НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут атомної та теплової енергетики

Кафедра інженерії програмного забезпечення в енергетиці

ДО ЗАХИСТУ ДОПУЩЕНО
В.о. завідувача кафедри
_____Олександр КОВАЛЬ
«_____»_____2023 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем і веб-технологій»
спеціальності 121 Інженерія програмного забезпечення
на тему: «Модуль візуалізації стану гідравлічного стану Нового Безпечного Конфайнменту ЧАЕС»

Виконав:
студент IV курсу, групи ТВ-91
Болдир Максим Сергійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник:
професор, д. т. н. Гаврилко Є. В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент:
професор, д. т .н, доцент Шушура О. М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2023

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної та теплової енергетики
Кафедра інженерії програмного забезпечення в енергетиці
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення інтелектуальних кібер-фізичних систем і веб-технологій»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Олександр КОВАЛЬ
(підпис)

«____»_____ 2023 р.

ЗАВДАННЯ
на дипломну роботу студенту

Болдирю Максиму Сергійовичу

(прізвище, ім'я, по батькові)

1. Тема роботи
Модуль візуалізації стану гідравлічного стану Нового Безпечного Конфайнменту ЧАЕС_____
керівник роботи Гаврилко Євген Володимирович, доктор технічних наук, професор
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від "__"_____ 2023 року №_____
3. Строк подання студентом роботи _____
3. Вихідні дані до роботи мови програмування C# і TypeScript, програмна платформа .NET, середовища розробки Visual Studio 2022, Visual Studio Code та SQL Server Management Studio._____
_____
4. Зміст (дипломної роботи) пояснювальної записки (перелік завдань, які потрібно розробити) Аналіз моделі гідравлічного стану НБК, аналіз сценаріїв використання додатку, розвиток архітектури додатку, розробка додатку для візуалізації гідравлічного стану НБК, тестування._____
_____
5. Перелік ілюстративного матеріалу робота містить 16 ілюстрацій.
6. Дата видачі завдання «30» вересня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Строки виконання етапів роботи	Примітка
1	Отримання завдання	30.09.2022	Виконано
2	Дослідження предметної області	02.10.2022 - 20.02.23	Виконано
3	Постановка вимог до проєктування системи	21.02.2023- 28.02.2023	Виконано
4	Дослідження існуючих рішень	29.02.2023- 16.04.2023	Виконано
5	Розробка програмного продукту	17.04.2023- 10.05.2023	Виконано
6	Тестування	11.05.2023- 18.05.2023	Виконано
7	Захист програмного продукту	19.05.2023	Виконано
8	Оформлення дипломної роботи	20.05.2023- 05.06.2023	Виконано
9	Передзахист	06.06.23	Виконано
10	Захист	19.06.23	Виконано

Студент _____ Максим БОЛДИР
(підпис) (ім'я, прізвище)

Керівник роботи _____ Євген ГАВРИЛКО
(підпис) (ім'я, прізвище)

РЕФЕРАТ

Структура та обсяг дипломної роботи. Робота містить 58 сторінок, 16 рисунків, 1 додаток та 10 посилань.

Метою роботи є дослідження сучасних засобів візуалізації даних та можливостей розробки інтерфейсу користувача для візуалізації даних стану гідравлічного стану Нового Безпечного Конфайнменту.

Для досягнення поставленої мети виконано такі завдання:

- проаналізовано предметну область, розглянуто можливі варіанти взаємодії користувача з системою, обрано архітектурний підхід до розробки та засоби розробки;
- розроблено API для обробки даних на сервері з урахуванням можливості подальшого розширення функціоналу та інтеграції з іншими сервісами та модулями;
- розроблено користувацький інтерфейс для відображення історичних даних натурних вимірювань НБК.

Практичне значення одержаних результатів полягає в тому, що розроблений веб-додаток дозволяє переглядати візуалізовані дані натурних вимірювань НБК, що спрощує їх аналіз. Зроблено огляд можливих підходів до представлення даних та розглянуто проблеми, пов’язані з цими підходами. Визначено можливості до подальшого розвитку системи та наведено рекомендації щодо подальших досліджень.

Ключові слова: візуалізація даних, новий безпечний конфайнмент, модель гідравлічного стану.

ABSTRACT

Structure and scope of the thesis. The thesis consists of 58 pages, 16 figures, 1 appendix, and 10 references.

The purpose of this work is to explore modern data visualization tools and the possibilities of developing a user interface for visualizing the data of the hydraulic state of the New Safe Confinement (NSC).

To achieve this goal, the following tasks were accomplished:

- Analyzed the subject area, considered possible user-system interactions, selected an architectural approach for development, and identified development tools.
- Developed an API for data processing on the server, taking into account the potential for future functionality expansion and integration with other services and modules.
- Designed a user interface for displaying historical data of physical measurements in the NSC.

The practical significance of the obtained results is that the developed web application allows viewing visualized data of physical measurements in the NSC, thereby simplifying their analysis. An overview of possible approaches to data representation has been provided, and associated challenges have been discussed. The potential for further system development has been identified, and recommendations for future research have been provided.

Keywords: data visualization, New Safe Confinement, hydraulic state model.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД - база даних.

НБК - Новий Безпечний Конфайнмент.

КП - кільцевий простір.

ОО - основний об'єм.

ОС - оточуюче середовище.

ОУ - об'єкт «Укриття».

РА - радіоактивні аерозолі.

API – прикладний програмний інтерфейс.

JSON - формат обміну даними.

JWT - JSON Web Token. Компактний формат обміну даними у вигляді JSON-об'єкта, який використовується для безпечної передачі інформації між сторонами.

UI - користувацький інтерфейс.

UX - користувацький досвід.

SPA - Single-Page Application. Веб-додаток, що працює на одній сторінці, без перезавантаження.

ВСТУП

У квітні 1986 році внаслідок аварії 4-й реакторний блок ЧАЕС був повністю зруйнований, що призвело до великої техногенної катастрофи. У листопаді 1986 р. зруйнований блок був покритий тимчасовою, негерметичною спорудою, названою «Об'єктом Укриття», термін експлуатації якої було оцінено у 30 років. Через нещільності в конструкції ОУ відбувалося постійне забруднення навколишнього середовища. Крім того мала місце загроза збільшення радіаційного забруднення внаслідок можливого обвалу нестабільних конструкцій реактора та ОУ.

Починаючи з 1996 р. почалося обговорення подальшої ізоляції зруйнованого реактора і можливість вилучення, переробки та поховання радіоактивних матеріалів, що залишилися в ньому. В результаті такого обговорення було запропоновано концепцію будівництва споруди, яка б ізолювала зруйнований реактор разом з ОУ від навколишнього та дозволило витягувати радіоактивні матеріали протягом часу. Така споруда була названа «Новий Безпечний Конфайнмент» (англ. New Safe Confinement). До 2000 року концепцію форми та розмірів НБК було визначено у вигляді арочної конструкції, яка має бути побудована на відстані від ОУ та насунута на ОУ. У 2016 році НБК було введено в експлуатацію.

З метою забезпечення функціонування НБК протягом тривалого періоду часу існує необхідність у постійному спостереженні за спорудою та її обслуговуванні. Оскільки НБК є конструкцією, що захищає навколишнє середовище від радіоактивних викидів, він піддається впливу як зовнішніх так і внутрішніх факторів, серед яких важливу роль відіграють атмосферні умови.

Сучасний темп розвитку інформаційних технологій та зростання цифровізації виробничих і дослідницьких процесів зумовило появу концепції цифрових двійників – цифрових копій фізичних об’єктів та процесів, що дозволяють моделювати, аналізувати та потенційно прогнозувати роботу фізичного об’єкту чи процесу. Зокрема і для НБК, розробка та застосування цифрового

двійника є актуальною проблемою. Створення цифрового двійника конфайнменту дозволить оптимізувати ефективність збереження конструкцій та захисту навколишнього середовища від радіоактивних викидів. Одним із питань розробки цифрового двійника НБК є оптимізація вітрових установок, що забезпечують вентиляцію всередині конструкції, зменшуючи при цьому об’єм радіоактивних викидів назовні.

Для обчислення оптимальних об’ємів витрат вітрових установок в залежності від різних погодних умов використовується гідравлічна модель НБК, для повної оцінки стану якої є необхідність у візуалізації цієї моделі. Метою роботи є дослідження сучасних засобів візуалізації даних та можливостей розробки інтерфейсу користувача для візуалізації даних стану гідравлічного стану НБК.

1 МЕТА ТА ПОСТАНОВКА ЗАДАЧІ РОЗРОБКИ МОДУЛЯ ВІЗУАЛІЗАЦІЇ

Метою дипломної роботи є дослідження та розробка системи візуалізації стану моделі гідравлічного стану Нового Безпечного Конфайнменту ЧАЕС. При розробці системи такого типу необхідно розуміти специфіку предметної області, сформулювати пріоритетні задачі та окреслити можливості до подальшого розвитку розроблюваної системи. У цьому розділі буде визначено предметну область та основні задачі розробки.

1.1 Визначення предметної області

Метою проектування та будівництва Нового Безпечного Конфайнменту Чорнобильської атомної електростанції є захист оточуючого середовища при виконанні робіт по вилученню радіоактивних матеріалів із зруйнованого 4-го енергоблоку та демонтажу нестабільних конструкцій Об’єкта «Укриття» (ОУ).

У конструкції такого розміру (ШхВхД=250х110х165 м) неможливо уникнути негерметичностей, тому в оболонках НБК і між Західними і Східними стінами та будівельними конструкціями під ними існують чисельні протічки повітря із НБК в оточуюче середовище (рис. 1.1).

Для вирішення цієї проблеми було обрано науково-експериментальний підхід з застосуванням моделі гідравлічного стану НБК, параметри якої визначаються на основі даних натурних вимірювань параметрів гідравлічного стану НБК [1].

Дані про гідравлічний стан Нового безпечного конфайнменту (НБК), включаючи вимірювання перепаду тисків між Кільцевим Простором (КП) і Основним Об'ємом (ОО), ОО і Оточуючим Середовищем (ОС), КП і ОС, а також вимірювання витрат вентиляторів в КП, ОО та кліматичних умов (напрямок і швидкість вітру), надходять в модель НБК. Ця модель оцінює правильність гідравлічних коефіцієнтів. Якщо коефіцієнти оцінені правильно, модель розраховує

неорганізовані витрати повітря з ОО НБК в ОС через Західні і Східні протічки (криві сині лінії на рис. 1.1). У разі неправильної оцінки коефіцієнтів, проводиться калібрування моделі (уточнення гідравлічних коефіцієнтів), і розраховуються неорганізовані витрати повітря з НБК в ОС з концентрацією радіоактивних матеріалів, яка також вимірюється. Організовані витрати повітря очищаються від радіоактивних матеріалів за допомогою витяжних вентиляторів та спеціальних фільтрів перед виходом в ОС.

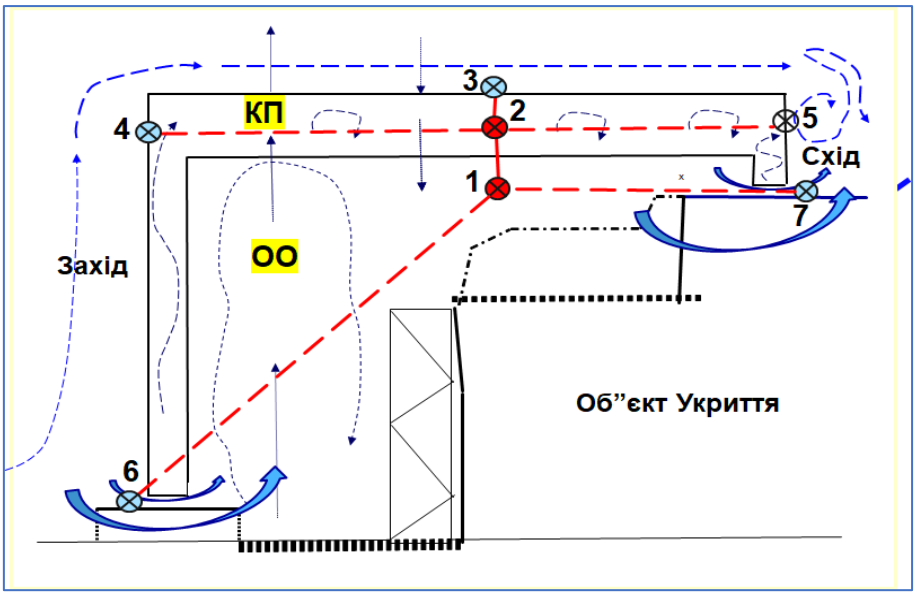


Рисунок 1.1 – Схематичне представлення моделі гідравлічного стану НБК у поздовжньому перерізі: 1 – вузол основного об'єму, 2 – вузол кільцевого простору, 3 – вузол на зовнішній циліндричній поверхні, 4 – вузол на зовнішній поверхні західної стіни, 5 – вузол на зовнішній поверхні східної стіни, 6 – вузол на зовнішній частині західних протічок, 7 – вузол на зовнішній частині східних протічок

Проблема полягає у тому, що оцінювати роботу моделі, користуючись табличним представленням даних досить складно. Існує необхідність у візуалізації даних у вигляді діаграм.

Завдяки співпраці з Інститутом технічної теплофізики НАНУ в рамках дослідження цифрових двійників критичної інфраструктури, маємо доступ до експлуатаційних вимірювань гідравлічного стану НБК. У рамках даної дипломної

роботи було використано уже оброблений набір даних. Розглянемо наявні у наборі даних параметри:

- DPE (Data Point Element) – поле набору даних, що є одночасно часом проведення виміру та його унікальним ідентифікатором в системі моніторингу гідравлічного стану НБК;
- швидкість вітру – виміри швидкості вітру в метрах на секунду з метеостанції НБК на висоті 15м;
- напрям вітру – виміри напрямку вітру в градусах з метеостанції НБК на висоті 15м;
- тиск КП – тиски в кільцевому просторі;
- тиск ОО – тиски в основному об’ємі;
- витрати КП+ – закачування повітря в кільцевий простір;
- витрати ОО- – викачування повітря з основного об’єму;
- витрати ОО+ – закачування повітря в основний об’єм;
- перепад КП-ОС – перепад тиску між кільцевим простором та оточуючим середовищем;
- перепад ОО-ОС – перепад тиску між основним об'ємом та оточуючим середовищем;
- перепад КП-ОО – перепад тиску між кільцевим простором та основним об’ємом.

