МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ЛУХВЕРЧИК СЕРГІЙ АНДРІЙОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
канд. техн. наук, доцент
_____ О. В. Зелінська
«____»_____ 20__ р.

РОЗРОБКА СИСТЕМИ КЕРУВАННЯ ЕФЕКТИВНІСТЮ ВИРОБНИЧИХ АКТИВІВ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
С. М. Цирульник, доцент кафедри
інформаційних технологій,
к.т.н., доцент
_____

Оцінка:____/ ____/ _____

(бали за шкалою СКТС/за національною шкалою)

Голова ЕК: _____

(підпис)

Вінниця 2024

АНОТАЦІЯ

Лухверчик С.А. Розробка системи керування ефективністю виробничих активів. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця 2024.

У кваліфікаційній (бакалаврській) роботі досліджено та проаналізовано поняття системи керування ефективністю виробничих активів, їх роль в сучасних веб застосунках та переваги, а також надано практичний приклад застосування.

Ключові слова: моніторинг, бази даних, системи керування, JWT, EPM, HTTPS, архітектура системи.

76 с., 31 рис., 21 джерело.

ABSTRACT

Lukhverchyk S.A. Development of a system for managing the efficiency of production assets. Specialty 122 "Computer Science", educational program "Computer Science". Vasyl' Stus Donetsk National University, Vinnytsia, 2024.

The qualification (bachelor's) thesis investigates and analyzes the concept of a production asset performance management system, their role in modern web applications and advantages, and provides a practical example of application.

Keywords: monitoring, databases, management systems, JWT, EPM, HTTPS, system architecture.

76 p., 31 pict., 21 source.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Grafana – Платформа для моніторингу та аналізу даних з різноманітних джерел

EAM – (Enterprise asset management) це система управління активами підприємства, яка допомагає організаціям відслідковувати, обслуговувати та оптимізувати фізичні активи протягом їх життєвого циклу

Java – Мова програмування високого рівня, що використовується для розробки серверної частини додатку, відома своєю переносимістю між різними платформами

JavaScript – це динамічна мова програмування, що використовується для створення інтерактивних веб-сторінок

Dashboard – це інтерфейс, який надає візуалізацію ключових показників та даних для моніторингу, аналізу та управління діяльністю організації або системи в реальному часі

IoT – (Internet of Things) це мережа фізичних пристроїв, оснащених датчиками, програмним забезпеченням та іншими технологіями, які дозволяють їм підключатися до Інтернету, збирати та обмінюватися даними

SCADA – (Supervisory Control and Data Acquisition) це система нагляду та збору даних, яка використовується для контролю та управління процесами у промислових об'єктах.

CRM – (Customer Relationship Management) це система управління відносинами з клієнтами, яка допомагає компаніям залучати, зберігати і задовольняти клієнтів через автоматизацію та аналіз даних

ПЗ – програмне забезпечення

API – (Application Programming Interface) це набір визначень взаємодії різнотипного програмного забезпечення

React – JavaScript бібліотека для побудови динамічного користувацького інтерфейсу

Redux – це бібліотека для управління станом у JavaScript-застосунках, яка допомагає централізовано зберігати та керувати станом, спрощуючи відстеження змін та відлагодження

TypeScript – розширення JavaScript, що забезпечує строгу типізацію, покращує роботу над кодом та зменшує кількість помилок

Redux Thunk – Middleware для Redux, що дозволяє писати асинхронні дії, спрощує управління складними асинхронними операціями та дозволяє легко взаємодіяти з сервером

React Router – бібліотека для маршрутизації в React додатках. Дозволяє створювати динамічні та зручні для використання інтерфейси

Axios – бібліотека для виконання HTTP-запитів у браузері. Легка взаємодія з сервером та отримання потрібних даних, що робить комунікацію між клієнтом і сервером зручною та ефективною

React-Jhipster – набір інструментів для швидкої розробки React-додатку, який включає в себе генерацію інтерфейсів, функціонал перекладу (i18n), валідацію, пагінацію, локальне сховище та інше

Leaflet та React Leaflet – бібліотеки для інтерактивних карт

React Hook Form – бібліотека для керування формами в React за допомогою хуків

Joi – бібліотека для валідації даних в JavaScript. Надає засоби для визначення схем даних та їх валідації, що дозволяє забезпечити правильність та цілісність даних у додатку

React-toastify – бібліотека для створення повідомлень (toast) у React. Дозволяє зручно та ефективно сповіщати користувачів про різні події у додатку

React-Select – бібліотека для створення та стилізації селектів у React

React-fontawesome – бібліотека для використання іконок FontAwesome у React

Swiper – бібліотека для створення динамічних та інтерактивних слайдерів у React, надаючи можливість налаштовувати та стилізувати слайдери, роблячи їхнє використання приємним та зручним для користувачів

Day.js – бібліотека для зручного виконання операцій з датами та часом

HTML – (HyperText Markup Language) мова розмітки гіпертексту, що використовується для створення структури та вмісту веб-сторінок

CSS – мова стилізації, що використовується для задання зовнішнього вигляду веб-сторінок

