Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Автоматики і комп'ютеризованих технологій
(повна назва)
Кафедра Комп'ютерно–інтегрованих технологій, автоматизації та робототехніки

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Розроблення комп'ютеризованої системи для візуалізації інформації з баз даних на вебсайті
(тема)

Виконав:
здобувач другого року навчання,
групи КТРСМ–23–1
Гургуц М. Д.
(прізвище, ініціали)

Спеціальність 174 Автоматизація та комп'ютерно–інтегровані технології та робототехніка
(код і повна назва спеціальності)

Тип програми освітньо–професійна
(освітньо–професійна або освітньо–наукова)

Освітня програма Комп'ютеризовані та робототехнічні системи
(повна назва освітньої програми)

Керівник доц. Іванов Л. С
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Невлюдов І.Ш.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет	Автоматики і комп'ютеризованих технологій
Кафедра	Комп'ютерно–інтегрованих технологій, автоматизації та робототехніки
Рівень вищої освіти	другий (магістерський)
Спеціальність	174 Автоматизація та комп'ютерно–інтегровані технології та робототехніка
Тип програми	освітньо–професійна
Освітня програма	174 Автоматизація та комп'ютерно–інтегровані технології та робототехніка

(код і повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____»_____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Гургуц Микиті Дмитровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення комп'ютеризованої системи для візуалізації інформації з баз даних на вебсайті
затверджена наказом по університету від 25.11. 2024 р. № 1239 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 12. 01. 2025 р.
3. Вихідні дані до роботи
3.1 Тип СУБД – MySQL
3.2 Мова програмування – Java
3.3 Призначення комп'ютеризованої сис.,теми, що розроблюється: візуалізація інформації з баз даних на вебсайті
4. Зміст розрахунково–пояснювальної записки (перелік питань, які потрібно розробити)
4.1 Вступ, 4.2 Аналіз предметної області, 4.3 Обгрунтування важливості дослідження, 4.4 Розробка автоматизованого пристрою, 4.5 Робота системи на практиці, 4.6 Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Демонстраційний матеріал представлений у форматі презентації PowerPoint (*.ppt) – 12 с. формату А4

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Керівник (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз літератури за темою кваліфікаційної роботи	28.07.2024	Виконано
2	Розроблення структури комп'ютеризованої системи	28.09.2024	Виконано
3	Розробка бази даних	28.011.2024	Виконано
4	Розроблення програмного модуля	28.12.2024	Виконано
5	Оформлення пояснювальної записки	1.01.2025	Виконано
6	Подання роботи на перевірку Інтернет–сервісом Unicheck	12.01.2025	Виконано
7	Подання роботи на рецензію	13.01.2025	Виконано
8	Подання роботи на підпис зав. кафедри	14.01.2025	Виконано
9	Подання роботі до ЕК	15.01.2025	Виконано

Дата видачі завдання

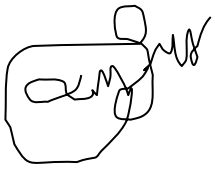
Здобувач _____ Гургуц М.Д.

(підпис) (прізвище, ініціали)

Керівник роботи доц. Іванов Л. С.

(підпис) (посада, прізвище, ініціали)

Я, як студент ХНУРЕ, розумію і підтримую політику закладу із академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

«12» січня 2025 р.  Гургуц М. Д.

РЕФЕРАТ

Пояснювальна записка: 90 с., 17 рис., 2 дод., 53 джерел.

БАЗА ДАНИХ, АВТОМАТИЗАЦІЯ, КОМП'ЮТЕРИЗОВАНА СИСТЕМА, ВІДОБРАЖЕННЯ ДАННИХ.

Об'єкт дослідження: процеси інтеграції баз даних із вебсайтами для візуалізації інформації.

Предмет дослідження: методи, технології та програмні інструменти для створення веборієнтованих систем візуалізації даних.

Мета роботи: удосконалення комп'ютеризованої системи для візуалізації інформації з баз даних на веб–сайті шляхом створення програмного засобу, який забезпечить зручність користування, динамічне оновлення даних і надійність інтеграції.

Досягнення мети передбачає комплексний підхід, що включає кілька етапів. Першим кроком є аналіз існуючих методів інтеграції баз даних із веборієнтованими системами. Це важливо для визначення найкращих практик та можливих проблем, які можуть виникнути під час реалізації проекту. На основі цього аналізу здійснюється вибір оптимальних технологій для забезпечення ефективної взаємодії баз даних із вебінтерфейсом. Це включає вибір відповідних фреймворків, бібліотек та інших інструментів, які відповідають потребам проекту.

Після вибору технологій розробляється структура вебсайту, яка має забезпечити візуалізацію інформації у зручному та зрозумілому вигляді для користувачів. Важливо врахувати інтуїтивність інтерфейсу та його адаптивність до різних пристроїв. Реалізація функціоналу динамічного оновлення та обробки даних із бази є наступним етапом, що забезпечує актуальність представленої інформації та можливість її оперативного оновлення.

Останнім, але не менш важливим етапом є тестування та оптимізація розробленої системи для забезпечення її коректної роботи. Це включає виявлення та виправлення можливих помилок, а також покращення продуктивності та безпеки системи. Впровадження цих кроків дозволяє підвищити ефективність представлення інформації та забезпечити її доступність для користувачів завдяки сучасним технологіям веброзробки.

Також, отримані результати роботи можно віднести до Цілі сталого розвитку 15 “Захист та відновлення екосистем суші”, а саме п. 15.3 “ Відновити деградовані землі та ґрунти з використанням інноваційних технологій”.

ABSTRACT

Explanatory note: 90 p., 17 fig., 53 sources, 2 appendix.

DATABASE, AUTOMATION, COMPUTERIZED SYSTEM, DATA DISPLAY.

Object of research: processes of database integration with websites for information visualization.

Subject of research: methods, technologies and software tools for creating web–based data visualization systems.

Purpose of work: development of a computerized system for visualization of information from databases on a website, which will ensure ease of use, dynamic data updating and reliability of integration.

Achieving the goal involves a comprehensive approach that includes several stages. The first step is to analyze existing methods of database integration with web–based systems. This is important for identifying best practices and possible problems that may arise during the project implementation. Based on this analysis, the optimal technologies are selected to ensure effective interaction of databases with the web interface. This includes the selection of appropriate frameworks, libraries, and other tools that meet the needs of the project.

After selecting the technologies, the website structure is developed, which should provide visualization of information in a convenient and understandable form for users. It is important to take into account the intuitiveness of the interface and its adaptability to different devices. The implementation of the functionality of dynamic updating and data processing from the database is the next stage, which ensures the relevance of the presented information and the possibility of its prompt updating.

The last, but no less important stage is testing and optimizing the developed system to ensure its correct operation. This includes identifying and correcting possible

errors, as well as improving the performance and security of the system. The implementation of these steps allows you to increase the efficiency of information presentation and ensure its accessibility for users thanks to modern web development technologies.

Also, the results of the work can be taken to the Goal of development 15 “Protection and renewal of terrestrial ecosystems”, and also paragraph 15.3 “Renew degraded lands and soils with the use of innovative innovative technologies”.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

СУБД – систем управління базами даних;

API – Application Programming Interface;

HTML – HyperText Markup Language;

RESTful – Representational State Transfer;

SQL – Structured Query Language.

ВСТУП

Актуальність теми. У сучасному світі обсяг даних, що генеруються різними системами, постійно зростає. Ефективна візуалізація цих даних є ключовим фактором для прийняття обґрунтованих рішень в різних сферах діяльності – від бізнесу та фінансів до науки та медицини. Здатність швидко та наочно представити великі обсяги інформації дозволяє виявити закономірності, тенденції та аномалії, які можуть бути непомітними при аналізі даних у табличному вигляді. Автоматизація процесу візуалізації даних, особливо в контексті веб–додатків, значно підвищує ефективність роботи з інформацією та спрощує доступ до неї для широкого кола користувачів. Тому розробка автоматизованого пристрою візуалізації даних з бази даних на веб–сайті є актуальною та перспективною задачею. Існуючі рішення часто мають обмежену функціональність, не забезпечують реального часу оновлення або потребують значних зусиль для налаштування та обслуговування. Ця робота спрямована на створення більш ефективного та зручного інструменту для візуалізації даних.

Мета і задачі дослідження. Метою даної роботи є удосконалення комп’ютеризованої системи для візуалізації інформації на веб–сайті шляхом створення програмного засобу, отриманих з бази даних, який забезпечує швидке, зручне та інтерактивне відображення інформації в режимі реального часу. Для досягнення цієї мети необхідно вирішити наступні задачі:

– провести аналіз існуючих методів та технологій візуалізації даних;

– визначити оптимальний набір технологій для розробки веб–пристрою;

– розробити архітектуру системи, що забезпечує автоматичне оновлення візуалізацій;

– спроектувати зручний та інтуїтивно зрозумілий інтерфейс користувача;

– розробити алгоритми обробки та візуалізації даних;

– забезпечити безпеку та надійність роботи системи;

– провести тестування та оцінку ефективності розробленого веб–пристрою.

Об’єкт і предмет дослідження. Об’єктом дослідження є процес візуалізації даних, отриманих з бази даних. Предметом дослідження є розробка програмного забезпечення та веб–інтерфейсу для автоматизованої візуалізації даних, що забезпечує зручний та ефективний доступ до інформації для користувачів.

Методи дослідження. У даній роботі будуть використані наступні методи дослідження:

– аналіз літератури з питань візуалізації даних та веб–розробки;
– порівняльний аналіз існуючих рішень;
– моделювання архітектури системи;
– проектування інтерфейсу користувача;
– розробка програмного забезпечення;
– тестування та оцінка ефективності.

Робота оформлена згідно з ДСТУ 3008–2015 [1] та методичнми вказівками [2]. По цій роботі було проведено дослідження, яке опубліковане в статті [3].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Визначення комп'ютеризованої системи

Комп'ютеризована система – інформаційно–технічний комплекс метою якого є обробка, збереження, ввід–вивід інформації. До складу комп'ютерної системи входять комп'ютери, принтери, сервери тощо із програмним забезпеченням. Через комп'ютерну мережу за допомогою локальної або глобальної системи передачі даних здійснюється обмін інформацією. При описуванні систем використовують технічні, організаційні, документальні, функціональні, алгоритмічні, програмні та інформаційні структури. Задачі, що розв'язуються в комп'ютерних інформаційних системах, мають ряд характерних особливостей, що впливають на технологію автоматизованої обробки даних. Комп'ютерна система має можливість інтегрувати з іншими інженерними технологіями, розширювати можливості й створювати єдине середовище для керування завдяки різноманіттю і уніфікації комп'ютерного устаткування.

Комп'ютерні системи відіграють ключову роль у сучасному світі, проникаючи практично в усі сфери життя суспільства. Вони стали незамінними не лише для менеджерів, але й для багатьох інших фахівців. Завдяки комп'ютерним системам можна вирішувати різноманітні завдання у конкретних предметних галузях, таких як технологічна підготовка, управління, облік та автоматизація процесів. Важливим аспектом стало їх застосування у дистанційному навчанні, яке еволюціонувало від традиційного заочного формату до сучасних методів, що використовують кейс–, ТБ– і мережеві технології.

Сьогодні користувачі комп'ютерів мають доступ до різноманітного програмного забезпечення, включаючи системні та прикладні програми, як–от компілятори, текстові редактори, системи управління базами даних та інші. Ці програми функціонують у взаємодії з операційною системою, яка забезпечує

управління всіма процесами на комп'ютері. Таким чином, сучасні комп'ютерні системи значно спрощують виконання багатьох завдань, підвищуючи ефективність та продуктивність у різних сферах діяльності.

У контексті дипломної роботи, додаток, розроблений на основі Spring Framework з використанням MySQL та HTML, можна розглядати як "комп’ютеризовану систему". Він є ключовим елементом автоматизованої системи, що забезпечує автоматизацію процесу візуалізації даних з бази даних на вебсайті. Враховуючи сучасні тенденції в інформаційних технологіях, де межа між апаратним та програмним забезпеченням стає дедалі розмитішою, такий підхід є цілком обґрунтованим і відповідає загальноприйнятим визначенням автоматизованих систем.

1.2 Основи візуалізації даних

Візуалізація даних – це потужний інструмент, що дозволяє перетворювати абстрактні числові дані у зрозумілі та доступні візуальні образи. Вона є ключовим елементом у процесі аналізу даних, допомагаючи виявляти закономірності, тенденції та аномалії, які можуть бути непомітними при аналізі числових таблиць. Ефективна візуалізація не тільки спрощує сприйняття інформації, а й покращує комунікацію результатів дослідження з різною аудиторією, включаючи як експертів, так і широку публіку [4].

Візуалізацію даних можна визначити як процес перетворення цифрових даних у візуальні форми, такі як графіки, діаграми, карти, анімації та інтерактивні елементи, для покращення розуміння та комунікації інформації. Вона включає в себе вибір відповідного типу візуалізації, дизайн візуальних елементів, представлення даних та інтерпретацію отриманих результатів.

Класифікація методів візуалізації даних є багатогранною задачею, оскільки існує безліч підходів до представлення інформації візуально. Один з найпоширеніших способів класифікації базується на типі даних, які необхідно візуалізувати. Для кількісних даних, що представляють величину або кількість,

найбільш ефективними є стовпчикові діаграми, які чудово демонструють порівняння різних категорій за допомогою довжини стовпчиків, що прямо пропорційні величині даних. Подібними за своєю суттю є гістограми, але вони зосереджуються на відображенні розподілу частот, показуючи, як часто зустрічаються певні значення у наборі даних. Для демонстрації змін значень з часом або залежно від іншої змінної, ідеальним вибором стають лінійні графіки, які наочно ілюструють тренди та динаміку. Якщо ж потрібно показати взаємозв'язок між двома змінними, то зручно використовувати точкові діаграми (scatter plots), де кожна точка представляє пару значень, дозволяючи виявити кореляцію або залежність між ними. Для детального аналізу розподілу даних, включаючи медіану, квартилі та викиди, ефективно застосовуються box plots (ящики з вусами), що дають уявлення про центральну тенденцію та розкид даних [5].

На відміну від кількісних, якісні дані описують характеристики або категорії. Для їх візуалізації часто використовують кругові діаграми (pie charts), що наочно демонструють частку кожної категорії від загальної суми. Стовпчикові діаграми також можуть бути ефективними для порівняння частот категорій. Для відображення ієрархічних структур та взаємозв'язків між категоріями добре підходять дерева, які візуалізують структуру даних у вигляді розгалуженої схеми.

Окремо варто розглянути візуалізацію часових рядів – даних, що змінюються з часом. Для них найкраще підходять лінійні графіки, що чітко демонструють тренди та зміни з часом. Свічкові діаграми, поширені в аналізі фінансових ринків, дозволяють представити відкриття, закриття, максимум та мінімум значень за певний період. Ареальні діаграми, в свою чергу, зосереджуються на відображенні сукупного показника за певний період [6].

Просторові дані, пов'язані з географічним розташуванням, найефективніше візуалізуються за допомогою карт, що дозволяють наочно представити розподіл даних на географічній площині. Теплові карти доповнюють карти, відображаючи інтенсивність даних за допомогою

кольорового кодування, що дозволяє швидко ідентифікувати зони з високою та низькою концентрацією даних [7].

Крім класифікації за типом даних, методи візуалізації можна розділити за метою їх застосування. Експлуаторна візуалізація використовується на початкових етапах аналізу для пошуку закономірностей та формулювання гіпотез. Підтверджувальна візуалізація, навпаки, спрямована на перевірку сформульованих гіпотез та демонстрацію отриманих результатів. Комунікативна візуалізація має на меті передачу інформації широкій аудиторії, тому наголос робиться на простоті та зрозумілості візуального представлення даних. Вибір методу візуалізації завжди залежить від конкретних завдань та контексту аналізу.

Також є візуалізація за типом візуальних елементів. Ця класифікація базується на конкретних візуальних елементах, що використовуються для представлення даних. Вона включає велику кількість різних типів діаграм, графіків та інших візуальних форм, кожна з яких має свої переваги та недоліки, залежно від типу даних та мети візуалізації. Наприклад, мережеві графіки, діаграми Вороного, панорамні графіки, та багато інших.

Це лише деякі з можливих класифікацій методів візуалізації даних. Вибір конкретного методу залежить від типу даних, мети візуалізації та аудиторії. Важливо пам'ятати, що ефективна візуалізація повинна бути зрозумілою, точною та естетично привабливою.

1.3 Вибір відповідних методів візуалізації

Вибір оптимальних методів візуалізації для будь–якого проекту, в тому числі й для розробки веб–пристрою візуалізації даних з бази даних, є критично важливим етапом. Неправильний вибір може призвести до незрозумілої або навіть спотвореної картини даних, ускладнюючи аналіз та прийняття рішень. Оптимальне рішення визначається на основі кількох ключових факторів: типу даних, мети візуалізації та очікуваної аудиторії [8].

Тип даних. Перш за все, необхідно визначити тип даних, що підлягають візуалізації. Це може бути:

– кількісні дані: числові значення, що вимірюють величину або кількість (наприклад, температура, прибуток, кількість проданих товарів). Для них добре підходять стовпчикові діаграми, гістограми, лінійні графіки, точкові діаграми та box plots. Вибір конкретного методу залежить від того, що потрібно показати: порівняння категорій, розподіл частот, динаміку змін, взаємозв'язок між змінними або розкид даних [9];

– якісні дані: категоріальні дані, що описують характеристики або атрибути (наприклад, колір, тип товару, країна походження). Для них доречні кругові діаграми, стовпчикові діаграми та дерева. Кругові діаграми добре показують частку кожної категорії від загальної суми, стовпчикові – порівнюють частоти категорій, а дерева – ієрархічні структури [10];

– часові ряди: дані, що змінюються з часом (наприклад, температура протягом дня, курс валюти). Для них найбільш підходять лінійні графіки, які наочно демонструють тренди та динаміку змін. Свічкові діаграми використовуються для візуалізації фінансових даних [11];

– просторові дані: дані, пов'язані з географічним розташуванням (наприклад, розподіл населення, кількість дощів). Для них використовуються карти та теплові карти [12].

Мета візуалізації є ключовим фактором, що визначає вибір відповідного методу. Якщо головна задача – порівняння різних категорій або груп даних, то найефективнішим рішенням будуть стовпчикові діаграми, які наочно демонструють відносні розміри величин за допомогою довжини стовпчиків, або кругові діаграми, що показують частку кожної категорії від загальної суми. Для візуалізації трендів та змін значень з часом або залежно від іншої змінної найбільш підходящим є лінійний графік, що дозволяє чітко простежити динаміку змін та виявити тенденції. Якщо ж метою є виявлення кореляцій та взаємозв'язків між двома змінними, то необхідно використовувати точкові діаграми (scatter plots), де кожна точка представляє пару значень, а їхнє

розташування на графіку показує взаємозв'язок між змінними. Для детального аналізу розподілу даних, виявлення центральної тенденції, розкиду та наявності аномалій, найбільш інформативними будуть гістограми, що показують частоту значень у різних інтервалах, та box plots (ящики з вусами), що наочно ілюструють медіану, квартилі та викиди. Нарешті, для відображення географічних даних, їх розподілу та інтенсивності на певній території, найефективнішими методами будуть карти, що дозволяють наочно представити просторові дані, та теплові карти, які використовуються для візуалізації інтенсивності даних за допомогою кольорової шкали, відображаючи концентрацію даних на карті. Вибір методу залежить від того, яку саме інформацію потрібно передати та які аспекти даних необхідно підкреслити.