Саме ці дані лежать в основі обчислення моделі гідравлічного стану НБК.

1.2 Постановка задач дослідження та розробки

Задача даної бакалаврської дипломної роботи полягає у дослідженні сучасних інструментів візуалізації даних та розробці програмного застосунку для візуалізації моделі гідравлічного стану Нового Безпечного Конфайнменту ЧАЕС на основі даних історичних натурних вимірювань.

Пріоритетом в розробці інформаційної системи завжди є створення зручного інтерфейсу користувача (UI) задля забезпечення зручного користувацького досвіду (UX). Для кожної спеціалізованої системи дизайн користувацького досвіду потребує дослідження. Для предметної області модуля візуалізації моделі гідравлічного стану потрібно обов'язково врахувати не тільки різні форми подання даних, а і такі аспекти користувацького досвіду, як зручна навігація, відображення підказок та додаткової інформації та можливість фільтрувати та сортувати дані.

Не дивлячись на те, що концептуально розроблювана система є лише одним з модулів цифрового двійника НБК, розробка такої системи в перспективі потребує можливості довгострокової підтримки та розширення. Основними кроками для досягнення цього є вибір технологій та архітектури системи. При підборі технологій необхідно врахувати їхню актуальність, швидкодію та зручність використання, а також перспективи їхнього подальшого розвитку. У виборі архітектури ж необхідно враховувати необхідність у подальшій інтеграції з іншими системами та забезпечити можливість швидкої зміни та розширення функціоналу системи.

На даному етапі дослідження концепція модуля візуалізації гідравлічного стану не передбачає можливості взаємодії із апаратними засобами НБК, але в роботі з цифровими двійниками критичної інфраструктури потрібно враховувати, що існуватиме необхідність роботи із чутливими, потенційно засекреченими даними. Тож при розробці системи також врахуємо і безпеку додатку та реалізуємо механізми автентифікації, авторизації та адміністрування користувачів. Оскільки система не передбачена для загального доступу, то відкритої реєстрації в системі бути не може. Натомість передбачимо механізм додавання нових користувачів в систему для адміністратора.

Одним із найцікавіших аспектів застосування цифрових двійників є можливість моніторингу роботи системи в реальному часі. Очевидно, що для реалізації моніторингу даних у реальному часі для розроблюваної системи необхідно мати доступ до сенсорів, розташованих безпосередньо на території НБК.

Хоча наразі доступ до сенсорів відсутній, існує потреба в моделюванні такого функціоналу. Таким чином, необхідно дослідити можливості до роботи системи із даними, отриманими в реальному часі. Це відкриє можливість до потенційної інтеграції модуля візуалізації із системами, що моделюватимуть роботу та оптимізацію вітрових установок НБК в реальному часі.

У ході роботи буде розглянуто різні архітектурні підходи та можливості до реалізації застосунку відображення статистичних даних із застосуванням технологій веб-розробки та інструментів відображення даних у вигляді графіків та таблиць. Крім того, буде розглянута можливість моделювання сценарію отримання даних з НБК в режимі реального часу на основі наявних історичних даних.

Результат виконання роботи буде складатися з серверної та клієнтської частини. Серверна частина буде виконана із застосуванням платформи .NET та мови програмування C# та реляційної бази даних в системі управління базами даних MS SQL Server. Для розробки клієнтської частини буде застосовано мови програмування JavaScript та TypeScript, фреймворк Angular та бібліотеку для створення графіків Chart.Js. Також буде розглянута бібліотека для асинхронного обміну повідомленнями SignalR, як інструмент обміну між клієнтом та сервером в режимі реального часу.

Висновки до розділу 1

У даному розділі була сформульована предметна область розробки, її актуальність та мета. Визначені основні параметри, що потребують візуалізації. Поставлена задача, що включає в себе дослідження сучасних інструментів візуалізації даних та розробки програмного застосунку. Основними об’єктами дослідження є дизайн користувацького досвіду вибір технологій та архітектури системи. Враховуючи потребу у довгостроковій підтримці та розширенні, необхідно вибрати актуальні технології з можливістю швидкої зміни та розширення функціоналу. Розроблювана система не передбачає взаємодії з апаратними засобами

НБК, але безпека та механізми автентифікації, авторизації та адміністрування користувачів є важливими аспектами дослідження. Також розглядається можливість роботи з даними в реальному часі для моніторингу системи та потенційної інтеграції з іншими системами моделювання. Результатом роботи буде серверна та клієнтська частини, реалізовані з використанням визначених технологій і інструментів

2 АНАЛІЗ ІСНУЮЧИХ СИСТЕМ

Серед великої кількості сучасних інструментів візуалізації даних найпопулярнішими є:

- Grafana;
- Tableau;
- Google Data Studio;
- Microsoft Power BI;
- Microsoft Power View.

Розглянемо кожен з них індивідуально.

2.1 Grafana

Grafana - це інструмент для візуалізації та аналізу даних у реальному часі. Він надає можливість створювати інтерактивні діаграми з різних джерел даних [2].

Grafana має широкий набір різних форм візуалізації даних, включаючи лінійні графіки, стовпчасті графіки, кругові діаграми, теплові карти, географічні карти та надає можливість налаштовувати кожну діаграму, вибираючи тип даних, кольори, масштаби, вісі та інші параметри.

Grafana підтримує різні джерела даних, включаючи бази даних (наприклад, MySQL, PostgreSQL, InfluxDB), сервіси моніторингу (наприклад, Prometheus, Zabbix), сервіси логування (наприклад, Elasticsearch, Loki).

Також Grafana дозволяє взаємодіяти з діаграмами та організовувати їх в інформаційні панелі. Дані можна фільтрувати збільшувати масштаб, наводити курсор на елементи графіків для отримання деталей, створювати спливаючі вікна з додатковою інформацією та виконувати інші операції, щоб глибше аналізувати дані.

В Grafana представлені можливості спільної роботи, що дозволяє кільком користувачам працювати з даними та інформаційними панелями одночасно.

Крім того, Grafana має багато розширень та плагінів, які дозволяють розширити його функціональність та інтегрувати з іншими інструментами.

Grafana використовується в різних сферах, включаючи моніторинг систем, аналітику даних, IoT-проекти та багато інших.

2.2 Tableau

Tableau - це інструмент для візуалізації та аналізу даних, який дозволяє створювати інтерактивні діаграми [3].

Tableau представляє можливості до візуалізації даних, подібні до тих, що наявні в Grafana. Відмінність полягає в тому, що Grafana надає більше можливостей до відображення даних в режимі реального часу. Tableau ж в свою чергу має більше можливостей для підключень до різних джерел даних. Інструмент дозволяє підключати різні джерела даних, включаючи бази даних, ексель-файли, веб-джерела, Hadoop, сервіси хмарних обчислень та інші.

Також Tableau надає можливості автоматизувати процеси оновлення та розповсюдження даних. Користувач має можливість налаштовувати регулярні оновлення даних, запускати завдання автоматичного оновлення та надсилати звіти на електронну пошту або публікувати їх в інтернеті.

Tableau є потужним інструментом для аналізу та візуалізації даних, який використовується в таких сферах, як бізнес-аналітика, фінанси, маркетинг та наука.

2.3 Google Looker Studio

Looker Studio - це безкоштовний веб-інструмент для візуалізації та аналізу даних, розроблений компанією Google. Він дозволяє створювати красиві та інтерактивні звіти, дашборди та інфографіку з різних джерел даних [4].

Looker Studio має простий інтерфейс, що дозволяє легко створювати візуалізації без необхідності програмування або складних налаштувань. Управління відбувається через інтерфейс користувача.

Looker Studio інтегрується з різними джерелами даних, включаючи бази даних, Google Sheets, Google Analytics, Google Ads, BigQuery, YouTube, CSV-файли та багато інших.

Інструмент надає можливсть ділитися звітами, дашбордами та інфографікою з іншими користувачами, дозволяючи їм переглядати та редагувати дані. Також ви можете публікувати ваші візуалізації в Інтернеті та вбудовувати їх на веб-сайти.

2.4 Microsoft Power BI

Power BI - це комплексна платформа бізнес-аналітики та візуалізації даних від Microsoft. Вона дозволяє користувачам підключатися до різноманітних джерел даних, обробляти їх, створювати звіти та інтерактивні візуалізації, а також ділитися ними з іншими користувачами. Power BI пропонує різноманітні інструменти та функції для аналізу та відображення даних, дозволяючи здійснювати широкий спектр бізнес-аналітичних задач [5].

Основні компоненти та функції Power BI:

- Power BI Desktop: Це настільний інструмент для розробки звітів та візуалізацій. Power BI Desktop дозволяє збирати дані з різних джерел, моделювати їх, створювати візуалізації та додавати розрахунки. Він надає широкі можливості для налаштування та настройки звітів перед їх публікацією;
- Power BI Service: Це хмарна платформа Power BI, де можна публікувати, керувати та ділитися звітами та візуалізаціями. Power BI Service дозволяє завантажувати звіти, створені в Power BI Desktop, та розміщувати їх у веб-середовищі, доступному для спільної роботи з іншими користувачами. Він також має функції планування оновлення даних та можливості для налаштування безпеки та доступу;

- Power BI Mobile: Це мобільний додаток Power BI для iOS та Android, який дозволяє отримувати доступ до звітів та візуалізацій на мобільних пристроях. Користувачі можуть переглядати та взаємодіяти з даними в режимі реального часу, а також отримувати сповіщення та оновлення;
- Power Query: Цей інструмент дозволяє підключатися до різних джерел даних, виконувати операції з їхнім перетворенням та очищенням перед їх використанням у звітах Power BI. Power Query дозволяє імпортувати, з'єднувати та очищувати дані з різних форматів файлів, баз даних та веб-джерел;
- Power Pivot: Це інструмент для моделювання даних, який дозволяє користувачам створювати складні зв'язки між таблицями та створювати розрахунки на основі даних. Power Pivot дозволяє створювати потужні моделі даних з великим обсягом інформації та ефективно аналізувати дані.

Power BI дозволяє організаціям та користувачам робити обґрунтовані рішення на основі даних, аналізувати тренди, виявляти проблеми та можливості, а також ділитися даними та звітами зі співробітниками.

2.5 Microsoft Power View

Power View (Відображення даних) - це інструмент візуалізації даних та інтерактивних звітів у Microsoft Excel [6]. Він є частиною пакета Microsoft Power BI, який також включає Power Pivot і Power Query. Power View дозволяє користувачам створювати високоякісні візуалізації даних, звіти та презентації з їх даних.

Power View дозволяє користувачам досліджувати та аналізувати дані шляхом створення інтерактивних візуалізацій. Користувачі можуть вибирати та фільтрувати дані, докладатися до деталей та взаємодіяти з візуальними елементами для отримання висновків.

Фільтри (slicers) є інтерактивними елементами керування, що дозволяють користувачам фільтрувати дані в звітах Power View. Користувачі можуть легко

створювати фільтри, щоб обмежувати дані, що відображаються, що допомагає аналізувати конкретні підмножини інформації в межах візуалізацій.

Power View підтримує функції анімації, що дозволяють користувачам візуалізувати зміни з часом. За допомогою функції "Play" користувачі можуть створювати динамічні візуалізації, які автоматично переходять через різні періоди часу або категорії.

Користувачі можуть клацати на точки даних, докладатися до деталей, фільтрувати дані та виділяти певні елементи для дослідження підлеглої інформації. Ця можливість взаємодії спрощує аналіз даних з різних точок зору.

Інтеграція з Power Pivot і Power Query: Power View може використовувати можливості Power Pivot для моделювання даних і Power Query для трансформації та підготовки даних. Користувачі можуть імпортувати дані з різних джерел, створювати зв'язки між таблицями та покращувати модель даних перед їх візуалізацією в Power View.

Висновки до розділу 2

Під час розробки програмного забезпечення для візуалізації даних потрібно спиратися на досвід існуючих інструментів візуалізації даних та забезпечувати зручне відображення та інтерактивність діаграм. Також важливо розвивати додаток таким чином, щоб забезпечити гнучкість в роботі з різними форматами даних.

3 ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

В цьому розділі будуть представлені та проаналізовані технології, які були застосовані під час розробки цього програмного продукту. Також будуть наведені основні переваги цих технологій, що зумовили їх вибір.

3.1 .Серверна частина

3.1.1 .NET

.NET є платформою розробки програмного забезпечення, що розробляється компанією Microsoft. Вона надає розробникам інструменти та середовище для створення різноманітних програм, включаючи веб-додатки, мобільні додатки, настільні програми, хмарні рішення та ігри.

В основі платформи .NET лежить середовище виконання CLR (Common Language Runtime), яке забезпечує виконання програмного коду та управління оперативною пам’яттю. Розробники можуть використовувати різні мови програмування, такі як C#, VB.NET, F# і C++/CLI, для створення додатків на платформі .NET. Ці мови підтримують об'єктно-орієнтоване програмування (ООП).

Однією з головних переваг .NET є його кросплатформність. Код, написаний для .NET, може працювати на різних платформах, таких як Windows, macOS і Linux, за допомогою платформи .NET Core без значних змін в коді.

.NET надає набір бібліотек і фреймворків, які спрощують розробку програм та прискорюють процес розробки. У ході розробки було застосовано такі компоненти .NET, як ASP.NET, Entity Framework та ASP.NET Identity.

ASP.NET є фреймворком призначеним для створення веб-додатків та веб-сайтів. Основою ASP.NET є середовище ASP.NET Runtime і ряд компонентів,

необхідних для обробки веб-запитів і відповідей. ASP.NET пропонує три головні моделі розробки: Web Forms, MVC та Web API.

ASP.NET Web Forms - це модель розробки, яка забезпечує розробку веб-додатків за допомогою елементів управління на основі подій.

ASP.NET MVC - це модель розробки, яка базується на патерні Model-View-Controller. Вона забезпечує розділення логіки додатку на модель (дані), представлення (відображення) і контролер (логіка обробки запитів). Це сприяє більшому контролю над поведінкою і шаблонами веб-додатків.

