодії з базами даних через інтерфейси REST, SOAP, або GraphQL.

Історично API використовувалися переважно для внутрішньої взаємодії між серверами і програмами, але з розвитком веб-технологій API стали основним інструментом для інтеграції зовнішніх джерел даних. Веб-сервіси, як-от Google Maps, Weather API, або соціальні медіа API, дозволяють розробникам інтегрувати дані з різних джерел у свої веб-додатки, надаючи користувачам доступ до корисної інформації у реальному часі.

API можуть мати різні типи, але в контексті веб-розробки найпоширенішими є RESTful API та GraphQL API. RESTful API ґрунтуються на принципах архітектури REST (Representational State Transfer), що дозволяє отримувати доступ до ресурсів через HTTP-запити (GET, POST, PUT, DELETE), тоді як GraphQL дає змогу клієнту запитувати лише необхідні дані, мінімізуючи надлишкові запити [6].

Завдяки використанню API, веб-розробники можуть інтегрувати з різними зовнішніми даними, такими як фінансові потоки, курси валют, погода, або навіть інші веб-додатки, що значно покращує функціональність і корисність веб-сайтів.

1.2.2 Інтерактивна візуалізація даних

Інтерактивна візуалізація даних – це процес створення графічних інтерфейсів, які дозволяють користувачам взаємодіяти з даними та змінювати вигляд візуалізації у реальному часі. Візуалізація даних є потужним інструментом для розуміння та аналізу великих обсягів інформації, допомагаючи користувачам швидко знайти важливі закономірності, тренди або аномалії в даних.

Інтерактивні візуалізації дозволяють користувачам змінювати вид графіків, фільтрувати дані, а також виконувати дії, які приводять до динамічного оновлення графіків або карт. Наприклад, користувач може вибрати певний період часу для аналізу даних або налаштувати фільтри для відображення тільки тих елементів, які йому цікаві.

Веб-технології, такі як HTML5, CSS3 та JavaScript, дозволяють створювати інтерактивні графіки без необхідності використання додаткових плагінів або програмного забезпечення. Це зробило візуалізацію даних більш доступною для широкого кола розробників і користувачів [7].

1.2.3 Основні бібліотеки для інтерактивної візуалізації даних

Серед найбільш популярних бібліотек для інтерактивної візуалізації даних на JavaScript можна виділити:

D3.js — велика бібліотека для маніпулювання даними та їх візуалізацією. Вона дозволяє створювати інтерактивні графіки та діаграми, маніпулюючи елементами DOM з даними. D3.js надає багатий набір інструментів для побудови адаптивних, анімованих та динамічних графічних елементів, а також для роботи з географічними даними.

ECharts — це потужна бібліотека для побудови графіків, яка пропонує широкий набір інструментів для розробки інтерактивних графічних візуалізацій даних. Вона підтримує не лише стандартні типи графіків, як лінійні, стовпчикові або кругові діаграми, але й складніші формати, такі як теплові карти, географічні карти та графи. ECharts забезпечує анімацію та інтерактивність, що дозволяє користувачам взаємодіяти з даними без необхідності перезавантаження сторінки.

Highcharts – ще одна популярна бібліотека для побудови інтерактивних графіків. Вона пропонує великий вибір типів графіків та розширених можливостей для налаштування інтерактивних елементів [8].

Ці бібліотеки дозволяють не тільки візуалізувати дані, але й інтерактивно взаємодіяти з ними, роблячи досвід користувачів більш динамічним і зручним.

1.2.4 Застосування інтерактивної візуалізації в реальних додатках

Інтерактивна візуалізація даних має широке застосування в різних галузях:

Фінанси – візуалізація фондових ринків, динаміки цін на акції, валютних курсів. Це дозволяє інвесторам приймати швидкі рішення на основі реальних даних.

Охорона здоров'я – візуалізація медичних даних, таких як результати аналізів, показники здоров'я пацієнтів, допомагаючи лікарям краще розуміти стан пацієнтів та ухвалювати обґрунтовані рішення.

Інтернет речей (IoT) – інтерактивні візуалізації для моніторингу стану пристроїв IoT, таких як датчики температури, вологості, або навіть моніторинг енергоспоживання.

Економіка і бізнес-аналітика – візуалізація великих обсягів даних для аналізу трендів, прогнозів та інших важливих метрик, що дозволяє компаніям приймати стратегічні рішення.

Інтерактивна візуалізація також важлива в науці та освіті, де складні концепти можуть бути краще зрозумілі завдяки візуальному представленню даних. Вона допомагає не лише в аналізі даних, але й у навчанні та поширенні знань серед широкої аудиторії [9].

1.3 Опис предметної області

1.3.1 Визначення предметної області

Предметною областю цього дослідження є створення інтерактивного веб-інтерфейсу для візуалізації даних, що отримуються з API, з використанням сучасних веб-технологій, таких як JavaScript, HTML5, CSS3 та бібліотек для візуалізації даних. Метою є розробка системи, яка дозволяє користувачам візуалізувати різноманітні дані, отримані через API, в інтерактивному форматі, що дає можливість здійснювати детальний аналіз і взаємодіяти з даними в реальному часі.

Цей підхід зосереджений на створенні веб-додатків, які інтегрують зовнішні джерела даних через API, що дозволяє не тільки відображати дані, але й надавати користувачам можливість налаштовувати вигляд візуалізації, фільтрувати інформацію та взаємодіяти з графіками та іншими елементами інтерфейсу. Таким чином, предметна область охоплює як технічну сторону інтеграції API, так і аспекти проектування ефективних і зручних інтерфейсів для візуалізації та взаємодії з даними.

1.3.2 Мета і завдання предметної області

Основною метою предметної області є розробка веб-інтерфейсу, який дозволяє ефективно візуалізувати дані з різних джерел за допомогою API, а також надавати можливість взаємодії з цими даними через інтерактивні елементи інтерфейсу.

Для досягнення цієї мети необхідно вирішити кілька основних завдань:

- Збір і обробка даних через API: Розробити механізм інтеграції з зовнішніми джерелами даних через API. Це включає в себе вибір відповідних API, створення запитів та обробку отриманих даних у зручному форматі для відображення на веб-інтерфейсі.
- Інтерактивна візуалізація даних: Створити інтерактивні елементи, які дозволяють користувачам взаємодіяти з графіками та іншими візуальними елементами. Наприклад, можливість фільтрації даних, налаштування параметрів відображення або зміна вигляду візуалізації.
- Розробка зручного і адаптивного інтерфейсу: Забезпечити зручний інтерфейс для користувача, що дозволяє легко зрозуміти і взаємодіяти з даними. Інтерфейс повинен бути адаптованим до різних пристроїв і екранних розмірів, забезпечуючи комфортне використання як на десктопах, так і на мобільних пристроях.

1.3.3 Визначення основних компонентів

Предметна область включає кілька основних компонентів, кожен з яких є важливим для досягнення мети проекту:

API-інтерфейси це основа для отримання даних, які використовуються для візуалізації. API можуть бути різних типів, в залежності від джерела даних.

Наприклад, для отримання даних про фінансові ринки можуть використовуватись API для біржових котирувань, для візуалізації географічних даних – API картографічних сервісів (наприклад, Google Maps або OpenStreetMap).

Інтерактивні графіки та діаграми. Інтерактивність може включати можливість масштабування, фільтрації, виділення окремих елементів або груп даних.

Користувацький інтерфейс (UI), через який користувачі взаємодіють з даними. Це можуть бути різні елементи управління (кнопки, фільтри, поля для введення) і візуальні компоненти (графіки, таблиці, карти).

Функціонал фільтрації і налаштування, це важливий компонент, який дозволяє користувачам налаштовувати, які саме дані вони хочуть бачити, змінювати параметри візуалізації, отримувати детальнішу інформацію по окремих елементах.

1.3.4 Типи даних для візуалізації

Для інтерактивної візуалізації можуть використовуватися різноманітні типи даних, зокрема:

- Часові ряди: Це дані, які змінюються в часі, наприклад, фінансові котирування акцій, температура повітря, рівень вологості тощо. Для візуалізації таких даних використовуються лінійні графіки або діаграми.
- Кількісні дані: Дані, що вимірюються в числових величинах, наприклад, кількість відвідувачів сайту, обсяги продажу. Для таких даних зазвичай використовують стовпчикові або кругові діаграми [10].
- Категоріальні дані: Дані, що представляють різні категорії, наприклад, типи продуктів, регіони, групи користувачів. Для візуалізації таких даних використовують гістограми або діаграми розподілу.

- Геопросторові дані: Дані, що мають географічні координати, наприклад, місце розташування користувачів, дані з картографічних сервісів. Для таких даних зазвичай використовуються інтерактивні карти .

1.4 Постановка задачі розробки

Розробка веб-інтерфейсу для інтерактивної візуалізації API-даних передбачає кілька важливих етапів, на яких слід зосередитися для забезпечення ефективної роботи системи. Постановка задачі охоплює визначення основних вимог до функціональності системи, зокрема до інтерактивності, точності візуалізації та ефективності роботи з великими об'ємами даних.

1.4.1 Загальні вимоги до системи

Користувач повинен мати можливість взаємодіяти з даними в реальному часі. Це охоплює фільтрацію, масштабування, вибір підмножини даних та налаштування відображення візуалізацій. Ідея інтерактивності полягає в тому, щоб дати користувачеві максимальний контроль над процесом відображення та аналізу даних, що дозволяє йому отримувати потрібну інформацію з графіків і діаграм у реальному часі, адаптуючи їх до своїх потреб і вимог.

Веб-інтерфейс має бути зручним для користувачів на різних пристроях, включаючи десктопи та мобільні телефони. Система повинна автоматично адаптуватися до любих розмірів екранів, гарантуючи комфортне використання та відображення контенту, а також підтримку функціональності незалежно від типу пристрою.

Враховуючи обробку великих обсягів даних, система повинна працювати швидко і без затримок при завантаженні, маніпулюванні або оновленні даних. Це особливо важливо при роботі з великими наборами даних або при здійсненні складних запитів до API.

Система повинна включати механізми для захисту даних, які надходять і відправляються через API. Важливим аспектом є також забезпечення безпеки доступу до інтерфейсу, особливо в разі роботи з конфіденційною інформацією.

1.4.2 Конкретні задачі проекту

Необхідно розробити механізми для інтеграції та взаємодії з різними API. Це включає створення запитів до API, а також обробку і форматування отриманих даних для подальшої візуалізації. Важливо враховувати типи даних, що передаються, і оптимізувати запити, щоб забезпечити ефективний доступ до необхідної інформації.

Потрібно спроектувати інтерфейс, що дозволяє користувачеві ефективно взаємодіяти з даними. Це включає розробку панелей керування для налаштування фільтрів, відображення різних типів візуалізацій (графіки, діаграми, таблиці) і налаштування вигляду інтерфейсу для зручності користувача.

Також одним з ключових завдань є вибір відповідних бібліотек для створення інтерактивних графіків та діаграм. Це можуть бути такі популярні бібліотеки, як D3.js, EChart.js, Plotly, або Highcharts, які дозволяють створювати різноманітні типи графіків, що можна інтерактивно налаштовувати.

Враховуючи, що інтерфейс буде працювати з великими обсягами даних, необхідно розробити механізми для оптимізації процесів завантаження та відображення інформації. Це може включати використання таких методів, як lazy loading, кешування даних, асинхронні запити та інші техніки для покращення швидкодії [11].

Інтерфейс повинен бути інтуїтивно зрозумілим і простим у використанні. Для цього потрібно провести тестування користувачів і вдосконалити інтерфейс на основі їх відгуків. Це включає створення зрозумілих і зрозумілих елементів управління, а також надання користувачам чітких інструкцій.

1.5 Вибір технології розробки

Оскільки від вибору технологій для створення веб-інтерфейсу для інтерактивної візуалізації API-даних залежатимуть функціональність, продуктивність, масштабованість та зручність використання системи, цей етап є надзвичайно важливим. Для реалізації такого проекту слід врахувати вимоги до обробки великих обсягів даних, наявність інтерактивних елементів, а також потребу в забезпеченні високої продуктивності та доступності на різних пристроях.

1.5.1 Основні технології для фронтенду

HTML, CSS і JavaScript — це основні технології для розробки сучасних веб-інтерфейсів. HTML формує структуру веб-сторінки, CSS відповідає за її зовнішній вигляд та стилі, а JavaScript забезпечує інтерактивність. Враховуючи потреби проекту, JavaScript буде основною мовою для розробки динамічних елементів інтерфейсу та взаємодії з API.

Новітня версія мови HTML надає потужні можливості для створення адаптивних веб-сторінок завдяки новим тегам, таким як <article>, <section>, <canvas>, а також API для роботи з мультимедіа та геолокацією. Використання HTML5 дозволяє створювати легкі та швидкі веб-сторінки, що є важливим для продуктивності інтерфейсу [12].

З використанням CSS3 можна реалізувати адаптивний дизайн, а також застосовувати новітні технології, такі як медіа-запити, анімації, трансформації та градієнти. Це дозволяє створити інтерфейс, який чудово виглядає на різних пристроях.