Аудиторія. Важливо враховувати рівень знань та досвід аудиторії, для якої призначена візуалізація. Для неспеціалістів краще використовувати прості та зрозумілі методи, такі як стовпчикові діаграми або лінійні графіки. Для експертів можна використовувати більш складні методи, що дозволяють показати більше деталей [13].

Обґрунтування оптимального рішення для проекту. Для веб–пристрою візуалізації даних з бази даних, оптимальний вибір методів буде залежати від конкретних даних та функціональних можливостей системи. Однак, бажано забезпечити гнучкість, дозволяючи користувачеві вибирати з декількох методів візуалізації залежно від його потреб. Це може включати стовпчикові діаграми, лінійні графіки, точкові діаграми, кругові діаграми та, залежно від типу даних, інші методи. Важливо забезпечити інтерактивність, дозволяючи користувачеві масштабувати, змінювати масштаб та фільтрувати дані. Для великих обсягів даних необхідно використовувати ефективні алгоритми та бібліотеки, що забезпечують швидке відображення та оновлення візуалізацій [14].

1.4 Технології веб–розробки для візуалізації даних

Розробка веб–додатку для візуалізації даних вимагає ретельного вибору технологій, враховуючи їх можливості, продуктивність та відповідність потребам проекту. Ключовими аспектами є вибір JavaScript бібліотек для візуалізації, фреймворків для розробки інтерфейсу, систем баз даних та серверних технологій [15].

JavaScript бібліотеки для візуалізації. Вибір JavaScript бібліотеки є критичним, оскільки вона безпосередньо впливає на якість та можливості візуалізації. Серед популярних варіантів:

– D3.js: потужна та гнучка бібліотека, що дозволяє створювати складні та інтерактивні візуалізації. Вона надає повний контроль над кожним елементом, але вимагає глибокого розуміння JavaScript та SVG. Це робить її складною для новачків, але ідеальною для розробки унікальних та кастомних візуалізацій. (Reddit) підкреслює її складність, пропонуючи альтернативи, такі як Plot, побудовані на основі D3, але з спрощеним інтерфейсом. (Monterail), (wpDataTables) та інші ресурси також згадують D3.js як один з найкращих варіантів [16];

– Chart.js: проста у використанні бібліотека, ідеальна для швидкого створення різних типів діаграм. Надає широкий набір готових до використання компонентів, що спрощує розробку. Однак, вона може бути обмежена у можливостях кастомізації порівняно з D3.js. Багато ресурсів, включаючи (DataBrain) та (GeeksforGeeks), включають Chart.js до списку найкращих бібліотек [17];

– Highcharts: комерційна бібліотека з широким набором функцій та можливостей кастомізації. Надає високу якість візуалізації та підтримку різних типів діаграм. Однак, вимагає ліцензії для комерційного використання. (LogRocket) та інші ресурси згадують Highcharts як один з популярних варіантів [18];

– Plotly.js: ще одна потужна бібліотека, що дозволяє створювати інтерактивні та анімовані візуалізації. Підтримує широкий спектр типів діаграм та має зручний API. Часто згадується в порівняннях, як наприклад, у (LogRocket) [19];

– ApexCharts: безкоштовна та відкрита бібліотека з простим API та широким набором функцій. Висока продуктивність та легкість використання роблять її популярним вибором. (G2) пропонує порівняння ApexCharts з іншими бібліотеками [20].

Вибір фреймворку для розробки інтерфейсу веб–додатку візуалізації даних є критичним рішенням, що залежить від багатьох факторів, серед яких досвід розробників та масштаб проекту. Для великих та складних проектів, що вимагають високої продуктивності, масштабованості та широкого набору функцій, часто обирають Angular – потужний фреймворк від Google, який пропонує багатий набір інструментів та можливостей для створення складних та інтерактивних інтерфейсів. Його структура, орієнтована на компоненти, дозволяє ефективно управляти великими проектами та забезпечує високу якість коду. Однак, Angular має досить високий поріг входу, тому вимагає від розробників певного рівня досвіду [21]. На противагу цьому, React, ще один надзвичайно популярний фреймворк, відомий своєю гнучкістю та ефективністю, також є чудовим вибором для складних проектів. Його компонентна архітектура та віртуальний DOM забезпечують високу продуктивність та зручність розробки, при цьому надаючи достатньо можливостей для кастомізації та розширення функціональності [22]. Для менших за масштабом проектів або для команд з обмеженим досвідом, Vue.js може бути більш підходящим варіантом. Він відрізняється простотою вивчення та використання, пропонуючи при цьому достатньо можливостей для створення якісних та функціональних інтерфейсів. Його легкість та гнучкість роблять його ідеальним вибором для швидкої розробки та прототипування, особливо для проектів, де пріоритетом є швидкість розробки та простота обслуговування [23]. В кінцевому підсумку, оптимальний вибір фреймворку залежить від

специфічних потреб проекту, досвіду розробників та балансу між функціональністю, продуктивністю та швидкістю розробки.

Вибір системи баз даних для веб–додатку візуалізації даних є критичним рішенням, що безпосередньо впливає на продуктивність, масштабованість та загальну архітектуру системи. Це рішення залежить від характеру та об'єму даних, які необхідно зберігати та обробляти. Якщо дані мають чітку структуру, з визначеними полями та зв'язками між ними, то найбільш підходящим варіантом будуть реляційні бази даних SQL, такі як MySQL або PostgreSQL. Вони забезпечують високу надійність, цілісність даних та ефективний пошук за допомогою структурованих запитів. SQL–бази даних добре підходять для проектів з великою кількістю структурованих даних, де важлива цілісність та транзакційність. Однак, для великих об'ємів даних або даних з динамічною структурою, SQL–бази даних можуть бути неефективними [24]. В таких випадках доречніше використовувати NoSQL–бази даних, такі як MongoDB або Cassandra. NoSQL–бази даних краще справляються з неструктурованими або напівструктурованими даними, дозволяючи гнучкіше моделювання даних та масштабування системи відповідно до зростання об'єму даних. MongoDB, наприклад, є документо–орієнтованою базою даних, що добре підходить для проектів з великою кількістю даних змінної структури, тоді як Cassandra – це розподілена база даних, що забезпечує високу доступність та масштабованість для великих та розподілених систем. Вибір між SQL та NoSQL залежить від специфічних потреб проекту, враховуючи тип даних, об'єм даних, вимоги до продуктивності та масштабованості, а також досвід розробників. Важливо ретельно оцінити всі фактори, щоб прийняти оптимальне рішення, що забезпечить ефективну роботу веб–додатку та його здатність обробляти великі об'єми даних [25].

Вибір серверної технології для веб–додатку візуалізації даних тісно пов'язаний з вибором бази даних та загальною архітектурою проекту, а також залежить від навичок та переваг розробників. Node.js, наприклад, є популярною платформою, що дозволяє використовувати JavaScript як на стороні клієнта, так

і на стороні сервера, забезпечуючи єдиний технологічний стек та спрощуючи розробку. Це особливо зручно, якщо фронтенд також розробляється на JavaScript, оскільки це дозволяє використовувати спільний код та спрощує комунікацію між клієнтом та сервером. Однак, Node.js може бути менш ефективним для задач, що вимагають інтенсивних обчислень, оскільки він є однопотоковим. Python [26], з іншого боку, є універсальною мовою програмування з великим та активним ком'юніті, що пропонує широкий вибір бібліотек для роботи з даними, машинного навчання та інших задач, пов'язаних з обробкою та аналізом даних. Його простота та читабельність коду роблять його зручним для розробки як невеликих, так і великих проектів. Крім того, Python має потужні фреймворки для веб–розробки, такі як Django та Flask, що спрощують процес створення веб–додатків. PHP [27], ще одна поширена мова веб–розробки, має довгу історію та велику кількість готових рішень, що може бути перевагою для проектів з обмеженим бюджетом та часом. Однак, PHP може бути менш гнучким та масштабованим порівняно з Python або Node.js, особливо для великих та складних проектів. В кінцевому підсумку, вибір серверної технології залежить від конкретних вимог проекту, досвіду команди та балансу між продуктивністю, масштабованістю, швидкістю розробки та витратами. Важливо враховувати сумісність обраної технології з обраною базою даних та іншими компонентами системи для забезпечення ефективної та надійної роботи веб–додатку.

Оптимальний вибір технологій для розробки веб–додатку візуалізації даних є критично важливим рішенням, яке залежить від безлічі факторів, включаючи масштаб проекту, складність візуалізацій, об'єм та тип даних, вимоги до продуктивності та масштабованості, а також досвід розробників. Для невеликих проектів з простими потребами та обмеженим об'ємом даних, де пріоритетом є швидкість розробки та простота використання, можна успішно використовувати легкі та прості у вивченні технології, такі як Chart.js для візуалізації та Vue.js для розробки інтерфейсу. Ці технології забезпечують достатню функціональність для створення простих, але ефективних візуалізацій.

Однак, для великих та складних проектів з великим об'ємом даних, високими вимогами до продуктивності та інтерактивності, потрібні більш потужні інструменти. У таких випадках D3.js, відома своєю гнучкістю та можливостями створення складних та кастомних візуалізацій, може стати оптимальним вибором. Разом з D3.js, React або Angular можуть забезпечити високу продуктивність та масштабованість інтерфейсу, дозволяючи ефективно управляти великими об'ємами даних та створювати складні та інтерактивні візуалізації. Для обробки та зберігання великих об'ємів даних знадобиться потужна база даних, вибір якої залежить від характеру даних (SQL для структурованих даних або NoSQL для неструктурованих). Серверна технологія також повинна бути обрана відповідно до вимог проекту та обраних інших технологій, забезпечуючи ефективну обробку запитів та взаємодію з базою даних. Продуктивність системи залежить від безлічі факторів, включаючи об'єм даних, складність візуалізацій, оптимізацію коду, вибір технологій та ефективність їх взаємодії. Хоча існують різні бенчмарки та порівняння продуктивності різних технологій, наприклад, на GitHub [28], результати таких порівнянь слід розглядати в контексті конкретних умов тестування та не повинні бути єдиним критерієм вибору. Оптимальний вибір технологій – це завжди компроміс між різними факторами, і найкращий підхід – це ретельний аналіз вимог проекту та вибір технологій, що найкраще відповідають цим вимогам.

1.5 Аналіз архітектури веб–додатків

Вибір архітектурного патерну для веб–додатку візуалізації даних є фундаментальним рішенням, що значно впливає на його довгострокову життєздатність. Ефективна архітектура забезпечує масштабованість, високу продуктивність, легкість підтримки та загальну якість коду. Неправильний вибір може призвести до складнощів у розробці, обслуговуванні та подальшому розвитку проекту. Оптимальний патерн залежить від специфічних потреб

проекту, включаючи об'єм даних, складність візуалізацій, вимоги до інтерактивності та масштабованості, а також досвіду розробників.

Розглянемо детальніше класичний патерн MVC (Model–View–Controller) та його придатність для веб–додатку візуалізації. MVC розділяє додаток на три чітко визначені компоненти: модель, вид та контролер. Модель відповідає за представлення даних та їхню логіку, абстрагуючись від деталей їхнього зберігання та обробки. В контексті візуалізації, модель міститиме дані, які будуть візуалізовані, та логіку їхньої підготовки до відображення. Вид відповідає за візуальне представлення даних, тобто за те, як дані будуть відображені користувачу. Для веб–додатку це може бути HTML–шаблон, який використовує дані з моделі для рендерингу графіків, діаграм та інших візуальних елементів. Контролер, в свою чергу, виступає посередником між моделлю та видом, керуючи потоком даних та взаємодією користувача. Він отримує запити від користувача (наприклад, вибір фільтрів, зміна масштабу), обробляє їх, взаємодіє з моделлю для отримання необхідних даних та передає оновлені дані виду для відображення [29].

Хоча MVC є добре зрозумілим та широко використовуваним патерном, він має свої обмеження. Для великих та складних проектів, взаємодія між компонентами може стати заплутаною, що призводить до труднощів у підтримці та розширенні коду. В контексті веб–додатку візуалізації, MVC може бути ефективним для відносно простих візуалізацій з невеликим об'ємом даних та обмеженою інтерактивністю. Однак, для складних інтерактивних візуалізацій з великою кількістю даних та динамічних елементів, MVC може виявитися недостатньо гнучким та масштабованим. В таких випадках, більш досконалі патерни, такі як MVVM чи MVP, можуть бути більш підходящими.

MVVM (Model–View–ViewModel) – це архітектурний патерн, який є еволюційним розвитком MVC, що вирішує деякі його недоліки, особливо стосовно складності та тестування. В MVVM додається додатковий шар – ViewModel, який виступає посередником між Моделлю та Видом. Цей шар абстрагує логіку представлення даних від логіки їх обробки. Модель, як і в

MVC, залишається відповідальною за зберігання та маніпулювання даними, але взаємодія з нею відбувається виключно через ViewModel [30].

ViewModel презентує дані у форматі, зручному для Виду. Він обробляє дані, отримані з Моделі, перетворюючи їх у структуру, яка легко може бути використана для рендерингу у Виді. Це може включати форматування дат, обчислення сумарних значень, фільтрування та сортування даних тощо. Крім того, ViewModel обробляє команди від користувача, отримані через Вид. Наприклад, якщо користувач натискає кнопку "Фільтрувати дані", ViewModel отримує цю команду, обробляє її, звертається до Моделі для отримання відфільтрованих даних та оновлює свій стан, що призводить до оновлення Виду.

Ключова перевага MVVM – це розділення відповідальності та спрощення тестування. Вид залежить тільки від ViewModel, а не від Моделі. Це дозволяє тестувати кожен компонент окремо, без залежності від інших. Зміна в Моделі не вплине на Вид, якщо ViewModel не зміниться. Це спрощує розробку та підтримку коду, оскільки зміни в одному компоненті менш імовірно призведуть до помилок в інших.

Для веб–додатку візуалізації даних, MVVM є особливо ефективним, особливо для складних візуалізацій з великою кількістю інтерактивних елементів. ViewModel може обробляти складні логічні операції, пов'язані з візуалізацією, та забезпечувати простий інтерфейс для Виду. Це дозволяє розробникам зосередитися на створенні ефективних та інтерактивних візуалізацій, не заглиблюючись в деталі обробки даних і логіки взаємодії з Моделлю. Загалом, MVVM пропонує кращу структуру та підтримуваність коду порівняно з MVC, що робить його вигідним вибором для більшості веб–додатків візуалізації даних, особливо для великих та складних проектів.

MVP (Model–View–Presenter) – це архітектурний патерн, який поділяє додаток на три основні компоненти: модель, вид та презентер. Цей патерн надає ще більш чітке розділення відповідальності, ніж MVC, що призводить до кращої підтримуваності та спрощує тестування [31].

Модель (Model) в MVP, як і в інших патернах, відповідає за представлення даних та бізнес–логіку. Вона абстрагує доступ до даних та їхню обробку від представлення. В контексті веб–додатку візуалізації, модель містить дані, які будуть візуалізовані, та методи для їхньої обробки.

Вид (View) в MVP є пасивним компонентом, який відповідає виключно за візуальне представлення даних. Він не містить ніякої логіки обробки даних або взаємодії з користувачем. Вид просто відображає дані, отримані від Презентера. В веб–додатку це може бути HTML–шаблон, що використовує дані, отримані від Презентера, для рендерингу графіків, діаграм тощо.

Презентер (Presenter) є ключовим компонентом в MVP. Він відповідає за обробку даних, взаємодію з Моделлю та керування Видом. Презентер отримує запити від користувача через Вид, обробляє їх, взаємодіє з Моделлю для отримання необхідних даних та передає оновлені дані Виду для відображення. Презентер також відповідає за валідацію даних та обробку помилок.

Ключова перевага MVP – це висока ступінь розділення відповідальності. Кожен компонент має чітко визначену роль, що спрощує розробку, тестування та підтримку коду. Тестування стає простішим, оскільки кожен компонент можна тестувати окремо, без залежності від інших. Однак, MVP може бути більш складним в розробці, ніж MVC або MVVM, оскільки вимагає більшої кількості коду та більш складної архітектури.

Для веб–додатку візуалізації даних, MVP може бути доречним, якщо потрібна висока ступінь розділення відповідальності та простота тестування є пріоритетом. Однак, потрібно враховувати більшу складність розробки та підтримки коду. Вибір між MVC, MVVM та MVP залежить від конкретних вимог проекту та компромісу між складністю розробки та якостю коду.

Вибір архітектурного патерну тісно пов'язаний з підходами до розробки. Монолітна архітектура, де всі компоненти додатка об'єднані в єдиний блок, проста у розробці, але може бути обмежена у масштабованості та продуктивності для великих проектів. Мікросервісна архітектура, де додаток розділений на

незалежні сервіси, забезпечує кращу масштабованість та продуктивність, але вимагає більших зусиль у розробці та обслуговуванні. Вибір підходу залежить від масштабу проекту та його вимог до продуктивності та масштабованості.

В заключенні, вибір архітектури та підходу до розробки веб–додатку візуалізації даних є важливим рішенням, що вимагає ретельного аналізу вимог проекту та врахування досвіду розробників. Оптимальний вибір забезпечить високу якість, продуктивність та масштабованість додатка.

1.6 Забезпечення безпеки веб–додатків

Безпека веб–додатку візуалізації даних є критично важливим аспектом, який вимагає ретельного планування та реалізації заходів захисту від різних типів загроз. Ігнорування безпеки може призвести до серйозних наслідків, включаючи втрату даних, фінансові втрати та пошкодження репутації. Цей розділ охоплює ключові аспекти забезпечення безпеки веб–додатку.

Захист від поширених вразливостей [32]:

– SQL–ін'єкції: це один з найпоширеніших типів атак, що дозволяє зловмисникам виконувати власні SQL–запити до бази даних. Захист від SQL–ін'єкцій досягається за допомогою параметризованих запитів (prepared statements) або використання ORM (Object–Relational Mapper), які автоматично екранують шкідливі символи. Важливо уникати безпосереднього вбудовування даних користувача в SQL–запити;

– XSS (Cross–Site Scripting): цей тип атак дозволяє зловмисникам вводити шкідливий JavaScript–код на веб–сторінку, що виконується в браузері користувача. Захист від XSS досягається за допомогою екранування виводу (output encoding), використання Content Security Policy (CSP) та інших методів, що обмежують виконання небезпечного коду [33];

– CSRF (Cross–Site Request Forgery): цей тип атак змушує користувача виконувати небажані дії на веб–сайті, використовуючи його авторизовану сесію.

Захист від CSRF досягається за допомогою токенів CSRF (CSRF tokens), які є унікальними для кожного запиту та перевіряються на сервері [34];

– захист від інших атак: крім перерахованих вище, існують інші типи атак, таких як DoS (Denial of Service), Clickjacking, Session Hijacking тощо. Захист від цих атак вимагає комплексного підходу, що включає використання firewall'ів, WAF (Web Application Firewall), регулярне оновлення програмного забезпечення та інші заходи безпеки.