ASP.NET Web API - це фреймворк для розробки веб-сервісів, який дозволяє будувати API (Application Programming Interface) для обміну даними між різними додатками. ASP.NET Web API спрощує розробку API шляхом надання зручного середовища для визначення маршрутів, обробки запитів та відповідей. Він використовує протокол HTTP для взаємодії з клієнтами, надаючи можливість використовувати HTTP запити, такі як GET, POST, PUT та DELETE, для здійснення операцій з ресурсами. Крім того, він підтримує стандартні коди стану HTTP та дозволяє налаштовувати заголовки запитів і відповідей. Також Web API надає можливість вибору формату передачі даних. Серіалізацію та десеріалізацію об'єктів, що передаються через API, можна виконувати як у форматі JSON, так і в форматі XML. Обмін даними у форматі JSON робить дану модель розробки найгнучкішою серед трьох наявних у ASP.NET, тому в ході роботи будемо використовувати саме метод розробки Web API.

Ще одним компонентом платформи .NET, застосованим у цій роботі є Entity Framework. Entity Framework (EF) - це об'єктно-орієнтований маппер даних (ORM) [7]. Він дозволяє розробникам працювати з даними, використовуючи об'єктно-орієнтований підхід, замість прямої роботи з SQL-запитами. EF дозволяє представляти таблиці бази даних як класи та зв'язки між ними як властивості об'єктів. Це дозволяє розробникам працювати з даними у вигляді об'єктів і використовувати властивості, методи та спадковість для зручного взаємодії з даними. Крім того, EF автоматично виконує мапінг (встановлення відповідності)

між об'єктами та таблицями бази даних. У фреймворку наявні можливості як визначати мапінг явно, так і використовувати конвенції назв, що дозволяють EF автоматично вирішувати правила мапінгу. В EF входить мова запитів LINQ (Language-Integrated Query), яка дозволяє розробникам писати запити до бази даних, використовуючи розширення мови C#. Це спрощує написання складних запитів та забезпечує перевірку на етапі компіляції. Наявний в EF механізм міграцій дозволяє автоматично створювати таблиці у базі даних на основі програмного коду на мові C# та вносити зміни до схеми бази даних, не втрачаючи наявних даних.

ASP.NET Identity є фреймворком аутентифікації та авторизації, який входить в склад платформи ASP.NET. Він надає інструменти для роботи зі збереженням, управлінням та перевіркою облікових записів користувачів, ролей та дозволів у веб-додатках.

3.1.2 C#

Основною мовою програмування платформи .NET є мова C#. Це мова із синтаксисом, подібним до синтаксису C, C++ та Java. C# підтримує концепції об'єктно-орієнтованого програмування, такі як спадкування, поліморфізм, інкапсуляція та абстракція. Пам’ять у C# виділяється та очищується автоматично, що спрощує роботу з пам'яттю та допомагає уникнути багатьох типових помилок. Також мова підтримує розширення функціональності за допомогою розширень (extensions) та делегатів. Це дозволяє додавати нові методи до існуючих типів без модифікації їхнього вихідного коду.

3.1.3 Microsoft SQL Server

Microsoft SQL Server (MS SQL Server) є реляційною базою даних, розробленою компанією Microsoft. Він надає потужні можливості для зберігання, керування та обробки даних.

SQL Server використовує мову запитів Transact-SQL для взаємодії з базою даних. Transact-SQL (T-SQL) - це процедурне розширення мови SQL, створене компанією Microsoft (для Microsoft SQL Server) і Sybase (для Sybase ASE). У порівнянні із іншими діалектами SQL, Transact-SQL був розширений додатковими можливостями, такими як керуючі оператори, локальні і глобальні змінні, різні додаткові функції для обробки рядків, дат, математики, тощо, підтримка аутентифікації Microsoft Windows.

MS SQL Server може працювати з великими обсягами даних і підтримує розподілені системи баз даних. Він забезпечує можливість масштабування горизонтально (додавання нових серверів) та вертикально (збільшення потужності і ресурсів сервера).

MS SQL Server має вбудовані механізми безпеки для захисту даних, такі як автентифікація, авторизація, шифрування даних та аудит. Він дозволяє налаштовувати рівні доступу до даних для різних користувачів та ролей.

SQL Server надає інструменти для резервного копіювання та відновлення баз даних, що забезпечує захист даних від втрати або пошкодження.

3.1.4 SignalR

У ході дослідження можливостей імплементації отримання даних в реальному часі було розглянуто бібліотеку SignalR. SignalR - це відкрита бібліотека для реалізації додатків в реальному часі. Вона забезпечує зручний спосіб побудови взаємодії між клієнтами та серверами з допомогою веб-сокетів або інших механізмів зворотного зв'язку, таких як Server-Sent Events або Long Polling.

Основним принципом роботи SignalR є двостороння комунікація між клієнтом і сервером. Клієнт може ініціювати взаємодію, надсилати запити серверу та отримувати оновлення в режимі реального часу від сервера. Це дозволяє створювати додатки, які миттєво реагують на зміни даних і сповіщають користувачів про події в режимі реального часу.

SignalR надає розробникам інструменти для роботи з різними платформами, включаючи .NET, JavaScript, Java і Python. Вона підтримує автоматичне виявлення доступних транспортних протоколів та вибір найоптимальнішого для кожного клієнта, що дозволяє забезпечити максимальну сумісність та продуктивність в різних середовищах.

SignalR дозволяє надсилати повідомлення від сервера до всіх підключених клієнтів або до певних груп клієнтів. Клієнти можуть бути організовані в групи, і сервер може надсилати повідомлення всім учасникам групи. SignalR також дозволяє відстежувати зміни в реальному часі на пристроях, таких як смартфони або планшети.

Бібліотека SignalR може бути використана в різних сценаріях, таких як веб-додатки, мобільні додатки, групові чати та моніторинг систем.

3.2 Клієнтська частина

3.2.1 Angular

Angular є фреймворком для розробки веб-додатків, створеним компанією Google. Він базується на мові програмування TypeScript і надає інструменти для розробки масштабованих та динамічних односторінкових додатків (Single Page Applications) [8]. Angular побудований на основі компонентної архітектури, де весь інтерфейс користувача поділений на невеликі та перевикористовувані компоненти. Кожен компонент включає HTML-шаблон, CSS-стилі та логіку, пов'язану з ним. Angular надає широкий набір директив, які дозволяють маніпулювати DOM-елементами, контролювати видимість елементів та обробляти події. Він також дозволяє створювати власні директиви для вирішення конкретних задач. Angular використовує бібліотеку RxJS для реалізації реактивного програмування. Це дозволяє зручно працювати з асинхронними операціями, подіями та потоками даних. Angular має вбудовану систему впровадження залежностей (dependency

injection), яка спрощує керування залежностями між компонентами. Це допомагає покращити перевикористання коду, тестування та розширюваність додатків. Angular надає можливість налаштування маршрутизації в додатку. Це дозволяє перехоплювати URL-адреси та відображати відповідні компоненти, забезпечуючи навігацію між різними сторінками.

Життєвий цикл компонентів Angular розділений на декілька етапів, що називаються хуками життєвого циклу (lifecycle hooks). Вони представляють собою, що функції дозволяють впливати на поведінку компонента в певні моменти його життєвого циклу, наприклад, підключати або відключати ресурси, або працювати зі змінами даних. Розглянемо найпоширеніші з них:

- ngOnChanges - викликається, коли вхідні дані компонента змінюються, але лише для вхідних даних, що мають примітивний тип, або є об’єктами, у яких змінюється посилання;
- ngOnInit - викликається один раз, після того, як Angular ініціалізував вхідні дані компонента та побудував його представлення (view) та дає можливість виконати початкове налаштування компонента;
- ngOnDestroy - викликається перед тим, як компонент буде знищено. Він дозволяє вам відключити підписки RxJS, очистити ресурси або виконати будь-які інші дії, що необхідні для коректного звільнення пам'яті;
- ngAfterViewInit - викликається після того, як Angular ініціалізував представлення компонента (view) та всі дочірні представлення (child views). Тут можна виконувати операції які потребують доступу до DOM-елементів;
- ngDoCheck – викликається при будяких змінах в додатку та використовується для перевірки змін, які не може відстежити ngOnChanges. Проте викликається цей хук дуже часто протягом існування компоненту, тому його варто використовувати лише у крайньому випадку.

На рисунку 3.3.1 зображено порядок виконання хуків життєвого циклу.

Рисунок 3.1 – Порядок виконання хуків життєвого циклу

Важливим елементом фреймворку є Angular CLI (Command Line Interface) - інструмент командного рядка, розроблений для спрощення і прискорення розробки додатків на Angular. Він надає набір команд для автоматизації рутинних завдань, таких як створення компонентів, модулів, сервісів та тестів. Angular CLI дозволяє легко створювати нові Angular проекти з встановленою структурою та конфігурацією, швидко генерувати компоненти, модулі та сервіси. Також він включає в себе вбудований сервер розробки, який автоматично перезавантажує додаток при зміні файлів. Це дозволяє вам швидко бачити результати внесених змін без необхідності вручну перезавантажувати додаток. Крім того, в Angular CLI наявна вбудована підтримка автоматизованого тестування з використанням фреймворку Karma.

Для швидкого створення інтерфейсів користувача в Angular доступна бібліотека Angular Material. Angular Material надає широкі можливості для створення інтерфейсу користувача в Angular додатках. Бібліотека містить велику кількість готових компонентів, таких як кнопки, форми, таблиці, навігаційні панелі, картки, діалогові вікна, спливаючі меню, підказки, тощо. Для доступних компонентів легко змінювати кольори, типографію, розміри, тіні та інші стилі

компонентів за допомогою готових тем або власних налаштувань, а також додавати анімацію. Angular Material має вбудовані шаблони та макети для швидкої розробки, що надає можливість використовувати готові компоненти та макети, щоб створювати стандартні структури сторінок, такі як навігаційна панель, бічна панель, тощо. Angular Material також пропонує додаткові функції, такі як форми з валідацією, робота з таблицями, графіки, робота з даними. Крім готових компонентів, Angular Material надає Angular Component Dev Kit (CDK), який містить різні використовувані рішення та інструменти. CDK дозволяє створювати власні розширені компоненти, роботу з жестами, клавіатурою, розташуванням елементів та іншими розширеними можливостями.

3.2.2 TypeScript

TypeScript є статично типізованою мовою програмування, яка розширює JavaScript, додаючи до нього типи та нові функції. Вона розроблена компанією Microsoft і є одним з основних інструментів для розробки веб-додатків та великих проектів.

TypeScript дозволяє оголошувати типи для змінних, параметрів функцій, об'єктів тощо. Це допомагає виявляти помилки на етапі розробки, полегшує рефакторинг коду і забезпечує більшу надійність програми. У TypeScript підтримуються всі можливості JavaScript, а також додані нові функції, такі як класи, модулі, інтерфейси, зворотні виклики (callbacks) тощо. Це дозволяє писати більш структурований та організований код.

Також TypeScript надає розширений інструментарій розробника, включаючи автодоповнення коду, перевірку синтаксису, рефакторинг, навігацію по коду. Це полегшує розробку, забезпечує більшу продуктивність та допомагає збільшити якість коду.

3.2.3 Chart.Js

Chart.js - це бібліотека візуалізації даних, яка дозволяє створювати графіки та діаграми на основі HTML та JavaScript на веб-сторінках [9]. Chart.js має простий API, що дозволяє легко створювати та налаштовувати графіки.

Бібліотека підтримує лінійні, кругові, стовпчасті, гістограми, кольорові карти, а також змішані діаграми. API дає можливість налаштовувати різні аспекти графіків, включаючи кольори, шрифти, маркери, легенди та інші елементи.

Графіки, створені з використанням Chart.js, автоматично адаптуються до розмірів HTML контейнера, що дозволяє їх коректне відображення на різних пристроях та екранах.

Chart.js надає можливість взаємодії з графіками, через наведення мишки, натиску, перегляду підказок, що дозволяє користувачам отримувати додаткову інформацію про дані під час взаємодії з графіком.

Бібліотека підтримується усіма платформами, які можуть інтерпретувати JavaScript. Таким чином, вона дозволяє використовувати графіки в мобільних додатках або веб-сторінках, що переглядаються на мобільних пристроях.

Chart.js є має відкритий вихідний код і може бути легко інтегрований в проекти, що використовують JavaScript або TypeScript. На офіційному сайті бібліотеки доступна детальна документація, що включає приклади використання і описом параметрів кожного графіку.

chartjs-plugin-zoom є доповненням (плагіном) для Chart.js, яке надає можливість збільшувати та переміщуватися графіками для більш детального аналізу даних.

Плагін дозволяє користувачам збільшувати або зменшувати масштаб графіка за допомогою масштабування осей X та Y. Це дає можливість зосередитися на конкретних ділянках графіка та розглядати їх більш детально. Крім збільшення, плагін дозволяє користувачам переміщувати графік по осі X або Y, щоб відображати

різні частини даних без зміни масштабу. Область збільшення можна налаштувати, обмеживши користувача вибором певного діапазону даних.

chartjs-plugin-zoom додає інтерактивність до графіків Chart.js, і є особливо корисним для великих обсягів даних, де детальний аналіз вузьких діапазонів є важливим.

3.3 Середовища розробки

3.3.1 Visual Studio 2022

Для розробки серверної частини було використано середовище Visual Studio 2023. Visual Studio є інтегрованим середовищем розробки (IDE) від Microsoft, яке надає розробникам зручні інструменти для створення різноманітних програмних продуктів. Visual Studio підтримує різні мови програмування, включаючи C#, Visual Basic, C++, F#, JavaScript, TypeScript та Python.

Visual Studio підтримує розробку різних типів додатків, включаючи десктопні програми, веб-додатки, мобільні додатки, хмарні додатки та ігри.

Однією із основних переваг Visual Studio є IntelliCode. Він надає покращену функціональність IntelliSense за допомогою штучного інтелекту. IntelliCode використовує машинне навчання для аналізування шаблонів коду з великого обсягу публічних репозиторіїв, щоб виявляти типові шаблони та рекомендувати найбільш ймовірні продовження коду. IntelliCode також може підкреслювати рекомендовані варіанти коду, щоб виділити їх у списку доступних доповнень IntelliSense. Це допомагає вибору правильної підказки з першого разу, що збільшує продуктивність розробника та допомагає уникнути потенційних помилок. Загалом, IntelliCode розширює можливості IntelliSense, надаючи більш інтелектуальні та контекстуальні рекомендації під час написання коду, що значно збільшує швидкість розробки.