Мова програмування, що відповідає за логіку взаємодії користувача з веб-сторінкою. Вона дозволяє обробляти події, працювати з API, створювати динамічні графіки та інші елементи інтерфейсу.

1.5.2 Фреймворки і бібліотеки для фронтенду

Для створення складних і масштабованих веб-додатків можна використовувати бібліотеку React, яка дозволяє ефективно керувати станом інтерфейсу і відображенням компонентів на сторінці. React дозволяє швидко розробляти інтерактивні елементи інтерфейсу, ідеально підходить для створення динамічних компонентів, таких як таблиці або графіки, що реагують на зміни в API.

Завдяки JSX розробка інтерфейсу стає більш зручною, оскільки компоненти можна описувати безпосередньо в JavaScript-коді. JSX синтаксис дозволяє легко інтегрувати HTML та JavaScript, що підвищує ефективність розробки.

Для створення багатосторінкових додатків, де потрібно змінювати вміст без перезавантаження сторінки, React Router забезпечує навігацію між різними компонентами.

Для реалізації інтерактивних візуалізацій на основі API-даних доцільно використовувати ECharts. Це потужна бібліотека для побудови інтерактивних графіків, діаграм та інших візуальних елементів, яка підтримує численні типи візуалізацій, від лінійних графіків до складних карт. ECharts дозволяє легко налаштовувати відображення даних і інтерактивність, зокрема підтримує масштабування, панорамування, підсвічування та інші елементи взаємодії з

користувачем, що робить його ідеальним для створення інтерактивних та динамічних візуалізацій [13].

Також є популярними Chart.js та Plotly це інструменти для створення візуалізацій даних. Вони мають простий API, що дозволяє швидко створювати стандартні графіки (лінійні, стовпчикові, кругові та інші). Вибір між D3.js, Chart.js і Plotly залежить від складності візуалізацій та вимог до їх інтерактивності.

Chart.js: Легкий у використанні для базових графіків. Має хорошу документацію і активну спільноту.

Plotly: Підтримує більш складні графіки та інтерактивні візуалізації, такі як 3D-графіки, що можуть бути корисні для відображення великих наборів даних.

1.5.3 Технології для роботи з API

Для роботи з API даними, важливими є не лише вибір способу взаємодії з сервером, а й вибір підходу до обробки та візуалізації отриманих даних.

Однією з найбільш популярних бібліотек для здійснення HTTP-запитів у JavaScript є Axios. Axios спрощує роботу з асинхронними запитами, підтримує такі функції, як перехоплювачі запитів і відповіді, що дозволяє більш ефективно обробляти дані з API.

Axios працює з промісами, що дозволяє зручно обробляти асинхронні запити. Також можна легко налаштувати обробку помилок або додавати заголовки до кожного запиту, що робить цю бібліотеку гнучкою та потужною.

Як альтернатива традиційним REST API, можна використовувати GraphQL, який надає можливість запитувати тільки ті дані, які потрібні, що знижує обсяг переданих даних та покращує продуктивність. Це особливо корисно при роботі з великими обсягами даних і складними структурами.

1.5.4 Бекенд-технології

Для побудови бекенд-частини додатку, яка буде взаємодіяти з API, можна використовувати популярні технології на основі JavaScript, такі як Node.js. Node.js є потужним середовищем для розробки серверних додатків на JavaScript. Він має безліч вбудованих модулів для роботи з HTTP-запитами, базами даних, автентифікацією та іншими задачами. Node.js також має велику екосистему бібліотек, що дозволяє швидко розв'язувати багато типових завдань.

Одним із зручних інструментів для тестування і розробки API є MockAPI. Це інструмент, що дає змогу створювати та взаємодіяти з мок-сервісами API без потреби налаштовувати реальний сервер чи базу даних. MockAPI надає простий спосіб для створення фейкових RESTful API з можливістю визначення ендпоінтів, методів (GET, POST, PUT, DELETE) та структур даних, які повертаються в відповіді. Це особливо корисно на етапі фронтенд-розробки, коли потрібно протестувати взаємодію з API, але реальний бекенд ще не готовий.

MockAPI підтримує автентифікацію, фільтрацію даних та пагінацію, що робить його ефективним інструментом для тестування та розробки інтерфейсів. Крім того, він дозволяє легко інтегрувати мок-сервіси в проєкти, що дозволяє не зупиняти процес розробки через відсутність доступу до реального бекенду.

1.6 Опис проектування програмного забезпечення

Проектування програмного забезпечення для веб-інтерфейсу для інтерактивної візуалізації API-даних передбачає створення гнучкої та ефективної архітектури, яка дозволить інтерактивно відображати великі обсяги даних, а також забезпечити зручність користування для кінцевого користувача. У даному проекті використовуються сучасні технології, такі як Next.js та Tailwind для клієнтської

частини, а для серверної частини планується використання Node.js або MockAPI. Весь процес проектування складається з кількох етапів, починаючи від розробки архітектури системи до опису конкретних компонентів.

1.6.1 Архітектура програмного забезпечення

У випадку веб-інтерфейсу для інтерактивної візуалізації API-даних, архітектура програмного забезпечення побудована на клієнт-серверній моделі, де клієнтська частина відповідає за взаємодію з користувачем, а серверна забезпечує доступ до необхідних даних через API.

1.6.1.1 Клієнтська частина (Frontend)

Для створення інтерфейсу користувача буде застосовано бібліотеку Next.js, яка дозволяє ефективно організувати взаємодії з користувачем та обробку даних через стейт і компоненти.

Оскільки візуалізація даних є інтерактивною, Next.js надасть можливість швидко оновлювати стан інтерфейсу відповідно до змін у даних.

Основні елементи інтерфейсу, такі як кнопки, графіки, таблиці та фільтри, будуть стилізовані за допомогою Tailwind CSS. Цей CSS-фреймворк дозволить створити адаптивний і зручний для користувача інтерфейс, при цьому мінімізуючи обсяг написаного CSS коду.

Для отримання даних буде використовуватися бібліотека Axios або вбудоване API fetch для здійснення запитів на сервер та отримання необхідних даних у форматі JSON або іншому зручному форматі для візуалізації.

Для візуалізації API-даних використовуватимуться бібліотеки, такі як Chart.js для графіків або D3.js для більш складних і налаштованих візуалізацій, які потребують динамічних змін у процесі взаємодії з користувачем.

1.6.1.2 Серверна частина (Backend)

Серверна частина займається обробкою запитів від клієнтської частини, виконанням бізнес-логіки та наданням даних через API. Існує два можливі варіанти реалізації цієї частини.

Якщо обирається реалізація серверної частини за допомогою Node.js, використовуватиметься популярний серверний фреймворк Express.js, який дозволяє швидко розробляти RESTful API для взаємодії з клієнтом. Node.js є чудовим вибором, оскільки працює на JavaScript, що забезпечує зручність розробки як на сервері, так і на клієнті. Для зберігання даних можна використовувати реляційні або NoSQL бази даних, залежно від потреб проекту.

Альтернативним варіантом є використання MockAPI для швидкої розробки та тестування API без необхідності налаштування серверної частини. MockAPI надає можливість створювати фейкові API з можливістю визначення схеми даних і зручним доступом до них через HTTP запити. Це дає змогу швидко тестувати інтерфейс і взаємодію з даними без необхідності створення складної серверної логіки на початкових етапах розробки.

1.6.2 Основні принципи проектування

Для успішної розробки веб-інтерфейсу важливо дотримуватися кількох основних принципів проектування:

- Модульність: Кожен компонент системи має бути автономним і відповідати за окрему частину функціональності. Це дозволяє спростити тестування, оновлення та масштабування.
- Інтерфейс для користувача: Основна увага приділяється зручності інтерфейсу. Він повинен бути інтуїтивно зрозумілим, швидким у використанні та адаптивним до різних пристроїв.
- Продуктивність: Система має бути здатною обробляти великі обсяги даних у реальному часі, забезпечуючи швидке завантаження та оновлення візуалізацій.

1.7 Порівняння існуючих систем з інтерактивною візуалізацією даних

Інтерактивні платформи для роботи з даними мають свої переваги, але також стикаються з певними недоліками. Розглянемо два приклади:

1.7.1 Observable

Переваги:

- Потужне середовище для роботи з JavaScript і інтерактивною візуалізацією.
- Реальний час співпраці та можливість інтеграції зовнішніх API.
- Гнучкість у налаштуванні та розробці власних візуалізацій.

Недоліки:

- Складність для новачків: Для роботи необхідне базове знання JavaScript і методологій інтерактивної візуалізації.

- Відсутність оптимізації для широкого доступу: Інтерфейс платформи орієнтований більше на розробників і технічних спеціалістів.
- Обмеження в готових рішеннях: Багато компонентів необхідно створювати з нуля, що збільшує час на впровадження.

1.7.2 Our World in Data

Переваги:

- Простота у використанні завдяки інтуїтивному інтерфейсу.
- Великий обсяг даних із можливістю аналізу за багатьма параметрами.
- Підтримка інтерактивних графіків і карт із простими фільтрами.

Недоліки:

- Фіксованість функціоналу: Немає можливості гнучкого налаштування графіків для конкретних потреб.
- Обмеження у взаємодії: Інтерактивність в основному обмежена масштабуванням і базовими фільтрами.
- Не підходить для розробників: Відсутні інструменти для інтеграції API чи створення кастомізованих графіків.

Як видно на картинці, цей графік майже не має інтерактивних елементів (див. рис. 1.1), зображення взято із сайту – https://ourworldindata.org/coronavirus. Користувач може увімкнути функцію «Play time-lapse», щоб спостерігати зміну графіку за роками в реальному часі, а також додавати або видаляти країни зі списку відображення та переглядати дані у вигляді таблиці, але не має можливості взаємодіяти з ними.

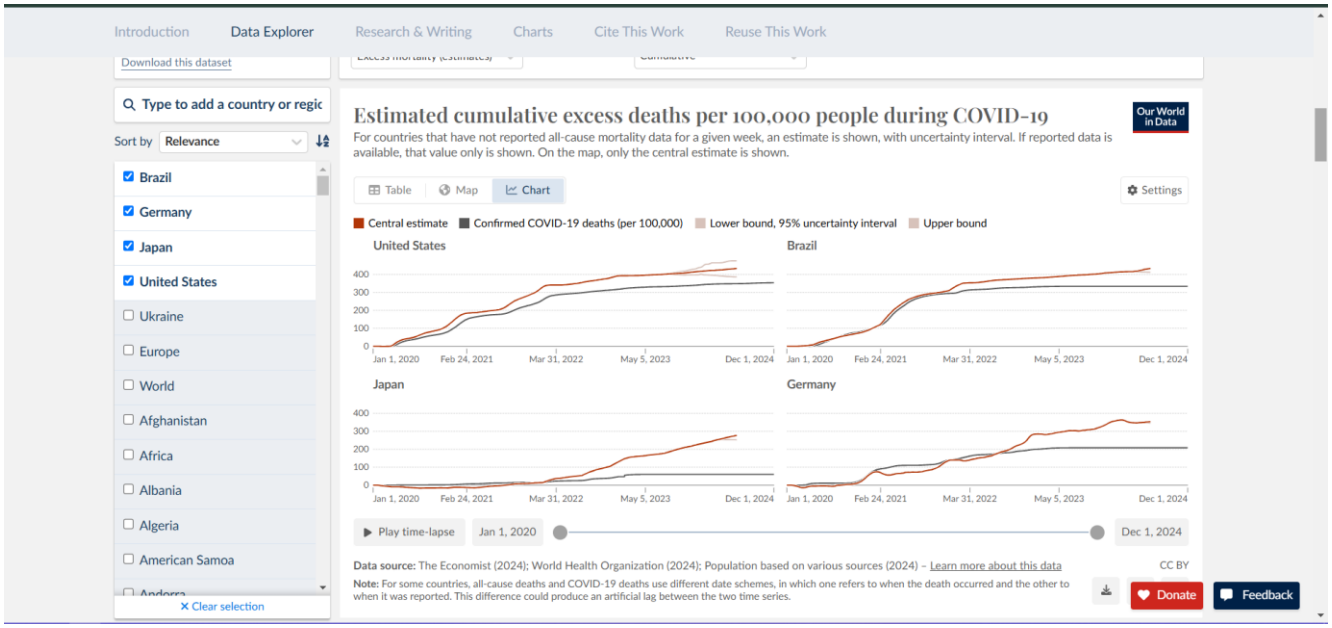

Рисунок 1.1 – Вигляд Our World in Data

Незважаючи на їхні переваги, кожна з платформ має свої недоліки, що обмежують її застосування в конкретних випадках. Observable підходить для розробників, але має високий поріг входу. Our World in Data простий і доступний, однак не підтримує глибокої кастомізації. Аналіз цих систем дозволяє врахувати найкращі практики й уникнути поширених недоліків у створенні власної платформи, пропонуючи баланс між функціональністю та зручністю.

Перший розділ охоплює основні аспекти аналізу предметної області для розробки веб-інтерфейсу для інтерактивної візуалізації API-даних. Було досліджено історію розвитку JavaScript та веб-інтерфейсів, а також їх еволюцію в контексті сучасних вимог до інтерактивності та продуктивності. Додатково, розглянуті основні поняття, що стосуються API-даних і їх використання в веб-додатках для візуалізації, підкреслюючи важливість ефективної обробки і презентації великих обсягів інформації.