Автентифікація та авторизація є двома критично важливими аспектами безпеки будь–якого веб–додатку, включаючи веб–додаток для візуалізації даних. Вони працюють разом, щоб забезпечити безпечний доступ до ресурсів та даних. Неправильна реалізація цих механізмів може призвести до серйозних порушень безпеки.

Автентифікація: це процес перевірки того, хто є користувач. Вона відповідає на питання: "Хто ти?". Існує багато методів автентифікації, кожен з яких має свої переваги та недоліки:

– імена користувачів та паролі: це найпоширеніший, але й найменш безпечний метод. Слабкі паролі та методи грубого підбору можуть легко зламати систему. Для підвищення безпеки важливо використовувати надійні алгоритми хешування паролів (наприклад, bcrypt, Argon2), що ускладнюють розшифровку паролів навіть при отриманні доступу до бази даних. Також важливо вимагати складних паролів, що містять великі та малі літери, цифри та спеціальні символи;

– многофакторна автентифікація (MFA): цей метод вимагає від користувача надати декілька факторів автентифікації, наприклад, пароль та код з SMS–повідомлення або з автентифікатора. MFA значно підвищує рівень безпеки, оскільки навіть якщо зловмисник отримає доступ до пароля, він не зможе увійти в систему без другого фактора [35];

– OAuth 2.0: це відкритий стандарт для делегування доступу до ресурсів без передачі пароля. Користувач авторизується через сторонню службу (наприклад, Google, Facebook), а додаток отримує токен доступу, який

використовується для доступу до ресурсів. OAuth 2.0 спрощує процес автентифікації та підвищує безпеку, оскільки пароль користувача ніколи не передається безпосередньо додатку;

– біометрична автентифікація: використання біометричних даних, таких як відбитки пальців, розпізнавання обличчя або сканування райдужної оболонки ока, для автентифікації. Цей метод є дуже безпечним, але може бути дорогим та вимагати спеціального обладнання [36].

Авторизація: це процес перевірки того, що користувач має право доступу до певних ресурсів. Вона відповідає на питання: "Що ти можеш робити?". Авторизація зазвичай базується на ролях та привілеях користувачів. Користувачі можуть мати різні ролі (наприклад, адміністратор, редактор, користувач), кожна з яких має певний набір прав доступу.

Ефективна система авторизації повинна забезпечувати:

– принцип найменших привілеїв: користувачі повинні мати доступ тільки до тих ресурсів, які їм необхідні для виконання своєї роботи;

– централізоване управління доступом: управління правами доступу повинно бути централізованим та простим у використанні;

– аудит доступу: система повинна вести журнали всіх дій користувачів, щоб можна було відстежувати та аналізувати доступ до ресурсів.

Комбінація надійної автентифікації та ретельно продуманої системи авторизації є ключем до забезпечення безпеки веб–додатку. Вибір конкретних методів залежить від специфічних вимог проекту та рівня безпеки, який потрібно забезпечити.

Захист даних та забезпечення конфіденційності є невід'ємними складовими безпеки веб–додатку візуалізації даних. З огляду на те, що додаток працює з даними, які можуть бути конфіденційними або критично важливими, ретельний підхід до захисту даних є обов'язковим. Це включає в себе комплексний набір заходів, що охоплюють різні аспекти обробки та зберігання даних. Шифрування є фундаментальним елементом захисту даних як під час передачі (в транзиті), так і під час зберігання (в спокої). Використання

протоколу HTTPS для всіх комунікацій між клієнтом та сервером є обов'язковим для захисту даних від перехоплення зловмисниками [37]. HTTPS забезпечує шифрування даних за допомогою SSL/TLS протоколу, що робить їх нечитабельними для сторонніх осіб. Крім того, важливо шифрувати дані, що зберігаються в базі даних, використовуючи відповідні алгоритми шифрування, що залежать від типу даних та рівня необхідної безпеки. Ретельний контроль доступу до даних на всіх рівнях – від бази даних до рівня додатку – є критично важливим. Система контролю доступу повинна бути добре спроектована та налаштована, щоб забезпечити доступ до даних тільки авторизованим користувачам з відповідними правами. Це включає в себе використання ролей та привілеїв, а також механізмів автентифікації та авторизації, що описані вище. Для ефективного моніторингу та розслідування інцидентів безпеки необхідна система журналювання, яка реєструє всі дії користувачів та події безпеки. Журнали повинні містити детальну інформацію про час, користувача, дію та інші релевантні дані. Регулярний аналіз журналів дозволяє виявляти підозрілу активність та своєчасно реагувати на загрози. Регулярне оновлення програмного забезпечення та бібліотек є невід'ємною частиною стратегії безпеки. Застаріле програмне забезпечення часто містить вразливості, які можуть бути використані зловмисниками для доступу до даних або порушення функціональності системи. Тому важливо своєчасно оновлювати всі компоненти системи, включаючи операційну систему сервера, базу даних, веб–сервер та сам веб–додаток, використовуючи останні версії з виправленнями безпеки. Крім того, регулярний аудит безпеки та пентест (пенетраційне тестування) дозволяють виявити потенційні вразливості та усунути їх до того, як вони будуть використані зловмисниками. Забезпечення конфіденційності даних також вимагає дотримання відповідних законів та нормативних актів щодо захисту даних, таких як GDPR (Загальний регламент про захист даних) або CCPA (Закон про конфіденційність споживачів в Каліфорнії). Це включає в себе надання користувачам чіткої та зрозумілої інформації про те, як їхні дані збираються, використовуються та захищаються.

Забезпечення безпеки веб–додатку – це безперервний процес, що вимагає постійного моніторингу та оновлення заходів безпеки. Важливо враховувати всі аспекти безпеки на етапі проектування та розробки додатку, а не лише після його завершення.

1.7 Висновки аналізу предметної області

Аналіз предметної області показав, що розробка комп'ютеризованої системи для візуалізації даних є складним процесом, що включає інтеграцію сучасних технологій веб–розробки, таких як HTML, CSS та JavaScript. Візуалізація даних потребує розуміння основних принципів та вибору методів, що найкраще підходять для представлених даних. Архітектура веб–додатків повинна бути масштабованою, продуктивною та безпечною, а забезпечення безпеки додатків має ключове значення для захисту інформації користувачів.

2 ОБГРУНТУВАННЯ ВАЖЛИВОСТІ ДОСЛІДЖЕННЯ

2.1 Аналіз існуючих рішень

Цей розділ присвячений детальному аналізу існуючих систем візуалізації даних, аналогічних до розроблюваної системи для аналізу продажів електроніки. Аналіз включає дослідження функціональності, архітектури, використаних технологій та порівняння з розроблюваною системою, обґрунтування її переваг.

Дослідження аналогічних систем: згідно з наданими результатами пошуку (Retool, ClickUp, Competitors.app, Intelemark, Qlik, PeerPanda, Improvado, Geckoboard, Salesforce, Klipfolio, Cvent, BDC.ca, Sprout Social, HubSpot), існують різні системи для аналізу конкурентів та продажів. Вони пропонують різні рівні функціональності, від простих дашбордів до складних систем бізнес–аналітики [38–52].

Переваги та недоліки існуючих рішень:

– переваги: багато існуючих рішень пропонують готові шаблони дашбордів, інтеграцію з різними джерелами даних та зручні інструменти для візуалізації даних. Деякі системи пропонують розширені можливості аналізу та прогнозування;

– недоліки: деякі системи можуть бути дорогими та складними у використанні. Функціональність деяких систем може бути обмеженою, не відповідаючи всім потребам користувачів. Інтеграція з певними системами може бути складною або неможливою. Не всі системи забезпечують достатній рівень безпеки та захисту даних.

Детальний аналіз функціональності, архітектури та технологій існуючих систем візуалізації даних вимагає окремого розгляду кожного конкретного рішення. Однак, загалом, можна виділити кілька поширених підходів та технологій. Щодо архітектури, багато сучасних систем використовують шаблон Model–View–Controller (MVC) або Model–View–ViewModel (MVVM), які

забезпечують розділення відповідальності та спрощують розробку та підтримку. Вибір між MVC та MVVM залежить від конкретних потреб проекту та переваг розробників. Бази даних, що використовуються в аналогічних системах, можуть варіюватися від реляційних баз даних (наприклад, PostgreSQL, MySQL) до NoSQL баз даних (наприклад, MongoDB), вибір залежить від характеру даних та вимог до продуктивності. Для візуалізації даних використовуються різні бібліотеки та інструменти. Популярними є комерційні рішення, такі як Tableau та Power BI, які пропонують широкі можливості для створення інтерактивних дашбордів та звітів. Крім того, широко використовуються JavaScript–бібліотеки, такі як D3.js, що дозволяють створювати складні та кастомізовані візуалізації. Вибір технології візуалізації залежить від вимог до функціональності, швидкодії та можливостей кастомізації. Порівняльний аналіз розроблюваної системи з існуючими рішеннями показує кілька ключових переваг. По–перше, спеціалізація на аналізі продажів електроніки дозволить створити систему, що враховує специфічні потреби компанії та особливості даних. Це забезпечить більш точний та ефективний аналіз, ніж використання універсальних рішень. По–друге, тісна інтеграція з існуючими системами компанії спростить обмін даними та автоматизує процеси, що призведе до економії часу та ресурсів. По–третє, гнучка та настроювана архітектура дозволить адаптувати систему під потреби конкретних користувачів, забезпечуючи зручний та ефективний інтерфейс для кожної категорії користувачів. По–четверте, розробка власної системи може бути більш економічно вигідною, ніж придбання та налаштування комерційних рішень, особливо враховуючи специфічні потреби компанії. Нарешті, система буде розроблена з урахуванням високих стандартів безпеки та захисту даних, що є критично важливим для захисту конфіденційної інформації. Детальний аналіз конкретних існуючих рішень та їх порівняння з розроблюваною системою буде проведено після завершення дослідження ринку та вивчення специфічних вимог компанії. Це дозволить більш точно оцінити переваги та недоліки кожного підходу та обґрунтувати вибір конкретних технологій та архітектурних рішень.

2.2 Вимоги до системи візуалізації

Цей розділ детально описує функціональні та нефункціональні вимоги до системи візуалізації даних про продажі електроніки. Вимоги обґрунтовуються з урахуванням потреб різних категорій користувачів (менеджери з продажу, маркетологи, аналітики, адміністратори) та загальних цілей системи – забезпечення ефективного аналізу даних для прийняття обґрунтованих бізнес–рішень.

Функціональні вимоги: функціональні вимоги визначають, які дії повинна виконувати система. Для системи візуалізації продажів електроніки це включає широкий спектр можливостей, розроблених з урахуванням потреб різних користувачів.

Завантаження та обробка даних: система повинна забезпечувати гнучкий та надійний імпорт даних з різних джерел. Це включає підтримку різних форматів даних (бази даних SQL та NoSQL, файли CSV, JSON, XML), а також можливість інтеграції через API з існуючими системами компанії (CRM, ERP, системи електронної комерції). Процес обробки даних повинен бути автоматизованим та включати в себе очищення даних (видалення дублікатів, корекція помилок), перетворення форматів та агрегацію даних на різних рівнях (щоденний, тижневий, місячний, річний) для зручного аналізу. Система повинна мати механізм контролю якості даних та повідомлення про помилки [53].

Візуалізація даних: система повинна пропонувати широкий спектр інтерактивних візуалізацій, адаптованих до різних типів даних та потреб користувачів. Це включає, але не обмежується: різні типи графіків (стовпчасті, лінійні, кругові, гістограми), діаграми (точкові, складені), географічні карти, таблиці та матриці даних. Кожна візуалізація повинна бути інтерактивною, дозволяючи користувачам фільтрувати, сортувати та виділяти окремі елементи даних для детального вивчення. Система повинна підтримувати кастомізацію візуалізацій (вибір кольорів, шрифтів, легенд тощо).

Фільтрація та сортування даних: система повинна надавати потужні інструменти для фільтрації та сортування даних за різними параметрами. Це включає фільтрацію за часовими періодами, географічними регіонами, категоріями товарів, ціновими діапазонами, маркетинговими кампаніями та іншими релевантними критеріями. Користувачі повинні мати можливість комбінувати різні фільтри для отримання точних результатів. Сортування даних повинно бути доступне за різними полями (наприклад, за кількістю продажів, прибутком, конверсією).

Генерація звітів: система повинна дозволяти генерувати настроювані звіти, що містять як візуалізації даних, так і текстові пояснення. Звіти повинні бути експортовані в різних форматах (PDF, Excel, CSV, HTML) для зручності використання та подальшої обробки. Користувачі повинні мати можливість зберігати шаблони звітів та автоматизувати їх генерацію за розкладом.

Керування користувачами та правами доступу: система повинна мати модуль керування користувачами, що дозволяє адміністраторам створювати нові акаунти, призначати ролі та встановлювати права доступу до різних частин системи та даних. Система повинна забезпечувати різні рівні доступу, забезпечуючи безпеку даних та конфіденційність.

Інтеграція з іншими системами: система повинна забезпечувати безперебійну інтеграцію з існуючими системами компанії (CRM, ERP, системи електронної комерції) через API або інші механізми обміну даними. Це дозволить автоматизувати процес збору та оновлення даних, уникнути дублювання інформації та забезпечити цілісність даних.

Нефункціональні вимоги: нефункціональні вимоги визначають якісні характеристики системи, тобто як вона повинна працювати, а не що вона повинна робити. Для системи візуалізації продажів електроніки це критично важливі аспекти, що впливають на її ефективність та зручність використання:

– продуктивність: система повинна демонструвати високу продуктивність, забезпечуючи швидке завантаження даних, генерування звітів та взаємодію з інтерактивними візуалізаціями. Час відгуку системи на запити користувача

повинен бути мінімальним (наприклад, менше 1 секунди для більшості операцій). Система повинна бути спроектована для ефективної обробки великих обсягів даних (кількість записів, обсяг даних) та одночасного доступу великої кількості користувачів (наприклад, до 100 одночасних користувачів без суттєвого зниження продуктивності). Це вимагає оптимізації бази даних, вибору ефективних алгоритмів та масштабованої архітектури;

– масштабованість: система повинна бути легко масштабованою для обробки зростаючих обсягів даних та кількості користувачів. Це означає, що архітектура системи повинна бути розроблена таким чином, щоб її можна було легко розширювати шляхом додавання нових серверів, баз даних та інших ресурсів без суттєвого перепроектування. Використання хмарних технологій та горизонтального масштабування є ключовими аспектами забезпечення масштабованості;

– надійність: система повинна бути надійною та стійкою до збоїв. Це вимагає використання резервного копіювання даних, механізмів відновлення після збоїв та моніторингу системи для своєчасного виявлення та усунення проблем. Високий рівень доступності (uptime) є критичним для забезпечення безперебійної роботи системи. Система повинна бути стійкою до помилок та забезпечувати захист даних від втрати та пошкодження;

– безпека: захист даних є критично важливим аспектом. Система повинна забезпечувати високий рівень безпеки даних, захищаючи їх від несанкціонованого доступу, модифікації та витоку інформації. Це включає використання надійних механізмів аутентифікації та авторизації, шифрування даних під час передачі та зберігання, контроль доступу на основі ролей та регулярне оновлення програмного забезпечення для усунення вразливостей. Система повинна відповідати відповідним стандартам безпеки та нормативним актам;

– зручність використання (юзабіліті): система повинна бути інтуїтивно зрозумілою та легкою у використанні для всіх категорій користувачів, незалежно від їхнього рівня технічної підготовки. Інтерфейс користувача

повинен бути простим, логічним та естетично привабливим. Навігація по системі повинна бути інтуїтивною, а функції – легкодоступними. Система повинна мати зрозумілу та детальну документацію та підтримку користувачів. Тестування юзабіліті з участю представників різних категорій користувачів є важливим етапом розробки.

Обґрунтування вимог: кожна вимога обґрунтовується потребами користувачів та цілями системи. Наприклад, вимога до швидкої продуктивності обґрунтовується тим, що менеджери потребують оперативного доступу до даних для прийняття рішень. Вимога до безпеки обґрунтовується необхідністю захисту конфіденційної інформації про продажі та клієнтів. Вимога до інтеграції з іншими системами обґрунтовується бажанням автоматизувати процеси та уникнути дублювання даних. Детальніше обґрунтування кожної вимоги буде надано в технічному завданні.

2.3 Причини вибору теми

У процесі цифровізації сучасного суспільства особливу роль відіграє розробка автоматизованих систем, які дозволяють ефективно обробляти, зберігати та візуалізувати дані. Щодня генерується величезна кількість інформації. Управління такими обсягами даних потребує створення інструментів, які не лише дозволяють зберігати інформацію, але й надавати її у зрозумілому вигляді для користувача. Візуалізація даних є одним із найефективніших способів подання інформації, що полегшує її аналіз, прийняття рішень та інтерпретацію. Сучасні технології пропонують різні методи візуалізації, такі як графіки, діаграми, карти, дашборди та інтерактивні звіти. Вони дозволяють представити складну інформацію у вигляді, який легко розуміється користувачем. Важливою складовою є також можливість взаємодії з даними, що надає змогу глибше занурюватися в аналіз та отримувати більш детальні висновки. Крім того, візуалізація даних сприяє комунікації та співпраці у командах, полегшуючи обговорення результатів та прийняття колективних

рішень. З розвитком технологій штучного інтелекту та машинного навчання, візуалізація даних стає ще потужнішим інструментом, який може автоматично адаптуватися до потреб користувача та надавати індивідуалізовані рекомендації.

Вебдодатки стали основним інструментом доступу до інформації для більшості користувачів. Це обумовлено їх зручністю, доступністю з будь–якого пристрою та мінімальними вимогами до користувацького обладнання. Тема розробки автоматизованого пристрою для візуалізації даних у вигляді вебдодатка відповідає сучасним тенденціям у сфері інформаційних технологій. Розробка такого пристрою передбачає використання сучасних фреймворків та бібліотек, що дозволяють створювати швидкі та адаптивні інтерфейси користувача. Це включає застосування таких технологій, як React, Angular або Vue.js, які забезпечують високу продуктивність та масштабованість додатків. Крім того, важливо приділяти увагу безпеці даних, щоб забезпечити надійний захист інформації від несанкціонованого доступу та кібератак. Автоматизація процесу візуалізації даних сприяє підвищенню ефективності роботи аналітиків та менеджерів, надаючи їм можливість швидко отримувати та інтерпретувати дані. Це особливо важливо в умовах швидкого розвитку бізнесу та необхідності прийняття оперативних рішень на основі актуальної інформації.

Ручна обробка даних часто є часозатратною та схильною до помилок. Автоматизовані рішення значно підвищують ефективність роботи, зменшують ймовірність помилок та дозволяють оперативно отримувати результати. Розробка автоматизованого пристрою вирішує завдання автоматизації збору, обробки та подання даних. У поєднанні з хмарними технологіями, такі пристрої можуть забезпечити масштабованість та доступність обробки даних незалежно від географічного розташування користувача. Хмарні сервіси, як–от Amazon Web Services, Google Cloud та Microsoft Azure, надають необхідні інструменти та ресурси для розгортання та підтримки автоматизованих систем візуалізації даних. Таким чином, впровадження автоматизованих рішень не тільки дозволяє зекономити час та ресурси, але й сприяє більш точному та швидкому прийняттю

рішень на основі актуальних даних, що є критичним для успіху в сучасному конкурентному середовищі.