SCSS – препроцесор для CSS, що додає додаткові можливості до звичайного CSS. Використання SCSS дозволяє створювати більш структурований та підтримуваний CSS-код, що спрощує розробку та підтримку стилів у проекті

Material UI – бібліотека компонентів для React на основі Material Design

Reactstrap – бібліотека Bootstrap компонентів для React, створюючи чучасний та привабливий вигляд додатку

ESLint – інструмент для перевірки стилю коду в JavaScript

Prettier – інструмент для форматування коду у визначений стиль, що дозволяє підтримувати єдиний стиль коду в рамках проекту

EditorConfig – формат для створення та підтримки конфігураційних файлів для редакторів коду

Husky – інструмент для запуску скриптів git hooks. Дозволяє автоматизувати виконання різних дій, таких як перевірка коду перед комітом, виконання тестів перед пушем на віддалений репозиторій

Git – система контролю версій, яка використовується для відстеження змін у коді та спільної роботи над проектом. Надає засоби для зберігання та керування версіями вашого коду

Java – мова програмування високого рівня, що використовується для розробки серверної частини додатку, відома своєю переносимістю між різними платформами

Spring Boot – фреймворк, що дозволяє спрощено створювати stand-alone, виробничі Spring-базові додатки з мінімальним зусиллям

Hibernate – один із провідних фреймворків для об'єктно-реляційного відображення (ORM), що дозволяє здійснювати взаємодію Java-додатку з базою даних

Maven – інструмент для автоматизації збірки проектів на Java, який також забезпечує управління залежностями

Docker – платформа для розробки, доставки та запуску додатків у контейнерах, що забезпечує легкість впровадження та масштабування додатків

Docker-compose – інструмент для визначення та запуску багатоконтейнерних додатків Docker, дозволяє з легкістю конфігурувати взаємозв'язки між контейнерами

RabbitMQ – проміжне програмне забезпечення для передачі повідомлень, що дозволяє асинхронну взаємодію між компонентами системи

ELK Stack – набір інструментів для пошуку, аналізу та візуалізації даних в реальному часі

Timescale – розширення для PostgreSQL, оптимізоване для зберігання та аналізу даних у часових рядах

Hazelcast – розподілена кеш-система та система управління інмеморі даними, що дозволяє підвищити швидкість обробки даних

MapStruct – інструмент для автоматизації мапінгу між Java-бінами, що дозволяє зменшити кількість шаблонного коду

TestContainers – бібліотека для Java, що дозволяє створювати легкі, переносні та відтворювані середовища для інтеграційного тестування за допомогою Docker контейнерів

H2 Database – вбудована база даних на Java, що підтримує SQL та JDBC API, використовується для розробки та тестування

Checkstyle – інструмент для перевірки вихідного коду Java на відповідність заданим стандартам форматування і стилю

Liquibase – інструмент для управління версіями баз даних, що дозволяє відстежувати, модифікувати та розгортати зміни схеми бази даних в контрольований та автоматизований спосіб

ВСТУП

Актуальність теми дослідження. У сучасних умовах індустріалізації та автоматизації виробництва, ефективне управління виробничими активами є критичним для успіху підприємств. Висока конкуренція на ринку вимагає від компаній оптимізації процесів та мінімізації витрат на обслуговування та ремонт обладнання. Це підкреслює необхідність впровадження систем керування ефективністю виробничих активів (EAM), що дозволяють здійснювати моніторинг, планування та управління ресурсами в режимі реального часу.

Метою цієї дипломної роботи є розробка та впровадження системи керування ефективністю виробничих активів, що поєднує сучасні інформаційні технології для забезпечення високого рівня продуктивності та надійності обладнання на виробництві. Система «SAKURA» складається з бекенду на Java та фронтенду на JavaScript, а також ряду інших використаних інструментів, забезпечуючи інтерактивний і зручний інтерфейс користувача.

Об’єктом дослідження є система керування ефективністю виробничих активів на підприємстві.

Предмет дослідження: є розробка та використання системи «SAKURA» для оптимізації процесів управління виробничими активами, зокрема функціонал моніторингу робочих місць, операційний дашборд для планування та організації завдань, а також енергомоніторинг обладнання.

Ключовим елементом дослідження є інтеграція сучасних інформаційних технологій, таких як Інтернет речей (IoT), хмарні сервіси, аналітика великих даних та мобільні додатки, що забезпечують високий рівень автоматизації та ефективності системи «SAKURA».

Основним завданням дипломної роботи є створення прототипу системи «SAKURA», який дозволить підприємствам ефективно керувати виробничими активами, знижувати витрати на обслуговування та підвищувати загальну продуктивність.

В якості теоретичного значення одержаних результатів полягає у розробці концептуальних основ EAM систем, принципів інтеграції сучасних IT технологій у процеси управління активами, а також аналізу специфічних вимог та функціональності для забезпечення ефективної роботи виробничих підприємств.

Апробація результатів дослідження. За тематикою дослідження було опубліковано одну наукову працю у збірнику матеріалів Всеукраїнської науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2024)» (Вінниця, 2024).