Важливою частиною аналізу стала постановка задачі розробки, де було визначено необхідні функціональні характеристики, такі як підтримка інтерактивних графіків, обробка даних через API, а також адаптивність інтерфейсу для різних пристроїв. Описані критерії вибору технології розробки, включаючи використання JavaScript, бібліотек для побудови візуалізацій, таких як D3.js і

Chart.js, а також серверних технологій для ефективної обробки запитів та роботи з великими обсягами даних.

Процес проектування програмного забезпечення також було детально розглянуто, з акцентом на архітектурні рішення, компоненти системи, принципи модульності та безпеки.

Розглянувши уже існуючі рішення було виділено переваги та недоліки щоб покращити власну розробку.

Загалом, розділ надає комплексне розуміння ключових аспектів і етапів розробки веб-інтерфейсу для інтерактивної візуалізації API-даних. Детальне вивчення історії розвитку технологій, а також вибір оптимальних методів і інструментів для створення інтерактивних та зручних інтерфейсів, дозволяє закласти міцну основу для подальшої реалізації проекту.

2 РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ

2.1 Node.js: Основні концепції та архітектура

Node.js — це сучасне середовище виконання JavaScript, призначене для створення серверних додатків. Воно дозволяє запускати JavaScript поза межами браузера, забезпечуючи оптимальну продуктивність, ефективність і масштабованість. Node.js, представлений Раяном Далем у 2009 році, швидко завоював популярність серед розробників завдяки своїй легкості у використанні, передовій архітектурі та активній підтримці спільноти.

Однією з основних характеристик Node.js є його однопотокова модель виконання з асинхронною обробкою задач. Це передбачає, що замість створення нових потоків для кожного запиту, Node.js обробляє їх асинхронно, дозволяючи одночасно виконувати велику кількість операцій. Наприклад, у традиційних серверних середовищах, таких як PHP або Java, кожен новий запит часто створює окремий потік, що може призводити до перевантаження сервера. У Node.js використовується подієвий цикл, який забезпечує виконання завдань у порядку їх надходження, дозволяючи системі працювати без блокувань навіть за великого навантаження.

Node.js побудований на основі рушія V8 від Google, який забезпечує швидке виконання JavaScript. Цей рушій компілює код безпосередньо в машинні інструкції, що значно прискорює його виконання. Крім цього, Node.js використовує бібліотеку libuv, яка відповідає за подієвий цикл і асинхронні операції вводу/виводу. Завдяки цьому розробники можуть створювати сервери, здатні обробляти тисячі запитів за секунду, використовуючи при цьому мінімум апаратних ресурсів.

Іншою важливою характеристикою Node.js є його модульна структура. Код у цьому середовищі легко розбити на окремі частини, що спрощує підтримку та повторне використання. Node.js має велику кількість вбудованих модулів, таких як fs для роботи з файлами чи http для створення веб-серверів. Крім того, завдяки npm

(Node Package Manager) доступний величезний репозиторій сторонніх бібліотек, які розширюють можливості Node.js та спрощують розробку.

Ще однією важливою перевагою Node.js є його інтеграція з JSON — форматом, що широко застосовується для передачі даних між клієнтом і сервером. Завдяки цьому Node.js стає чудовим вибором для створення RESTful API, які забезпечують інтерактивну взаємодію між фронтендом і бекендом.

Node.js є ідеальним вибором для створення реальних додатків, таких як чати чи платформи для потокового відео, завдяки своїй здатності ефективно обробляти численні одночасні підключення. Його архітектура також оптимально підходить для розробки мікросервісів, де кожен компонент відповідає за певну частину бізнес-логіки [14].

Таким чином, Node.js є потужним інструментом для сучасної серверної розробки. Його асинхронна модель, подієвий цикл, висока швидкість виконання та модульна структура роблять його одним із найкращих рішень для створення продуктивних і масштабованих веб-додатків. Це середовище надає розробникам усі необхідні інструменти для побудови сучасних серверних систем, які легко адаптуються до змінних потреб бізнесу.

2.2 MongoDB та її застосування

MongoDB — це одна з найпопулярніших NoSQL баз даних, яка дозволяє зберігати та обробляти великі обсяги структурованих та неструктурованих даних. Вона була розроблена компанією 10gen (тепер MongoDB Inc.) і випущена у 2009 році. MongoDB зазвичай використовують у проектах, де необхідно зберігати гнучкі, динамічно змінювані дані, зберігаючи при цьому високу продуктивність і масштабованість. Відмінною рисою MongoDB є її орієнтація на документи, що робить її потужною альтернативою реляційним базам даних для ряду сучасних веб-застосунків.

2.2.1 Переваги MongoDB

MongoDB є базою даних, яка зберігає дані у форматі документів. Це означає, що замість традиційних таблиць з рядками та стовпцями, як у реляційних базах даних, MongoDB використовує документи, схожі на JSON-об'єкти, які містять поля і значення. Кожен документ є незалежним і може мати свою структуру, що дозволяє зберігати дані, які не обов'язково повинні відповідати однаковій схемі.

Один документ у MongoDB може мати будь-яку кількість полів, включаючи вкладені об'єкти або масиви, що дає розробникам велику гнучкість. Це дозволяє зберігати не тільки структуровані дані, але й невизначені, наприклад, дані з різними типами значень чи з різними атрибутами. Замість використання SQL-запитів MongoDB використовує власну мову запитів, яка заснована на JSON-подібних об'єктах.

Однією з важливих особливостей MongoDB є її здатність до горизонтального масштабування, що дозволяє зберігати і обробляти величезні обсяги даних, розподіляючи їх по кількох серверах. Це особливо важливо для великих проектів з високим навантаженням, де потрібна висока доступність і швидкість обробки даних. Вона не вимагає від користувача чітко визначеної схеми бази даних перед початком роботи. Це дозволяє без проблем змінювати структуру даних або додавати нові поля, не порушуючи існуючий код та дані. Крім того, розробники можуть працювати з різними типами даних, такими як JSON, масиви, документи чи навіть двійкові дані.

MongoDB дозволяє налаштовувати реплікацію, що означає, що дані автоматично копіюються на кілька серверів. Це гарантує, що в разі відмови одного з серверів, інші сервери зможуть продовжити обробку запитів без втрати даних і часу.

MongoDB використовує індексацію для прискорення пошуку по великих обсягах даних. Також, оскільки MongoDB зберігає дані у форматі BSON (бінарний JSON), це дозволяє швидше здійснювати серіалізацію та десеріалізацію даних.

База даних має просту та інтуїтивно зрозумілу модель, що дає змогу розробникам швидко розпочати роботу без необхідності освоювати складну мову SQL чи заплутані структури реляційних баз даних.

2.2.2 Застосування MongoDB

MongoDB знайшла широке застосування в багатьох галузях, зокрема в таких типах проектів:

- Веб-додатки в реальному часі, завдяки своїй здатності обробляти великі обсяги запитів та швидко масштабуватись, MongoDB часто використовують для створення чат-систем, онлайн-ігор, фінансових платформ, де потрібно обробляти і зберігати велику кількість даних в реальному часі. Наприклад, MongoDB використовується у платформах, таких як Uber, для збереження даних про поїздки та користувачів у реальному часі.
- Інтернет-магазини та системи електронної комерції MongoDB дозволяє зберігати інформацію про продукти, замовлення та відгуки клієнтів у вигляді документів, що дуже зручно для створення кастомізованих елементів або рекомендацій. Наприклад, Etsy використовує MongoDB для зберігання даних про продукти і продавців.
- MongoDB ефективно використовується для зберігання даних у мобільних додатках, де критично важливий швидкий доступ до інформації. Завдяки підтримці гнучкої структури даних, MongoDB зручна для зберігання різноманітних даних користувачів та забезпечення функціонування в офлайн-режимі.
- Інтернет речей (IoT) Завдяки здатності працювати з великими обсягами даних, MongoDB часто використовують у системах Інтернету речей для зберігання та аналізу даних що надходять з датчиків і пристроїв.

- Аналітичні системи MongoDB часто використовується для зберігання аналітичних даних, оскільки вона дозволяє зберігати великий обсяг даних та виконувати запити на їх обробку без значних затримок. Наприклад, у таких галузях, як фінанси, охорона здоров'я та телекомунікації, MongoDB дозволяє зберігати і аналізувати історичні дані для прогнозування та прийняття рішень [15].

2.3 Розробка бекенду (створення схем, роутів та взаємозв'язків)

У даному підрозділі буде розглянуто, як реалізуються основні функціональні можливості бекенду для проекту, включаючи реєстрацію користувачів, авторизацію, завантаження файлів, видалення файлів та отримання списку файлів.

2.3.1 Реєстрація та авторизація користувачів

Для реалізації реєстрації та авторизації серверна частина застосовує базу даних для збереження відомостей про користувачів, зокрема їхньої електронної пошти, паролів та додаткових даних, необхідних для автентифікації й забезпечення безпеки (див. рис. 2.1).

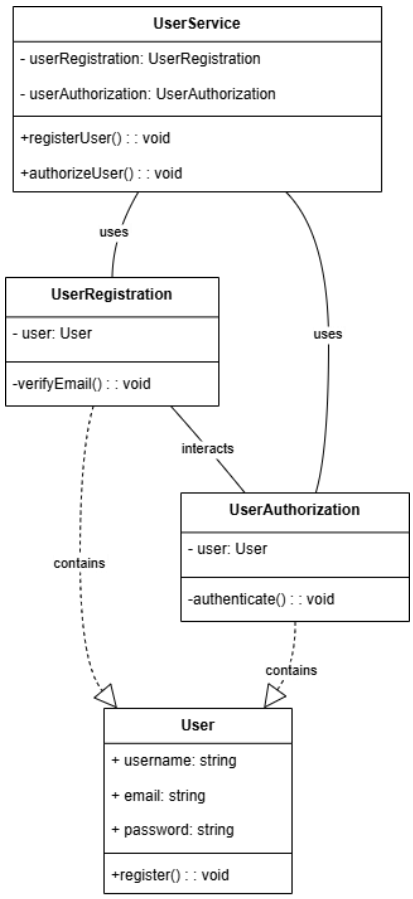

Рисунок 2.1 – Діаграма класів User

Клієнт (Frontend) → API (Backend): Користувач надає свої дані (електронну пошту, пароль) через інтерфейс реєстрації або авторизації.

API (Backend) → База даних (MongoDB): Сервер перевіряє наявність користувача в базі даних і, якщо користувач новий, створює новий запис, зберігаючи зашифрований пароль. У випадку авторизації система перевіряє відповідність введеного пароля за допомогою хешування.

API (Backend) → JWT (JSON Web Token): Після успішної авторизації генерується токен доступу (JWT), який повертається клієнту для подальших запитів до захищених ресурсів.

В результаті цього процесу клієнт отримує можливість взаємодіяти з бекендом через авторизаційний токен, що дозволяє обмежити доступ до певних API для неавторизованих користувачів.

2.3.2 Завантаження файлу на сервер

Для завантаження файлів на сервер використовується механізм прийому файлів через HTTP-запити, зазвичай через метод POST. Після завантаження файли зберігаються на сервері в певній директорії або в базі даних, а також зберігаються метадані файлів, як-от ім’я, розмір та тип (див. рис. 2.2).

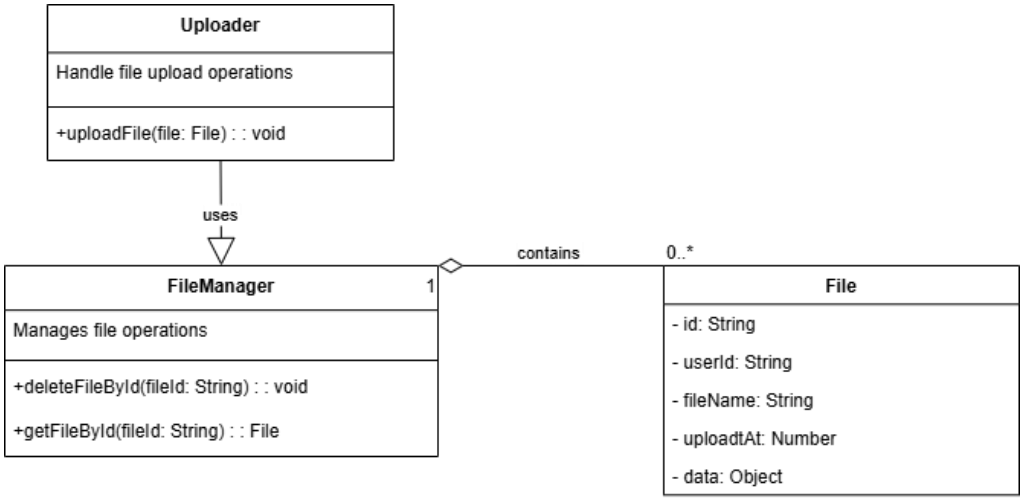

Рисунок 2.2 – Діаграма класів File

Клієнт (Frontend) → API (Backend): Користувач вибирає файл для завантаження і надсилає його через інтерфейс на сервер.