Вибір цієї теми зумовлений також прагненням отримати практичний досвід у розробці вебдодатків, роботі з базами даних та візуалізації даних. Це дозволяє розширити технічні навички, необхідні для роботи в сучасній IT–індустрії, та створити реальний продукт, який може бути використаний для вирішення конкретних завдань. Важливим аспектом цього проекту є можливість застосування знань на практиці, що сприяє глибшому розумінню теоретичних концепцій та їх застосуванню у реальних умовах. Робота над проектом включає не лише технічні навички, але й розвиток навичок командної роботи, управління проектами та вирішення проблем, що є критично важливими для успішної кар'єри в IT–сфері.

Існуючі рішення для візуалізації даних часто є універсальними і недостатньо адаптованими до специфічних потреб користувача. Обрана тема дозволяє створити гнучкий і адаптований інструмент, який буде ефективно виконувати поставлені завдання. Основною перевагою такого підходу є можливість налаштування інструменту під конкретні вимоги та умови роботи користувача. Це означає, що він буде враховувати особливості різних галузей, специфіку даних та індивідуальні потреби користувача. Завдяки цьому користувачі зможуть отримувати більш точну та релевантну інформацію, що сприятиме прийняттю обґрунтованих рішень. Крім того, створення власного візуалізаційного інструменту дозволяє інтегрувати його з іншими системами та базами даних, що вже використовуються в організації. Це забезпечує більш ефективну обробку та аналіз даних, зменшує ймовірність помилок та спрощує процеси взаємодії між різними системами.

Автоматизація процесів обробки даних сприяє підвищенню продуктивності праці, оптимізації бізнес–процесів та зниженню витрат. Такі рішення мають потенціал для впровадження у різних галузях, включаючи освіту, бізнес, медицину та державне управління. Впровадження автоматизованих систем в освіті може допомогти створити більш персоналізовані навчальні

програми, полегшити управління великими обсягами даних про студентів та покращити якість освітніх ресурсів. У бізнесі такі рішення дозволяють ефективніше аналізувати ринкові тенденції, керувати виробничими процесами та підвищувати задоволеність клієнтів шляхом надання більш точних та актуальних даних. У медицині автоматизація може сприяти покращенню діагностики, управління медичними записами та підтримки прийняття клінічних рішень, що зрештою підвищить якість медичного обслуговування. Державне управління, у свою чергу, може скористатися перевагами автоматизованих систем для покращення прозорості, підвищення ефективності надання послуг та оптимізації управлінських процесів.

Загалом, автоматизація обробки даних відкриває нові можливості для різних секторів економіки, сприяючи підвищенню їхньої продуктивності та конкурентоспроможності у сучасному світі.

2.4 Висновки обгрунтування важливості дослідження

Обґрунтування важливості дослідження базується на аналізі існуючих рішень, вимогах до системи візуалізації та причинах вибору теми. Аналіз існуючих рішень показав, що більшість з них мають обмежену функціональність або не задовольняють потреби користувачів у повній мірі. Тому важливо визначити чіткі вимоги до системи візуалізації, які включають інтерактивність, адаптивність до різних пристроїв та простоту у використанні. Причини вибору теми пов'язані з необхідністю покращення процесу візуалізації даних, що дозволить забезпечити більш ефективний аналіз та прийняття рішень на основі представлених даних.

3 РОЗРОБКА КОМП'ЮТЕРИЗОВАНОЇ СИСТЕМИ

3.1 Розробка бази даних

Для реалізації проекту було обрано MySQL як систему керування базами даних через її численні переваги, які роблять її оптимальним вибором для цього завдання. MySQL є однією з найпопулярніших баз даних у світі, що використовується багатьма розробниками та компаніями. Її популярність забезпечує широкий доступ до ресурсів, документації та підтримки спільноти, що є важливим для ефективного розвитку та обслуговування проекту.

Однією з головних причин вибору MySQL є її висока продуктивність і швидкість роботи, що дозволяє ефективно обробляти великі обсяги даних і виконувати складні запити. Важливим фактором також є простота використання цієї бази даних: зрозумілий синтаксис та зручні інструменти адміністрування роблять її доступною навіть для початківців. Крім того, MySQL відзначається масштабованістю, що дозволяє використовувати її як для невеликих проектів, так і для великих систем, які потребують обробки великих обсягів інформації.

Ще однією вагомою перевагою MySQL є її інтеграція з сучасними технологіями, такими як Spring Framework, який використовується у проекті. Це забезпечує простоту налаштування підключення та зручність взаємодії між компонентами системи. Безкоштовна ліцензія MySQL є додатковою перевагою, яка робить її ідеальним вибором для освітніх і невеликих комерційних проектів. Її надійність та стабільність підтверджені багаторічним використанням у критично важливих додатках, що забезпечує впевненість у збереженні та обробці даних.

Таким чином, вибір MySQL як бази даних обумовлений її високою швидкістю, простотою використання, можливістю масштабування, інтеграцією з іншими технологіями та безкоштовною ліцензією. Усі ці характеристики

забезпечують ефективне зберігання, обробку та доступ до даних, що є критично важливим для успішної реалізації проекту.

Розроблена схема бази даних обрана з урахуванням потреб проекту для зберігання, обробки та структурованого управління інформацією. Кожна таблиця виконує конкретну роль, забезпечуючи логічний поділ даних, що полегшує масштабування, підтримку та інтеграцію з вебдодатком. Схема представлена на рис. 3.1–3.2.

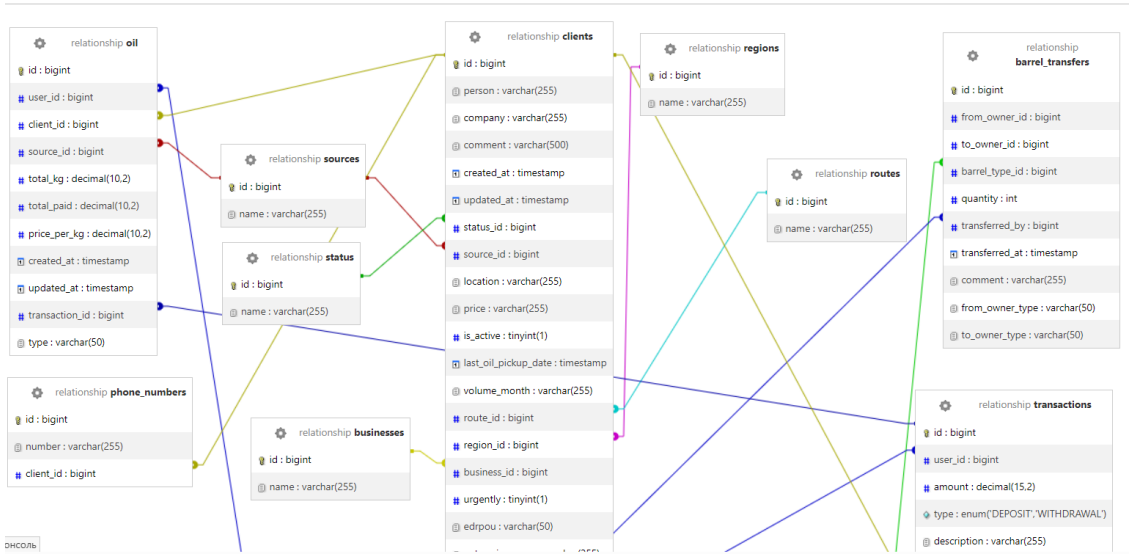


Рисунок 3.1 – Схема бази даних

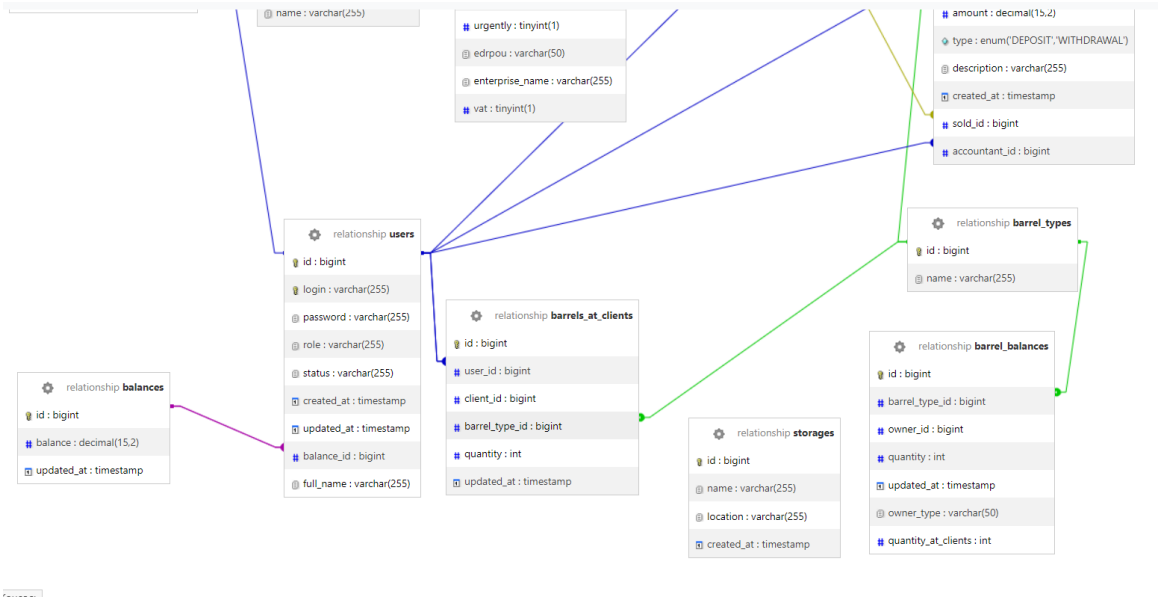


Рисунок 3.2 – Схема бази даних

Таблиця balances – ця таблиця використовується для зберігання фінансових балансів користувачів або інших об'єктів. Поля:

– balance зберігає точну суму в форматі з десятковими числами;

– updated_at відстежує останнє оновлення балансу.

Така структура дозволяє легко отримувати актуальну фінансову інформацію для звітів або операцій.

Таблиця barrels_at_clients– ця таблиця відображає кількість бочок різних типів, закріплених за клієнтами. Поля:

– user_id, client_id, barrel_type_id дозволяють відстежувати, хто та якому клієнту передав конкретний тип бочок;

– quantity вказує кількість бочок.

Ця таблиця дозволяє управляти даними щодо логістики та контролю активів.

Таблиця barrel_balances. Містить інформацію про залишки бочок на складах або в інших власників. Поле owner_type визначає, чи є власником клієнт, користувач чи інша організація. Це забезпечує універсальність і можливість легко розподіляти дані між різними суб’єктами.

Таблиця barrel_transfers. Відображає історію передачі бочок між власниками. Поля from_owner_type і to_owner_type дозволяють деталізувати типи учасників трансферу (клієнти, склади тощо). Це допомагає відслідковувати рух активів.

Таблиця barrel_types. Містить довідкову інформацію про типи бочок. Простота цієї таблиці забезпечує легкість розширення списку типів.

Таблиця businesses. Зберігає дані про компанії, які беруть участь у системі. Ця таблиця спрощує інтеграцію клієнтів і відокремлення їхніх бізнесових даних.

Таблиця clients. Основна таблиця для зберігання інформації про клієнтів. Велика кількість полів дозволяє зберігати:

– ім'я особи або компанії (person, company);

– дані про статуси, джерела, маршрути, регіони, а також специфічні фінансові параметри.

Це забезпечує гнучкість у роботі з клієнтами.

Таблиця oil. Фіксує операції, пов'язані з олією: її кількість, вартість та транзакції. Поле type дозволяє відрізняти різні категорії олії або операцій.

Таблиця phone_numbers. Зберігає контактні номери клієнтів, що дозволяє зв’язувати їх із таблицею clients. Ця структура спрощує роботу з багатьма номерами для одного клієнта.

Таблиця regions. Містить перелік регіонів для маршрутизації та зручної фільтрації клієнтів за місцем розташування.

Таблиця routes. Містить інформацію про маршрути. Використовується для організації логістики та оптимізації процесів доставки або передачі товарів.

Таблиця sources. Ця таблиця містить джерела походження клієнтів або товарів. Вона корисна для аналізу та маркетингових цілей.

Таблиця status. Довідкова таблиця статусів клієнтів або об’єктів у системі. Полегшує підтримку єдиних стандартів статусів.

Таблиця storages. Зберігає дані про склади, включаючи назви та місце розташування. Використовується для відстеження активів і логістичних операцій.

Таблиця transactions. Реалізує облік фінансових операцій. Поле type дозволяє розділяти депозити та зняття коштів, що важливо для звітності.

Таблиця users. Центральна таблиця для зберігання даних про користувачів системи. Поля role та status дозволяють керувати правами доступу і відображати активність користувачів.

Така структура бази даних забезпечує логічний розподіл інформації між таблицями, що спрощує управління даними, підвищує продуктивність запитів і дозволяє легко масштабувати систему в майбутньому.

3.2 Архітектура безпеки системи

У даному розділі розглядається реалізація механізмів безпеки в автоматизованому пристрої для візуалізації даних з бази даних на веб–сайті.

Безпека є критично важливим аспектом будь–якої системи, яка працює з конфіденційними даними, тому для забезпечення захисту від несанкціонованого доступу та інших загроз було використано фреймворк Spring Security у поєднанні з JWT (JSON Web Token) для автентифікації та авторизації користувачів.

Система безпеки побудована на основі наступних компонентів.

Spring Security – фреймворк, який надає інструменти для захисту веб–додатків.

JWT (JSON Web Token) – токени, які використовуються для автентифікації користувачів та передачі інформації про їхні ролі.

BCryptPasswordEncoder – для хешування паролів користувачів.

CSRF–захист – вимкнений через використання JWT, оскільки токени забезпечують достатній рівень захисту від CSRF–атак.

Конфігурація Spring Security. Основна конфігурація безпеки реалізована в класі WebSecurityConfig. Цей клас містить наступні ключові елементи.

Вимкнення CSRF–захисту:

.csrf(AbstractHttpConfigurer::disable)

CSRF–захист вимкнений, оскільки JWT токени забезпечують достатній рівень захисту від CSRF–атак. Крім того, використання токенів у заголовках HTTP–запитів робить систему менш вразливою до таких атак.

Налаштування сесій:

.sessionManagement(session –>
session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))

Сесії в системі є безстатусними, що означає, що сервер не зберігає стан користувача між запитами. Це дозволяє масштабувати систему, оскільки кожен запит містить всю необхідну інформацію для автентифікації (JWT токен).

Налаштування доступу до ресурсів:

```
.authorizeHttpRequests(auth –> auth
    .requestMatchers(
        "/login",
        "/favicon.ico",
```

```
            "/api/v1/user/login",
            "/favicon/**",
            "/favicon/site.webmanifest",
            "/js/login.js",
            "/css/**"
        ).permitAll()
        .anyRequest().authenticated()
    )
```

Доступ до певних ресурсів (наприклад, сторінка входу, статичні файли) дозволений без автентифікації. Всі інші запити вимагають автентифікації користувача.

Хешування паролів:

```
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder(12);
}
```

Для зберігання паролів користувачів використовується алгоритм хешування BCrypt з силою 12. Це забезпечує високий рівень захисту паролів від зловмисників.

JWT–автентифікація. Для автентифікації користувачів використовується JWT токени. Основні компоненти для роботи з JWT.

JwtTokenProvider – Цей клас відповідає за створення, валідацію та розшифрування JWT токенів. Токени містять інформацію про користувача (логін та роль) та час дії токена. Створення токену:

```
 Copy
public String createToken(String login, String role) {
    Claims claims = Jwts.claims().setSubject(login);
    claims.put("role", role);
    Date now = new Date();
    Date validity = new Date(now.getTime() + validityMilliseconds * 1000);
```

```
        byte[] decodedKey =
Base64.getDecoder().decode(secretKeyConfig.getSecretKey());
        SecretKey originalKey = new SecretKeySpec(decodedKey, 0,
decodedKey.length, "HmacSHA256");

        return Jwts.builder()
                .setClaims(claims)
                .setIssuedAt(now)
                .setExpiration(validity)
                .signWith(originalKey, SignatureAlgorithm.HS256)
                .compact();
    }
```

Токен створюється з використанням секретного ключа, який зберігається в конфігурації системи. Токен містить інформацію про користувача (логін та роль), а також час дії токена.

Валідація токену:

Copy

```
    public boolean validateToken(String token) {
        try {
            byte[] decodedKey =
Base64.getDecoder().decode(secretKeyConfig.getSecretKey());
            SecretKey originalKey = new SecretKeySpec(decodedKey, 0,
decodedKey.length, "HmacSHA256");

            Jws<Claims> claimsJws = Jwts.parserBuilder()
                    .setSigningKey(originalKey)
                    .build()
                    .parseClaimsJws(token);
            return !claimsJws.getBody().getExpiration().before(new Date());
```

```
        } catch (JwtException | IllegalArgumentException e) {
            throw new JwtAuthenticationException("JWT token is expired or
invalid", HttpStatus.UNAUTHORIZED);
        }
    }
```

Токен валідується за допомогою секретного ключа. Якщо токен недійсний або прострочений, система повертає помилку 401 Unauthorized.

JwtTokenFilter – Цей фільтр перехоплює кожен HTTP–запит і перевіряє наявність JWT токена. Якщо токен валідний, система автентифікує користувача та надає доступ до ресурсів.

```
 Copy
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse
servletResponse, FilterChain filterChain)
            throws IOException, ServletException {
        HttpServletRequest httpRequest = (HttpServletRequest) servletRequest;
        HttpServletResponse httpResponse = (HttpServletResponse)
servletResponse;

        String token = jwtTokenProvider.resolveToken(httpRequest);

        if (token != null && jwtTokenProvider.validateToken(token)) {
            Authentication authentication =
jwtTokenProvider.getAuthentication(token);
            SecurityContextHolder.getContext().setAuthentication(authentication);
        }

        filterChain.doFilter(servletRequest, servletResponse);
    }
```

Обробка помилок автентифікації. У випадку невдалої автентифікації (наприклад, прострочений або недійсний токен), система перенаправляє користувача на сторінку входу та очищає куки автентифікації:

Copy

```
private void handleAuthenticationException(HttpServletRequest httpRequest, HttpServletResponse httpResponse) throws IOException {
    SecurityContextHolder.clearContext();
    clearCookie(httpRequest, httpResponse);
    httpResponse.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Unauthorized");
}
```

Реалізована система безпеки забезпечує надійний захист від несанкціонованого доступу до даних. Використання JWT токенів дозволяє забезпечити безпечну автентифікацію та авторизацію користувачів, а також зробити систему масштабованою та легко інтегрованою з іншими сервісами. Вимкнення CSRF–захисту та використання безстатусних сесій дозволяють оптимізувати роботу системи та зменшити навантаження на сервер.