У **результаті** цього дослідження очікується розробка прототипу EPM системи, що дозволить ефективно керувати виробничими активами на підприємствах.

РОЗДІЛ 1 РОЗКРИТТЯ ПОНЯТТЯ СИСТЕМИ КЕРУВАННЯ ЕФЕКТИВНІСТЮ ВИРОБНИЧИХ АКТИВІВ

1.1 Поняття системи керування ефективністю виробничих активів

Визначення та призначення системи керування ефективністю виробничих активів:

Система керування ефективністю виробничих активів (англ. Enterprise Asset Management, EAM) є комплексним підходом до управління фізичними активами підприємства з метою максимізації їхньої ефективності, зниження витрат на обслуговування та підвищення загальної продуктивності [1]. Вона включає в себе стратегічне планування, моніторинг, аналіз та оптимізацію використання активів протягом їхнього життєвого циклу.

Основні функції системи керування ефективністю виробничих активів включають:

1. Моніторинг стану активів – збір та аналіз даних про поточний стан обладнання, його технічні характеристики, історію використання та обслуговування;

2. Планування обслуговування – автоматизація процесів планування профілактичного та коригуючого обслуговування для запобігання зупинок виробництва та продовження терміну служби активів;

3. Управління запасами та ресурсами – оптимізація використання запасних частин та матеріалів, управління людськими ресурсами, необхідними для обслуговування обладнання;

4. Аналіз та звітність – проведення аналізу ефективності використання активів, формування звітів для підтримки прийняття рішень на основі даних;

5. Інтеграція з іншими системами – зв'язок з ERP-системами, SCADA-системами та іншими корпоративними інформаційними системами для забезпечення комплексного підходу до управління активами [2];

Історія та розвиток:

Поняття системи керування ефективністю виробничих активів виникло в середині 20-го століття, коли підприємства почали усвідомлювати необхідність ефективного управління своїми фізичними активами для забезпечення стабільності виробничих процесів. Перші EAM-системи були простими базами даних, що містили інформацію про активи та їх обслуговування [3].

З розвитком інформаційних технологій EAM-системи еволюціонували, набуваючи нових функціональних можливостей та інтеграційних можливостей. Сучасні EAM-системи включають передові технології, такі як Інтернет речей (IoT), аналітика великих даних, штучний інтелект та хмарні обчислення, що дозволяють здійснювати моніторинг та аналіз активів у режимі реального часу.

З поширенням мобільних технологій EAM-системи стали доступними на різних пристроях, що дозволяє персоналу здійснювати управління та обслуговування активів незалежно від місця знаходження. Це значно підвищує оперативність реагування на потенційні проблеми та сприяє підвищенню загальної ефективності виробництва.

Основна ціль та мотивація для створення:

Основною ціллю створення системи керування ефективністю виробничих активів є забезпечення підприємствам можливості ефективно управляти своїми фізичними активами, зменшувати витрати на їх обслуговування та підвищувати загальну продуктивність виробництва. Це досягається шляхом впровадження сучасних технологій моніторингу та аналізу, автоматизації процесів обслуговування та інтеграції з іншими корпоративними системами.

Мотивація для створення таких систем виникла з потреби підприємств у зниженні простоїв обладнання, оптимізації витрат на обслуговування та підвищенні ефективності використання ресурсів. В умовах зростаючої конкуренції та технологічних викликів, підприємства прагнуть отримати максимальну віддачу від своїх активів, забезпечуючи їхню безперебійну роботу та продовження терміну експлуатації.

Методології та підходи до CRM:

Існує кілька методологій та підходів до впровадження систем керування ефективністю виробничих активів. Один із найбільш поширених - це підхід з фокусом на надійність (Reliability-centered maintenance, RCM). Цей підхід передбачає систематичний аналіз функцій активів, визначення можливих відмов та розробку стратегій обслуговування для мінімізації ризиків простоїв [4].

Інші підходи включають:

1. Технологічний підхід – використання передових інформаційних технологій для автоматизації процесів обслуговування та моніторингу активів;

2. Процесно-орієнтований підхід – інтеграція EAM з бізнес-процесами підприємства для забезпечення узгодженості дій та підвищення загальної ефективності;

3. Аналітичний підхід – використання аналітичних інструментів для аналізу даних про активи та прийняття обґрунтованих рішень щодо їхнього обслуговування та модернізації;

Ці підходи дозволяють підприємствам досягати високих показників ефективності та надійності своїх виробничих активів, забезпечуючи конкурентоспроможність на ринку та стійкий розвиток.

1.2 Огляд існуючих систем керування ефективністю виробничих активів

Для розуміння основних прийомів та функціональних можливостей, які слід враховувати під час розробки системи керування ефективністю виробничих активів (EAM), важливо проаналізувати поточні лідери цього ринку. Розглянемо основні особливості, можливості та варіативність функціоналу таких систем, як IBM Maximo, SAP EAM, Infor EAM, і Oracle EAM.