API (Backend) → База даних (MongoDB): Файл зберігається на сервері в директорії, а метадані файлу (ім’я, розмір, тип) зберігаються в базі даних.

API (Backend) → Клієнт (Frontend): Після успішного завантаження сервер повертає клієнту підтвердження з інформацією про файл, включаючи ID файлу для подальших операцій.

2.3.3 Видалення файлу за ID

Один з основних процесів у роботі з файлами — це їх видалення. Користувач може видалити файл, надавши його ID. Для цього сервер перевіряє, чи є файл з таким ID в базі даних, і, якщо є, видаляє файл із файлової системи та оновлює запис у базі даних.

Клієнт (Frontend) → API (Backend): Користувач ініціює запит на видалення файлу, надаючи його унікальний ID.

API (Backend) → База даних (MongoDB): Сервер шукає файл за ID у базі даних.

API (Backend) → Файлова система: Якщо файл існує, він видаляється з файлової системи.

API (Backend) → База даних (MongoDB): Запис про файл видаляється з бази даних.

API (Backend) → Клієнт (Frontend): Повертається відповідь, що файл успішно видалено.

2.3.4 Отримання списку усіх файлів на сервері

Для отримання списку усіх файлів, що зберігаються на сервері, клієнт може зробити запит до API. Бекенд відправляє список метаданих усіх файлів, включаючи їхні ID, імена, розміри та типи. Це може бути корисно для перегляду файлів або для подальших операцій, таких як завантаження чи видалення.

Клієнт (Frontend) → API (Backend): Клієнт надсилає запит на отримання списку всіх файлів.

API (Backend) → База даних (MongoDB): Сервер здійснює запит до бази даних для отримання метаданих усіх файлів, збережених на сервері.

API (Backend) → Клієнт (Frontend): Відповідь повертається клієнту у вигляді списку метаданих файлів, який можна відобразити на інтерфейсі користувача.

Загальна схема взаємозв'язку між компонентами виглядає наступним чином:

- Клієнт (Frontend): Взаємодіє через HTTP-запити з API для виконання операцій з файлами, реєстрації та авторизації користувачів.
- API (Backend): Приймає запити від клієнта, виконує необхідні операції, взаємодіє з базою даних (MongoDB) та файловою системою для збереження, видалення та отримання файлів.
- База даних (MongoDB): Зберігає метадані файлів, інформацію про користувачів, а також інші необхідні дані для функціонування системи.
- Файлова система: Зберігає файли на сервері, забезпечуючи швидкий доступ до них при необхідності.

Ці операції дозволяють створити ефективну систему для управління файлами, що підходить для сучасних веб-додатків, де важливо мати можливість завантажувати, зберігати та видаляти файли через зручний інтерфейс.

2.4 Опис роботи з роутами та інтеграцією функціоналу для файлів

У цьому підпункті описуються основні маршрути для роботи з користувачами та файлами, що дозволяють забезпечити основні функціональності для реєстрації, авторизації, завантаження, отримання та видалення файлів. Код нижче демонструє, як було реалізовано ці маршрути та взаємодія з базою даних і файловою системою.

Реєстрація користувачів включає перевірку наявності існуючого користувача за email (див. рис. 2.3). У разі відсутності користувача, хешується пароль, створюється новий запис у базі даних, і користувач отримує повідомлення про успішну реєстрацію.

```
// Реєстрація
router.post("/register", async (req, res) => {
  try {
    const { email, password, name } = req.body;

    // Перевірка, чи користувач вже існує
    const existingUser = await User.findOne({ email });
    if (existingUser) {
      return res.status(400).json({ message: "Користувач вже існує" });
    }

    // Хешування паролю      You, 3 weeks ago • registration
    const hashedPassword = await bcrypt.hash(password, 10);

    // Створення нового користувача
    const newUser = new User({ email, password: hashedPassword, name });
    await newUser.save();

    res.status(201).json({ message: "Реєстрація успішна" });
  } catch (error) {
    res.status(500).json({ message: "Помилка сервера", error });
  }
});
```

Рисунок 2.3 – Код реєстрації користувача

Авторизація здійснюється шляхом перевірки користувача за email та валідації введеного паролю (див. рис. 2.4). Після успішної перевірки генерується JWT токен, який клієнт використовує для наступних запитів.

```
32  // Авторизація
33  router.post("/login", async (req, res) => {
34    try {
35      const { email, password } = req.body;
36
37      // Перевірка користувача
38      const user = await User.findOne({ email });
39      if (!user) {
40        return res.status(400).json({ message: "Неправильний email або пароль" });
41      }
42
43      // Перевірка паролю
44      const isPasswordValid = await bcrypt.compare(password, user.password);
45      if (!isPasswordValid) {
46        return res.status(400).json({ message: "Неправильний email або пароль" });
47      }
48
49      // Генерація JWT з ім'ям користувача в payload
50      const token = jwt.sign(
51        { id: user._id, name: user.name }, // Додаємо ім'я користувача в payload
52        process.env.JWT_SECRET,
53        { expiresIn: "1d" } // Токен діє 1 день
54      );
55
56      // Форматування відповіді з користувачем
57      const userResponse = {
58        id: user._id,
59        name: user.name,
60        email: user.email,
61      };
62
63      res.status(200).json({
64        message: "Авторизація успішна",
65        token,
66        user: userResponse,
67      });
68    } catch (error) {
69      res.status(500).json({ message: "Помилка сервера", error });
70    }
71  });
```

Рисунок 2.2 – Код авторизації користувача

Маршрут для завантаження файлів на сервер використовує бібліотеку multer, що дозволяє зберігати файли у тимчасовій директорії (див. рис. 2.5). Після завантаження файл перетворюється з формату CSV у JSON, а потім зберігається в базі даних разом з метаданими. Після цього тимчасовий файл видаляється з файлової системи.

```
24  // POST маршрут для завантаження файлу
25  router.post("/upload-file", upload.single("file"), async (req, res) => {
26    try {
27      if (!req.file) {
28        return res.status(400).json({ message: "Файл не завантажено" });
29      }
30      const filePath = req.file.path;
31      // Читання CSV і перетворення в JSON
32      const jsonData = await csvtojson().fromFile(filePath);
33      // Вибір тільки необхідних полів
34      const filteredData = filterCovidData(jsonData);
35      const covidDataEntry = new CovidData({
36        user: req.user.id,
37        fileName: req.file.originalname,
38        uploadedAt: new Date(),
39        data: filteredData,
40      });
41      // Зберігаємо у базі даних
42      const savedData = await covidDataEntry.save();
43      // Видаляємо тимчасовий файл
44      fs.unlinkSync(filePath);
45      res.status(201).json({ message: "Дані завантажено успішно", savedData });
46    } catch (error) {
47      res.status(500).json({ message: "Помилка при завантаженні файлу", error });
48    }
49  });
```

Рисунок 2.5 – Код завантаження файлу

Для отримання файлів, які належать авторизованому користувачу, використовується GET маршрут, що звертається до бази даних і повертає всі файли, що належать користувачу (див. рис. 2.6).

```
51  // GET маршрут для отримання даних для авторизованого користувача
52  router.get("/covid-data", async (req, res) => {
53    try {
54      const covidData = await CovidData.find({ user: req.user.id });
55      if (!covidData) {
56        return res.status(404).json({ message: "Дані не знайдено" });
57      }
58      res.status(200).json(covidData);
59    } catch (error) {
60      res.status(500).json({ message: "Помилка при отриманні даних", error });
61    }
62  });
```

Рисунок 2.6 – Код отримання даних

Для видалення файлу за його унікальним ID реалізовано маршрут, який перевіряє, чи є файл у базі даних. Якщо файл існує, він видаляється з файлової системи, а також видаляється запис з бази даних (див. рис. 2.7).

```
64  // DELETE маршрут для видалення файлу за ID
65  router.delete("/delete-file/:id", async (req, res) => {
66    try {
67      const { id } = req.params;
68      const dataToDelete = await CovidData.findOne({
69        _id: id,
70        user: req.user.id,
71      });
72      if (!dataToDelete) {
73        return res
74          .status(404)
75          .json({ message: "Дані не знайдено або у вас немає доступу до них" });
76      }
77      const filePath = `uploads/${dataToDelete.fileName}`;
78      if (fs.existsSync(filePath)) {
79        fs.unlinkSync(filePath);
80      }
81      await CovidData.deleteOne({ _id: id });
82      res.status(200).json({ message: "Дані та файл видалено успішно" });
83    } catch (error) {
84      res.status(500).json({ message: "Помилка при видаленні даних", error });
85    }
86  });
```

Рисунок 2.7 – Код видалення файлу

Завдання та отримання файлів: Кожен завантажений файл зберігається на сервері, а його метадані зберігаються в базі даних MongoDB. Це дозволяє зберігати інформацію про файли навіть після їх видалення з файлової системи.

Всі операції з файлами потребують аутентифікації користувача. Тому кожен запит до API перевіряється на наявність валідного токену JWT, що гарантує захист даних користувачів.

Вся обробка даних, яка відбувається при завантаженні файлів (наприклад, перетворення CSV в JSON), здійснюється на сервері, що дозволяє користувачеві отримувати лише необхідні та очищені дані.

Ці маршрути і функціональності забезпечують основні операції для опрацювання файлів на сервері, що є важливою частиною роботи бекенду для інтеграції з фронтендом і базою даних.

3. РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ

У цьому розділі детально розглянуто процес розробки клієнтської частини веб-додатку, що забезпечує інтерактивну візуалізацію даних, отриманих з API.

Клієнтська частина є важливою складовою проекту, оскільки вона безпосередньо взаємодіє з користувачем, надаючи зручний інтерфейс для взаємодії з даними. Розробка цієї частини передбачає створення компонувальної, адаптивної та інтерактивної системи, яка забезпечує ефективну роботу додатку на різних пристроях і дозволяє користувачам зручно та інтуїтивно здійснювати візуалізацію та аналіз даних.

В рамках цього розділу будуть розглянуті основні аспекти розробки, такі як архітектура клієнтської частини, реалізація окремих компонентів, а також обмеження та потенціал для подальшого розвитку додатку.

3.1. Архітектура клієнтської частини

Архітектура клієнтської частини базується на модульному підході, що дозволяє розділити логіку додатку на окремі компоненти. Це забезпечує високу гнучкість системи, можливість повторного використання компонентів та спрощує процес їх тестування й розширення. Для побудови клієнтської частини було використано Next.js, що дозволило створити динамічний та інтуїтивно зрозумілий інтерфейс для користувача [16].

3.1.1 Основні компоненти архітектури

Компоненти верхнього рівня включають основну структуру додатку, такі як бічна панель, футер, та контейнер для рендерингу контенту. Ці компоненти забезпечують базовий каркас інтерфейсу та виконують основні функції.

Контейнери відповідають за отримання даних із серверної частини через REST API або WebSocket. Вони передають отримані дані до дочірніх компонентів у вигляді пропсів, забезпечуючи чіткий поділ логіки й візуального представлення.

Презентаційні компоненти це UI-елементи, які відповідають виключно за відображення даних. Вони отримують пропси від контейнерів і не містять власної логіки, що робить їх легко тестованими. Презентаційні компоненти включають таблиці даних, графіки, форми, елементи фільтрації тощо. Наприклад, для візуалізації даних використовувалася бібліотека ECharts, яка інтегрується з React-компонентами.

3.1.2 Комунікація з сервером

Для взаємодії клієнтської частини з сервером використовується axios, що дозволяє виконувати HTTP-запити для отримання й відправлення даних. Запити організовані в окремих сервісних модулях, таких як signIn.js та signUp.js для автентифікації або dashboard.js для роботи з файлами. Це сприяє повторному використанню логіки запитів у багатьох частинах додатку.

Комунікація з сервером побудована з дотриманням принципів REST:

- Запити на отримання списків файлів виконуються через GET /files.
- Завантаження файлів – POST /upload.
- Видалення конкретного файлу – DELETE /files/:id.

Для керування станом додатку був використаний Redux toolkit, що забезпечує ефективне управління глобальним станом, таким як автентифікація користувача, фільтрація даних або збереження стану графіків.

3.2 Реалізація клієнтської частини

Розробка клієнтської частини здійснюється із застосуванням передових технологій, включаючи Next.js, Redux Toolkit, TailwindCSS, ECharts, React Hook Form та axios. У цьому розділі наведено структуру та ключові аспекти її реалізації.

Клієнтська частина побудована за принципами компонентного підходу, що забезпечує чітке розмежування відповідальностей між компонентами та модульність коду. Стан додатку централізовано управляється за допомогою Redux Toolkit, що спрощує передачу даних між компонентами, а також обробку асинхронних запитів.

3.2.1 Основні сторінки

Сторінка реєстрації містить форму з трьома полями: name, email, та password (див. рис. 3.1). Для валідації даних використовується бібліотека React Hook Form, що забезпечує простий і ефективний механізм обробки форм.

Рисунок 3.1 – Сторінка реєстрації

Після успішної реєстрації користувача перенаправляє на сторінку авторизації.

Сторінка авторизації містить два поля: email та password (див. рис. 3.2). Після успішного входу користувач переходить на головну сторінку, де його зустрічає персоналізована панель.