3.3 Створення RESTful сервісів

Контролери є ключовими компонентами в архітектурі RESTful веб–сервісів, оскільки вони відповідають за обробку HTTP–запитів, взаємодію з сервісами бізнес–логіки та повернення відповідей користувачеві. У даній системі контролери реалізовані з використанням фреймворку Spring, що забезпечує простоту та гнучкість в обробці запитів. У цьому розділі я розгляну реалізацію контролера для сутності "Тип Бочки" (BarrelType), що слугує прикладом для інших контролерів у системі.

Опис контролера BarrelTypeController: контролер BarrelTypeController відповідає за обробку запитів, пов'язаних з типами бочок у системі. Він

використовує анотації Spring для визначення маршруту запитів та обробки даних. Нижче наведено ключові аспекти реалізації контролера:

```
Copy
@RestController
@RequestMapping("/api/v1/barrel/type")
@RequiredArgsConstructor
@Slf4j
public class BarrelTypeController {
    private final IBarrelTypeService barrelTypeService;
    private final BarrelTypeMapper barrelTypeMapper;
```

Анотація @RestController – ця анотація позначає клас як контролер, який обробляє REST–запити. Вона автоматично перетворює об'єкти, повернуті з методів контролера, у JSON–формат, що є стандартом для веб–сервісів.

Анотація @RequestMapping("/api/v1/barrel/type") – встановлює базовий шлях для всіх методів контролера. Це дозволяє зручно організувати маршрути, що стосуються типів бочок.

Контролер використовує два основні компоненти:

IBarrelTypeService – сервіс, що реалізує бізнес–логіку, пов'язану з типами бочок.

BarrelTypeMapper – маппер, що відповідає за перетворення між сутностями та їх DTO.

Контролер реалізує кілька методів для обробки різних HTTP–запитів.

Отримання типу бочки за ID:

```
Copy
@GetMapping("/{id}")
public ResponseEntity<BarrelTypeDTO> findById(@PathVariable Long id) {
    BarrelTypeDTO barrelTypeDTO =
barrelTypeMapper.barrelTypeToDTO(barrelTypeService.getBarrelTypeById(id));
    return ResponseEntity.ok(barrelTypeDTO);
}
```

Цей метод обробляє GET–запити на отримання типу бочки за його ідентифікатором. Він використовує @PathVariable для отримання ID з URL і повертає DTO типу бочки у відповіді.

Отримання всіх типів бочок:

```
Copy
@GetMapping
public ResponseEntity<List<BarrelTypeDTO>> findAll() {
    List<BarrelType> barrelTypes = barrelTypeService.getAllBarrelTypes();
    List<BarrelTypeDTO> barrelTypeDTOS = barrelTypes.stream()
            .map(barrelTypeMapper::barrelTypeToDTO)
            .toList();
    return ResponseEntity.ok(barrelTypeDTOS);
}
```

Цей метод обробляє GET–запити для отримання списку всіх типів бочок. Він перетворює сутності в DTO перед поверненням.

Створення нового типу бочки:

```
Copy
@PostMapping
public ResponseEntity<BarrelTypeDTO> create(@RequestBody
BarrelTypeDTO barrelTypeDTO) {
    BarrelType barrelType =
barrelTypeMapper.barrelTypeToEntity(barrelTypeDTO);
    BarrelType response = barrelTypeService.createBarrelType(barrelType);
    return ResponseEntity.ok(barrelTypeMapper.barrelTypeToDTO(response));
}
```

Цей метод обробляє POST–запити для створення нового типу бочки. Він приймає дані в форматі JSON, перетворює їх у сутність, передає їх у сервіс для збереження, а потім повертає створений об'єкт у DTO–форматі.

Оновлення існуючого типу бочки:

```
Copy
```

```
@PutMapping("/{id}")
public ResponseEntity<BarrelTypeDTO> update(@RequestBody
BarrelTypeDTO barrelTypeDTO) {
    BarrelType barrelType =
barrelTypeMapper.barrelTypeToEntity(barrelTypeDTO);
    BarrelType response = barrelTypeService.updateBarrelType(barrelType);
    return ResponseEntity.ok(barrelTypeMapper.barrelTypeToDTO(response));
}
```

Метод обробляє PUT–запити для оновлення типу бочки. Він приймає DTO через тіло запиту, перетворює його в сутність і передає у сервіс для оновлення.

Видалення типу бочки:

Copy

```
@DeleteMapping("/{id}")
public ResponseEntity<Void> delete(@PathVariable Long id) {
    barrelTypeService.deleteBarrelType(id);
    return ResponseEntity.noContent().build();
}
```

Цей метод обробляє DELETE–запити для видалення типу бочки за його ID. Після виконання операції повертається статус 204 No Content, що вказує на успішне виконання запиту без повернення тіла.

Контролер BarrelTypeController реалізує основні CRUD–операції для роботи з типами бочок у системі. Застосування аннотацій Spring робить код чистим та зрозумілим, а використання мапперів дозволяє ефективно перетворювати дані між сутностями та DTO. Аналогічна реалізація контролерів для інших сутностей у системі дозволяє підтримувати єдиний стиль коду та спрощує подальший розвиток і підтримку системи.

3.4 Реалізація бізнес–логіки

Сервіси в архітектурі веб–додатків виконують роль посередників між контролерами та репозиторіями. Вони реалізують бізнес–логіку, обробляють дані та забезпечують узгодженість між різними частинами системи. У цьому розділі я розгляну реалізацію сервісу для сутності "Тип Бочки" (BarrelType), який слугує прикладом для інших сервісів у системі.

Опис сервісу BarrelTypeService. Сервіс реалізує інтерфейс IBarrelTypeService і відповідає за бізнес–логіку, пов'язану з типами бочок. Основні функції сервісу включають створення, оновлення, видалення та отримання типів бочок. Нижче наведено ключові аспекти реалізації сервісу:

```
Copy
@Service
@RequiredArgsConstructor
@Slf4j
public class BarrelTypeService implements IBarrelTypeService {
    private final BarrelTypeRepository barrelTypeRepository;
```

Анотація @Service позначає клас як сервіс, що дозволяє Spring автоматично визначати його як компонент, який може бути впроваджений в інші частини програми (наприклад, у контролери).

BarrelTypeRepository – репозиторій, що забезпечує доступ до бази даних для операцій CRUD з типами бочок.

Сервіс реалізує кілька методів, які відповідають за основні операції з типами тари.

Створення нового типу тари:

```
Copy
@Override
@Transactional
public BarrelType createBarrelType(BarrelType barrelType) {
    log.info("Creating BarrelType: {}", barrelType);
```

```
        return barrelTypeRepository.save(barrelType);
    }
```

Цей метод обробляє створення нового типу тари. Він відзначений анотацією @Transactional, що забезпечує атомарність операції – або всі зміни зберігаються, або жодні. Логування інформації про створюваний тип бочки допомагає в моніторингу та налагодженні.

Оновлення існуючого типу бочки:

```
 Copy
@Override
@Transactional
public BarrelType updateBarrelType(BarrelType barrelType) {
    log.info("Updating BarrelType: {}", barrelType);
    if (!barrelTypeRepository.existsById(barrelType.getId())) {
        throw new BarrelException("BarrelType with id " + barrelType.getId() +
" not found");
    }
    return barrelTypeRepository.save(barrelType);
}
```

Метод для оновлення типу бочки спочатку перевіряє, чи існує тип бочки з вказаним ID. Якщо тип не знайдено, генерується виключення BarrelException. Це забезпечує надійність операції та запобігає неочікуваним помилкам.

Видалення типу бочки:

```
 Copy
@Override
@Transactional
public BarrelType deleteBarrelType(Long id) {
    log.info("Deleting BarrelType with id: {}", id);
    BarrelType barrelType = barrelTypeRepository.findById(id)
            .orElseThrow(() –> new BarrelException("BarrelType with id " + id + "
not found"));
```

```
    barrelTypeRepository.delete(barrelType);
    return barrelType;
}
```

Метод для видалення типу бочки спочатку знаходить тип за ID. Якщо тип бочки не існує, генерується виключення. Цей підхід гарантує, що система не видалить тип бочки, який не існує.

Отримання всіх типів бочок:

```
 Copy
@Override
public List<BarrelType> getAllBarrelTypes() {
    log.info("Retrieving all BarrelTypes");
    return barrelTypeRepository.findAll();
}
```

Цей метод повертає список усіх типів бочок з бази даних. Логування також допомагає відстежувати дії, що виконуються в системі.

Отримання типу бочки за ID:

```
 Copy
@Override
public BarrelType getBarrelTypeById(Long id) {
    log.info("Retrieving BarrelType with id: {}", id);
    return barrelTypeRepository.findById(id)
            .orElseThrow(() –> new BarrelException("BarrelType with id " + id + " not found"));
}
```

Метод для отримання типу бочки за ID, який генерує виключення, якщо тип не знайдено, забезпечуючи валідацію даних.

Отримання типу бочки за назвою:

```
 Copy
@Override
public BarrelType getBarrelTypeByName(String name) {
```

```
        log.info("Retrieving BarrelType by name: {}", name);
        return barrelTypeRepository.findByName(name)
                .orElseThrow(() –> new BarrelException("BarrelType with name " +
name + " not found"));
    }
```

Цей метод дозволяє отримувати тип бочки за його назвою, також генеруючи виключення у випадку, якщо тип не знайдено.

Сервіс BarrelTypeService реалізує основну бізнес–логіку для роботи з типами бочок у системі. Використання анотацій Spring, таких як @Service і @Transactional, робить реалізацію зручною для впровадження та підтримки. Логування інформації про виконувані операції допомагає у відстеженні та налагодженні. Аналогічна реалізація сервісів для інших сутностей у системі забезпечує узгодженість та спрощує подальший розвиток і підтримку.

3.5 Пагінація даних

Пагінація є важливою складовою сучасних веб–додатків, яка дозволяє ефективно управляти великими обсягами даних, розбиваючи їх на менші, зручні для перегляду частини. У цьому розділі я розгляну реалізацію пагінації для сутності "Олії" (Oil), яка слугує прикладом для інших сутностей у системі. Використання пагінації сприяє покращенню продуктивності, зручності використання та зменшенню навантаження на сервер.

Реалізація пагінації в системі здійснюється через метод HTTP GET, який відповідає за пошук об'єктів типу Oil з можливістю фільтрації, сортування та пагінації. Нижче наведено ключові аспекти реалізації методу пошуку:

```
 Copy
@GetMapping("/search")
public ResponseEntity<PageResponse<OilPageDto>> searchOil(
        @RequestParam(name = "q", required = false) String query,
        @RequestParam(name = "size", defaultValue = "100") int size,
```

@RequestParam(name = "page", defaultValue = "0") int page,

@RequestParam(name = "sort", defaultValue = "updatedAt") String sortProperty,

@RequestParam(name = "direction", defaultValue = "DESC") Sort.Direction sortDirection,

@RequestParam(name = "filters", required = false) String filtersJson) throws JsonProcessingException {

Параметри запиту:

– query: рядок для пошуку, що дозволяє шукати об'єкти за різними критеріями;

– size: кількість об'єктів на сторінці (за замовчуванням 100);

– page: номер сторінки (за замовчуванням 0);

– sortProperty: властивість для сортування (за замовчуванням updatedAt);

– sortDirection: напрямок сортування (за замовчуванням DESC);

– filtersJson: JSON–рядок для фільтрації.

Метод обробляє параметри, перевіряючи, чи не є query порожнім, і формує об'єкт для фільтрації з JSON–рядка:

Copy

```
ObjectMapper objectMapper = new ObjectMapper();
Map<String, List<String>> filters = new HashMap<>();
if (filtersJson != null) {
    filters = objectMapper.readValue(filtersJson, new TypeReference<>() {});
}
```

Створюється об'єкт Pageable, що містить інформацію про номер сторінки, розмір сторінки та параметри сортування:

Copy

```
Pageable pageable = PageRequest.of(page, size, Sort.by(sortDirection, sortProperty));
```

Метод передає параметри запиту до сервісу для виконання пошуку:

Copy

Page<OilPageDto> oil = oilService.searchOil(query, pageable, filters)

.map(oilMapper::oilToOilPageDTO);

Формування відповіді. Результати пошуку упаковані в об'єкт PageResponse, який повертається як відповідь:

Copy

PageResponse<OilPageDto> response = new PageResponse<>(oil);

return ResponseEntity.ok(response);

Реалізація методу пошуку в сервісі. Сервісний метод searchOil реалізується в сервісі відповідним чином:

Copy

@Override

public Page<Oil> searchOil(String query, Pageable pageable, Map<String, List<String>> filterParams) {

return oilRepository.findAll(new OilSpecification(query, filterParams), pageable);

}

Цей метод викликає репозиторій для виконання запиту, передаючи специфікацію пошуку.

Клас OilSpecification реалізує інтерфейс Specification<Oil> і забезпечує логіку формування запиту на основі параметрів пошуку:

Copy

public class OilSpecification implements Specification<Oil> {

private final String query;

private final Map<String, List<String>> filterParams;

public OilSpecification(String query, Map<String, List<String>> filterParams) {

this.query = query;

this.filterParams = filterParams;

}

```
}
```

Метод toPredicate формує предикат на основі запиту та фільтрів.

Перш за все, перевіряється наявність текстового запиту. Якщо є, додаються умови до предиката на основі значень полів, таких як totalKg, totalPaid, pricePerKg, а також на основі зв’язків з іншими сутностями (користувачі, клієнти, джерела).

Для кожного фільтра, переданого в запиті, додаються умови до предиката:

```
 Copy
for (Map.Entry<String, List<String>> entry : filterParams.entrySet()) {
   String key = entry.getKey();
   List<String> values = entry.getValue();
   // Логіка додавання умов для кожного фільтра
}
```

Реалізація пагінації у системі забезпечує зручний та ефективний доступ до великих обсягів даних. Використання специфікацій дозволяє гнучко формувати запити на основі різних критеріїв пошуку та фільтрації. Таке рішення не лише підвищує продуктивність системи, але й значно покращує користувацький досвід. Аналогічна реалізація пагінації для інших сутностей у системі забезпечує узгодженість та спрощує подальший розвиток і підтримку.

3.6 Синхронізація даних між системами

Синхронізація даних між різними системами є важливим аспектом для забезпечення цілісності інформації та зручності роботи з даними. У даній системі реалізована синхронізація даних з CRM системою, що дозволяє підтримувати актуальність даних клієнтів, а також забезпечує можливість переходу до CRM системи у випадку, якщо основна система перестане працювати. У цьому розділі я розгляну механізм синхронізації даних, що включає створення клієнтів у CRM та обробку відповідей від CRM системи.

Синхронізація даних з CRM системою реалізована у методі createClient, який відповідає за створення нового клієнта. Основні етапи процесу наведені нижче:

```
Copy
@Override
public Client createClient(Client client) {
    log.info("Creating new client: {}", client);
    Long id = keepinCrmService.createClient(client);
    client.setId(id);
```

Метод починає з логування інформації про створюваного клієнта, що дозволяє відслідковувати дії в системі.

Метод keepinCrmService.createClient(client) викликає CRM сервіс для створення клієнта. Цей виклик повертає ідентифікатор нового клієнта, який отримується з CRM системи.

Якщо у клієнта є телефонні номери, вони прив’язуються до об'єкта клієнта:

```
Copy
if (client.getPhoneNumbers() != null) {
    for (PhoneNumber phoneNumber : client.getPhoneNumbers()) {
        phoneNumber.setClient(client);
    }
}
```

Після отримання ідентифікатора клієнта з CRM, об'єкт клієнта зберігається у локальній базі даних через репозиторій:

```
Copy
return clientRepository.save(client);
```

Метод createClient у сервісі CRM виконує HTTP POST запит для створення нового клієнта в CRM системі:

```
Copy
try {
```

ResponseEntity<Map> response = restTemplate.postForEntity(apiUrl + "/clients", requestEntity, Map.class);

Для взаємодії з CRM системою використовується RestTemplate, що дозволяє відправляти HTTP запити.

Метод обробляє відповідь, отриману від CRM системи. Якщо статус відповіді є HTTP_CREATED або HTTP_OK, перевіряється наявність ідентифікатора у відповіді:

```
Copy
if (statusCode == HttpStatus.CREATED || statusCode == HttpStatus.OK) {
    Map<String, Object> responseBody = response.getBody();
    assert responseBody != null;
    if (responseBody.containsKey("id")) {
        addPhoneNumbers(client, Long.parseLong(responseBody.get("id").toString()));
        return Long.parseLong(responseBody.get("id").toString());
    } else {
        log.error("CRM returned a successful status code but missing 'id' in response: {}", responseBody);
        throw new RuntimeException("CRM response missing 'id' field.");
    }
}
```

Якщо CRM повертає статус, що не є успішним, або відсутній ідентифікатор, метод генерує виключення:

```
Copy
else {
    String errorMessage = extractErrorMessageFromCRMResponse(Objects.requireNonNull(response.getBody()));
    throw new RuntimeException("Failed to create client in CRM: " + statusCode + ", " + errorMessage);
```

```
}
```

У випадку виникнення помилок під час створення клієнта у CRM системі, вони логуються для подальшого аналізу:

```
 Copy
catch (Exception e) {
   log.error("Error creating client in CRM", e);
   throw new RuntimeException("Error creating client in CRM", e);
}
```

Реалізація синхронізації з CRM системою забезпечує надійний механізм підтримки актуальності даних клиентів. У разі збою основної системи, дані залишаються доступними в CRM, що дозволяє користувачам безперешкодно продовжувати роботу. Процес синхронізації включає логування, обробку помилок та інтеграцію з REST API CRM, що робить його гнучким та надійним. Подібний підхід може бути реалізований і для інших сутностей у системі, що дозволяє підтримувати цілісність даних і забезпечувати безперервність бізнес–процесів.

3.7 Розробка та інтеграція користувацького інтерфейсу

У рамках розробки автоматизованого пристрою для візуалізації даних з бази даних на веб–сайті, важливо правильно спроектувати інтерактивний і зручний інтерфейс користувача. У цьому розділі я розгляну одну з веб–сторінок, яка реалізує функціонал для управління даними клієнтів, включаючи можливість фільтрації, перегляду деталей клієнта, а також створення нових записів.

Фронтенд веб–інтерфейсу складається з кількох ключових компонентів, які слугують для взаємодії користувача з даними. Розглянемо основні елементи структури сторінки.

Секція фільтрації:

```
 Copy
<div id="filter–section">
```

```
<span id="filter–counter">0</span>
<button id="filter–clear">×</button>
<input type="text" id="lastPurchaseDate" placeholder="Дата останньої закупівлі олії">
<button id="apply–filters">Застосувати фільтри</button>
</div>
```

Лічильник фільтрів: показує кількість активних фільтрів.

Кнопка очищення фільтрів: дозволяє скинути застосовані фільтри.

Поле вводу для дати: дозволяє вибрати дату останньої закупівлі олії.

Кнопка застосування фільтрів: викликає обробку фільтрації даних на основі введених значень.

Список клієнтів:

Copy

```
<table id="client–table">
  <thead>
    <tr>
      <th><input type="checkbox" id="select–all"></th>
      <th>Компанія</th>
      <th>Область</th>
      <th>Статус</th>
      <th>Збір</th>
      <th>Телефони</th>
      <th>Залучення</th>
      <th>Маршрут</th>
      <th>Коментар</th>
      <th>Адреса</th>
    </tr>
  </thead>
  <tbody id="client–table–body">
    <!— Дані клієнтів заповнюються динамічно через JavaScript —>
```

```
    </tbody>
</table>
<button id="delete–selected">Delete Selected</button>
```

Таблиця: відображає список клієнтів з можливістю вибору кількох записів. Checkbox для вибору всіх: дозволяє вибрати всі клієнти для подальших дій. Кнопка видалення: видаляє вибрані записи з таблиці.