Visual Studio є розширюваним середовищем, яке дозволяє встановлювати розширення та плагіни для розширення функціональності.

Також, Visual Studio має інтеграцію із NuGet - пакетним менеджером для платформи .NET, який дозволяє легко встановлювати, оновлювати та керувати залежностями в своїх проектах через інтерфейс користувача.

3.3.2 Visual Studio Code

Для розробки клієнтської частини було використано середовище Visual Studio Code. Visual Studio Code (VS Code) - це безкоштовне, легке середовище розробки, що так само розроблене компанією Microsoft і, так само, як Visual Studio 2022 підтримує усі популярні мови програмування.

VS Code має масивну екосистему розширень, які дозволяють розробникам налаштовувати та розширювати функціональність редактора під їхні потреби. Розширення доступні для підсвічування синтаксису, автоматичного завершення коду, налагодження та інтеграції з різними сервісами.

Так само, як і Visual Studio, VS Code має вбудовану інтеграцію системи контролю версій Git, що дозволяє працювати з репозиторіями Git, за допомогою інтерфейсу користувача.

VS Code має вбудований термінал, який дозволяє виконувати команди в контексті проекту, встановлювати залежності, запускати тестування та інші операції без необхідності відкривати зовнішні термінали. Оскільки для ефективної роботи із фреймворком Angular необхідне постійне використання інтерфейсу командного рядку Angular CLI, наявність зручного вбудованого терміналу є дуже важливою для середовища розробки.

Наявна у VS Code також і підтримка Emmet - інструменту для розширення можливостей роботи із розмітками веб-сторінок. Він надає короткі абревіатури, які можна розгортати в розширений HTML або CSS код.

Зручним є також те, що VS Code підтримується на різних операційних системах, таких як Windows, macOS та Linux, і навіть доступний у браузері (з певними обмеженнями функціоналу).

3.3.3 SQL Server Management Studio

SQL Server Management Studio є інтегрованим середовищем для управління базами даних Microsoft SQL Server. Воно надає розширені засоби для адміністрування баз даних, розробки SQL-запитів, налагодження, виконання скриптів, створення та управління об'єктами бази даних, налаштування безпеки.

SSMS дозволяє підключатися до локальних або віддалених баз даних SQL Server та управляти ними, а також має вбудований візуальний конструктор, який дозволяє створювати таблиці, зв'язки, процедури, функції та інші об'єкти бази даних безпосередньо у графічному інтерфейсі.

Висновки до розділу 3

Розглянуто та проаналізовано особливості обраних засобів розробки. У якості засобів для розробки серверної частини обрано платформу .NET з застосуванням мови C# та SQL Server у якості системи управління базами даних. Це дозволить швидко працювати з базою даних за допомогою бібліотеки Entity Framework. Клієнтська частина виконується з використанням фреймворку для розробки односторінкових веб-додатків Angular та бібліотеки візуалізації даних Chart.Js. Також було проаналізовано особливості застосованих середовищ розробки.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Програмна реалізація передбачає клієнт, реалізований з використанням Angular, що відображає сторінки та компоненти, з якими користувач взаємодіє в браузері. Компоненти Angular обробляють події користувача та взаємодіють з сервером, який побудований на платформі ASP.NET. Сервер виконує бізнес-логіку, отримує та обробляє дані з бази даних MS SQL. Взаємодія між клієнтом і сервером здійснюється за допомогою HTTP-запитів та відповідей, що передаються між ними у форматі JSON. Ця взаємодія дозволяє клієнту отримувати та відображати дані з сервера, а також надсилати дані для збереження та оновлення у базі даних.

4.1 Архітектура системи

У якості архітектури системи було обрано підхід SPA.

SPA (Single-Page Application) - це архітектурний підхід, в якому веб-додаток складається з однієї HTML-сторінки, а весь необхідний контент та логіка завантажуються динамічно за допомогою JavaScript. Взаємодія з користувачем відбувається без перезавантаження сторінки, що робить додаток більш швидкодійним та зручним для користувача. Схема SPA архітектури зображена на рисунку 4.1.

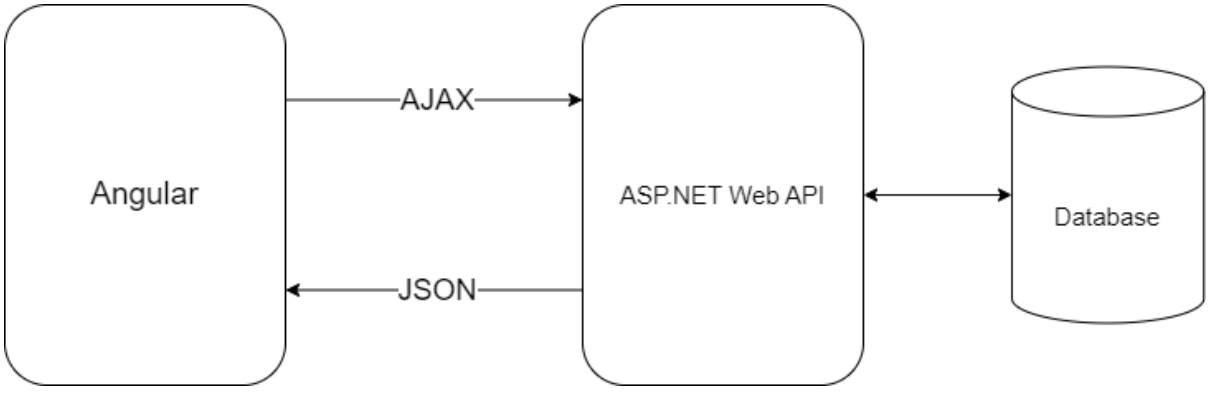


Рисунок 4.1 – SPA архітектура

SPA архітектуру для цього додатку було обрано з причини того, що такий підхід дозволяє використовувати один інтерфейс користувача для відображення даних із різних джерел. В свою чергу, API дозволяє обробляти інформацію із різних джерел і передавати у декілька різних клієнтів. Таким чином, додаток виконує функцію сервісу. Подібний підхід дозволяє як використовувати даний додаток у роботі інших додатків, так і розширювати функціональні можливості нашого додатку за рахунок інших додатків.

У SPA архітектурі клієнтська сторона виконує більшу частину роботи, включаючи відображення даних, обробку подій та навігацію. В розроблюваному додатку це досягається за допомогою фреймворку Angular. Коли користувач взаємодіє з додатком, клієнтська сторона відправляє запити на сервер для отримання необхідних даних, а сервер відповідає збіркою даних у форматі JSON або XML. Після цього клієнтська сторона відповідальна за відображення цих даних на сторінці.

Одна з переваг SPA архітектури - це більш плавна та швидка взаємодія з додатком, оскільки немає потреби в перезавантаженні сторінки при кожній дії користувача. Крім того, вона спрощує розробку, оскільки можна створювати повторно використовувані компоненти та керувати станом додатка з більшою легкістю.

SPA архітектура включає в себе реалізацію серверної частини з застосуванням RESTful архітектури.

RESTful (Representational State Transfer) - це архітектурний стиль, який використовується при розробці веб-сервісів. RESTful базується на наборі принципів та обмежень, які спрямовані на створення, масштабованих сервісів.

Основні обмеження RESTful включають:

1. кешування - сервер може позначати відповіді як кешовані, що дозволяє клієнтам кешувати ці відповіді та повторно використовувати їх, щоб зменшити навантаження на сервер;

2. відсутність стану - кожен запит клієнта до сервера повинен містити всю необхідну інформацію для обробки цього запиту, а сервер не повинен зберігати жодного стану про клієнта між запитами. Це дозволяє зробити систему більш масштабованою та зменшити навантаження на сервер;

3. однорідний інтерфейс - RESTful використовує стандартний набір HTTP-методів, таких як GET, POST, PUT та DELETE, для взаємодії з ресурсами. Це спрощує розуміння та використання API, робить його більш зрозумілим та передбачуваним;

4. шари абстракції - система може бути складена з різних рівнів (шарів), при цьому кожен рівень не знає про існування інших рівнів, окрім того, з яким безпосередньо взаємодіє. Це дозволяє створювати модульні та масштабовані системи.

Для розроблюваного застосунку RESTful архітектура реалізується за рахунок використання фреймворку ASP.NET Web API, в якому за замовчуванням враховані основні обмеження.

Використання RESTful архітектури спрощує розробку та підтримку веб-сервісів, робить їх більш масштабованими та незалежними від платформи або технологій реалізації клієнтського застосунку.

4.2 Модель даних

Існує два основних підходи до програмної реалізації моделі даних - Code First та Database First. Розглянемо обидва підходи у контексті фреймворку ASP.NET.

Code First – підхід, згідно якому розробка починається з опису класів сутностей та взаємозв'язків між ними. Після визначення моделі даних за допомогою класів, Entity Framework автоматично генерує відповідну базу даних на основі цих класів. У Code First використовується механізм міграцій, що дозволяє здійснювати зміни в структурі бази даних під час розвитку додатку.

Database First – підхід, в якому розробка починається з наявної бази даних. Entity Framework використовує об'єктно-реляційний мапінг, щоб автоматично згенерувати модель даних (класи сутностей) на основі структури бази даних. За допомогою Database First можна використовувати побудовану модель даних для взаємодії з базою даних, виконувати запити та виконувати зміни в даних.

Відповідно до наявного набору даних із датчиків НБК, для обліку вимірів ми використовуватимемо лише одну сутність. Таким чином, побудувавши базу даних на основі набору даних ми отримаємо одну таблицю. При цьому, необхідно врахувати необхідність реалізації механізму автентифікації та авторизації, що в ASP.NET реалізований за допомогою фреймворку Identity [10]. Специфіка Identity полягає у тому, що даний фреймворк використовує вбудовані класи (сутності), для збереження яких потрібно мати ряд таблиць в базі даних. У випадку використання підходу Database First нам необхідно власноруч написати SQL скрипт для створення усіх необхідних таблиць. В свою чергу, при підході Code First ми лише створюємо сутність користувача додатку – ApplicationUser, що наслідує вбудований клас IdentityUser та вказуємо в сутності користувача потрібні нам поля, відсутні в батьківському класі. У ході виконання міграцій EF створює в базі даних усі необхідні таблиці, а в таблицю користувачів додає не тільки вказані нами поля, а і поля, що використовуються в механізмі авторизації. Виходячи з цього, правильніше обрати підхід Code First для створення моделі даних.

Після створення усіх необхідних сутностей, створимо контекст бази даних. Контекст бази даних являє собою клас, що відповідає за взаємодію з базою даних і виконання операцій збереження, отримання, оновлення та видалення даних. Він включає набір полів - об’єктів класу DbSet, які представляють відповідність сутностей в додатку та таблиць в базі даних. Якщо назва таблиці та назва сутності в нашому проекті співпадають, то явно створювати поле у контексті бази даних не є необхідним. Після виконання міграцій в базу даних, можемо завантажити тестовий набір даних, користуючись вбудованими інструментами імпорту даних. Конструювати запити до бази даних можемо за допомогою LINQ.

4.3 Розробка API

В розділі 4.1 ми визначили що для обробки запитів сервером нам потрібно реалізувати API. В ASP.NET немає необхідності явно описувати маршрутизацію, якщо дотримуватися структури запиту “api/[controller]/[action]”, де "[controller]" вказує на те, що ім'я контролера буде підставлено у місце "[controller]" в маршруті. Наприклад, у нас є контролер з назвою "AdminController", то "[controller]" буде замінено на "Admin". "[action]" вказує на те, що ім'я методу дії буде підставлено у місце "[action]" в маршруті. Маємо метод дії з назвою "GetUsers", то "[action]" буде замінено на "GetUsers".

Для отримання даних із БД в клас контролера додається контекст даних за допомогою механізму впровадження залежностей.

Впровадження залежностей (dependency injection, DI) - це концепція проектування програмного забезпечення, яка полягає в тому, щоб залежності об'єкту були впроваджені в нього ззовні, замість того, щоб він сам створював або залежав від них. У традиційному підході, коли об'єкт потребує використання іншого об'єкту або сервісу, він сам створює його екземпляр або залежить від конкретної реалізації цього об'єкту. Це може призводити до жорсткого зв'язку між компонентами та ускладнювати тестування та підтримку коду. Завдяки впровадженню залежностей, залежності об'єкту передаються через конструктор, методи або властивості від зовнішнього джерела (наприклад, контейнеру залежностей або фабричному класу). Об'єкт не має знати, як саме створюються або конфігуруються його залежності - він просто використовує їх.

В методах контролеру ми будуємо LINQ запити, які трансформуються в SQL запити за допомогою EF під час виклику метода контролера, тобто під час HTTP запиту.

4.3.1 Контролер для відображення графіків

Для побудови графіків визначимо контролер ChartsController. В ньому визначено У контролері визначено дві дії GetMeasurements та GetSpeedDirection, які відповідають на HTTP GET-запити:

- GetMeasurements отримує параметри startTime та endTime, перетворює їх у тип DateTime і отримує вимірювання з бази даних за цей період. Результат повертається у форматі JSON;
- GetSpeedDirection отримує параметри startTime та endTime, перетворює їх у тип DateTime і отримує дані про швидкість та напрямок вітру з бази даних за цей період. Далі виконується обробка даних, розрахунок середньої швидкості вітру для кожного напрямку, і результат повертається у форматі JSON.

4.3.2 Контролер для відображення таблиці даних вимірювань

Для побудови таблиці з даними про вимірювання використовується контролер MeasurementsController. Контролер надає

У контролері визначена одна дія GetMeasurements, яка відповідає на HTTP GET-запити до шляху /api/Measurements.

GetMeasurements отримує параметри:

- pageIndex – індекс сторінки таблиці;
- pageSize – кількість рядків на одній сторінці таблиці;
- sortColumn – колонка, за якою потрібно відсортувати дані в таблиці;
- sortOrder – порядок сортування.

Ці параметри використовуються для налаштування пагінації (механізму розділення списку або таблиці даних на сторінки) і сортування результатів. За замовчуванням, pageIndex та pageSize встановлені на значення 0 та 10 відповідно.

Далі, виконується перевірка, чи існують вимірювання в базі даних. Якщо вони відсутні, повертається HTTP відповідь зі статусом "Not Found".