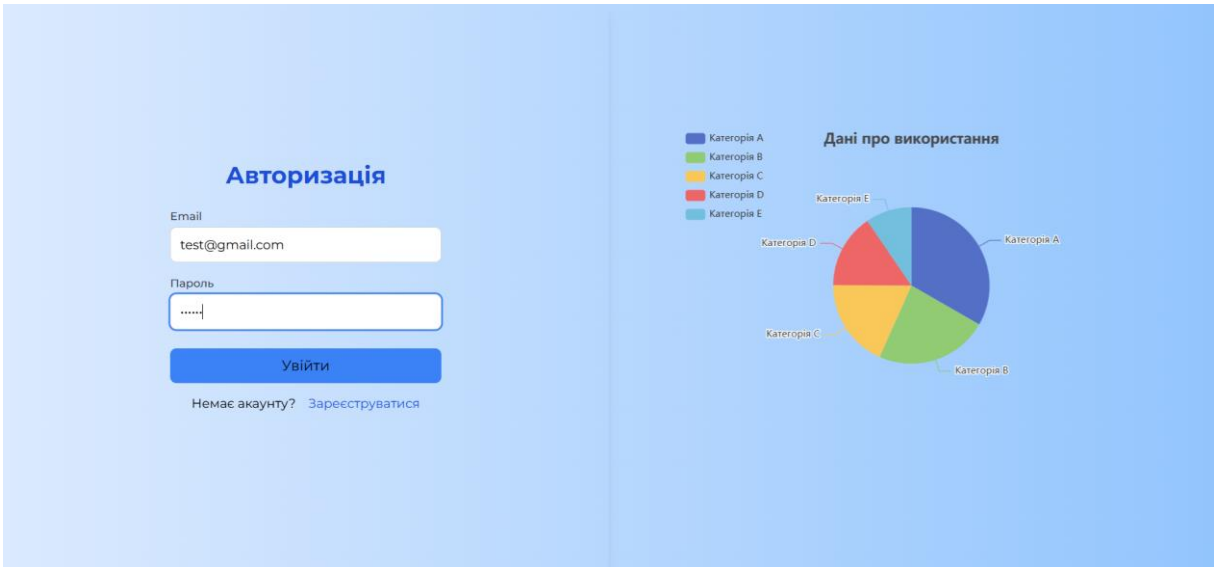

Рисунок 3.2 – Сторінка авторизації

Після авторизації користувач може завантажувати файли за допомогою кнопки у Sidebar. Завантаження здійснюється через axios, а інформація про файл зберігається у Redux.

Головна сторінка складається з двох ключових компонентів (див рис 3.3):

- Dashboard – використовується для відображення графіків на основі завантажених файлів. Коли користувач вибирає файл із бічної панелі, його ідентифікатор передається у Dashboard, і графік генерується за допомогою бібліотеки ECharts.

Також на панелі присутні фільтри, що дозволяють користувачеві налаштовувати відображення даних.

- Sidebar містить ім’я користувача, отримане після авторизації. Кнопку "Log Out", що дозволяє вийти з системи. Список завантажених файлів. При натисканні на файл його дані передаються у Dashboard. Кнопку для завантаження нового файлу. Завантажений файл додається до списку та відображається у Sidebar.

Дані для відображення було взято з сайту COVID Tracking Project Data explorer [17]

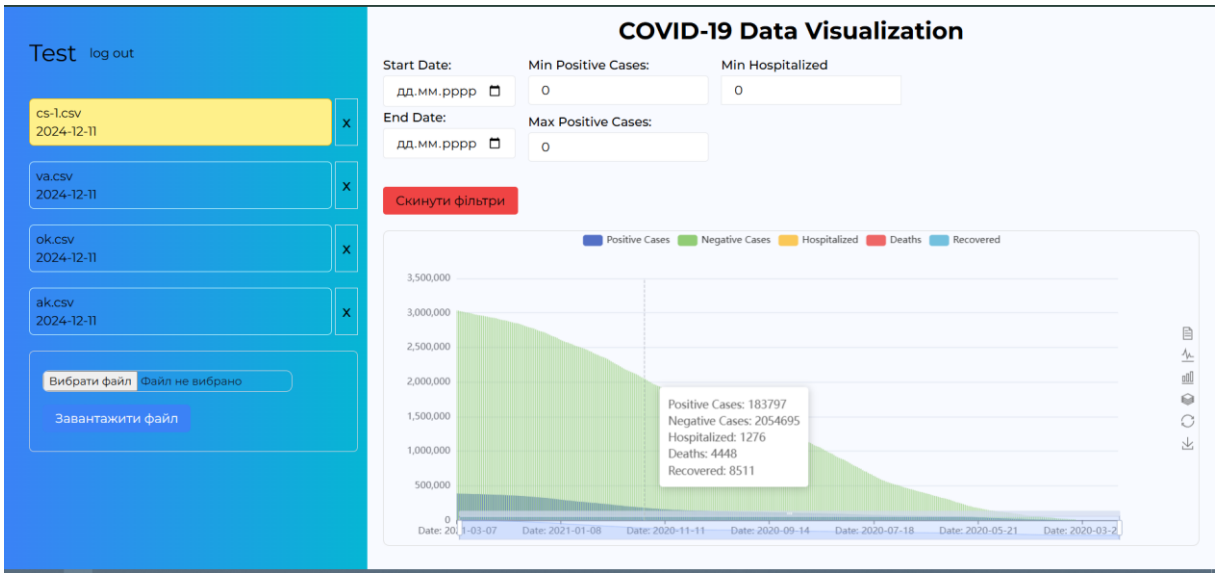

Рисунок 3.3 – Вигляд головної сторінки

Завантажені файли зберігаються у списку. Клік на файл викликає відповідний API-запит, дані з якого передаються у Dashboard для генерації графіку.

3.2.2 Функціонал головної сторінки

Користувач може вимикати або вмикати окремі категорії даних. Це дозволяє зосередитися на певних аспектах інформації, не перевантажуючи графік зайвими деталями. (див. рис. 3.4).

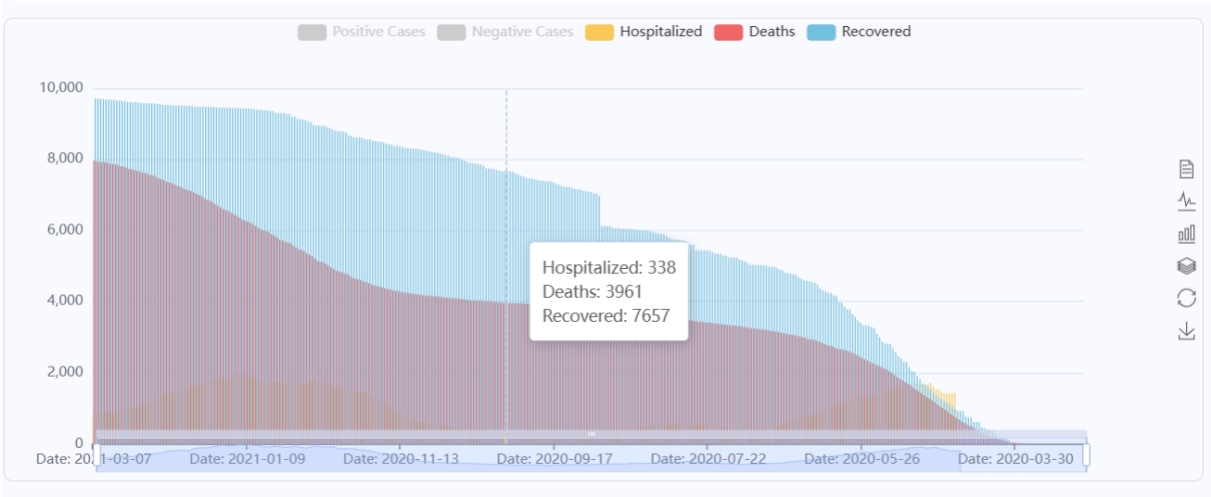

Рисунок 3.4 Вимкнення окремих категорій данних

Dashboard підтримує зміну типу графіку. Це забезпечує більшу гнучкість у візуалізації даних, адаптуючи відображення до потреб користувача.

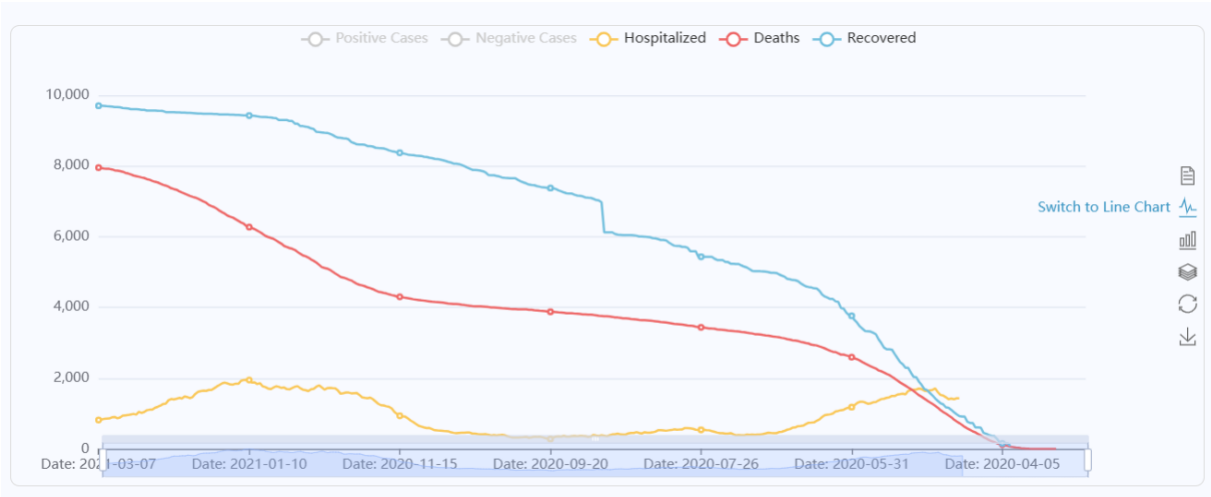

Рисунок 3.5 – Вигляд лінійного графіку

У Dashboard передбачена функція прямого перегляду та редагування бази даних. Це дає змогу користувачу вносити тимчасові зміни в дані без впливу на вихідний файл (див. рис. 3.6).

Усі зміни, зроблені користувачем, є зворотними: натискання кнопки "Restore" повертає дані до початкового стану.

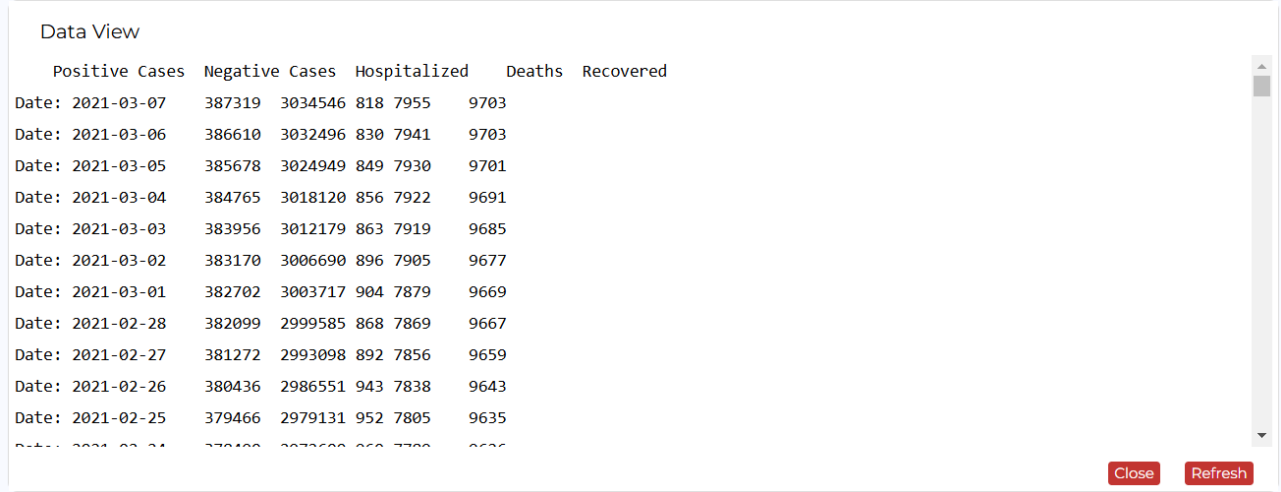

Рисунок 3.6 – Функція Data View

Для зручності користувач може завантажити поточний вигляд графіку у форматі зображення. Це корисно для подальшого використання, наприклад, у звітах або презентаціях.

Користувач може налаштовувати відображення даних за такими параметрами:

- Start Date і End Date: обмеження відображення даних за часовим періодом.
- Min Positive Cases і Max Positive Cases: фільтрація даних за кількістю підтверджених випадків.
- Min Hospitalized: фільтрація за мінімальною кількістю госпіталізованих пацієнтів.

Це дозволяє отримати деталізовану вибірку відповідно до визначених критеріїв, що спрощує аналіз великих обсягів даних (див. рис. 3.7).