Модальне вікно для деталей клієнта:

Copy

```
<div id="client–modal">
    <h2>Картка клієнта</h2>
    <p>ID: <span id="client–id"></span></p>
    <p>Компанія: <span id="client–company"></span> <button id="edit–company">✎</button></p>
    <p>Контактна особа: <span id="client–contact"></span> <button id="edit–contact">✎</button></p>
    <p>Коментар: <span id="client–comment"></span> <button id="edit–comment">✎</button></p>
    <p>Терміновий збір: <span id="urgent–collection"></span></p>
    <p>Останній збір: <span id="last–collection"></span></p>
    <p>Створено: <span id="created–at"></span></p>
    <p>Оновлено: <span id="updated–at"></span></p>
    <button id="save–changes">Зберегти зміни</button>
    <button id="cancel">Відмінити</button>
    <button id="delete–client">Видалити</button>
    <button id="full–delete">Full delete</button>
</div>
```

Модальне вікно: відображає деталі обраного клієнта з можливістю редагування.

Кнопки для редагування: дозволяють редагувати інформацію про компанію, контактну особу та коментар.

Кнопки для збереження та видалення: зберігають зміни або видаляють клієнта.

Модальне вікно для створення нового клієнта:

Copy

```
<div id="create–client–modal">
  <h2>Створення нового клієнта</h2>
  <form id="client–form">
    <label for="area">Область:</label>
    <input type="text" id="area" required>
    <label for="status">Статус:</label>
    <input type="text" id="status" required>
    <label for="engagement">Залучення:</label>
    <input type="text" id="engagement" required>
    <label for="route">Маршрут:</label>
    <input type="text" id="route" required>
    <label for="business–type">Тип бізнесу:</label>
    <input type="text" id="business–type" required>
    <button type="submit">Зберегти</button>
  </form>
</div>
```

Форма для створення клієнта: збирає необхідні дані для нового клієнта.

Поля вводу: вимагатимуть заповнення перед надсиланням форми.

Технології та структура

HTML: використовується для структурування контенту на веб–сторінці. Структура є логічною та семантичною.

JavaScript: обробка подій, динамічна взаємодія з елементами сторінки, відправка запитів до сервера, а також обробка відповідей від сервера (наприклад, заповнення таблиці даними клієнтів).

Представлений фронтенд для автоматизованого пристрою візуалізації даних забезпечує користувачеві зручний інтерфейс для управління клієнтами. Застосування модальних вікон для редагування та створення нових записів, а також можливість фільтрації та перегляду деталей клієнтів, роблять систему функціональною та інтуїтивно зрозумілою. З подальшими поліпшеннями в дизайні та функціональності, цей інтерфейс може стати потужним інструментом для візуалізації даних з бази даних на веб–сайті.

Також потрібно розуміти, що реалізований окремий фронтенд для водіїв, комірників, бухгалтерів і для всіх типів співробітників. При цьому підхід для кожного типу різний: наприклад, для водіїв інтерфейс реалізований під планшети, тому всі адаптації, функціонали заточені для роботи з планшетів. І така ж адаптована реалізація для всіх типів працівників.

3.8 Висновки розробки комп'ютеризованої системи

Розробка комп'ютеризованої системи включає кілька важливих етапів. Спочатку була розроблена база даних, що забезпечує надійне зберігання та швидкий доступ до інформації. Наступним кроком стала розробка архітектури безпеки системи, що гарантує захист даних від несанкціонованого доступу. Створення RESTful сервісів дозволило забезпечити взаємодію між різними компонентами системи. Реалізація бізнес–логіки забезпечила правильне виконання бізнес–процесів та операцій. Пагінація даних була впроваджена для зручної роботи з великими обсягами інформації. Синхронізація даних між системами забезпечує актуальність інформації у всіх підсистемах. Нарешті, була розроблена та інтегрована користувацький інтерфейс, що надає користувачам інтуїтивно зрозумілий доступ до всіх функцій системи.

4 РОБОТА СИСТЕМИ НА ПРАКТИЦІ

4.1 Перевірочні та демонстраційні тести

Для розглядання роботи системи пропоную почати розгляд системи зі сторони менеджера, тому що з нього починається перші внесення даних. Після авторизації в системі користувача за участю менеджер потрапляє на сторінку з клієнтами. Вид сторінки можна побачити на рис. 4.1.

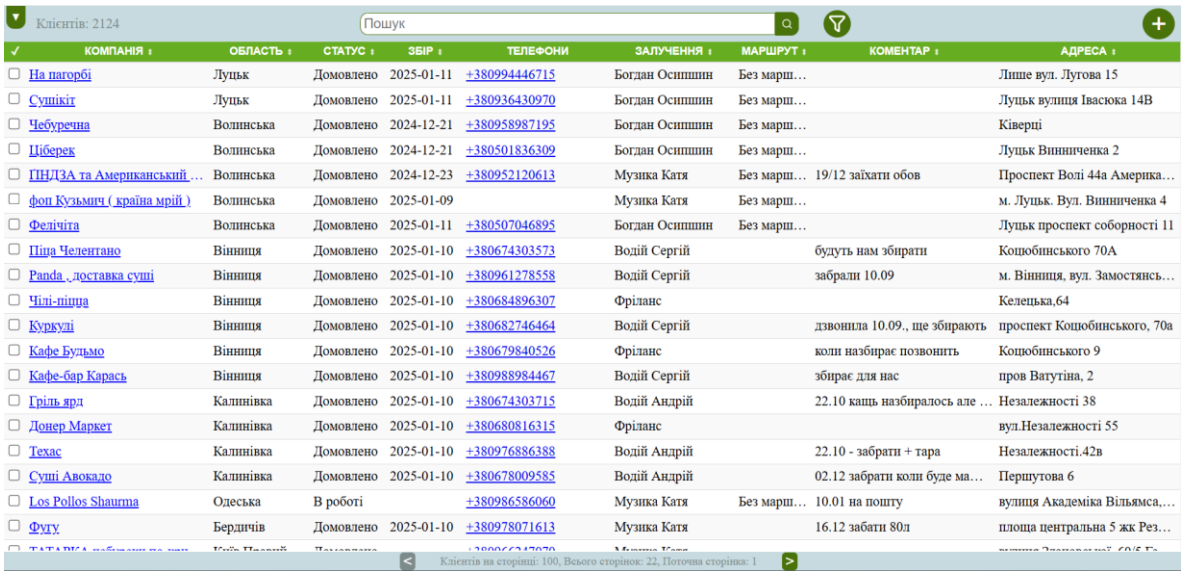


Рисунок 4.1 – Вид сторінки менеджера

Зліва зверху знаходиться кнопка для переходу на інші сторінки. Правіше загальна кількість клієнтів, що відображаються. У центрі є поле пошуку, пошук здійснюється по всіх існуючих полях в полях клієнта. Там же поруч кнопка фільтрів, при натисканні на яку можна виставити потрібні фільтри. У правому кутку кнопка створення нового клієнта. У центрі екрана відображається список усіх клієнтів. Нагадую, що реалізовано пагінацію, тому на одній сторінці можуть відображатися не всі клієнти. Інформацію про сторінки та переходи на інші можна побачити внизу екрана. У самому списку клієнтів менеджер може бачити частину інформації про клієнта, а при натисканні на назву відкривається

модальне вікно клієнта, де надана вся інформація. Вид модального вікна клієнта можна побачити рис. 4.2.

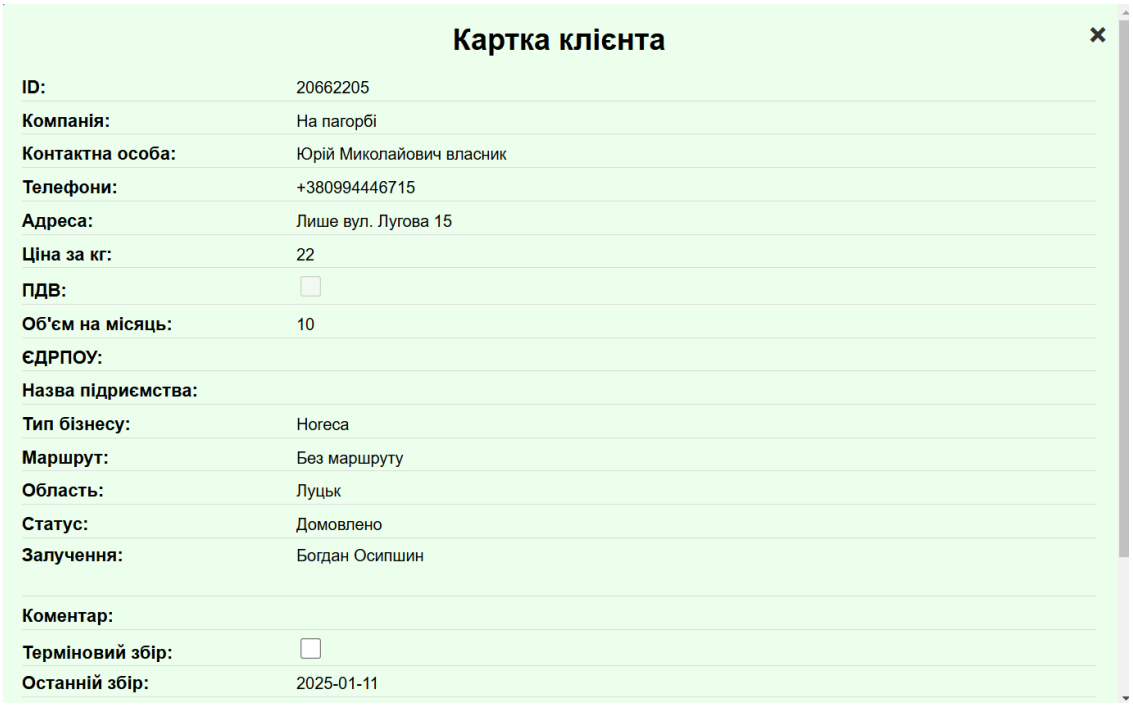


Рисунок 4.2 – Вид картки клієнта

Після створення списку клієнтів менеджером вони також потрапляють на сторінку водіїв, що забезпечує автоматичну передачу актуальних даних водію. При авторизації водія сторінка для них відрізняється, тому що вони працюють із планшетів рис. 4.3.

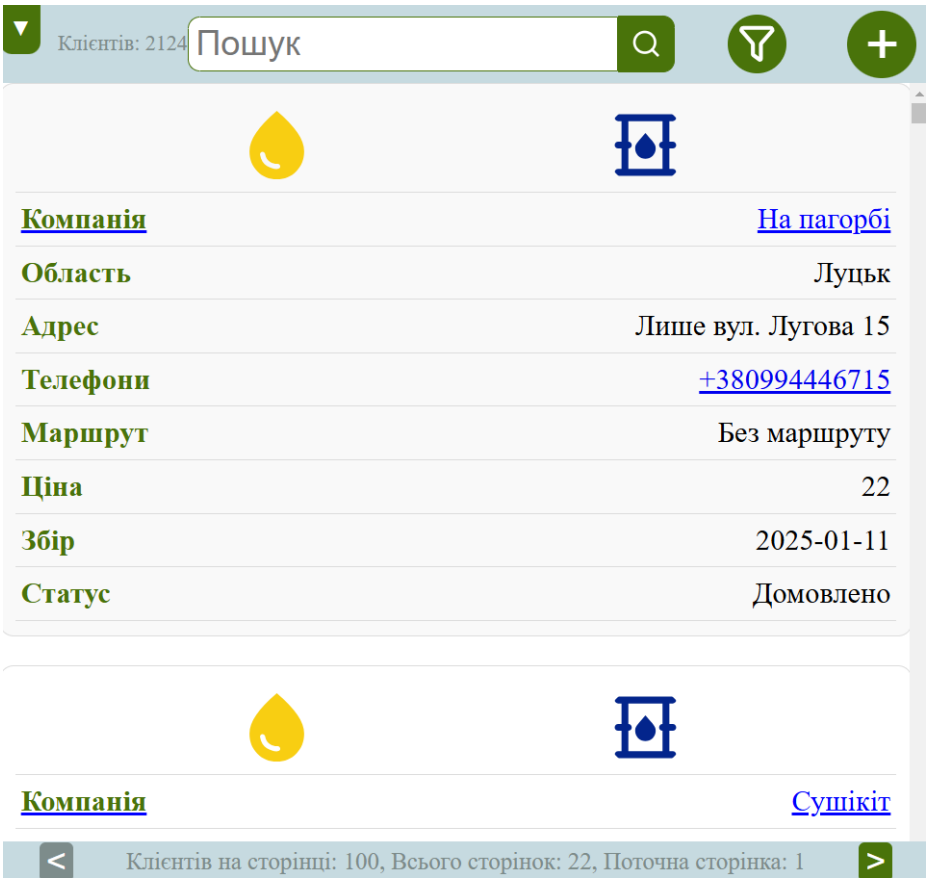


Рисунок 4.3 – Вид сторінки водія

По–перше, тепер список клієнтів тепер у вигляді таблиці, а вигляді карток. По–друге, з'явилися дві нові кнопки: збирання олії та дії з бочками. Розглянемо збирання олії, вид вікна надано на рис. 4.4.

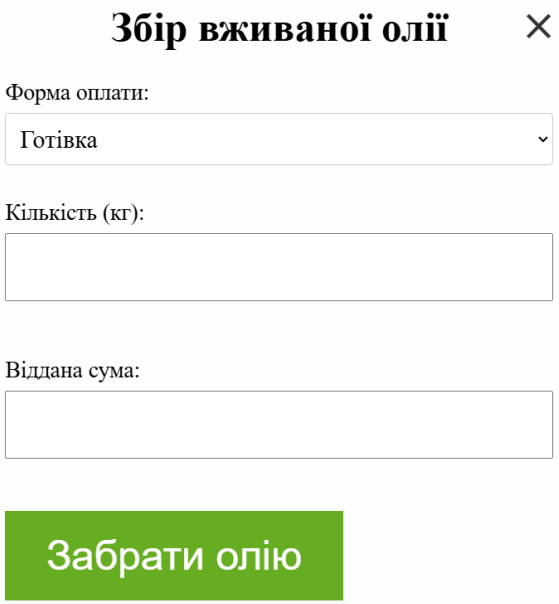


Рисунок 4.4 – Вид вікна збору олії

При внесенні збирання олії оновлюються дані клієнта, створюється сутність збирання олії, проводиться грошова транзакція. Тобто, у нас одним внесенням даних водієм проводиться кілька дій, що виключає зайву роботу для всіх: менеджер бачить актуальний стан клієнта, тобто чи в роботі він, всі користувачі можуть побачити збір олії, бухгалтер бачить грошову транзакцію, яка мало того що фіксується в системі, так ще й заноситься до 1С, тим самим автоматизуючи облік товару та грошей. Що так само цікаво для бухгалтерської частини: якщо водій обирає розрахунок безготівковими, тоді кошти знімаються з рахунку ТОВ, а не водія, що автоматизує роботу бухгалтера. Далі розглянемо вікно роботи з тарою на рис. 4.5.

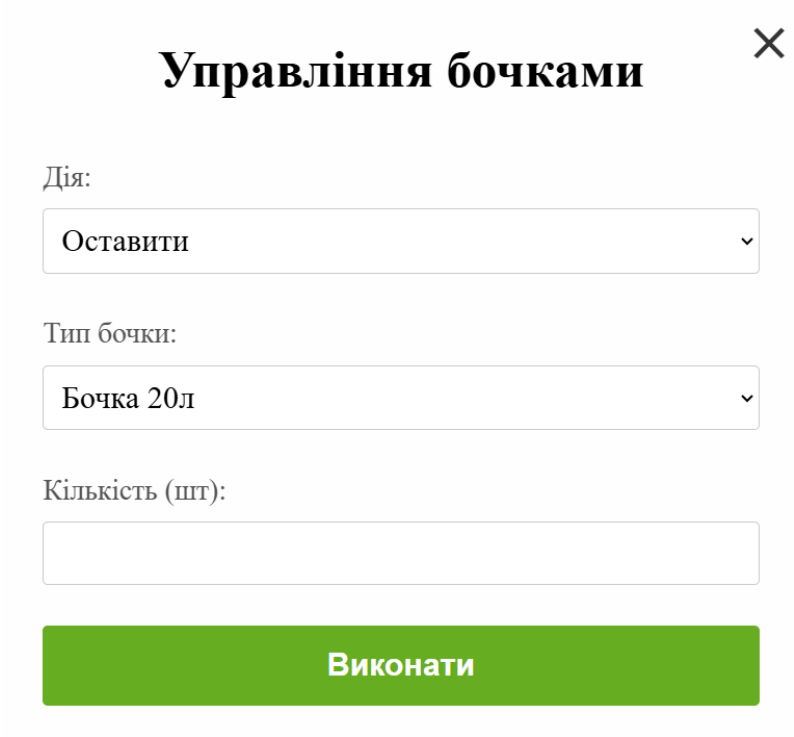


Рисунок 4.5 – Вид вікна роботи з тарами

За допомогою цього функціоналу водій може виконати події з тарою. Тобто залишити або забрати її з точки. Це дозволяє контролювати рух тари. І за допомогою цього функціоналу немає потреби вести минуловікові акти передачі тари, і легше відстежувати тару завдяки тому, що також створена окрема сторінка, де відображається наявність тари у клієнтів. Повертаючись до мастила,

для звіту зі збирання олії також є окрема сторінка, на якій відображаються всі збори. Вид можна побачити на рис. 4.6.

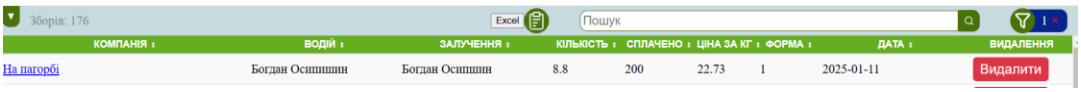


Рисунок 4.6 – Вид сторінки зборів олії

На цій сторінці можна побачити всі збори. Одним з основних функціоналів цієї сторінки є швидке відображення даних по зборах, вид надано на рис. 4.7.

Рисунок 4.7 – Вид звіта зборів олії

Це є важливою інформацією, за якою можуть орієнтувати всі користувачі, тому що виходячи з цих цифр налічується бонуси і так далі. І останнє що я вважав за потрібне показати зараз – це сторінку графічного аналізу даних, вид її можна переглянути на рис. 4.8–4.11.