Якщо вимірювання існують, викликається статичний метод CreateAsync класу TableApiResult<Measurement>, який отримує вимірювання з бази даних і створює об'єкт TableApiResult, який містить результати для пагінації та сортування. Результат повертається у форматі JSON.

Клас TableApiResult має конструктор, який приймає параметри data, count, pageIndex, pageSize, sortColumn та sortOrder. Ці параметри використовуються для ініціалізації властивостей об'єкта TableApiResult<T>.

Клас також має статичний метод CreateAsync, який приймає параметри source, pageIndex, pageSize, sortColumn та sortOrder. Метод виконує підрахунок кількості елементів count та застосовує сортування до source, якщо передані значення sortColumn та sortOrder. Далі метод виконує пагінацію даних та повертає новий об'єкт TableApiResult<T>.

Клас містить метод IsValidProperty, який перевіряє, чи існує властивість з заданою назвою propertyName в класі T. Метод використовує рефлексію для пошуку властивості та повертає true, якщо властивість знайдена, або false, якщо властивість не знайдена. Цей метод використовується для захисту бази даних від SQL ін’єкцій – типу атаки на веб-додатки, при якій використовується вразливість додатку, щоб вставити у запит свій SQL-код, який може виконатися базою даних. Така атака може призвести до витоку чутливої інформації, зміни або видалення даних, або навіть до повного компрометації системи.

Клас TableApiResult<T> є узагальненим класом, тобто він може використовуватися для побудови таблиці із будь-яким набором полів.

4.3.3 Автентифікація та авторизація

Контролер AccountController, який відповідає за обробку запитів, пов'язаних з аутентифікацією та управлінням обліковими записами.

У контролері окрім контексту даних використовуються наступні залежності:

- UserManager<ApplicationUser> - об'єкт для управління користувачами, заснований на класі ApplicationUser;
- JwtHandler - об'єкт для обробки JWT-токенів.

Конструктор контролера приймає ці залежності в якості параметрів і зберігає їх у відповідних приватних полях для подальшого використання.

У контролері є метод Login який відповідає на POST-запит з шляхом "api/Account/Login". Цей метод обробляє запит на вхід користувача і проводить аутентифікацію. Він отримує об'єкт LoginRequest, який містить дані для входу (електронну пошту та пароль). Метод перевіряє введені дані, шукає користувача в базі даних і перевіряє правильність паролю. Якщо аутентифікація успішна, метод генерує JWT-токен за допомогою об'єкта JwtHandler і повертає успішний результат з отриманим токеном. У випадку невірних облікових даних, метод повертає відповідну помилку з повідомленням.

Цей компонент дозволяє здійснювати аутентифікацію користувачів і отримувати JWT-токени для подальшого доступу до захищених ресурсів.

JWT (JSON Web Token) - це стандарт для передачі безпечної токенізованої інформації у веб-серверних та мобільних додатках. Він представляє собою компактний та безпечний спосіб обміну інформацією між сторонами у вигляді об'єкта JSON.

JWT складається з трьох основних частин: заголовка (header), корисного навантаження (payload) і підпису (signature). Кожна частина закодована в Base64 і розділена крапками. Розглянемо кожну частину JWT окремо:

- заголовок містить метадані про тип токена і тип шифрування, яке використовується для підпису токена;
- корисне навантаження містить дані, які передаються у токені. Воно може містити будь-яку корисну інформацію, наприклад ідентифікатор користувача, роль, термін дії автентифікації тощо;
- підпис використовується для перевірки цілісності токена. Він генерується за допомогою секретного ключа або приватного ключа, і лише сторона,

яка володіє відповідним публічним ключем, може перевірити підпис і довіряти токену.

JWT використовується для забезпечення механізму автентифікації та авторизації в розподілених системах, де немає необхідності зберігати стан аутентифікації на сервері. Клієнт отримує JWT після успішної аутентифікації і додає його до заголовка або запиту для доступу до захищених ресурсів на сервері. Сервер перевіряє підпис і достовірність токена, і якщо все вірно, надає доступ користувачеві до запитуваного ресурсу.

4.4 Клієнтський додаток

Основними структурними додатку, що розробляється за допомогою Angular є модулі, компоненти і сервіси.

Модуль є основною будівельною одиницею додатку Angular. Він групує компоненти, директиви, сервіси та імпортовані бібліотеки, пов'язані за спільною функціональністю. Модулі допомагають організувати код додатку та підтримувати розділення залежності.

Компоненти використовуються для створення користувацького інтерфейсу. Вони визначають розмітку шаблону (HTML), логіку поведінки (JavaScript/TypeScript) та стилі компонента. Компоненти можуть включати дочірні компоненти та взаємодіяти з їхніми даними за допомогою властивостей та подій.

Сервіси використовуються для організації спільної логіки та функціональності, яка не пов'язана з конкретним компонентом. Вони надаються в модулі та можуть бути використані компонентами для обробки даних, взаємодії з сервером, кешування тощо.

Основним модулем Angular є AppModule. Він створюється при створенні проекту разом із AppComponent - основним компонентом. Для налаштування маршрутизації в клієнтському додатку використовується AppRoutingModule. Він містить конфігурацію маршрутів і визначає, який компонент буде відображатися

для кожного маршруту. Для відображення даних реалізовуємо компоненти DashboardInfo та MeasurementTable.

4.4.1 Інформаційна панель

Інформаційна панель, або дешборд реалізована у компоненті DashboardInfoComponent.

Коли компонент ініціалізується, метод ngOnInit встановлює початкові значення для властивостей компонента та виконує ініціалізацію необхідних залежностей.

Після завантаження шаблону, метод ngAfterViewInit викликається. Він виконує рендеринг графіків на інформаційній панелі, використовуючи функції RenderChartWindSpeed, RenderRadarSpeedDiretion, RenderChartDrop, RenderChartSpend, RenderChartPressureBC і RenderChartPressure. Ці функції отримують початкові дані з порожніх масивів, оновлюють графіки та відображають їх на панелі.

Компонент також має функцію updateCharts, яка викликається при оновленні даних на панелі. Ця функція очищає масиви даних графіків, отримує нові дані з сервера за допомогою функцій transformData і transformSpeedDirection, а потім викликає функції рендерингу графіків для оновлення відображення.

transformData і transformSpeedDirection є функціями, які обробляють дані з сервера. transformData отримує дані про показники (наприклад, температуру, вологість) за певний період часу і заповнює масиви даних для кожного показника. transformSpeedDirection отримує дані про швидкість та напрямок вітру, обробляє їх і створює окремі масиви даних для швидкості та напрямку вітру.

ChartDataService є сервісом, який надає функціональність для отримання даних для графіків. Він використовується для отримання вимірювань та швидкості/напрямку.

У ChartDataService визначені дві властивості: data та speedDirection. data є масивом об'єктів ChartData, який містить дані для графіків, а speedDirection є масивом об'єктів SpeedDirection, що містить дані про швидкість та напрямок.

Конструктор ChartDataService ін'єктує http з типом HttpClient, що дозволяє здійснювати HTTP-запити до сервера.

Метод getMeasurements(start: Date | null, end: Date | null) використовується для отримання вимірювань за заданий період. Він створює об'єкт HttpParams, до якого додаються параметри startTime та endTime, що вказують на початок та кінець періоду. Потім відбувається HTTP-запит методом get до з передачею параметрів. Результатом є об’єкт класу Observable, що містить отримані дані вимірювань і перетворюється уже безпосередньо в компоненті DashboardInfoComponent.

Метод getSpeedDirection(start: Date | null, end: Date | null) використовується для отримання даних про швидкість та напрямок за заданий період. Він також створює об'єкт HttpParams з параметрами startTime та endTime, а потім здійснює HTTP-запит методом get до серверу з передачею параметрів. Аналогічно до getMeasurements , результатом є об’єкт класу Observable.

Таким чином, ChartDataService надає можливість отримати дані для графіків, здійснюючи відповідні HTTP-запити та оброблюючи отримані результати.

4.4.2 Таблиця з даними вимірювань

MeasurementsTableComponent є компонентом, відповідальним за відображення таблиці з вимірами.

Компонент має властивість measurements, яка є об'єктом MatTableDataSource<Measurement>. Цей об'єкт використовується для збереження та відображення даних в таблиці.

У компоненті також визначені різні властивості, які використовуються для налаштування таблиці. Наприклад, defaultPageIndex, defaultPageSize, defaultSortColumn та defaultSortOrder встановлюють значення за замовчуванням для

індексу сторінки, розміру сторінки, стовпця сортування та порядку сортування відповідно. displayedColumns - це масив рядків, який містить імена стовпців, які будуть відображатись у таблиці.

Компонент також має декоратор ViewChild, який використовується для отримання доступу до екземплярів MatPaginator та MatSort з шаблону.

У методі ngOnInit викликається функція loadData, яка завантажує дані для таблиці.

Метод loadData створює об'єкт PageEvent, встановлює значення індексу сторінки та розміру сторінки за замовчуванням, а потім викликає функцію getData з цим об'єктом PageEvent.

Метод getData виконує HTTP-запит до сервера, використовуючи HttpClient, для отримання даних вимірів. Він передає параметри запиту, такі як індекс сторінки, розмір сторінки, стовпець сортування та порядок сортування. Після отримання результату запиту, він оновлює властивості paginator та measurements для налаштування пагінації та відображення даних у таблиці.

Отже, загальна мета MeasurementsTableComponent полягає в завантаженні даних вимірів з сервера та їх відображенні у вигляді таблиці з можливістю сортування та пагінації.

4.4.3 Автентифікація та авторизація

Трьома основними елементами в реалізації аутентифікації в клієнтському додатку є AuthGuard, AuthService та AuthInterceptor.

AuthGuard є сервісом, який реалізує інтерфейс CanActivate з Angular Router і використовується для захисту маршрутів від неавторизованого доступу.

У сервісі визначені дві властивості: authService і router. authService є екземпляром AuthService, який використовується для перевірки автентифікації користувача. router є екземпляром Router, який використовується для перенаправлення користувача на інші маршрути.

Метод canActivate є обов'язковим методом, який має бути реалізованим в AuthGuard. Цей метод приймає два параметри: route (об'єкт ActivatedRouteSnapshot), який містить інформацію про активний маршрут, і state (об'єкт RouterStateSnapshot), який містить інформацію про поточний стан маршрутизатора.

У методі canActivate спочатку перевіряється, чи користувач автентифікований шляхом виклику методу isAuthenticated() у authService. Якщо користувач має валідну автентифікацію, метод повертає true, що означає, що користувач має доступ до маршруту.

Якщо користувач не має валідної автентифікації, метод виконує перенаправлення на маршрут '/login' з параметром returnUrl, який містить URL поточного стану. Це забезпечує коректне перенаправлення користувача на бажаний маршрут після автентифікації.

У разі неавтентифікованого доступу метод повертає false, що вказує на відмову в доступі до маршруту.

AuthService - це сервіс, який забезпечує функціональність для автентифікації користувачів у додатку.

Метод isAuthenticated() використовується для перевірки, чи користувач має дійсну автентифікацію. Він повертає значення true, якщо користувач автентифікований, або false, якщо він неавтентифікований.

Метод getToken() дозволяє отримати токен автентифікації користувача. Він перевіряє наявність токена у локальному сховищі браузера і повертає його значення. Якщо токен відсутній, метод повертає null.

Метод init() викликається для ініціалізації сервісу. Він перевіряє, чи є автентифікація користувача, і встановлює відповідний статус автентифікації.

Метод login(item: LoginRequest) виконує процес входу користувача за допомогою наданих даних LoginRequest. Він надсилає запит на сервер для автентифікації та повертає результат у вигляді Observable.

Метод logout() виконує вихід користувача з системи. Він видаляє токен автентифікації та оновлює статус автентифікації.

Приватний метод setAuthStatus(isAuthenticated) використовується для встановлення статусу автентифікації відповідно до переданого значення isAuthenticated. Він оновлює внутрішній стан сервісу та повідомляє підписників про зміну статусу автентифікації.

AuthInterceptor є класом, який реалізує інтерфейс HttpInterceptor з Angular для перехоплення та обробки HTTP-запитів. Його основна мета - забезпечити автоматичну обробку автентифікації під час взаємодії з сервером. Основні засоби, які використовуються в AuthInterceptor, включають AuthService, Router та різні класи з @angular/common/http.

У конструкторі AuthInterceptor ін'єктується authService та router. authService є екземпляром AuthService, необхідним для отримання токену автентифікації, тоді як router є екземпляром Router, використовується для перенаправлення користувача на інші маршрути.

Метод intercept(req: HttpRequest<any>, next: HttpHandler) є обов'язковим методом, який необхідно реалізувати в AuthInterceptor. Цей метод приймає HTTP-запит req та об'єкт next з типом HttpHandler, який дозволяє продовжити обробку запиту ланцюжком інших перехоплювачів.

У методі intercept спочатку отримується токен автентифікації з authService за допомогою методу getToken(). Якщо токен існує, то створюється новий клонований запит req, до якого додається заголовок Authorization з використанням токену. Це дозволяє передавати токен автентифікації з кожним запитом, що вимагає автентифікації.

Після цього метод викликає next.handle(req) для передачі запиту до наступного обробника у ланцюжку. Результат цього обробника - Observable<HttpEvent<any>>, який ми отримуємо через pipe.

У pipe метод next.handle(req) обробляється за допомогою catchError, який перехоплює будь-які помилки, що виникли під час обробки запиту. Якщо помилка є екземпляром HttpErrorResponse та має статус 401 (Неавторизований), то виконується вихід користувача шляхом виклику методу logout() у authService та

перенаправлення користувача на сторінку входу (login) за допомогою router.navigate(['login']).

На завершення, throwError(error) повертає помилку до підписника, що дозволяє обробити помилку на вищому рівні обробки запиту.

4.5 Симуляція отримання даних в режимі реального часу

В ході роботи було проведено дослідження з можливостей симуляції отримання даних з НБК в режимі реального часу. Спробою було використати перевірку бази даних на наявність змін та відправку оновлених даних до клієнта через SignalR. Однак, такий підхід виявився хибним з кількох причин.

Постійна перевірка бази даних на наявність змін призводить до зайвого навантаження на базу даних і знижує швидкість обробки запитів. Крім того, затримка в оновленні даних на клієнтському боці може призводити до відображення застарілих даних.