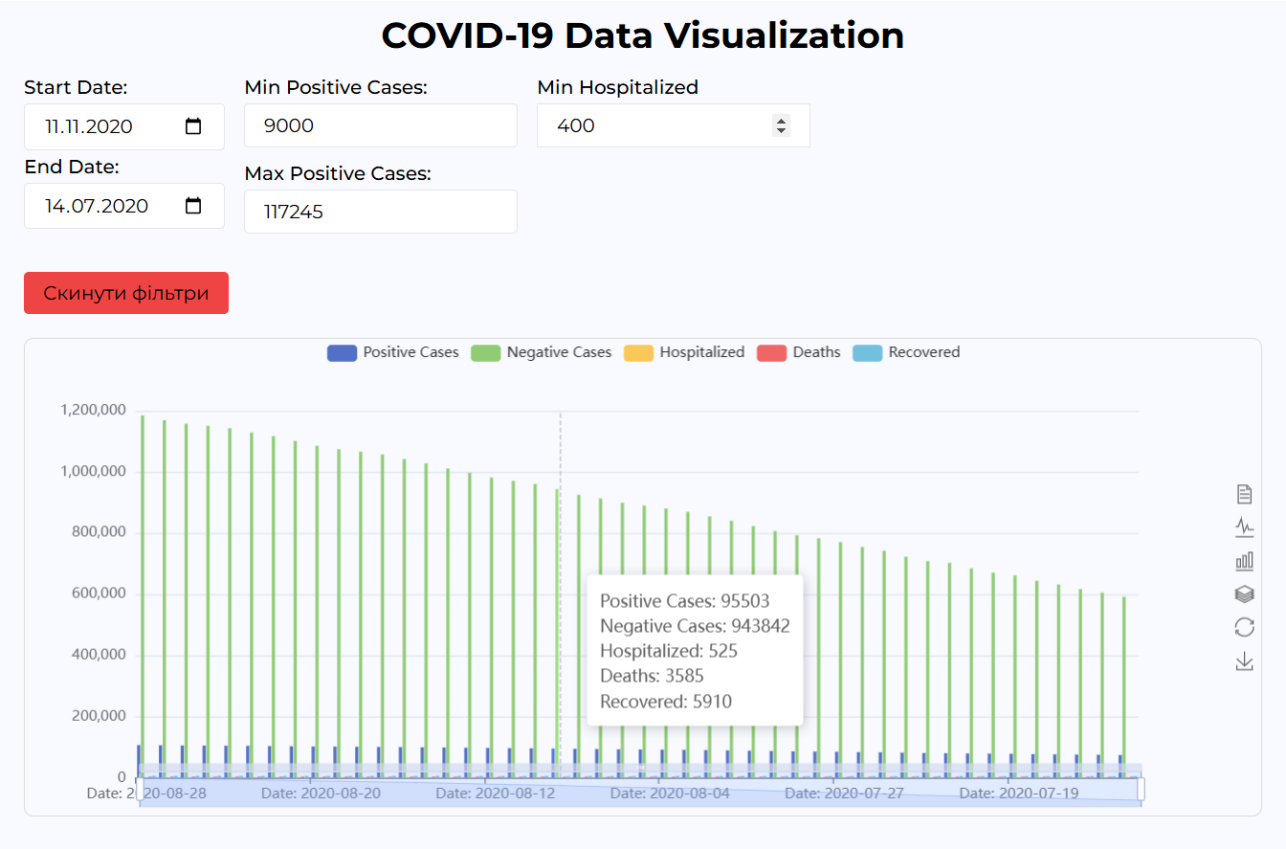

Рисунок 3.7 Відфільтровані дані

Також користувач може використовувати функцію масштабування графіка за допомогою скролу. Це особливо корисно для аналізу окремих ділянок графіка з великим обсягом даних або дрібних деталей.

Для централізованого управління станом використовується Redux Toolkit. Основні ред'юсери обробляють такі дії:

- Авторизація/вихід користувача.
- Додавання нового файлу до списку.
- Видалення файлу із списку

Для відображення помилок і повідомлень використовується бібліотека Notyflix.

3.3 Обмеження та потенціал для подальшого розвитку

У цьому підрозділі розглядаються поточні обмеження реалізованої клієнтської частини веб-інтерфейсу, а також визначаються напрями, які можуть бути розвинуті або вдосконалені в майбутньому.

3.3.1 Обмеження

Поточна реалізація може бути недостатньо ефективною для роботи з надвеликими наборами даних через обмеження клієнтської частини браузера. Це може вплинути на швидкодію під час рендерингу складних графіків або застосування фільтрів.

Використання Redux Toolkit і ECharts забезпечує базові потреби, але для підтримки більш складних сценаріїв може знадобитися інтеграція додаткових інструментів або бібліотек.

Поточна система потребує стабільного серверного API для обробки даних. У разі його нестабільності або затримок з боку сервера це може негативно вплинути на якість користувацького досвіду.

Наявні типи графіків і варіанти їх налаштування можуть бути недостатніми для деяких користувачів, які потребують більшої кастомізації або специфічних форматів візуалізації.

3.3.2 Потенціал для подальшого розвитку

Інтеграція веб-воркерів для розподілу навантаження між потоками може значно підвищити швидкодію під час роботи з великими обсягами даних.

Також можливе додавання нових типів графіків (наприклад, 3D-візуалізація або теплові карти).

Інтеграція більш гнучких фільтрів і кастомізації, таких як налаштування кольорів, легенд, або шкал.

Крім того, варто розглянути використання алгоритмів машинного навчання для аналізу завантажених даних, що дозволить автоматично виявляти ключові інсайти, наприклад, аномалії або прогнозувати майбутні тенденції. Можливе створення покрокових інструкцій або інтерактивного туторіалу для нових користувачів та реалізація експорту в різних форматах (CSV, JSON, PDF) для збереження оброблених даних і графіків.

Подальший розвиток проєкту може бути спрямований на усунення наявних обмежень, розширення функціональності та покращення зручності користувача. Це дозволить значно підвищити цінність інструменту для різних категорій користувачів.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

Магістерська кваліфікаційна робота зосереджена на дослідженні та розробці веб-інтерфейсу для відображення даних API за допомогою JavaScript. Оскільки розроблена система була реалізована з використанням комп’ютерної техніки, важливим аспектом стало дотримання вимог охорони праці та техніки безпеки.

Охорона праці в сфері інформаційних технологій передбачає суворе дотримання всіх вимог і норм, викладених у законодавчих актах з охорони праці. Ці нормативні документи спрямовані на забезпечення безпечної та ефективної експлуатації обладнання і приміщень, дотримання санітарно-гігієнічних умов праці та захист працівників від інших небезпечних факторів на підприємстві [18]. Ці положення є невід'ємною частиною вивчення методів і систем управління зрілістю вимог при виконанні процесів життєвого циклу. У ключових законодавчих актах з охорони праці значна увага приділяється покращенню умов праці у всіх галузях господарства, впровадженню сучасних засобів техніки безпеки та забезпеченню таких санітарно-гігієнічних умов, які мінімізують ризики виробничого травматизму і професійних захворювань.

Збереження життя і здоров’я людини є пріоритетним напрямом соціальної політики держави. В Україні діє закон прямої дії «Про охорону праці», що забезпечує захист конституційного права працівників на безпечні умови праці . Законодавча база України з охорони праці включає загальні закони та спеціальні нормативно-правові акти [18]. До загальних законів, що закріплюють основні положення щодо охорони праці, належать Конституція України, Закон України «Про охорону праці», Кодекс законів про працю (КЗпП) та Закон України «Про загальнообов'язкове державне соціальне страхування від нещасного випадку на виробництві та професійного захворювання, що спричинили втрату працездатності» [18].

Під час розробки веб-інтерфейсу для відображення даних API за допомогою JavaScript, площа та об'єм одного робочого місця оператора повинні відповідати нормам НПАОП 0.00-7.15-18. Зокрема, площа робочого місця має бути не менше 6,0 квадратних метрів, а об'єм — не менше 20,0 кубічних метрів [19].

Відповідно до вимог НПАОП 0.00-7.15-18, стіни, стеля та підлога приміщень, у яких розміщені комп’ютери, мають бути виготовлені з матеріалів, дозволених для оформлення приміщень органами державного санітарно-епідеміологічного нагляду.

У приміщеннях, де одночасно експлуатуються понад п'ять електронно-обчислювальних машин, на видному та доступному місці повинні бути встановлені аварійні резервні вимикачі, які забезпечують повне відключення електричного живлення приміщення, за винятком освітлення.

Заземлені конструкції, що знаходяться у приміщеннях з робочими місцями операторів (такі як батареї опалення, водопровідні труби, кабелі із заземленим відкритим екраном), повинні бути надійно ізольовані за допомогою діелектричних щитків та сіток, щоб запобігти потраплянню працівника під напругу. Організація робочого місця оператора повинна забезпечувати ергономічну відповідність усіх його елементів та їх розташування згідно з вимогами НПАОП 0.00-7.15-18.

Для захисту від прямих сонячних променів, які спричиняють відблиски на екранах персональних комп’ютерів і клавіатурах, необхідно передбачити сонцезахисні пристрої; вікна приміщень повинні бути обладнані жалюзі або шторами.

Умови праці мають відповідати вимогам нормативних актів з охорони праці, зокрема:

- забезпечення належних умов праці на кожному робочому місці;
- безпека технологічних процесів, машин, механізмів, обладнання та інших засобів виробництва;
- відповідний стан засобів колективного та індивідуального захисту;
- дотримання санітарно-побутових умов.

Приміщення з електронно-обчислювальними машинами повинні бути оснащені переносними вуглекислотними вогнегасниками з розрахунку 1 вогнегасник на кожні 50 м², але не менше 2 на приміщення. Підходи до засобів пожежогасіння повинні бути вільними [19].

Отже, під час розробки веб-інтерфейсу для відображення даних API за допомогою JavaScript було проаналізовано та враховано необхідні норми охорони праці при використанні електронно-обчислювальної техніки, а також забезпечено умови для комфортної та ефективної роботи працівників.

4.2 Безпека в надзвичайних ситуаціях

Надзвичайні ситуації, такі як природні катастрофи, техногенні аварії, кібератаки чи відмови в роботі систем, створюють значну загрозу для надійності комп’ютерних систем та безпеки їхніх користувачів. У процесі розробки веб-інтерфейсу для інтерактивної візуалізації API-даних на основі JavaScript важливо враховувати потенційні ризики, які можуть виникнути в таких умовах, а також передбачити способи їх мінімізації.

Дослідження стійкості роботи об’єкта — це всебічне вивчення обстановки, яка може скластися під час надзвичайної ситуації, та визначення її впливу на виробничу діяльність підприємства. Мета дослідження полягає у виявленні слабких місць в роботі об’єкта та розробленні найбільш ефективних пропозицій, спрямованих на підвищення його стійкості [20].

На першому, підготовчому, етапі визначають склад учасників, розробляють керівні документи та організовують підготовку дослідницьких груп. Другий етап присвячений оцінці стійкості: аналізуються вразливі елементи виробництва, досліджується стійкість будівель, обладнання, технологічних процесів, систем енергозабезпечення та логістики. Особливу увагу приділяють захисту працівників і готовності об’єкта до роботи в умовах НС.

На завершальному етапі узагальнюють результати, формулюють практичні пропозиції щодо підвищення стійкості та розробляють план заходів, які виконуються як завчасно, так і при загрозі чи виникненні НС. Результати перевіряються на спеціальних навчаннях.

Дослідження також охоплюють оцінку надійності комп'ютерних систем, зокрема часу відновлення після збоїв, стабільності роботи та частоти оновлення резервних даних. Усе це дозволяє підприємству краще підготуватися до потенційних загроз, зменшити ризики та мінімізувати наслідки можливих порушень [20].

Оцінка надійності комп'ютерної системи проводиться шляхом аналізу таких параметрів, як час відновлення після збою (RTO), який визначає тривалість простою, та середній час до виникнення відмови (MTTF), що дозволяє спрогнозувати стабільність роботи системи. Також особливу увагу слід приділити частоті оновлення резервних даних, яка впливає на можливість відновлення актуальної інформації [20].

Комп'ютерні системи можуть зазнавати негативного впливу внаслідок перебоїв електропостачання, що призводить до втрати даних і збоїв у роботі серверів. Ризики також пов'язані з можливими кібератаками, які спрямовані на компрометацію даних або блокування доступу до них через уразливості веб-додатків. Додаткові загрози виникають у разі фізичних пошкоджень обладнання внаслідок природних катастроф, таких як пожежі, повені чи землетруси, а також у разі перевантаження систем через різке зростання кількості запитів під час кризових ситуацій [21].

Для забезпечення надійності роботи системи в умовах надзвичайних ситуацій важливо впровадити низку заходів:

- Регулярне резервне копіювання даних на віддалених серверах або у хмарних сховищах дає змогу зберігати інформацію навіть у разі пошкодження основних баз даних.
- Реалізація плану аварійного відновлення (Disaster Recovery Plan), який включає кроки для швидкого відновлення роботи, є необхідною умовою. У межах

такого плану варто використовувати реплікацію серверів та автоматичне перенаправлення трафіку на резервні ресурси.