IBM Maximo - є однією з найпопулярніших і найпотужніших EAM систем у світі. Вона пропонує комплексний підхід до управління активами, включаючи технічне обслуговування, управління запасами, закупівлями та трудовими

ресурсами. Система відзначається високим рівнем налаштовуваності та гнучкістю, що дозволяє адаптувати її до специфічних потреб будь-якого підприємства. Основні особливості IBM Maximo включають [5]:

- моніторинг стану активів у режимі реального часу;
- інтеграція з IoT пристроями для збору даних про роботу обладнання;
- іналітика та звітність на основі великих даних;
- підтримка мобільних додатків для доступу до системи з будь-якого місця.

SAP EAM – є частиною SAP S/4HANA та забезпечує комплексне управління життєвим циклом активів. Система пропонує інструменти для планування, виконання та аналізу всіх аспектів управління активами, що допомагає підприємствам оптимізувати витрати та підвищити ефективність. Ключові можливості SAP EAM [6]:

- планування профілактичного та коригуючого обслуговування;
- управління запасами та закупівлями;
- аналітика даних для виявлення тенденцій та прогнозування відмов;
- інтеграція з іншими модулями SAP для забезпечення єдиного інформаційного простору.

Infor EAM - є сучасною системою управління активами, що використовує хмарні технології та мобільні додатки. Вона забезпечує високий рівень автоматизації процесів управління активами, що дозволяє підприємствам швидко реагувати на зміни у стані обладнання та оптимізувати витрати на його обслуговування. Основні функціональні можливості Infor EAM включають [7]:

- мобільний доступ до даних про активи та процеси обслуговування;
- інтеграція з IoT для збору та аналізу даних у режимі реального часу;
- управління життєвим циклом активів від планування до утилізації;
- аналітичні інструменти для підтримки прийняття рішень на основі даних.

Oracle EAM пропонує потужний набір інструментів для управління активами на підприємствах різних галузей. Система інтегрується з іншими продуктами Oracle, що дозволяє забезпечити комплексний підхід до управління бізнес-процесами та оптимізації використання активів. Ключові можливості Oracle EAM [8]:

- планування та виконання обслуговування активів;
- управління запасами та ланцюгами постачання;
- Аналітика та звітність для підтримки прийняття рішень;
- інтеграція з іншими системами Oracle для забезпечення єдиного інформаційного простору.

Кожна з цих систем має свої унікальні переваги та недоліки, які слід враховувати при розробці власної EAM системи. Вивчення досвіду та функціональних можливостей лідерів ринку допоможе створити більш ефективне та зручне рішення для управління виробничими активами. Унікальність нашої роботи полягає у створенні системи «SAKURA», яка поєднує найкращі практики з використанням сучасних технологій для забезпечення максимального рівня ефективності та зручності для користувачів.

1.3 Основні принципи роботи систем керування ефективністю активів

Системи керування ефективністю виробничих активів (EAM) базуються на ряді ключових принципів, які забезпечують оптимальне використання ресурсів, мінімізацію витрат на обслуговування та максимізацію продуктивності активів. Розглянемо основні принципи роботи таких систем [9].

1. Централізоване управління даними – одним з головних принципів є централізоване управління всіма даними про активи, включаючи їх технічні характеристики, історію обслуговування, стан та продуктивність. Це дозволяє отримувати повну картину про стан активів у режимі реального часу, що є основою для прийняття обґрунтованих рішень.

2. Прогнозне обслуговування – сучасні EAM системи використовують аналітичні інструменти та алгоритми машинного навчання для прогнозування можливих відмов обладнання. Це дозволяє проводити обслуговування не за фіксованим графіком, а за потребою, що значно знижує ризик незапланованих зупинок та оптимізує витрати на обслуговування.

3. Інтеграція з IoT та датчиками – інтеграція з Інтернетом речей (IoT) дозволяє збирати дані про роботу активів у режимі реального часу за допомогою датчиків. Це забезпечує постійний моніторинг стану обладнання та дозволяє швидко реагувати на будь-які відхилення від нормальної роботи.

4. Мобільний доступ – для підвищення оперативності та ефективності роботи, сучасні EAM системи підтримують мобільний доступ, що дозволяє співробітникам отримувати необхідну інформацію та виконувати свої задачі з будь-якого місця та у будь-який час. Це особливо важливо для польових працівників та технічного персоналу.

5. Автоматизація робочих процесів – EAM системи автоматизують багато рутинних процесів, таких як створення та управління заявками на обслуговування, закупівля запасних частин, планування робіт та контроль за їх виконанням. Це дозволяє знизити навантаження на персонал та підвищити загальну ефективність управління активами.

6. Аналіз та звітність – системи керування ефективністю виробничих активів включають потужні інструменти для аналізу даних та формування звітів. Це дозволяє керівництву підприємства отримувати аналітичні дані про ефективність використання активів, виявляти проблемні зони та приймати обґрунтовані рішення щодо їх оптимізації.

7. Гнучкість та налаштовуваність – сучасні EAM системи відзначаються високим рівнем гнучкості та налаштовуваності, що дозволяє адаптувати їх до специфічних потреб кожного підприємства. Це забезпечує ефективне управління активами незалежно від галузі та масштабу виробництва.