Замість цього, у випадку подальшої роботи над імплементацією такого функціоналу, рекомендується створити окремий додаток, який буде надсилати симуляційні дані в API серверу для запису до бази даних. Одночасно, сервер може використовувати SignalR для передачі цих оновлених даних до клієнта. Такий підхід зменшить навантаження на базу даних і забезпечить синхронізацію даних між сервером і клієнтом у реальному часі, зменшуючи використання мережевого трафіку.

Висновки до розділу 4

Для розробки застосунку обрано архітектурний підхід SPA. Серверна частина виконана з використанням підходу RESTful API. Розглянуто різні підходи до реалізації моделі даних та обрано підхід Code First, за якого БД генерується на основі існуючого коду серверної частини. Реалізовано контролери, що обробляють

HTTP запити від клієнта. Впроваджено механізм авторизації та автентифікації користувачів. Створено користувацький інтерфейс, який виконує задачу з візуалізації історичних даних натурних вимірювань у вигляді інтерактивних графіків з можливістю фільтрації даних за періодом часу. Реалізовано також відображення даних у вигляді таблиці із можливістю сортування даних за будь яким з полів. При розробці враховано необхідність у подальшій підтримці додатку, розширення його функціоналу та інтеграції із іншими сервісами. Також досліджено можлвість отримання даних в реальному часі та наведено рекомендації щодо можливого підходу до реалізації такого функціоналу у разі подальших досліджень.

5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

В цьому розділі буде описано сценарії взаємодії користувача з програмною системою, а саме, автентифікація, взаємодія з графіками та таблицями.

5.1 Автентифікація

Неавторизований користувач обов’язково потрапляє на до форми авторизації. Форма авторизації зображена на рисунку 5.1.

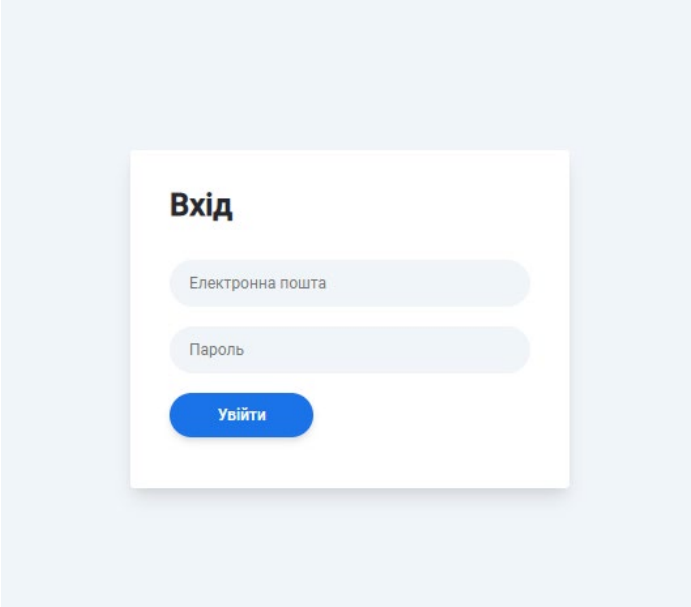


Рисунок 5.1 – Форма авторизації

Реєструватися в системі будь-який користувач не може. Додавати нових користувачів до системи може лише адміністратор.

5.2 Взаємодія з графіками

Після успішної авторизації користувач потрапляє на «дешборд» - сторінку, де відображаються графіки із даними історичних вимірювань, як зображено на рисунку 5.3.

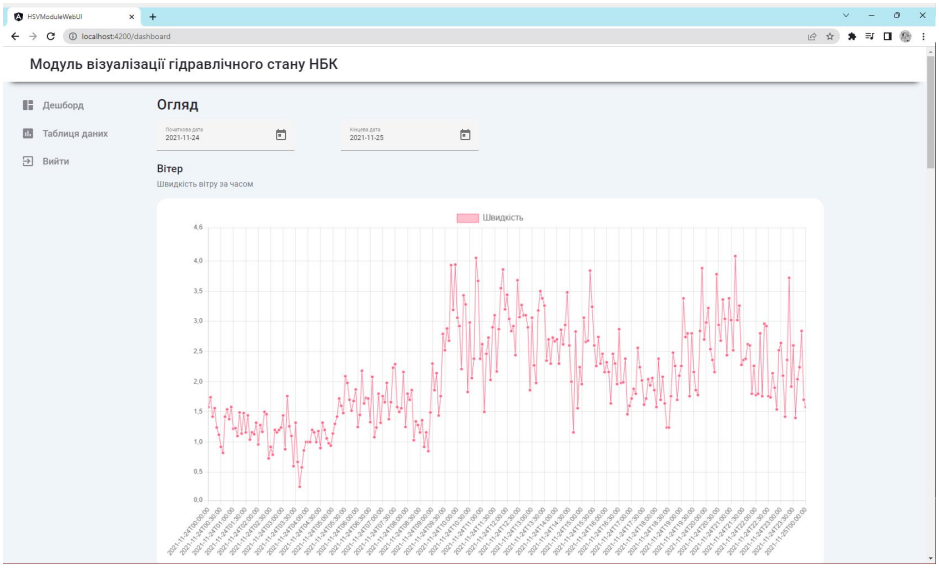


Рисунок 5.2 – Графік швидкості вітру

Користувач має можливість обирати період часу, за який будуть відображатися дані вимірювань. Вибір даних відбувається за допомогою календаря, що відкривається за натиском мишкою на іконку календаря (рис. 5.3).

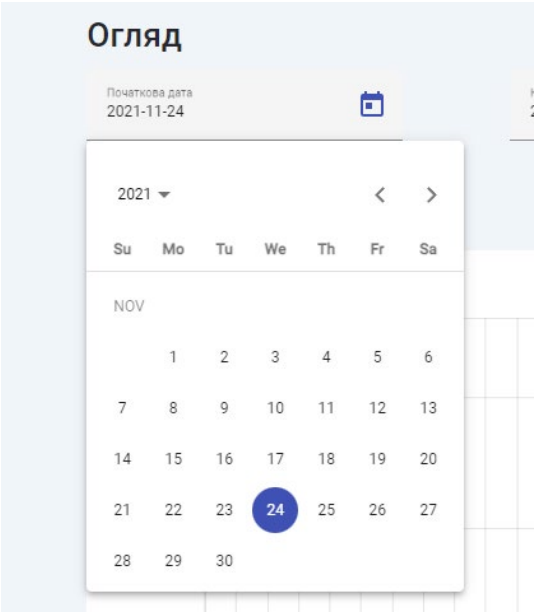


Рисунок 5.3 – Вибір дня місяця

Користувач може виконувати навігацію календарем за допомогою стрілок, перемикаючись на попередній або наступний місяць, або натиснувши на номер року у правому кутку календаря, після чого меню вибору дати зміниться на меню вибору року (рис. 5.4).

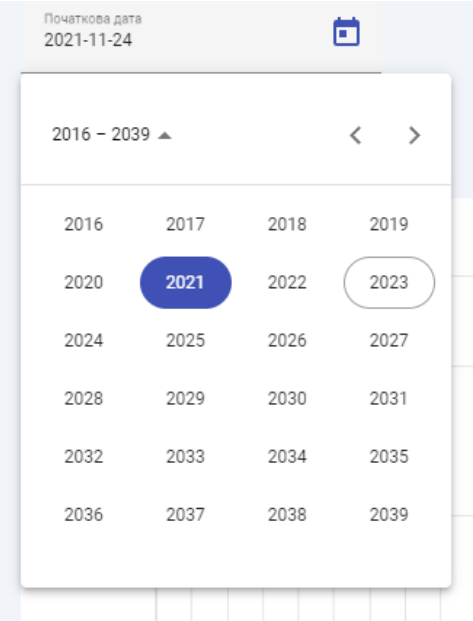


Рисунок 5.4 – Вибір року

Після вибору року відкривається меню вибору місяця (рис. 5.5).

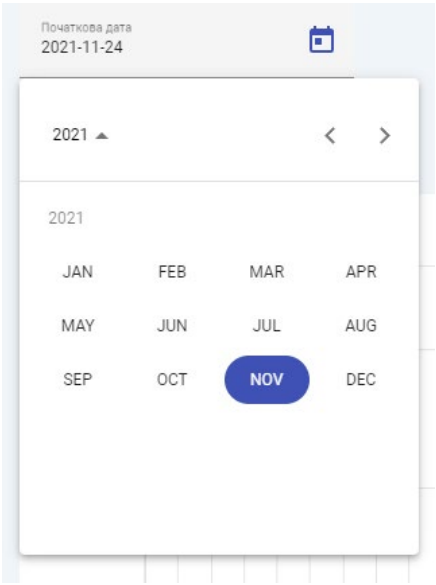


Рисунок 5.5 – Вибір місяця

Після вибору місяця відкривається меню вибору дня для вибраного місяця і року (рис. 5.3).

Користувачу не обов’язково обирати обидві дати для завантаження даних за обраний діапазон часу. Дані на сторінці оновлюються як тільки користувач обирає одну з дат.

На рисунку 5.6 можна побачити радар, що відображає середнє значення швидкості вітру для кожного з напрямків за обраний період часу.

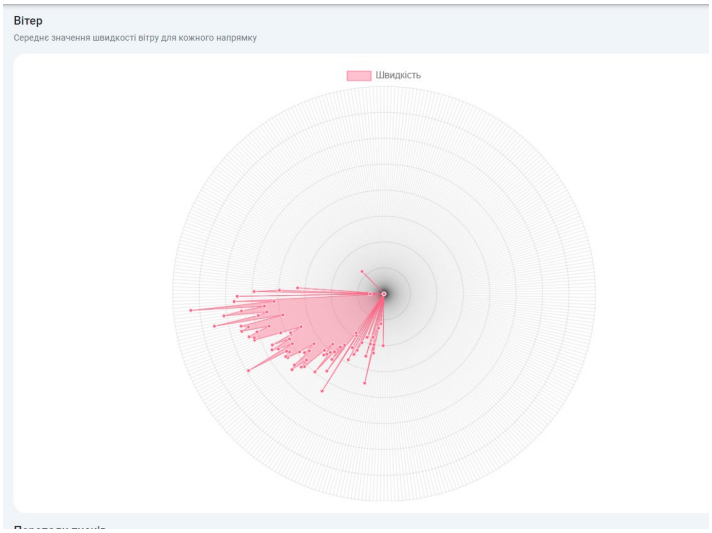


Рисунок 5.6 – Діаграма середньої швидкості вітру для кожного з напрямків

Розроблені графіки є інтерактивними та передбачають наступні дії:

- зміна масштабу графіку;
- навігація графіком;
- увімкнення та вимкнення кожного окремого графіка на площині;
- перегляд значень в кожній окремій точці.

Зміна масштабу графіку відбувається по прокручуванню колесика мишки вперед або назад. При прокручуванні мишки вперед, графік наближується, а при прокручуванні назад – віддаляється. Крім того, зміна масштабу залежить від позиції мишки на графіку, або на його осі. Якщо мишка знаходиться безпосередньо на координатній площині, то графік збільшується або зменшується відносно позиції мишки. Якщо мишка знаходиться на координатній осі, то масштаб графіка змінюється тільки за цією віссю. Крім того, передбачено обмеження, які не дають графіку зменшуватися занадто сильно. На рисунку 5.7 зображено графік перепаду тисків в оригінальному масштабі.

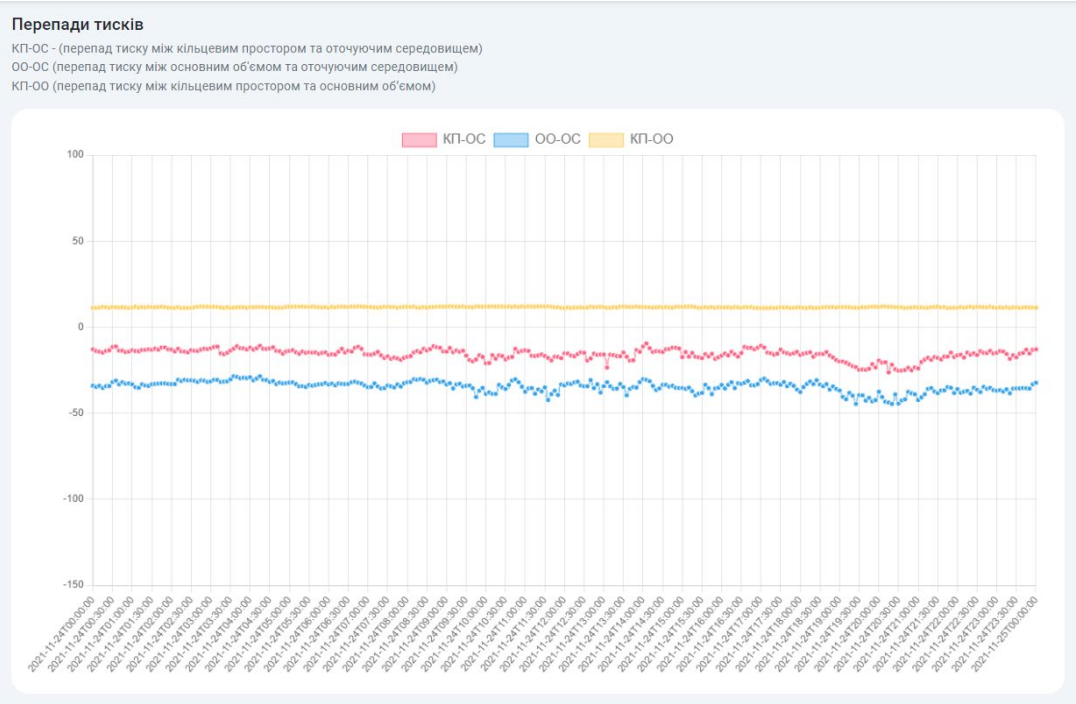


Рисунок 5.7 – Графік перепаду тисків в початковому масштабі

Зменшимо масштаб графіку до найменшого допустимого. Результат можемо побачити на рисунку 5.8.