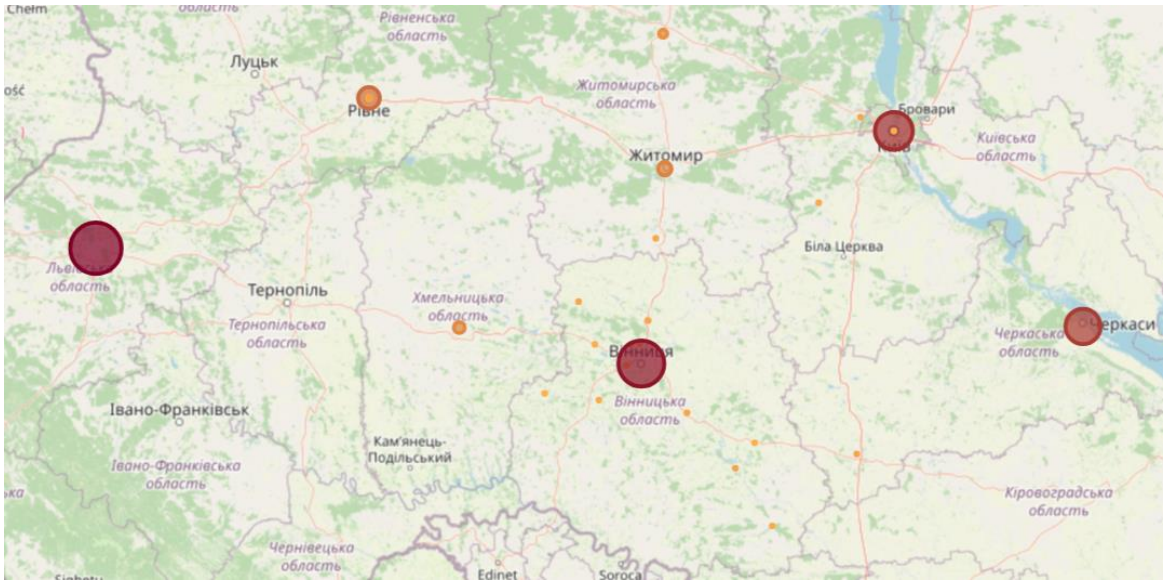


Рисунок 4.8 – Карта зборів олії по регіонам

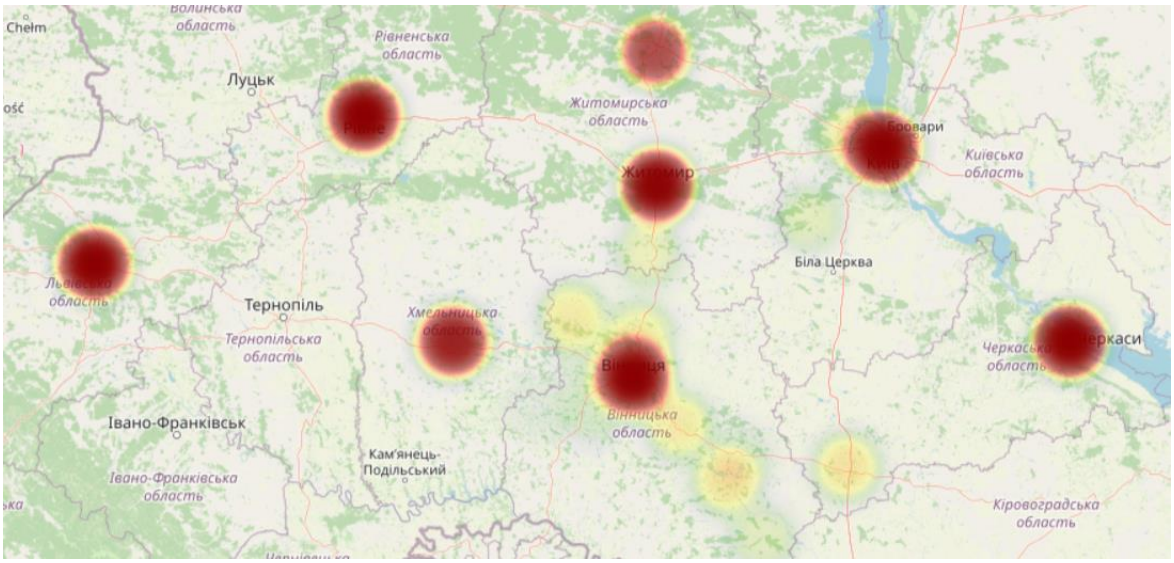


Рисунок 4.9 – Теплова карта зборів олії

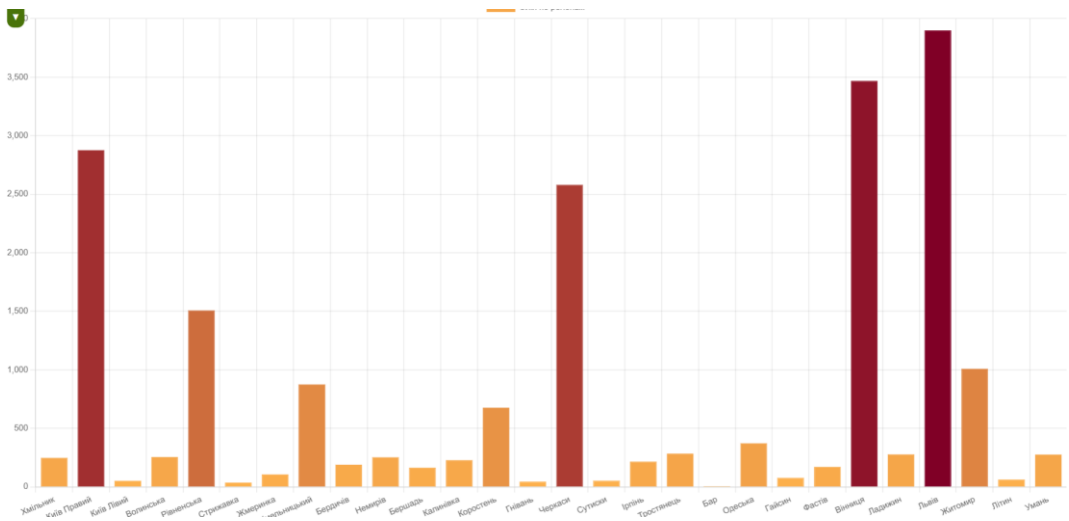


Рисунок 4.10 – Діаграма зборів олії

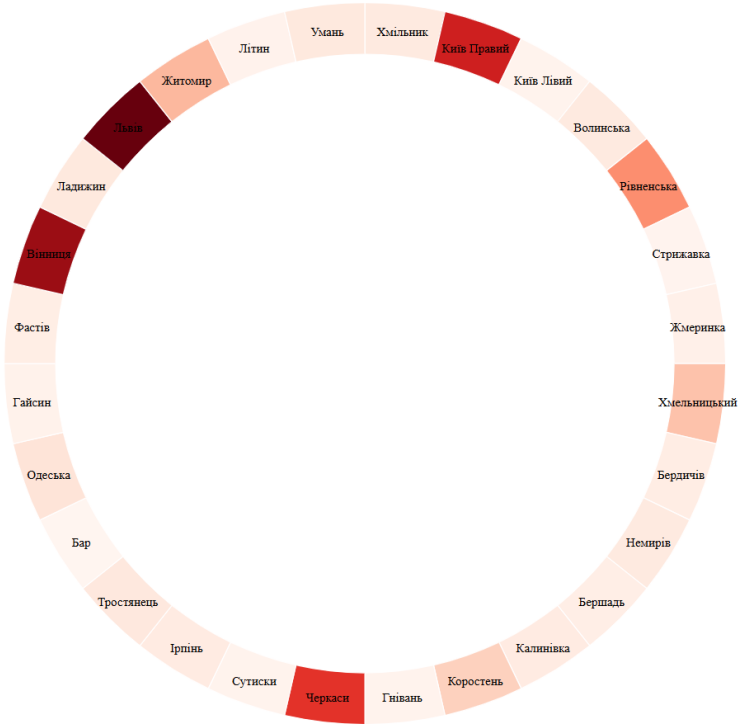


Рисунок 4.11 – Кругова діаграма зборів олії

Сторінка аналітики є потужним інструментом для візуалізації та аналізу даних, пов'язаних із виробництвом або розподілом олії в різних регіонах. Вона складається з кількох ключових компонентів, які взаємодіють між собою для надання користувачу зручного та інформативного інтерфейсу. Давайте детально розглянемо кожен елемент сторінки та його функціональність.

Сторінка починається з двох полів для введення дат.

"Від:" (start–date) – початкова дата для аналізу.

"До:" (end–date) – кінцева дата для аналізу.

Ці поля дозволяють користувачу обмежити часовий період, за який будуть збиратися дані. Після введення дат користувач натискає кнопку "Відобразити" (fetch–data), яка ініціює завантаження даних з сервера.

Після завантаження даних на сторінці відображається карта (map), яка використовує бібліотеку Leaflet. На карті кожен регіон представлений у вигляді круга, розмір і колір якого залежать від обсягу олії в цьому регіоні.

Розмір круга обчислюється на основі нормалізованих значень обсягу олії (totalOil), де найменший обсяг відповідає радіусу 100, а найбільший – 15000.

Колір круга визначається за допомогою функції getColor, яка перетворює нормалізоване значення обсягу олії в градієнт від світлого до темного відтінку.

Кожен круг має поп–ап, який відображає назву регіону та обсяг олії при наведенні курсора.

Поруч із основною картою розташована теплова карта (heatmap), яка також використовує Leaflet. Вона візуалізує інтенсивність виробництва або розподілу олії за допомогою градієнта кольорів.

Градієнт починається з синього (низька інтенсивність) і закінчується темно–червоним (висока інтенсивність).

Радіус та розмиття точок налаштовані для кращої візуалізації.

Теплова карта допомагає швидко визначити регіони з найвищою концентрацією олії.

На сторінці присутній стовпчастий графік (oil–chart), створений за допомогою бібліотеки Chart.js. Графік відображає обсяг олії для кожного регіону.

Вісь X містить назви регіонів.

Вісь Y відображає обсяг олії.

Кольори стовпців відповідають кольорам на карті, що забезпечує узгодженість візуалізації.

Графік дозволяє порівняти обсяги олії між регіонами в зручному форматі.

Радіальна діаграма (radial–chart–container) створена за допомогою бібліотеки D3.js. Вона представляє дані у вигляді кільця, розділеного на сектори.

Кожен сектор відповідає одному регіону.

Колір сектора залежить від обсягу олії, використовуючи градієнт від світлого до темного червоного.

Текст всередині сектора відображає назву регіону.

Радіальна діаграма надає альтернативний спосіб візуалізації даних, який може бути корисним для порівняння відносних часток регіонів.

Під всіма графіками розташована таблиця (table–container), яка відображає детальну інформацію про кожен регіон, назва регіону, обсяг олії, широта та довгота (координати регіону).

Таблиця дозволяє користувачу швидко знайти конкретні дані, які можуть бути неочевидними на графіках або картах.

Сторінка використовує JavaScript для динамічного оновлення всіх елементів на основі введених даних.

Запит до сервера: функція fetchOilData відправляє запит на сервер з вказаними датами та отримує дані у форматі JSON.

Оновлення візуалізації: отримані дані передаються до функцій renderMap, renderChart, renderTable, renderHeatmap та renderRadialChart, які оновлюють відповідні елементи сторінки.

Обробка помилок: у разі помилки під час завантаження даних, у консоль виводиться повідомлення про помилку.

Нормалізація даних: функція normalize перетворює значення обсягу олії в діапазон, який використовується для розрахунку розміру круга або кольору.

Генерація кольорів: функція getColor створює градієнт кольорів на основі нормалізованих значень.

Сторінка використовує кілька популярних бібліотек для візуалізації:

Leaflet: для створення карт та теплових карт.

Chart.js: для побудови стовпчастого графіка.

D3.js: для створення радіальної діаграми.

Ця сторінка аналітики є прикладом сучасного веб–додатку, який поєднує різні типи візуалізації для аналізу даних. Вона дозволяє користувачу: вибирати часовий період для аналізу, переглядати дані на карті, тепловій карті, графіку, радіальній діаграмі та таблиці, швидко порівнювати дані між регіонами, отримувати детальну інформацію про кожен регіон.

Такий підхід робить сторінку зручною та ефективною для роботи з великими обсягами даних.

4.2 Порівняння з існуючими рішеннями

У цьому розділі буде здійснено порівняння розробленої мною комп'ютеризованої системи з одним із популярних рішень на ринку – KeepinCRM. Метою цього порівняння є виявлення переваг і недоліків обох систем, що допоможе зрозуміти унікальні аспекти моєї розробки та її конкурентні переваги. Аналіз буде зосереджений на функціональних можливостях, зручності використання, гнучкості налаштування, а також продуктивності. Порівняння дозволить визначити, наскільки інноваційним є нове рішення в контексті існуючих пропозицій на ринку CRM систем.

Для аналізу швидкості обробки запитів на початковій сторінці ми можемо врахувати кілька ключових параметрів, таких як час завантаження сторінки, кількість запитів до сервера, і обсяг переданих даних. Це допоможе оцінити ефективність і оптимізацію вашої CRM системи порівняно з KeepinCRM. Першочергово хочу відмітити, що загальна кількість клієнтів однакова, як і поля до них. І при однаковій корисній інформації для користувача ми бачимо результати, представлені на рис. 4.12–4.13.

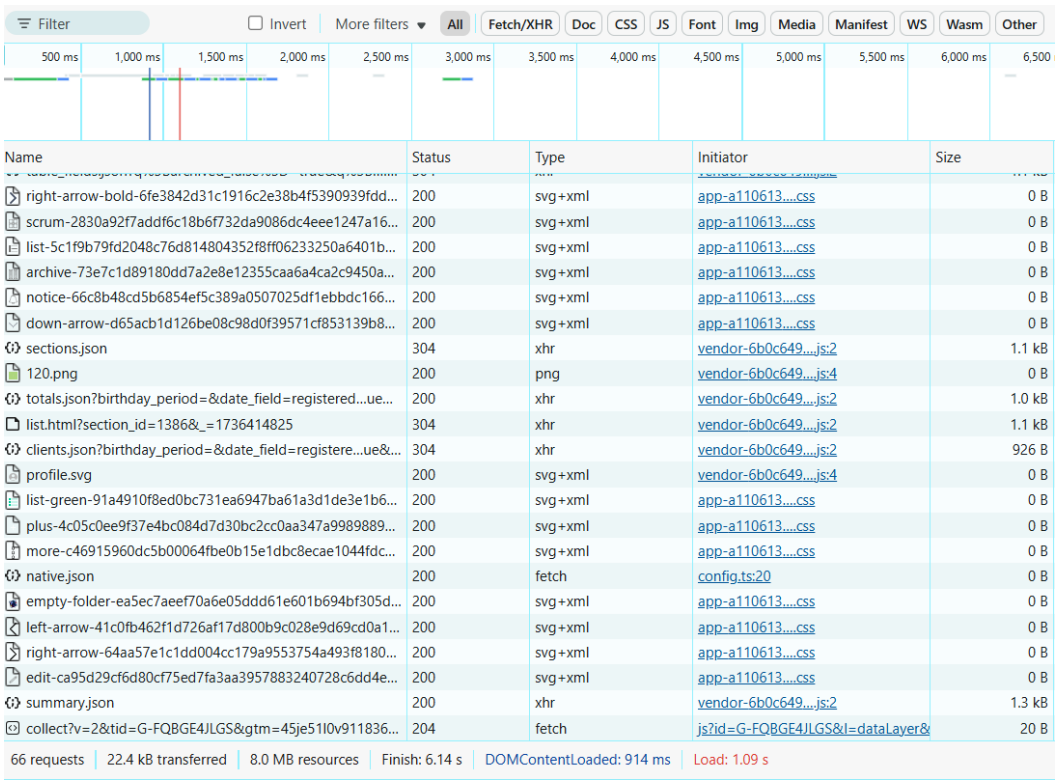


Рисунок 4.12 – Показники завантаження сторінки KeepinCRM

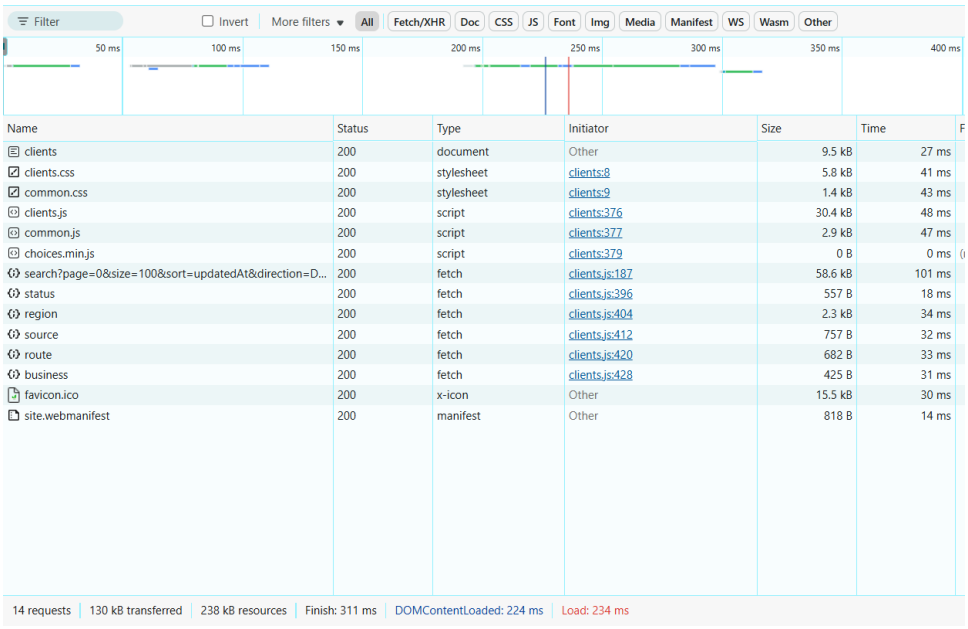


Рисунок 4.13 – Показники завантаження сторінки розробленої системи

Як ми можемо бачити, для завантаження сторінки з клієнтами KeepinCRM знадобилось зробити 66 запитів до сервера. При цьому моя система робить тільки 14. Тобто, моя система робить більше чим в 4 рази менше запитів для отриманні базової сторінки, з якої починає робити менеджер. Але потрібно відмітити, що кількість запитів не є найкращим показником, адже при великому бажанні можна зробити завантаження сторінки і за один запит. Тому розглядаємо показники далі, а саме Transferred – це показник, який вказує на фактичний обсяг даних, що передається від сервера до клієнта під час завантаження сторінки. Він враховує лише ті дані, які були реально передані через мережу. І ми бачимо, що тут KeepinCRM завантажує лише 22.4kB, тоді як моя система завантажила 130kB. І можна було б подумати, що моя система завантажує в 5 разів більше даних, але це не так, тому що якщо брати показники першого завантаження сторінки, рис. 4.14, то там взагалі можна побачити набагато гірші показники по всім критеріям, а по Transferred взагалі 2MB, що означає завантаження в більше ніж в 15 разів більше даних. При тому, що моя система завжди завантажує 130kB, незалежно від першого чи останнього завантаження сторінки. Тобто, можна відмітити, що при налаштуванні систем кешування даних, їх оптимізацію, а ці два пункти не були виконані, то моя сторінка буде грузити ще менше даних.