Застосування цих принципів у системі «SAKURA» дозволяє забезпечити високу ефективність управління виробничими активами, зниження витрат та підвищення продуктивності підприємства.

1.4 Особливості впровадження систем керування ефективністю активів

Впровадження систем керування ефективністю виробничих активів (EAM) є складним процесом, що вимагає ретельного планування та врахування специфічних потреб підприємства. Розглянемо основні етапи та особливості цього процесу.

1. Аналіз поточних процесів та визначення вимог – першим етапом є проведення детального аналізу існуючих процесів управління активами. Це включає оцінку технічного стану обладнання, існуючих методів обслуговування, та визначення ключових проблем. На основі цього аналізу формулюються вимоги до нової системи EAM.

2. Вибір системи та постачальника – на цьому етапі відбувається вибір системи EAM, яка найбільше відповідає вимогам підприємства. Важливо оцінити різні системи на ринку, їх функціональні можливості, гнучкість, інтеграційні можливості та підтримку з боку постачальника.

3. Пілотне впровадження – пілотний проект дозволяє протестувати обрану систему на невеликій частині підприємства. Це допомагає виявити можливі проблеми та визначити необхідні налаштування та адаптації перед повномасштабним впровадженням.

4. Інтеграція з існуючими системами – інтеграція з іншими інформаційними системами підприємства, такими як ERP, CRM, або SCADA, є ключовим аспектом впровадження EAM. Це дозволяє забезпечити єдиний інформаційний простір та синхронізувати всі дані про активи.

5. Налаштування та кастомізація – система EAM повинна бути налаштована відповідно до специфічних потреб підприємства. Це включає

налаштування процесів обслуговування, управління запасами, планування робіт та інших функцій системи.

6. Навчання персоналу – ефективне використання системи EAM вимагає належного навчання персоналу. Працівники повинні знати, як використовувати систему для виконання своїх завдань, та розуміти, як нові процеси покращують управління активами.

7. Повномасштабне впровадження – інтеграція з іншими інформаційними системами підприємства, такими як ERP, CRM, або SCADA, є ключовим аспектом впровадження EAM. Це дозволяє забезпечити єдиний інформаційний простір та синхронізувати всі дані про активи.

8. Моніторинг та оптимізація – після впровадження системи EAM, необхідно постійно моніторити її ефективність та вносити необхідні корективи. Це дозволяє постійно покращувати процеси управління активами та адаптувати систему до змінних умов виробництва.

Впровадження системи керування ефективністю виробничих активів є комплексним і багатоступеневим процесом, що потребує ретельного планування та адаптації до конкретних умов підприємства. Дотримання зазначених етапів забезпечить успішне впровадження системи та підвищення ефективності управління виробничими активами.

1.5 Інформаційні технології у системах керування активами

Інформаційні технології (IT) відіграють ключову роль у системах керування ефективністю виробничих активів (EAM), забезпечуючи їхню ефективність, інтеграцію та аналітичні можливості. Розглянемо основні аспекти використання IT в таких системах.

1. Інтернет речей (IoT) – інтеграція IoT дозволяє підключати виробниче обладнання до мережі, що дає можливість в режимі реального часу збирати та аналізувати дані про роботу активів [10]. Датчики, встановлені на обладнанні, забезпечують безперервний моніторинг

його стану, допомагають виявляти несправності та проводити прогнозне обслуговування.

2. Хмарні технології – хмарні рішення забезпечують доступ до даних і системи управління активами з будь-якого місця та в будь-який час. Це дозволяє легко масштабувати систему відповідно до потреб підприємства, знижуючи витрати на інфраструктуру та забезпечуючи високу доступність і безпеку даних.
3. Аналітика великих даних – завдяки технологіям великих даних (Big Data) EAM системи можуть обробляти великі обсяги інформації, отриманої від різних датчиків та джерел. Це дозволяє проводити глибокий аналіз, виявляти тренди та робити прогнози, що сприяє прийняттю обґрунтованих рішень щодо управління активами.

4. Штучний інтелект (ШІ) та машинне навчання – ШІ та машинне навчання допомагають автоматизувати процеси управління активами, такі як прогнозування відмов, оптимізація графіків обслуговування та управління запасами. Ці технології дозволяють підвищити точність прогнозів і знизити витрати на обслуговування.

5. Мобільні застосунки – мобільні застосунки забезпечують доступ до системи EAM з мобільних пристроїв, що підвищує мобільність та оперативність співробітників. Це особливо важливо для технічного персоналу, який може здійснювати обслуговування та ремонт обладнання безпосередньо на місці.

6. Інтеграція з ERP та іншими системами – EAM системи інтегруються з ERP (Enterprise Resource Planning) та іншими корпоративними системами, що забезпечує єдиний інформаційний простір для управління всіма аспектами діяльності підприємства. Це сприяє узгодженості дій та підвищенню загальної ефективності.

7. Безпека даних – інформаційні технології забезпечують високий рівень захисту даних у системах EAM. Використовуються методи шифрування, системи контролю доступу та резервного копіювання, що забезпечує безпеку і цілісність даних.