Важливим аспектом є захист від кібератак, що включає шифрування даних, багаторівневу аутентифікацію, а також моніторинг і виявлення аномальної активності у веб-додатку. Крім того, надійність системи підвищується завдяки дублюванню обладнання і серверів, що розміщуються у географічно віддалених центрах обробки даних. Для користувачів доцільно впровадити механізми інформування про надзвичайні ситуації, а також надати доступ до інструкцій із використання системи в кризових умовах [21].

Таким чином, забезпечення безпеки в умовах надзвичайних ситуацій є критично важливим завданням при розробці веб-інтерфейсів для роботи з API-даними. Впровадження механізмів резервного копіювання, захисту від атак та аварійного відновлення сприяє мінімізації ризиків і забезпечує стабільну роботу системи навіть за умов кризи.

ВИСНОВОК

В процесі дослідження був створений веб-інтерфейс для інтерактивної візуалізації даних, отриманих через API. Основний акцент був зроблений на інтеграцію сучасних технологій для створення продуктивних, гнучких і ефективних рішень для обробки великих обсягів даних у реальному часі. У рамках дослідження було виконано аналіз сучасних підходів до візуалізації даних, визначено найефективніші методи для інтерактивного представлення інформації та розроблено архітектуру веб-додатка для інтеграції API, що забезпечує оперативний доступ до інформації та її якісну обробку. Методика охоплює етапи проектування, програмування.

У результаті дослідження були досягнуті значущі результати, що стосуються не лише створення функціонального інтерфейсу для візуалізації даних, але й розробки та впровадження нових підходів до інтерактивної візуалізації в реальному часі. Вперше було реалізовано методи інтеграції персоналізованих функцій візуалізації, що дозволяють користувачам налаштовувати відображення даних в залежності від їхніх індивідуальних потреб. Це значно підвищує зручність роботи з даними, адже кожен користувач може адаптувати візуалізацію відповідно до своїх завдань і уподобань. Усі ці функціональні можливості були реалізовані завдяки використанню таких сучасних технологій, як Next.js, Node.js та ECharts, що забезпечує відмінну продуктивність і надійність роботи додатку.

Розроблені підходи до інтерактивної візуалізації даних мають велике значення для численних сфер, де необхідна обробка та аналіз великих обсягів даних. Це може стосуватися таких галузей, як бізнес-аналітика, наукові дослідження, фінансові послуги, соціальні мережі та інші. Візуалізація даних у реальному часі значно полегшує прийняття рішень, надаючи користувачам можливість оперативно отримувати актуальну інформацію та аналізувати її на ходу. Зокрема, використання персоналізованих функцій дозволяє максимально налаштувати інтерфейс під потреби кожного користувача, що в результаті сприяє

підвищенню ефективності роботи з системою та покращенню досвіду кінцевого користувача.

Розроблене рішення може бути ефективно впроваджене для створення інформаційних систем у різноманітних галузях, таких як бізнес, наукові дослідження, освіта та державне управління. Усі функціональні можливості інтерактивної візуалізації дозволяють обробляти та представляти дані у зручному вигляді, що є важливим для різних користувачів: аналітиків, дослідників, бізнесменів та інших. Крім того, запропоновані методи можуть слугувати основою для створення персоналізованих інформаційних систем, які адаптуються до конкретних потреб користувачів, забезпечуючи збір і аналіз необхідної інформації з різноманітних джерел.

Враховуючи використання сучасних відкритих технологій, таких як Next.js, Node.js та ECharts, вартість розробки та підтримки системи значно знижена. Ці технології не тільки знижують витрати на розробку, але й забезпечують високу продуктивність додатку, що важливо для роботи з великими обсягами даних. Крім того, високий рівень гнучкості та масштабованості дозволяє швидко адаптувати систему під нові вимоги і розширювати її функціональність без значних додаткових витрат. Така система є економічно вигідною як для організацій, що займаються аналізом даних, так і для кінцевих користувачів, яким вона забезпечує зручний та ефективний інструмент для роботи з великими даними.

Подальші дослідження в цій галузі можуть зосередитися на розширенні можливостей інтерактивної візуалізації, зокрема шляхом інтеграції більш складних методів аналізу даних, таких як машинне навчання чи штучний інтелект. Використання цих технологій може допомогти покращити точність прогнозування та візуалізації, що стане важливим кроком для створення передових інформаційних систем. Крім того, перспективним напрямком є розробка інтерфейсів, які б інтегрувались із новими джерелами даних, такими як IoT-пристрої чи системи моніторингу в реальному часі. Також є можливість вдосконалення функціоналу персоналізованих налаштувань, що дозволить створити ще більш гнучкі й адаптивні платформи для кінцевих користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Boyko, I., Petryk, M., Mudryk, I., Stoianov, Y., Tsupryk, H. Mathematical Model of the Capacitor Based on Zeolite Material Proceedings - International Conference on Advanced Computer Information Technologies, ACIT, 2021. – С. 45–48.

2. Бойко І. В., Петрик М. Р., Цуприк Г. Б. Інформаційні технології видобутку даних (Data mining, високопродуктивні обчислення у складних системах): навч. посіб. — Тернопіль: ТНТУ, 2020. — 62 с.

3. Zakas, N. C. Understanding ECMAScript 6: The Definitive Guide for JavaScript Developers. — No Starch Press, 2016.

4. Flanagan, D. JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language. 7th ed. — O'Reilly Media, 2020.

5. Дубовик, О. В. JavaScript та сучасні веб-технології. — Київ: Видавництво КНУ, 2019.

6. Morales, J. Practical API Design: Designing and Developing Scalable Web APIs. — Packt Publishing, 2022.

7. Кузьмін, О. В. Сучасні веб-технології JavaScript: Практичний посібник. — Львів: Видавництво Львівської політехніки, 2018.

8. Bostock, M. Interactive Data Visualization for the Web: An Introduction to Designing with D3. — O'Reilly Media, 2019.

9. Бриж, В. В. Інтерактивна візуалізація даних у веб-додатках: Методичний посібник. — Київ: Національний університет біоресурсів і природокористування, 2022.

10. Кондратюк, А. І., Шевчук, В. І. Основи роботи з API у веб-розробці. — Тернопіль: ТНТУ імені Івана Пулюя, 2020.

11. Murray, S. Data Visualization with Python and JavaScript: Scrape, Clean, Explore, and Transform Your Data. — O'Reilly Media, 2021.

12. Osmani, A. Learning JavaScript Design Patterns: A JavaScript and Node.js Developer's Guide. — O'Reilly Media, 2021.

13. Echart.js Documentation. Interactive Charting and Data Visualization Library. [Електронний ресурс]. URL: https://echarts.apache.org/.

14. Tilkov, S., Vinoski, S. Node.js: Using JavaScript to Build High-Performance Network Applications // IEEE Internet Computing, 2010.

15. MongoDB Documentation. Building Scalable Applications with MongoDB. [Електронний ресурс]. URL: https://www.mongodb.com/docs/.

16. Зюзін, В. І. Next.js і React: Практичні аспекти розробки веб-застосунків. — Дніпро: ДНУ імені Олеся Гончара, 2021.

17. COVID Tracking Project Data explorer URL: https://explore.covidtracking.com/

18. Желібо Є., Заверуха Н., Зацарний В. Безпека життєдіяльності. — Київ, 2001. — 483 с.

19. НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями». — Київ, 2018. URL: https://zakon.rada.gov.ua/laws/show/z0508-18#Text.

20. Стручок, В. С. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання «БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ» — Тернопіль: ФОП Паляниця В. А., 156 с. URL: https://elartu.tntu.edu.ua/handle/lib/39196.

21. Стручок, В. С. Навчальний посібник «ТЕХНОЕКОЛОГІЯ ТА ЦИВІЛЬНА БЕЗПЕКА. ЧАСТИНА «ЦИВІЛЬНА БЕЗПЕКА». — Тернопіль: ФОП Паляниця В. А., 156 с. URL: http://elartu.tntu.edu.ua/handle/lib/39424.

ДОДАТКИ

Додаток А

Програмний код компоненту SideBar

```
const SideBar: React.FC<SideBarProps> = ({
  register,
  errors,
  onSubmit,
  handleSubmit,
  activeFileId,
  handleDeleteFile,
  handleFileId,
}) => {
  const [userName, setUserName] = useState<string | null>(null);

  const router = useRouter();
  const fileList = useAppSelector(selectorCovidDataData);

  useEffect(() => {
    const token = localStorage.getItem("token");

    if (token) {
      try {
        const decoded = jwtDecode<DecodedToken>(token);
        setUserName(decoded.name);
      } catch (error) {
        console.error("Не вдалося розшифрувати токен", error);
      }
    }
  }, []);

  // функція logOut
  const handleLogOut = () => {
    localStorage.removeItem("token");
    router.push("/");
  };

  // функція для форматування поля uploadedAt
  const formatDate = (dateString: string) => {
    const date = new Date(dateString);
    const year = date.getFullYear();
    const month = (date.getMonth() + 1).toString().padStart(2,
"0"); // Додаємо 0 перед місяцем, якщо менше 10
    const day = date.getDate().toString().padStart(2, "0"); //
Додаємо 0 перед днем, якщо менше 10
    return `${year}-${month}-${day}`;
  };

  return (
    <div className="h-screen w-[30%] bg-gradient-to-r from-blue-
500 to-cyan-500 py-10 pl-8 pr-1">
      <div className="mb-10 flex gap-x-4">
```

```
            <p className="text-3xl capitalize">
              {userName ? userName : "Завантаження..."}
            </p>
            <button onClick={handleLogOut}>log out</button>
          </div>

          <div  className="custom-scrollbar  flex   flex-col  gap-y-5
pr-2">
            {fileList?.map((item) => {
              return (
                <div className="flex gap-x-1" key={item._id}>
                  <div
                    className={cn(
                      "w-full   cursor-pointer   rounded-md   border
border-cyan-100 px-2 py-2 transition-all hover:border-yellow-500",
                      {
                        "border-yellow-500 bg-yellow-200":
                          activeFileId === item._id,
                      },
                    )}
                    onClick={() => {
                      handleFileId(item._id);
                    }}>
                    <p className="flex flex-col gap-y-1">
                      <span>{item.fileName}</span>
                      <span>{formatDate(item.uploadedAt)}</span>
                    </p>
                  </div>

                  <button
                    className="flex   items-center   justify-center
border p-2 text-xl font-medium hover:bg-red-500 hover:text-white-base"
                    onClick={() => handleDeleteFile(item._id)}>
                    x
                  </button>
                </div>
              );
            })}

            <div className="mb-10 flex flex-col gap-4">
              <form
                onSubmit={handleSubmit(onSubmit)}
                className="flex flex-col items-start gap-4 rounded-
md border border-gray-300 px-4 py-6">
                <input
                  type="file"
                  {...register("file", {
                    required: "Файл обов'язковий для завантаження",
                  })}
                  className="block  w-min  cursor-pointer  rounded-lg
border border-gray-300 text-sm text-gray-900 focus:outline-none"
                />
                {errors.file && (
```

```
                    <p          className="text-sm          text-red-
500">{errors.file.message}</p>
                  )}

                  <button
                    type="submit"
                    className="rounded-md bg-blue-500 px-4 py-2 text-
slate-50 hover:bg-blue-600">
                    Завантажити файл
                  </button>
                </form>
              </div>
            </div>
          </div>
        );
      };

      export default SideBar;
```