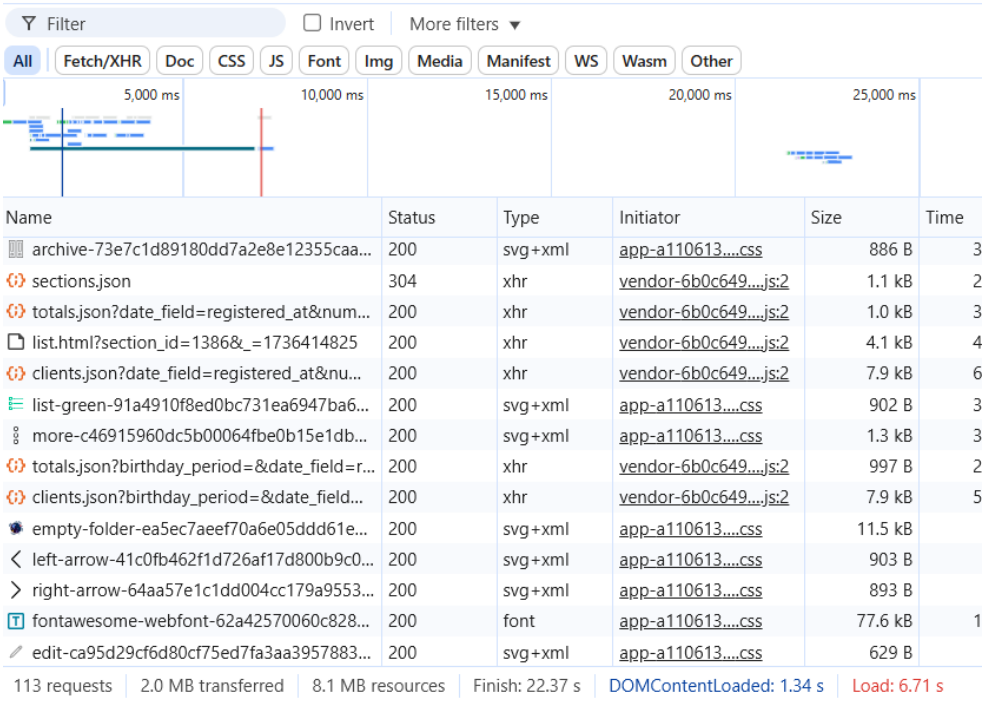


Рисунок 4.14 – Показники першого завантаження сторінки KeepinCRM

Наступний показник – це Resources, тут в мене показник стабільно кращий в 33 рази, при такому розриві немає сенсу щось взагалі розписувати. Двома словами: повна домінація.

Далі іде Finish, але його немає сенсу тут порівнювати, так як KeepinCRM постійно щось завантажує, тому він може доходити навіть до декількох хвилин, а користувач може починати працювати набагато раніше, тому він не впливає на користувальницький досвід.

Показник DOMContentLoaded – цей показник відображає момент, коли HTML–документ був повністю завантажений і розпарсений, не враховуючи стилі та зображення. І хоча моя система знову тут отримує перевагу, так як бачимо 914ms проти 224ms, більше ніж в 3 рази перевага, але є важливий момент: для користувача не має значення цей показник, адже важливо коли вся сторінка і всі її ресурси (включаючи стилі, зображення, скрипти і т.д.) повністю завантажені – а це показник Load. Тут моя система також має перевагу більше ніж в 3 рази, але є важливий момент: користувач KeepinCRM не може почати користуватися системою після фіксації показника Load, орієнтовно проходить ще 1–2 секунди, тільки тоді з'явиться список клієнтів і все інше. Тому

сподіватись на цей показник при користуванні KeepinCRM не можемо – для її роботи ще підгружаються файли, в той час як моя система може одразу функціонувати після завантаження Load.

Інші сторінки мають схожі показники, тобто моя система є швидшою мінімум в 3 рази при завантаження сторінок списку клієнтів. Також моя система є більш оптимізованою як ми можемо бачити по завантажених ресурсах, а саме в 33 рази незалежно від першого або повторного завантаження.

Ще можна зробити акцент на фільтрах та сортуванні даних. В моїй системі це навантаження віддано серверу, тому при перерахованих діях в мене швидкість відображення даних відбувається зі швидкістю запиту до сервера, а саме 113ms рис. 4.15.

Рисунок 4.15 – Показники завантаження фільтрів розробленої системи

А ось KeepinCRM робить ці дії на стороні клієнта, що розтягує ці дії до декількох секунд, що дуже погано впливає на користувальницький досвід.

4.3 Охорона праці

Освітлення та мікроклімат: робоче місце повинно бути облаштоване з урахуванням правил ергономіки та відповідати нормативним вимогам щодо освітлення та мікроклімату. Природне освітлення є найкращим варіантом, оскільки воно зменшує втому очей та підвищує продуктивність праці. У разі недостатності природного освітлення слід застосовувати штучне освітлення з достатньою яскравістю та правильним розташуванням світильників. Також важливо забезпечити комфортний мікроклімат на робочому місці, підтримуючи оптимальну температуру та вологість повітря.

Правила особистої гігієни: робота за комп'ютером передбачає тривале перебування на одному місці, тому важливо дотримуватися правил особистої гігієни. Це включає регулярне миття рук перед початком роботи та після її

завершення, щоб уникнути накопичення бактерій та бруду на клавіатурі та миші. Крім того, необхідно регулярно очищати робоче місце та обладнання від пилу та забруднень.

Перерви на відпочинок: тривала робота за комп'ютером може спричиняти втому очей, м'язову напругу та стрес. Для запобігання цьому важливо робити регулярні перерви на відпочинок. Оптимальним є режим роботи з перервами кожні 50–60 хвилин, під час яких слід виконувати прості фізичні вправи або короткі прогулянки. Це допоможе зменшити монотонність праці, полегшити нервово–емоційне напруження та підтримати фізичну активність.

Правила електробезпеки: працівники, що працюють за комп'ютером, повинні знати правила електробезпеки та володіти навичками надання першої допомоги при нещасних випадках. Важливо дотримуватися правил використання електрообладнання, уникати перевантажень електромережі та періодично перевіряти справність електроприладів. Розетки та подовжувачі повинні бути у справному стані та відповідати вимогам безпеки.

Пожежна безпека: оскільки комп'ютери та інше електронне обладнання можуть спричинити пожежу, важливо знати та дотримуватися правил пожежної безпеки. Працівники повинні бути ознайомлені з розташуванням первинних засобів пожежогасіння та вміти ними користуватися у разі необхідності. Крім того, на робочому місці повинні бути встановлені протипожежні датчики та сигналізація.

Медичні огляди: для забезпечення безпечних умов праці необхідно регулярно проходити медичні огляди. Це допоможе вчасно виявити та запобігти можливим проблемам зі здоров'ям, які можуть бути спричинені тривалою роботою за комп'ютером. Медичні огляди дозволяють контролювати стан зору, опорно–рухової системи та загальний стан здоров'я працівника.

Для посилання на нормативно–правові акти, що регулюють охорону праці при роботі за комп'ютером, ви можете використовувати наступні документи:

ДСанПіН 3.3.2.007–98 (Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно–обчислювальних машин),

які встановлюють вимоги до організації робочих місць, режимів праці та відпочинку, освітлення, мікроклімату та інших аспектів роботи за комп'ютером.

НПАОП 40.1–1.21–98 (Правила безпечної експлуатації електроустановок споживачів), що містять вимоги до забезпечення електробезпеки при експлуатації електрообладнання, включаючи комп'ютерну техніку.

Робота системи на практиці була перевірена за допомогою перевірочних та демонстраційних тестів, які підтвердили її ефективність та надійність. Ці тести дозволили виявити та усунути можливі недоліки, забезпечивши стабільну роботу системи. Охорона праці також була врахована під час розробки, включаючи дотримання норм безпеки та ергономічності, щоб гарантувати безпечне та комфортне використання системи для кінцевих користувачів [53].

ВИСНОВКИ

Вступ роботи висвітлює актуальність та значущість розробки комп'ютеризованої системи для візуалізації даних, визначає мету, основні завдання, об'єкт, предмет дослідження та методи роботи. Перший розділ аналізує предметну область, описуючи комп'ютеризовані системи, принципи та методи візуалізації даних, сучасні технології веб–розробки, архітектури веб–додатків та питання безпеки.

Другий розділ обґрунтовує важливість дослідження та підкреслює актуальність розробки системи візуалізації даних. Аналіз існуючих рішень виявив їхні недоліки та визначив напрями для покращення. Було сформульовано конкретні вимоги до системи для забезпечення ефективності та відповідності сучасним потребам. Вибір теми дослідження обґрунтовано потребою у створенні гнучких інструментів візуалізації даних та важливістю практичного досвіду у сфері веб–розробки.

Третій розділ описує процес розробки системи для візуалізації інформації з баз даних на вебсайті, включаючи створення структури бази даних, розробку архітектури безпеки, інтеграцію RESTful сервісів, реалізацію бізнес–логіки, пагінацію та синхронізацію даних, а також розробку користувацького інтерфейсу. Усі етапи забезпечили високу продуктивність, надійність та безпеку системи, досягнувши основну мету дослідження.

Четвертий розділ присвячено практичному застосуванню системи, включаючи перевірочні та демонстраційні тести, що підтвердили її функціональність та надійність. Розглянуто питання охорони праці при роботі за комп'ютером, забезпечуючи безпеку та комфорт для користувачів. Результати показали стабільну роботу системи, що відповідає визначеним вимогам, та її готовність до впровадження у реальних умовах.

Також, отримані результати роботи можно віднести до Цілі сталого розвитку 15 “Захист та відновлення екосистем суші”, а саме п. 15.3 “ Відновити деградовані землі та ґрунти з використанням інноваційних технологій”.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. ДСТУ 3008–2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлення. К.: Вид–во стандартів, 2016. 26 с.

2. Методичні вказівки з підготовки та захисту кваліфікаційної роботи здобувачами другого (магістерського) рівня вищої освіти спеціальності 174 Автоматизація, комп'ютерно–інтегровані технології та робототехніка, освітньо–професійних програм: «Комп'ютерно–інтегровані технологічні процеси і виробництва», «Комп'ютеризовані та робототехнічні системи» / Упоряд. І. Ш. Невлюдов, Р. В. Артюх, В. В. Безкоровайний, Н. П. Демська, В. В. Євсєєв, О. І. Филипенко, О. М. Цимбал. Харків: ХНУРЕ, 2024. 57 с.

3. Стаття [Електронний ресурс] / Режим доступу: https://www.ontu.edu.ua/download/konfi/2024/Collection–of–abstracts–of–the–conference–ITIA–2024.pdf (дата доступу: 09.11.2024).

4. Tableau. What Is Data Visualization? [Електронний ресурс] / Режим доступу: https://www.tableau.com/visualization/what–is–data–visualization (дата доступу: 09.11.2024).

5. Cambridge Dictionary. Classification. [Електронний ресурс] / Режим доступу: https://dictionary.cambridge.org/dictionary/english/classification (дата доступу: 09.11.2024).

6. DataLakeHouse.io. How to Visualize Time Series Data (With Examples). [Електронний ресурс] / Режим доступу: https://www.datalakehouse.io/blog/how–to–visualize–time–series–data–with–examples/ (дата доступу: 09.11.2024).

7. Tableau. 10 Examples of Interactive Map Data Visualizations. [Електронний ресурс] / Режим доступу: https://www.tableau.com/learn/articles/interactive–map–and–data–visualization–examples (дата доступу: 09.11.2024).

8. Harvard Business School Online. 17 Important Data Visualization Techniques. [Електронний ресурс] / Режим доступу:

https://online.hbs.edu/blog/post/data–visualization–techniques (дата доступу: 09.11.2024).

9. CareerFoundry. What Is Quantitative Data? [Електронний ресурс] / Режим доступу: https://careerfoundry.com/en/blog/data–analytics/what–is–quantitative–data/ (дата доступу: 09.11.2024).

10. QuestionPro. Qualitative Data: Definition, Types, Analysis and Examples. [Електронний ресурс] / Режим доступу: https://www.questionpro.com/blog/qualitative–data/ (дата доступу: 09.11.2024).

11. DataLakeHouse.io. How to Visualize Time Series Data (With Examples). [Електронний ресурс] / Режим доступу: https://www.datalakehouse.io/blog/how–to–visualize–time–series–data–with–examples/ (дата доступу: 09.11.2024).

12. Atlan. Spatial Data: Definition, Types, Examples, Use Cases & More! [Електронний ресурс] / Режим доступу: https://atlan.com/spatial–data/ (дата доступу: 09.11.2024).

13. TechChange. Why Your Audience Matters in Data Visualization. [Електронний ресурс] / Режим доступу: https://www.techchange.org/2015/05/21/audience–matters–in–data–visualization/ (дата доступу: 09.11.2024).

14. Examples Lab. Justification of a Project or Research. [Електронний ресурс] / Режим доступу: https://www.exampleslab.com/7–examples–of–justification–of–a–project–or–research/ (дата доступу: 09.11.2024).

15. MDN Web Docs. What is JavaScript? [Електронний ресурс] / Режим доступу: https://developer.mozilla.org/en–US/docs/Learn/JavaScript/First_steps/What_is_JavaScript (дата доступу: 09.11.2024).

16. D3 by Observable. What is D3? [Електронний ресурс] / Режим доступу: https://d3js.org/what–is–d3 (дата доступу: 09.11.2024).

17. Chart.js Documentation. [Електронний ресурс] / Режим доступу: https://www.chartjs.org/docs/latest/ (дата доступу: 09.11.2024).

18. Wikipedia. Highcharts. [Електронний ресурс] / Режим доступу: https://en.wikipedia.org/wiki/Highcharts (дата доступу: 09.11.2024).

19. Plotly. Plotly.js – Open Source JavaScript Graphing Library. [Електронний ресурс] / Режим доступу: https://plotly.com/javascript/ (дата доступу: 09.11.2024).

20. ApexCharts. ApexCharts.js – Open Source JavaScript Charts for your website. [Електронний ресурс] / Режим доступу: https://apexcharts.com/ (дата доступу: 09.11.2024).

21. Angular Official Documentation. What is Angular? [Електронний ресурс] / Режим доступу: https://angular.io/guide/what–is–angular (дата доступу: 09.11.2024)

22. React Official Documentation. Getting Started with React. [Електронний ресурс] / Режим доступу: https://legacy.reactjs.org/docs/getting–started.html (дата доступу: 09.11.2024).

23. Vue.js Official Documentation. Introduction to Vue.js. [Електронний ресурс] / Режим доступу: https://vuejs.org/guide/introduction.html (дата доступу: 09.11.2024).

24. LearnSQL.com. What Is an SQL Database? [Електронний ресурс] / Режим доступу: https://learnsql.com/blog/sql–database/ (дата доступу: 09.11.2024).

25. Wikipedia. NoSQL. [Електронний ресурс] / Режим доступу: https://en.wikipedia.org/wiki/NoSQL (дата доступу: 09.11.2024).

26. Wikipedia. Python (programming language). [Електронний ресурс] / Режим доступу: https://en.wikipedia.org/wiki/Python_%28programming_language%29 (дата доступу: 09.11.2024).

27. Wikipedia. PHP. [Електронний ресурс] / Режим доступу: https://en.wikipedia.org/wiki/PHP (дата доступу: 09.11.2024).

28. Wikipedia. GitHub. [Електронний ресурс] / Режим доступу: https://en.wikipedia.org/wiki/GitHub (дата доступу: 09.11.2024).

29. Wikipedia. Model–view–controller. [Електронний ресурс] / Режим доступу:

https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller (дата доступу: 09.11.2024).

30. Built In. What Is MVVM (Model–View–ViewModel)? [Електронний ресурс] / Режим доступу: https://builtin.com/software–engineering–perspectives/mvvm–architecture (дата доступу: 09.11.2024).

31. Wikipedia. Model–view–presenter. [Електронний ресурс] / Режим доступу: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter (дата доступу: 09.11.2024).

32. Cloudflare. What is Web Application Security? [Електронний ресурс] / Режим доступу: https://www.cloudflare.com/learning/security/what–is–web–application–security/ (дата доступу: 09.11.2024).

33. Cloudflare. What is Cross–Site Scripting (XSS)? [Електронний ресурс] / Режим доступу: https://www.cloudflare.com/learning/security/threats/cross–site–scripting/ (дата доступу: 09.11.2024).

34. OWASP Foundation. Cross–Site Request Forgery (CSRF). [Електронний ресурс] / Режим доступу: https://owasp.org/www–community/attacks/csrf (дата доступу: 09.11.2024).

35. TechTarget. Multi–factor Authentication (MFA). [Електронний ресурс] / Режим доступу: https://www.techtarget.com/search/security/multi–factor–authentication–mfa (дата доступу: 09.11.2024).

36. Okta. What Is Biometric Authentication? Biometrics Explained. [Електронний ресурс] / Режим доступу: https://www.okta.com/blog/2020/07/biometric–authentication/ (дата доступу: 09.11.2024).

37. Wikipedia. HTTPS. [Електронний ресурс] / Режим доступу: https://en.wikipedia.org/wiki/HTTPS (дата доступу: 09.11.2024).

38. Retool: Retool. [Електронний ресурс] / Режим доступу: https://retool.com/ (дата доступу: 09.11.2024).

39. ClickUp: ClickUp. [Електронний ресурс] / Режим доступу: https://clickup.com/ (дата доступу: 09.11.2024).

40. Competitors.app: Competitors.app. [Електронний ресурс] / Режим доступу: https://competitors.app/ (дата доступу: 09.11.2024).

41. Intelemark: Intelemark. [Електронний ресурс] / Режим доступу: https://intelemark.com/ (дата доступу: 09.11.2024).

42. Qlik: Qlik. [Електронний ресурс] / Режим доступу: https://qlik.com/ (дата доступу: 09.11.2024).

43. PeerPanda: PeerPanda. [Електронний ресурс] / Режим доступу: https://peerpanda.com/ (дата доступу: 09.11.2024).

44. Improvado: Improvado. [Електронний ресурс] / Режим доступу: https://improvado.io/ (дата доступу: 09.11.2024).

45. Geckoboard: Geckoboard. [Електронний ресурс] / Режим доступу: https://geckoboard.com/ (дата доступу: 09.11.2024).

46. Salesforce: Salesforce. [Електронний ресурс] / Режим доступу: https://salesforce.com/ (дата доступу: 09.11.2024).

47. Klipfolio: Klipfolio. [Електронний ресурс] / Режим доступу: https://klipfolio.com/ (дата доступу: 09.11.2024).

48. Cvent: Cvent. [Електронний ресурс] / Режим доступу: https://cvent.com/ (дата доступу: 09.11.2024).

49. BDC.ca: BDC.ca. [Електронний ресурс] / Режим доступу: https://bdc.ca/ (дата доступу: 09.11.2024).

50. Sprout Social: Sprout Social. [Електронний ресурс] / Режим доступу: https://sproutsocial.com/ (дата доступу: 09.11.2024).

51. HubSpot: HubSpot. [Електронний ресурс] / Режим доступу: https://hubspot.com/ (дата доступу: 09.11.2024).

52. Wikipedia. JSON (JavaScript Object Notation). [Електронний ресурс] / Режим доступу: https://en.wikipedia.org/wiki/JSON (дата доступу: 09.11.2024).

53. Інструкція з охорони праці [Електронний ресурс] / Режим доступу: https://pro-op.com.ua/article/485-nstruktsya-z-ohoroni-prats-pri-robot-na-personalnomu-kompyuter (дата доступу: 09.11.2024).