Застосування сучасних інформаційних технологій у системах керування ефективністю виробничих активів дозволяє значно підвищити продуктивність підприємства, оптимізувати витрати на обслуговування та забезпечити надійну роботу виробничого обладнання.

РОЗДІЛ 2 ПРОЕКТУВАННЯ СИСТЕМИ ТА ВИБІР ІНСТРУМЕНТІВ РОЗРОБКИ

2.1 Опис загальної архітектури

Архітектура веб-додатків описує взаємозв’язок між компонентами веб-додатку, їхню структуру та спосіб взаємодії. Вона визначає, як розподіляються функції та обов’язки між різними частинами додатку, щоб забезпечити його ефективну роботу та легкість розширення.

Для розробки ефективного та продуктивного додатку слід використовувати загальноприйняті підходи у розробці ПЗ. В першу чергу необхідно закласти клієнт-серверну модель, що забезпечує централізоване управління даними і логікою додатку, що спрощує синхронізацію даних між користувачами і зменшує ризик неконсистентної інформації. Це особливо важливо, оскільки система передбачає роботу з великим обсягом даних, що постійно оновлюються. Також, розподілення ролей між клієнтами та сервером дозволяє оптимізувати ресурси комп'ютерної системи, зменшуючи навантаження на клієнтські пристрої. Це забезпечує більш гладку та швидку роботу додатку на різноманітних пристроях користувачів. Клієнт-серверна архітектура також сприяє більшій безпеці системи через можливість централізованого управління доступом і аутентифікації користувачів, що є критичним аспектом для нашого проекту, який оперує чутливими даними. Нарешті, ця модель архітектури є масштабованою та гнучкою, дозволяючи легко додавати нові функціональні можливості та обслуговувати зростаючу кількість користувачів без потреби радикально перебудовувати усю систему. Зважаючи на зазначені переваги, клієнт-серверна архітектура є оптимальним вибором для нашого веб-додатку, забезпечуючи ефективне управління даними, високу продуктивність, масштабованість і безпеку [11].

Приклад архітектури трирівневої системи (three-tier architecture), що складається з клієнтського, серверу додатків та серверу баз даних наведено на рисунку 2.1

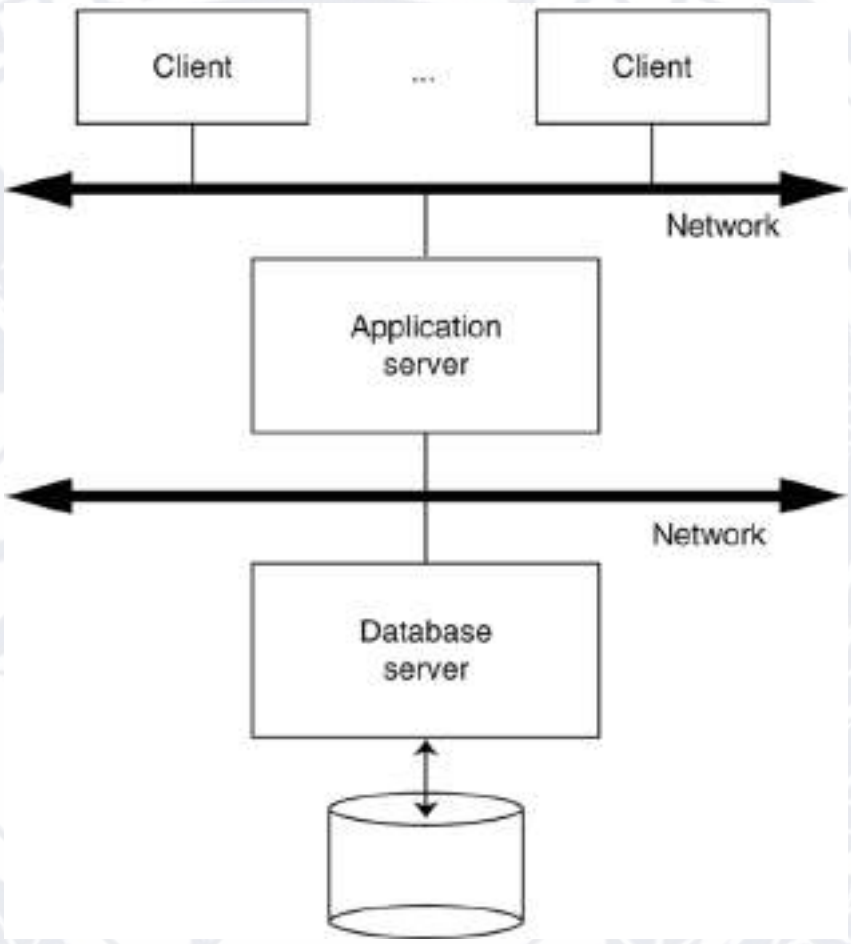


Рисунок 2.1 – Приклад архітектури трирівневої системи

В архітектурі серверного рішення необхідно впровадити n-шарову модель, що забезпечує високу гнучкість, масштабованість та легкість утримання системи. Основою архітектури є розділення відповідальностей між різними рівнями, кожен з яких відповідає за свою специфічну область функціональності. На найнижчому рівні розміщується шар даних, що відповідає за зберігання, вилучення та управління даними через ефективні засоби баз даних. Над ним розташовується шар логіки додатку, де реалізована основна бізнес-логіка, обробка даних та взаємодія з шаром даних. Цей шар є ключовим для реалізації функціональних вимог системи і може включати різні підсистеми або модулі, що