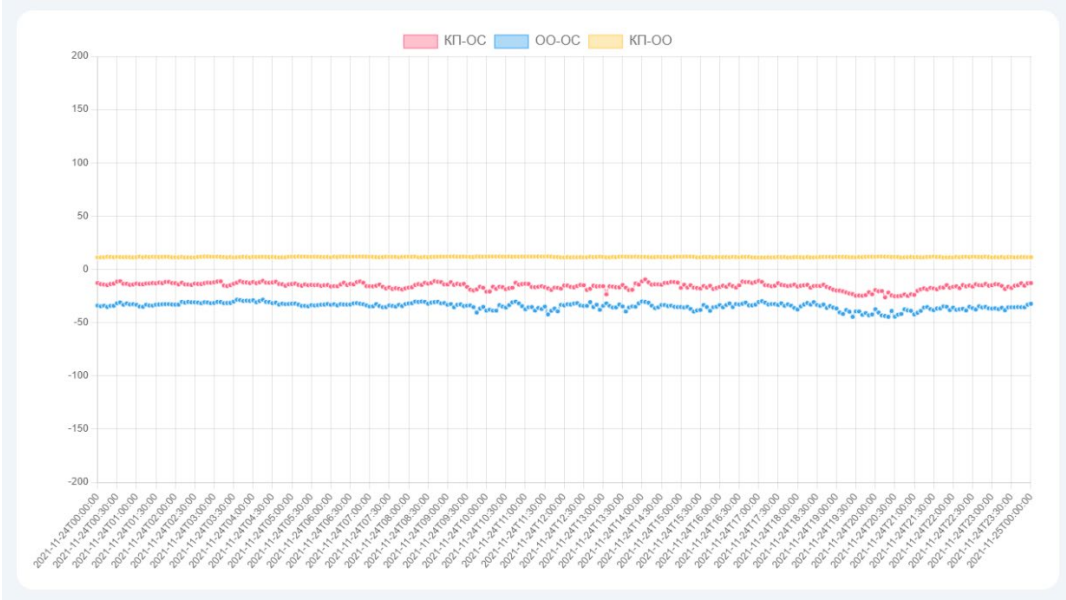


Рисунок 5.8 – Графік перепаду тисків в найменшому допустимому масштабі

Можемо спостерігати, що графік обмежений в зменшенні масштабу по осі X наявним набором даних, а по осі Y не виходить за межі значень -200 та 200.

Для порівняння збільшимо масштаб. Результат бачимо на рисунку 5.9.

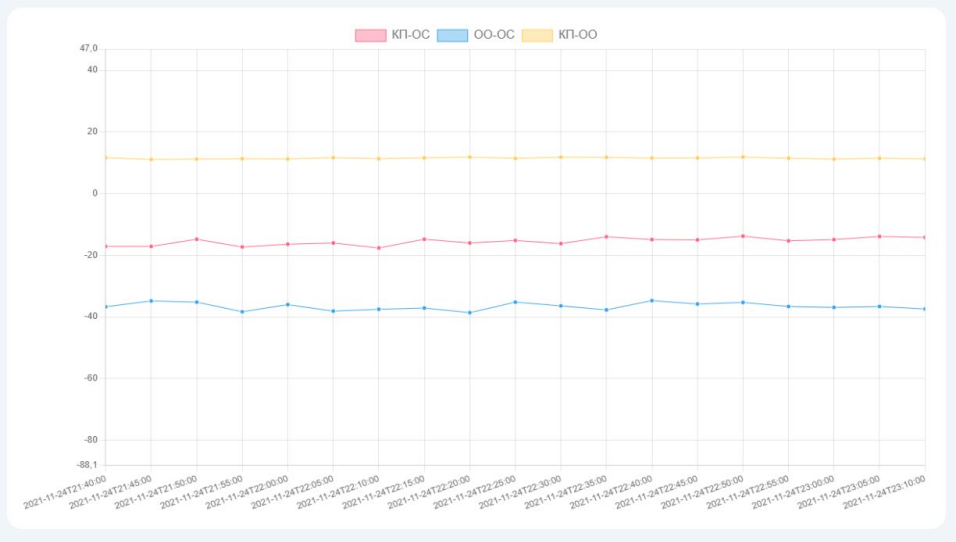


Рисунок 5.9 – Графік перепаду тисків в більшому масштабі

Навігація графіками відбувається за допомогою протягування мишки по графіку із затиснутою лівою кнопкою мишки. Для навігації актуальні ті ж обмеження на координатній площині, що і для масштабування.

Вимкнення або увімкнення окремих графіків відбувається за натиском лівої кнопки мишки по назві відповідного графіку. При цьому, назва вимкненого графіка закреслюється. Це можна побачити на рисунку 5.10.

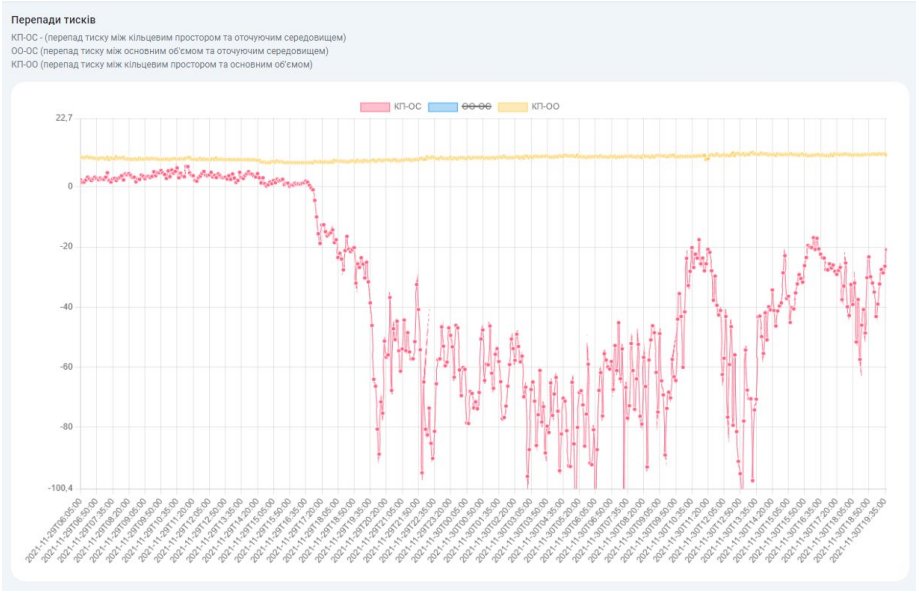


Рисунок 5.10 – Один графік вимкнено

На рисунку 5.11 зображено як виглядає відображення перегляду значення.

Рисунок 5.11 – Відображення значення в точці

Значення відображаються при наведенні курсора на точку графіку.

Для зручності порівняння значень із різних графіків на одній площині, графіки налаштовані таким чином, що при наведенні на одну точку, відображатимуться значення для усіх графіків в цій точці осі Y (рис. 5.12).

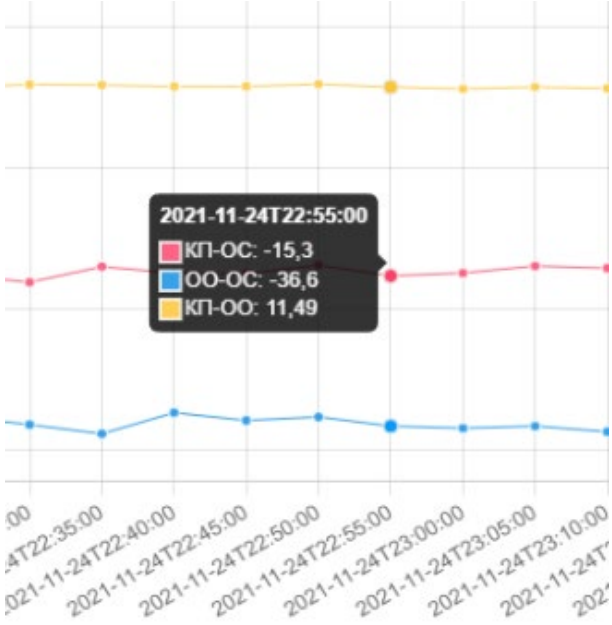
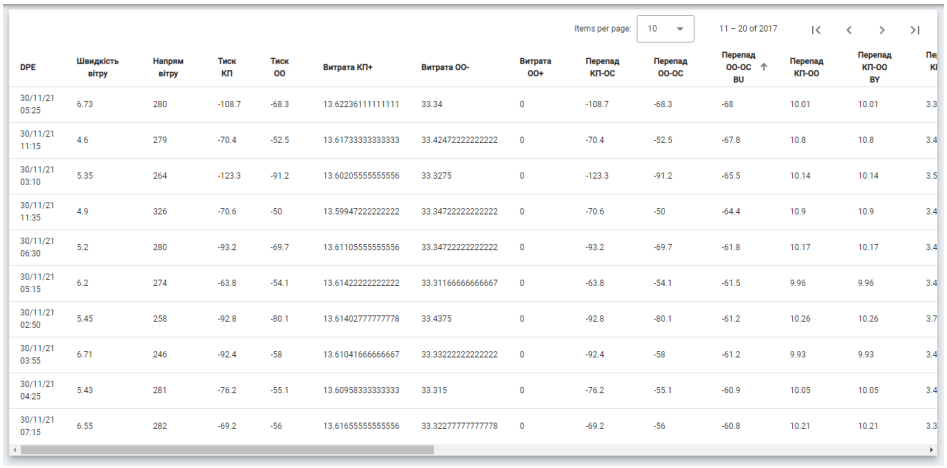


Рисунок 5.12 – Відображення кількох значень при наведенні на одну точку

5.3 Взаємодія з таблицями

Як можна побачити на рисунку 5.13, для користувача передбачена можливість сортування значень таблиці за кожною з колонок при натиску на неї в порядку спадання та зростання і прокручування таблиці по горизонталі.



Items per page: 10 | 11 – 20 of 2017

DPE	Швидкість вітру	Напрям вітру	Тиск КП	Тиск ОО	Витрата КП+	Витрата ОО-	Витрата ОО+	Перепад КП-ОС	Перепад ОО-ОС	Перепад ОО-ОС ВU	Перепад КП-ОО	Перепад КП-ОО ВУ	Пе... КІ
30/11/21 05:25	6.73	280	-108.7	-68.3	13.622361111111111	33.34	0	-108.7	-68.3	-68	10.01	10.01	3.3
30/11/21 11:15	4.6	279	-70.4	-52.5	13.61733333333333	33.42472222222222	0	-70.4	-52.5	-67.8	10.8	10.8	3.4
30/11/21 03:10	5.35	264	-123.3	-91.2	13.60205555555556	33.3275	0	-123.3	-91.2	-65.5	10.14	10.14	3.5
30/11/21 11:35	4.9	326	-70.6	-50	13.59947222222222	33.34722222222222	0	-70.6	-50	-64.4	10.9	10.9	3.4
30/11/21 06:30	5.2	280	-93.2	-69.7	13.61105555555556	33.34722222222222	0	-93.2	-69.7	-61.8	10.17	10.17	3.4
30/11/21 05:15	6.2	274	-63.8	-54.1	13.61422222222222	33.31166666666667	0	-63.8	-54.1	-61.5	9.96	9.96	3.4
30/11/21 02:50	5.45	258	-92.8	-80.1	13.61402777777778	33.4375	0	-92.8	-80.1	-61.2	10.26	10.26	3.7
30/11/21 03:55	6.71	246	-92.4	-58	13.61041666666667	33.33222222222222	0	-92.4	-58	-61.2	9.93	9.93	3.4
30/11/21 04:25	5.43	281	-76.2	-55.1	13.60958333333333	33.315	0	-76.2	-55.1	-60.9	10.05	10.05	3.4
30/11/21 07:15	6.55	282	-69.2	-56	13.61655555555556	33.32277777777778	0	-69.2	-56	-60.8	10.21	10.21	3.3

Рисунок 5.13 – Таблиця з даними вимірювань

Також для таблиці реалізовано механізм пагінації, тобто розбиття таблиці на сторінки. Користувач може вибрати кількість відображуваних на сторінці рядків таблиці між 10, 20 та 50 рядками. Також користувач може перемикати сторінки таблиці за натиском на стрілки у верхньому правому кутку таблиці. Елементи управління таблицею можна також побачити на рисунку 5.13.

Висновки до розділу 5

Розроблений додаток реалізовує базові функції із відображення даних у формі графіків та таблиць. Користувач має можливість сортувати та фільтрувати відображувані дані. Забезпечено інтерактивність відображуваних графіків через можливість їх масштабування та навігації ними за допомогою мишки, а також зручну взаємодію із даними через інтерфейс користувача.

ВИСНОВКИ

У ходів виконання роботи проаналізовано літературу за темою моделювання та аналізу гідравлічного стану Нового Безпечного Конфайнменту ЧАЕС. Досліджено різні архітектурні рішення та підходи до розробки проекту такого типу. Був визначений оптимальний архітектурний підхід. для розробки системи візуалізації даних із можливістю подальшого розширення цієї системи або її використання як модуля більшої системи цифрових двійників. Також досліджено сучасні методи відображення даних у веб-додатку.

В результаті реалізовано модуль візуалізації стану моделі гідравлічного стану НБК, що складається із веб-серверу та клієнту. Веб-сервер розроблено із застосуванням фреймворку ASP.NET та реляційної бази даних MS SQL Server. Клієнт розроблено із застосуванням фреймворку Angular з бібліотекою візуалізації даних Chart.Js. Розроблена система враховує вимоги до безпеки.

Крім того, розглянуті можливості до відображення даних в реальному часі та визначено подальший напрям дослідження за цим напрямком.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Круковський П. Г,. Дядюшко Є. В., Скляренко Д. І., Старовіт І. С. Неорганізовані викиди повітря з радіоактивними аерозолями із нового безпечного конфайнмента ЧАЕС в оточуюче середовище // Питання атомної науки і техніки. 2021. №6. С. 181–186.