Програмний код компоненту DashBoard

```
      const Dashboard: React.FC<DashboardProps> = ({ id }) => {
        const  covidData  =  useAppSelector(selectorCovidDataData);  //
Отримуємо дані через селектор
        const chartRef = useRef(null);
        const  [startDate,  setStartDate]  =  useState<string>("");  //
Початкова дата
        const [endDate, setEndDate] = useState<string>(""); // Кінцева
дата
        const      [minPositiveCases,      setMinPositiveCases]      =
useState<number>(0); // Мінімальна кількість позитивних випадків
        const      [maxPositiveCases,      setMaxPositiveCases]      =
useState<number>(0); // Максимальна кількість позитивних випадків
        const      [minHospitalized,      setMinHospitalized]      =
useState<number>(0); // Мінімальна кількість госпіталізованих

        // Фільтруємо дані за переданим id
        const  filteredData  =  covidData?.find((data)  =>  data._id  ===
id);

        // Функція для скидання фільтрів
        const resetFilters = () => {
          setStartDate("");
          setEndDate("");
          setMinPositiveCases(0);
          setMaxPositiveCases(0);
          setMinHospitalized(0);
        };

        // Функція для форматування дати
```

```
      const formatDate = (date: number): string => {
        const dateStr = date.toString(); // Перетворюємо в рядок
        const year = dateStr.slice(0, 4);
        const month = dateStr.slice(4, 6);
        const day = dateStr.slice(6, 8);
        return `${year}-${month}-${day}`;
      };

      useEffect(() => {
        if (!filteredData || !chartRef.current) return; // Якщо немає
даних, нічого не робимо

        const chart = echarts.init(chartRef.current);

        const filteredChartData = filteredData.data
          .filter(
            (record) =>
              (!startDate ||
                new     Date(formatDate(record.date))     >=     new
Date(startDate)) && // Фільтр за початковою датою
              (!endDate || new Date(formatDate(record.date)) <= new
Date(endDate)), // Фільтр за кінцевою датою
          )
          .filter((record) => record.positive >= minPositiveCases) //
Фільтр за позитивними випадками
          .filter(
            (record)  =>  !maxPositiveCases  ||  record.positive  <=
maxPositiveCases,
          )
          .filter((record)   =>   record.hospitalizedCurrently   >=
minHospitalized); // Фільтр за госпіталізованими

        const chartOptions = {
          tooltip: {
            trigger: "axis",
            formatter: (params: any) =>
              params
                .map((item:     any)     =>     `${item.seriesName}:
${item.value}`)
                .join("<br/>"),
          },
          legend: {
            data: [
              "Positive Cases",
              "Negative Cases",
              "Hospitalized",
              "Deaths",
              "Recovered",
            ],
          },
          toolbox: {
            show: true,
            orient: "vertical",
```

```
            left: "right",
            top: "center",
            feature: {
              mark: { show: true },
              dataView: { show: true, readOnly: false },
              magicType: { show: true, type: ["line", "bar", "stack"]
},
              restore: { show: true },
              saveAsImage: { show: true },
            },
          },
          dataZoom: [
            {
              type: "slider",
              start: 0,
              end: 100,
            },
            {
              type: "inside",
              start: 0,
              end: 100,
            },
          ],
          xAxis: [
            {
              type: "category",
              data: filteredChartData.map(
                (data) => `Date: ${formatDate(data.date)}`,
              ), // Форматуємо дату
            },
          ],
          yAxis: [{ type: "value" }],
          series: [
            {
              name: "Positive Cases",
              type: "bar",
              data: filteredChartData.map((data) => data.positive),
            },
            {
              name: "Negative Cases",
              type: "bar",
              data: filteredChartData.map((data) => data.negative),
            },
            {
              name: "Hospitalized",
              type: "bar",
              data:          filteredChartData.map((data)          =>
data.hospitalizedCurrently),
            },
            {
              name: "Deaths",
              type: "bar",
              data: filteredChartData.map((data) => data.death),
```

```
            },
            {
              name: "Recovered",
              type: "bar",
              data: filteredChartData.map((data) => data.recovered),
            },
          ],
          grid: {
            bottom: "3%",
            containLabel: true,
            left: "3%",
          },
        };

        chart.setOption(chartOptions);

        return () => chart.dispose();
      }, [
        filteredData,
        startDate,
        endDate,
        minPositiveCases,
        maxPositiveCases,
        minHospitalized,
      ]);

      if (!covidData || covidData.length === 0) {
        return    <div    className="text-center    text-xl">Оберіть
файл</div>;
      }

      if (!filteredData) {
        return (
          <div  className="text-center  text-xl">Дані  для  цього  ID
відсутні</div>
        );
      }

      return (
        <div className="w-full max-w-[70%] px-5">
          <h1 className="mb-4 w-full pt-3 text-center text-3xl">
            COVID-19 Data Visualization
          </h1>

          {/* Фільтри */}
          <div className="mb-4 flex flex-wrap gap-4">
            {/* Фільтр за датою */}
            <div className="flex flex-col gap-1">
              <div className="flex flex-col">
                <label  htmlFor="start-date"  className="mb-1  font-
medium">
                  Start Date:
                </label>
```

```
                <input
                  id="end-date"
                  type="date"
                  value={endDate}
                  onChange={(e) => setEndDate(e.target.value)}
                  className="rounded border px-4 py-2"
                />
              </div>
              <div className="flex flex-col">
                <label   htmlFor="end-date"   className="mb-1   font-
medium">
                  End Date:
                </label>

                <input
                  id="start-date"
                  type="date"
                  value={startDate}
                  onChange={(e) => setStartDate(e.target.value)}
                  className="rounded border px-4 py-2"
                />
              </div>
            </div>

            {/* Фільтр за позитивними випадками */}
            <div className="flex flex-col gap-3">
              <div className="flex flex-col">
                <label htmlFor="min-positive" className="mb-1 font-
medium">
                  Min Positive Cases:
                </label>
                <input
                  id="min-positive"
                  type="number"
                  value={minPositiveCases}
                  onChange={(e)                                    =>
setMinPositiveCases(Number(e.target.value))}
                  className="rounded border px-4 py-2"
                />
              </div>
              <div className="flex flex-col">
                <label htmlFor="max-positive" className="mb-1 font-
medium">
                  Max Positive Cases:
                </label>
                <input
                  id="max-positive"
                  type="number"
                  value={maxPositiveCases}
                  onChange={(e)                                    =>
setMaxPositiveCases(Number(e.target.value))}
                  className="rounded border px-4 py-2"
                />
```

```
            </div>
          </div>

          {/* Фільтр за госпіталізованими */}
          <div className="flex flex-col">
            <label   className="mb-1   font-medium"   htmlFor="min-
hispitalized">
              Min Hospitalized
            </label>
            <input
              id="min-hispitalized"
              type="number"
              value={minHospitalized}
              onChange={(e)                                          =>
setMinHospitalized(Number(e.target.value))}
              className="border px-4 py-2"
              placeholder="Min Hospitalized"
            />
          </div>
        </div>

        {/* Кнопка для скидання фільтрів */}
        <button
          onClick={resetFilters}
          className="text-white mt-4 rounded bg-red-500 px-4 py-2
hover:bg-red-600">
          Скинути фільтри
        </button>

        {/* Графік */}
        <div
          ref={chartRef}
          style={{
            width: "100%",
            height: "400px",
            border: "1px solid #ddd",
            borderRadius: "8px",
            marginTop: "20px",
          }}
        />
      </div>
    );
  };

  export default Dashboard;
```

Додаток Б
Тези

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
імені ІВАНА ПУЛЮЯ
НАУКОВЕ ТОВАРИСТВО ім. ШЕВЧЕНКА

М А Т Е Р І А Л И

XII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

18-19 грудня 2024 року

ТЕРНОПІЛЬ

УДК 001
М34

ПРОГРАМНИЙ КОМІТЕТ

Голова: Приймак Микола – професор кафедри комп'ютерних систем та мереж, д.т.н., професо**р.**

Співголови: Марущак Павло – проректор з наукової роботи, докт. техн. наук, професор.

Баран Ігор – канд. техн. наук, доцент, декан факультету ФІС.

Науковий секретар: Надія Крива– старший викладач.

Члени: Василь Кривень - завідувач кафедри математичних методів в інженерії д.ф.-м.н., професор; Галина Осухівська – завідувач кафедри комп'ютерних систем та мереж, к.т.н., доцент; Микола Карпінський - професор кафедри кібербезпеки, д.т.н., професор; Жанна Баб'як - завідувач кафедри української та іноземних мов, к.пед. н., доцент; Ярослав Литвиненко – професор кафедри комп'ютерних наук, д.т.н., професор; Михайло Петрик - завідувач кафедри програмної іженерії, д.ф.-м.н., професор; Наталія Загородна – завідувач кафедри кібербезпеки, к.т.н., доцент.

ОРГАНІЗАЦІЙНИЙ КОМІТЕТ

Голова: Скоренький Юрій Любомирович – канд. техн. наук, доцент кафедри фізики.

Члени: доцент кафедри комп'ютерних наук, к.т.н. В. Никитюк; доцент кафедри програмної інженерії, к.т.н. Д. Михалик; доцент кафедри кібербезпеки, к.т.н. М. Стадник; доцент кафедри комп'ютерних систем та мереж, к.т.н. Є. Тиш; ст. викладач Л. Джиджора.

Матеріали XII науково-технічної конфції «Інформаційні моделі, системи та технології» Тернопільського національного технічного університету імені Івана Пулюя, (Тернопіль, 18–19 грудня 2024 р.). – Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2024.

М34

Адреса оргкомітету: ТНТУ ім. І. Пулюя, м. Тернопіль, вул. Руська, 56, 46001, тел. (0352) 52-41-33, факс (0352) 25-49-83.

E-mail: conffis2024@gmail.com

Редагування, оформлення та верстка: Надія Крива

СЕКЦІЇ КОНФЕРЕНЦІЇ, ЯКІ ПРЕДСТВЛЕНІ В ЗБІРНИКУ

- Математичне моделювання
- Інформаційні системи та технології, кібербезпека
- Комп'ютерні системи та мережі
- Програмна інженерія та моделювання складних розподілених систем
- Новітні фізико-технічні та освітні технології

В збірнику надруковано тези доповідей XII науково-технічної конференції «Інформаційні моделі, системи та технології» (Тернопіль, 18–19 грудня 2024 р.) за такими науковими напрямками: математичне моделювання; інформаційні системи та технології, кібербезпека; комп'ютерні системи та мережі; програмна інженерія та моделювання складних розподілених систем; новітні фізико-технічні та освітні технології.

Розрахований на науковців, викладачів та студентів вузів.

За зміст тез та дотримання норм академічної доброчесності відповідальність несе автор.

УДК

В. В. Коваль, Г. Б. Цуприк, канд. техн. наук, доц.

Тернопільський національний технічний університет імені Івана Пулюя, Україна

РОЗРОБКА ВЕБ-ІНТЕРФЕЙСУ ДЛЯ ІНТЕРАКТИВНОЇ ВІЗУАЛІЗАЦІЇ API-ДАННИХ НА ОСНОВІ JAVASCRIPT

V. V. Koval, H. B. Tsupryk, PhD, Assoc.Prof

DEVELOPMENT OF A WEB INTERFACE FOR INTERACTIVE VISUALIZATION OF API DATA BASED ON JAVASCRIPT

У сучасному світі веб-інтерфейси відіграють ключову роль у створенні зручних та інтуїтивно зрозумілих інструментів для обробки та аналізу даних. Їх використання дозволяє ефективно інтегрувати візуалізацію великого обсягу інформації в різноманітних галузях – від бізнесу до наукових досліджень. Здатність адаптуватися до швидких змін і забезпечувати інтерактивність робить такі інструменти надзвичайно затребуваними.

Проблема обробки та візуалізації даних, отриманих через API, набула особливого значення через стрімке зростання обсягів інформації. Інтерфейси, які можуть відображати ці дані в графічному вигляді, сприяють кращому розумінню складної інформації і допомагають у прийнятті рішень. Саме це визначає актуальність створення інструментів, здатних забезпечити динамічну та гнучку візуалізацію.

JavaScript було обрано як основний інструмент для розробки через його універсальність та потужний набір інструментів для створення інтерактивних графічних елементів. Бібліотеки, такі як D3.js і Apache ECharts, дозволяють швидко реалізовувати складні механізми візуалізації, а також створювати адаптивні рішення, що підходять для різних типів даних.

Головні особливості розробленого інтерфейсу:

- Доступність – можливість використовувати систему через веб-браузер на будь-якому пристрої.
- Гнучкість – підтримка різних типів графіків та інтерактивних елементів, що дозволяють користувачу легко налаштовувати відображення даних.
- Ефективність – оптимізація запитів до API та обробки великих обсягів даних, що забезпечує швидкість роботи.

Для розробки було використано такі технології:

- JavaScript як мова для інтерактивної логіки.
- ECharts.js для побудови динамічних графіків.
- RESTful API для отримання та оновлення даних.
- Next.js для створення інтерфейсу, що адаптується до різних пристроїв.

Результатом роботи стала система, яка забезпечує інтерактивну візуалізацію даних у режимі реального часу. Реалізовано можливість масштабування, фільтрації та сортування інформації, що робить інструмент зручним для широкого кола користувачів.

Практична значущість цієї роботи полягає у створенні платформи, яка може застосовуватись у фінансовому аналізі, дослідженні ринкових трендів, наукових обчисленнях та інших галузях, де потрібна наочна візуалізація даних.

Таким чином, розроблений веб-інтерфейс не лише відповідає сучасним вимогам до інтерактивних інформаційних систем, але й демонструє великий потенціал для подальшого розвитку у сфері обробки та візуалізації даних.

Література

1. Петрик М.Р., Михалик Д.М., Петрик О.Ю., Цуприк Г.Б. Методичні вказівки до виконання атестаційної роботи магістра за спеціальністю 121 – «Інженерія програмного забезпечення» для усіх форм навчання – Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя – 2020 – 27 с.

2. «Data Visualization with JavaScript» від Stephen A. Thomas. https://blog.tomsawyer.com/exploring-javascript-libraries-for-data-visualization

3. Vadim Ogievetsky, Joseph Lowery. «JavaScript and jQuery for Data Analysis and Visualization»

4. Бойко І.В., Петрик М.Р., Цуприк Г.Б. Моделювання та видобуток даних (високопродуктивні обчислення у великих та числових системах, комбінаторному аналізі): навчальний посібник. Тернопіль: : ТНТУ 2019 – 62 с.

5. Бойко І.В., Петрик М.Р., Цуприк Г.Б. Інформаційні технології видобутку даних (Data mining, високопродуктивні обчислення у складних системах): навчальний посібник. Тернопіль: : ТНТУ 2020 – 62 с.

Додаток В
Диск з роботою