взаємодіють між собою. Далі, для забезпечення гнучкої взаємодії з користувачем та зовнішніми системами, введено шар презентації, який охоплює інтерфейси користувача, API для зовнішніх взаємодій, та інші механізми доставки вмісту до кінцевого споживача. Залежно від потреб проекту можуть бути додані додаткові шари, такі як шар безпеки для управління доступом та аутентифікацією, шар інтеграції для забезпечення взаємодії з іншими системами та сервісами, та шар кешування для оптимізації продуктивності шляхом зниження навантаження на базу даних та швидшого відгуку на запити користувачів. Використання такої багаторівневої структури дозволяє забезпечити високу адаптивність та масштабованість проекту, спростити процес розробки та підтримки, а також підвищити безпеку та надійність системи.

На прикладі, на рисунку 2.2 зображена архітектура багаторівневої програми (multilayered application architecture), зокрема, трирівнева архітектура [12], яка складається з інтерфейсу користувача, бізнес логіки та доступу до даних.

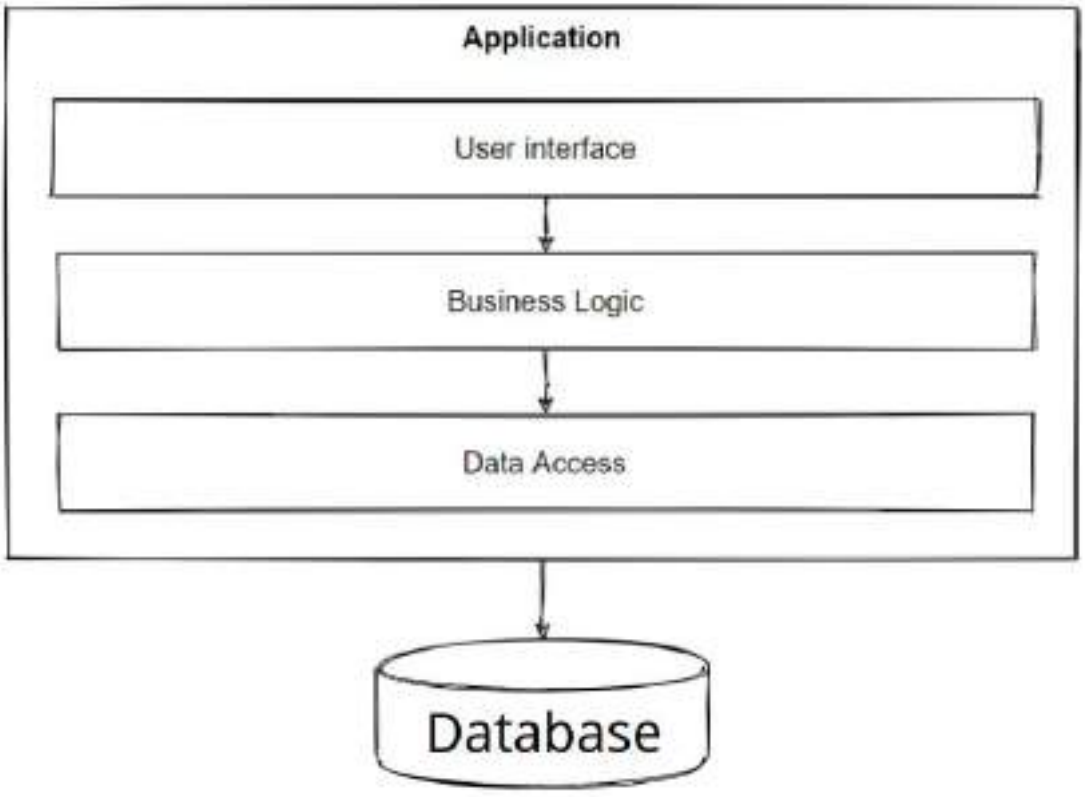


Рисунок 2.2 – Приклад архітектури багаторівневого застосунку

Для клієнтської частини проекту основним завданням є реалізація компонентної архітектури та створення гнучкої, масштабованої та легко підтримуваної системи. Основна ідея полягає у використанні модульного підходу до будівництва інтерфейсу, де кожен компонент є незалежним і може бути легко інтегрованим у різні частини системи без необхідності переписування коду. Це дозволяє швидко адаптувати інтерфейс до змінних вимог замовника, впроваджувати нові функції та вдосконалювати існуючі без втрати загальної продуктивності та стабільності системи.