2. Introduction to Grafana | Grafana documentation: веб-сайт. URL: https://grafana.com/docs/grafana/latest/introduction/

3. Tableau Desktop and Web Authoring Help: веб-сайт. URL: https://help.tableau.com/current/pro/desktop/en-us/default.htm

4. Learn how to view, edit and create a Looker Studio report: веб-сайт. URL: https://lookerstudio.google.com/u/0/reporting/0B5FF6JBKbNJxOWItcWo2SVVVeGc

5. Getting started with Power BI: веб-сайт. URL: https://powerbi.microsoft.com/en-us/getting-started-with-power-bi/

6. Charts and other visualizations in Power View: веб-сайт. URL: https://support.microsoft.com/en-us/office/charts-and-other-visualizations-in-power-view-141bd462-9853-4973-ac37-842e8345f51e

7. Entity Framework documentation: веб-сайт. URL: https://learn.microsoft.com/en-us/ef/

8. Angular documentation: веб-сайт. URL: https://angular.io/docs

9. Chart.js documentation: веб-сайт. URL: https://www.chartjs.org/docs/latest

10. Valerio De Sanctis. ASP.NET Core 6 and Angular Fifth Edition. Birmingham, 2023. 489-552 с.

ДОДАТОК А Програмний код

Компонент Angular DashboardInfoComponent

```
import { Component, OnInit, ChangeDetectorRef } from '@angular/core';
import { Chart, registerables } from "chart.js";
import { HttpClient } from '@angular/common/http';
import zoomPlugin from 'chartjs-plugin-zoom';
import { ChartDataService } from 'src/app/services/chart-data.service';
import { FormControl } from '@angular/forms';
import 'chartjs-adapter-date-fns';

Chart.register(zoomPlugin);
Chart.register(...registerables);

@Component({
  selector: 'app-dashboard-info',
  templateUrl: './dashboard-info.component.html',
  styleUrls: ['./dashboard-info.component.scss']
})


export class DashboardInfoComponent implements OnInit {
  data: any;
  speedDirection: any;
  isDataAvailable = false;
  start = new FormControl(new Date(Date.parse('01 Dec 2018')));
  end = new FormControl(new Date(Date.now()));

  labels: Date[] = [];
  windSpeed: number[] = [];
  windDirection: number[] = [];
  pressureRingSpace: number[] = [];
  pressureMainCapacity: number[] = [];
  spendRingSpacePlus: number[] = [];
  spendMainCapacityMinus: number[] = [];
  spendMainCapacityPlus: number[] = [];
  dropRingSpaceEnvironment: number[] = [];
  dropCapacityEnvironment: number[] = [];
  dropCapacityEnvironmentBU: number[] = [];
  dropRingSpaceCapacity: number[] = [];
  dropRingSpaceCapacityBy: number[] = [];
  dropRingSpaceCapacityBz: number[] = [];
  dropRingSpaceCapacityCa: number[] = [];
  airDensity: number[] = [];
  bcPressureWestWall: number[] = [];
  bcPressureEastWall: number[] = [];
  bcPressureCylindricalWall: number[] = [];
```

```
  bcPressureWestGap: number[] = [];
  bcPressureEastGap: number[] = [];
  bcPressureAuxiliarySystems: number[] = [];
  bcPressure008pBq: number[] = [];
  bcPressure009pBr: number[] = [];
  bcPressureRingSpaceEnvironmentBs: number[] = [];

  radarDirection: number[] = [];
  radarSpeed: number[] = [];

  windSpeedChart: Chart;
  speedDirectionRadar: Chart;
  spendChart: Chart;
  dropChart: Chart;
  pressureChart: Chart;


  constructor(private http: HttpClient, private chartData: ChartDataService, private
cdr: ChangeDetectorRef) {
  }

  async ngOnInit() {

    await this.transformData(this.start.value!, this.end.value!);
    await this.transformSpeedDirection(this.start.value!, this.end.value!);

  }

  ngAfterViewInit() {
  }

  async updateCharts() {
    await this.clearArrays();
  }

  RenderChartWindSpeed() {
    var chart = new Chart('line-chart-wind', {
      type: 'line',
      data: {
        labels: this.labels,
        datasets: [
          {
            label: 'Швидкість',
            data: this.windSpeed,
            borderWidth: 1
          },
        ]
      },
      options: {
```

```
      responsive: true,
      scales: {
        x: {
          display: true
        },
        y: {
          suggestedMax: 10,
          suggestedMin: 0.03,
          display: true
        }
      },
      plugins: {
        legend: {
          labels: {
            font: {
              size: 16
            }
          }

        },
        zoom: {
          limits: {
            y: { min: -20, max: 20 }
          },
          zoom: {
            wheel: {
              enabled: true,
              speed: 0.05
            },
            scaleMode: 'x',
            overScaleMode: 'y'
          },
          pan: {
            enabled: true,
            threshold: 20
          },
        }
      }
    }
  });

  return chart;
}

RenderRadarSpeedDiretion() {
  var chart = new Chart('radar-speed-direction', {
    type: 'radar',
    data: {
      labels: this.radarDirection,
```

```
      datasets: [
        {
          label: 'Швидкість',
          data: this.radarSpeed,
          borderWidth: 1
        },
      ]
    },
    options: {
      responsive: true,
      scales: {
        r: {
          pointLabels: {
            display: false,

          },
          ticks: {
            display: false
          }
        }
      },
      plugins: {
        legend: {
          labels: {
            font: {
              size: 16
            }
          }

        },
      }
    }
  });

  return chart;
}

RenderChartDrop() {
  return new Chart('line-chart-drop', {
    type: 'line',
    data: {
      labels: this.labels,
      datasets: [
        {
          label: 'КП-ОС',
          data: this.dropRingSpaceEnvironment,
          borderWidth: 1,
        },
        {
```

```
      label: 'ОО-ОС',
      data: this.dropCapacityEnvironment,
      borderWidth: 1,
    },
    {
      label: 'КП-ОО',
      data: this.dropRingSpaceCapacity,
      borderWidth: 1
    }
  ]
},
options: {
  responsive: true,
  interaction: {
    mode: 'index',
  },
  scales: {
    x: {

      display: true,
    },
    y: {
      suggestedMin: -150,
      display: true,
      suggestedMax: 60
    }
  },
  plugins: {
    legend: {
      labels: {
        font: {
          size: 16
        }
      }

    },
    zoom: {
      limits: {
        y: { min: -200, max: 200 }
      },
      zoom: {
        wheel: {
          enabled: true,
          speed: 0.05
        },
        scaleMode: 'x',
        overScaleMode: 'y'
      },
      pan: {
```

```
            enabled: true,
          }
        }
      }
    }

  });
}

RenderChartSpend() {
  return new Chart('line-chart-spend', {
    type: 'line',
    data: {
      labels: this.labels,
      datasets: [
        {
          label: 'КП+',
          data: this.spendRingSpacePlus,
          borderWidth: 0.5
        },
        {
          label: 'ОО-',
          data: this.spendMainCapacityMinus,
          borderWidth: 0.5
        },
        {
          label: 'ОО+',
          data: this.spendRingSpacePlus,
          borderWidth: 0.5
        }
      ]
    },
    options: {
      responsive: true,
      interaction: {
        mode: 'index',
      },
      scales: {
        x: {

          display: true,
        },
        y: {
          min: 0,
          display: true,
          suggestedMax: 60
        }
      },
      plugins: {
```

```
        legend: {
          labels: {
            font: {
              size: 16
            }
          }

        },
        zoom: {
          zoom: {
            wheel: {
              enabled: true,
              speed: 0.05
            },
            scaleMode: 'x',
            overScaleMode: 'y'
          },
          pan: {
            enabled: true,
          }
        }
      }
    }

  });
}

RenderChartPressure() {
  return new Chart('line-chart-pressure', {
    type: 'line',
    data: {
      labels: this.labels,
      datasets: [
        {
          label: 'Тиск КП',
          data: this.pressureRingSpace,
          borderWidth: 0.5
        },
        {
          label: 'Тиск ОО',
          data: this.pressureMainCapacity,
          borderWidth: 0.5
        }
      ]
    },
    options: {
      responsive: true,
      interaction: {
        mode: 'index',
```

```
      },
      scales: {
        x: {

          display: true,
        },
        y: {
          suggestedMin: 150,
          display: true,
          suggestedMax: 150,
        }
      },
      plugins: {
        legend: {
          labels: {
            font: {
              size: 16
            }
          }

        },
        zoom: {
          zoom: {
            wheel: {
              enabled: true,
              speed: 0.05
            },
            scaleMode: 'x',
            overScaleMode: 'y'
          },
          pan: {
            enabled: true,
          }
        }
      }
    }

  });
}

async transformData(start: Date, end: Date) {
  this.chartData.getMeasurements(start, end).subscribe(result => {
    this.data = result;

    if (this.data != null) {
      for (let i = 0; i < this.data.length; i++) {
        this.labels.push(this.data[i].dpe);
        this.windSpeed.push(this.data[i].wind_speed);
        this.windDirection.push(this.data[i].wind_direction);
```

```
          this.pressureRingSpace.push(this.data[i].pressure_ringspace);
          this.pressureMainCapacity.push(this.data[i].presssure_main_capacity);
          this.spendRingSpacePlus.push(this.data[i].spend_ringspace_plus);
          this.spendMainCapacityMinus.push(this.data[i].spend_main_capacity_minus);
          this.spendMainCapacityPlus.push(this.data[i].spend_main_capacity_plus);
          this.dropRingSpaceEnvironment.push(this.data[i].drop_ringspace_environment)
;
          this.dropCapacityEnvironment.push(this.data[i].drop_capacity_environment);
          this.dropCapacityEnvironmentBU.push(this.data[i].drop_capacity_environment_
bu);
          this.dropRingSpaceCapacity.push(this.data[i].drop_ringspace_capacity);
          this.dropRingSpaceCapacityBy.push(this.data[i].drop_ringspace_capacity_by);
          this.dropRingSpaceCapacityBz.push(this.data[i].drop_ringspace_capacity_bz);
          this.dropRingSpaceCapacityCa.push(this.data[i].drop_ringspace_capacity_ca);
          this.airDensity.push(this.data[i].air_density);
          this.bcPressureWestWall.push(this.data[i].bc_pressure_west_wall);
          this.bcPressureEastWall.push(this.data[i].bc_pressure_east_wall);
          this.bcPressureCylindricalWall.push(this.data[i].bc_pressure_cylindrical_wa
ll);
          this.bcPressureWestGap.push(this.data[i].bc_pressure_west_gap);
          this.bcPressureEastGap.push(this.data[i].bc_pressure_east_gap);
          this.bcPressureAuxiliarySystems.push(this.data[i].bc_pressure_auxiliary_sys
tems);
          this.bcPressure008pBq.push(this.data[i].bc_pressure_008p_bq);
          this.bcPressure009pBr.push(this.data[i].bc_pressure_009p_br);
          this.bcPressureRingSpaceEnvironmentBs.push(this.data[i].bc_pressure_ringspa
ce_environment_bs);

        }
      }

    this.windSpeedChart = this.RenderChartWindSpeed();
    this.spendChart = this.RenderChartSpend();
    this.dropChart = this.RenderChartDrop();
    this.pressureChart = this.RenderChartPressure();
    });
  }


  async clearArrays() {
    this.labels = [];
    this.windSpeed = [];
    this.windDirection = [];
    this.pressureRingSpace = [];
    this.pressureMainCapacity = [];
    this.spendRingSpacePlus = [];
    this.spendMainCapacityMinus = [];
    this.spendMainCapacityPlus = [];
    this.dropRingSpaceEnvironment = [];
```

```
    this.dropCapacityEnvironment = [];
    this.dropCapacityEnvironmentBU = [];
    this.dropRingSpaceCapacity = [];
    this.dropRingSpaceCapacityBy = [];
    this.dropRingSpaceCapacityBz = [];
    this.dropRingSpaceCapacityCa = [];
    this.airDensity = [];
    this.bcPressureWestWall = [];
    this.bcPressureEastWall = [];
    this.bcPressureCylindricalWall = [];
    this.bcPressureWestGap = [];
    this.bcPressureEastGap = [];
    this.bcPressureAuxiliarySystems = [];
    this.bcPressure008pBq = [];
    this.bcPressure009pBr = [];
    this.bcPressureRingSpaceEnvironmentBs = [];

    this.radarDirection = [];
    this.radarSpeed = [];


    await this.transformData(this.start.value!, this.end.value!);
    await this.transformSpeedDirection(this.start.value!, this.end.value!);
    this.windSpeedChart.destroy();
    this.speedDirectionRadar.destroy();
    this.spendChart.destroy();
    this.dropChart.destroy();
    this.pressureChart.destroy();
  }

  async transformSpeedDirection(start: Date, end: Date) {
    this.chartData.getSpeedDirection(start, end).subscribe(result => {
      this.speedDirection = result;
      if (this.speedDirection != null ) {
        for (let i = 0; i < this.speedDirection.length; i++) {
          this.radarDirection.push(this.speedDirection[i].windDirection);
          this.radarSpeed.push(this.speedDirection[i].windSpeed)

        }
      }
      this.speedDirectionRadar = this.RenderRadarSpeedDiretion();
    });
  }
}
```

Контролер ASP.NET ChartsController

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using HSVModule.API.Data;
using HSVModule.API.Data.Models;
using System.Data;
using Microsoft.AspNetCore.SignalR;
using HSVModuleAPI.Data.DTO;

namespace HSVModuleAPI.Controllers
{
    [Route("api/[controller]/[action]")]
    [ApiController]
    public class ChartsController : ControllerBase
    {
        private readonly ApplicationDbContext _context;

        public ChartsController(ApplicationDbContext context)
        {
            _context = context;
        }

        [HttpGet]
        public async Task<ActionResult<Measurement>> GetMeasurements(string
startTime, string endTime)
        {
            var start = DateTime.Parse(startTime);
            var end = DateTime.Parse(endTime);
            var data = await _context.Measurements
                .Where(m => m.DPE >= start && m.DPE <= end)
                .ToListAsync();

            return Ok(data);
        }

        [HttpGet]
        public async Task<ActionResult<SpeedDirectionDTO>> GetSpeedDirection(string
startTime, string endTime)
        {
            var start = DateTime.Parse(startTime);
            var end = DateTime.Parse(endTime);
            var radar = new List<SpeedDirectionDTO>();

            var data = await _context.Measurements
                .Where(m => m.DPE >= start && m.DPE <= end)
                .Select(m => new SpeedDirectionDTO
                {
                    WindSpeed = m.wind_speed,
                    WindDirection = m.wind_direction
                })
                .ToListAsync();


            foreach (var direction in Enumerable.Range(1, 360))
            {
```

```
                var speedList = data.Where(d => d.WindDirection ==
direction).ToList();
                double speed;

                if (speedList.Count > 0)
                {
                    speed = speedList.Select(s => s.WindSpeed).Average();
                }
                else
                {
                    speed = 0;
                }

                radar.Add(new SpeedDirectionDTO()
                {
                    WindDirection = direction,
                    WindSpeed = speed
                });
            }

            return Ok(radar);
        }

    }
}
```