Для забезпечення ефективної взаємодії між компонентами, необхідно використовувати сучасні технології та патерни проектування, такі як React або Vue для фронтенду. Використання цих фреймворків сприяє створенню інтуїтивно зрозумілого, динамічного користувацького інтерфейсу, який є легко налаштованим і забезпечує високу продуктивність додатку. Окрім того, важливо звернути увагу на систему управління станами (наприклад, Redux або Vuex), яка допоможе забезпечити консистентність даних між компонентами та спростить управління складними взаємодіями в межах аплікації.

Залучення до процесу розробки системи стилів і дизайн-системи сприятиме однорідності візуального представлення компонентів, що, в свою чергу, позитивно позначиться на загальному користувацькому досвіді. Такий підхід дозволяє забезпечити легку адаптацію інтерфейсу до різних екранів та пристроїв, забезпечуючи відмінну респонсивність та доступність додатку.

На завершення, ключовим аспектом розробки компонентної архітектури є впровадження автоматизованих тестів для кожного компоненту, що забезпечує високий рівень якості продукту та зменшує ризики при інтеграції нових функцій. Це включає як юніт-тести, так і інтеграційні тести, які допомагають швидко виявляти та виправляти помилки, забезпечуючи стабільність та надійність системи в цілому. Використання найкращих практик та сучасних інструментів розробки сприяє створенню ефективної, масштабованої та легко підтримуваної клієнтської частини, яка задовольнятиме потреби як розробників, так і кінцевих

користувачів [13]. На рисунку 2.3 зображена схема робочого процесу форм системи з використанням React і Redux.

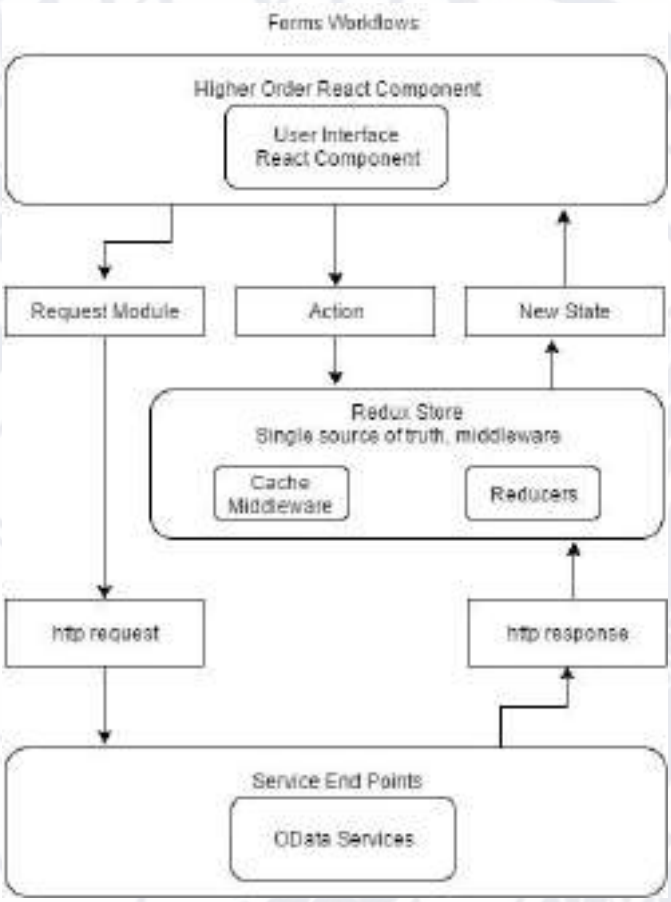


Рисунок 2.3 – Схема робочого процесу форм системи з використанням React і Redux.

2.2 Компоненти системи

Веб-застосунок повинен мати наступну архітектуру:

1. Клієнтська Частина (Frontend) – забезпечення зручного та інтуїтивно зрозумілого інтерфейсу для користувачів; ефективна взаємодія з серверною частиною для отримання та відправлення даних; відображення даних відповідно до вимог дизайну та користувацького досвіду; забезпечення безпеки даних користувачів.

Клієнтська частина програмного забезпечення повинна бути створена із застосуванням передових веб-технологій, таких як React для веб-додатків, і Swift для iOS; PWA, Kotlin для Android у разі мобільних додатків. Ця частина включатиме ряд ключових компонентів.

Інтерфейс користувача (UI) визначатиме візуальну структуру додатку та всі його елементи управління. Він забезпечить зручний та ефективний спосіб взаємодії користувача з додатком.

Клієнтська логіка (Frontend Logic) відповідатиме за обробку дій користувачів, взаємодію з сервером через API та обробку отриманих даних. Цей компонент забезпечить коректну та ефективну роботу додатку, забезпечуючи необхідні функціональні можливості.

Локальне зберігання даних використовуватиметься для кешування даних та збереження налаштувань користувача. Це дозволить забезпечити швидкий доступ до даних та підтримати персоналізовані налаштування для кожного користувача.

Клієнтська частина програми буде спілкуватися з сервером за допомогою RESTful API або GraphQL для забезпечення обміну даними. Це охоплює отримання інформації від сервера, відправлення запитів на виконання конкретних операцій, а також отримання повідомлень щодо стану та результатів виконання.

Клієнтська частина має бути оптимізована для забезпечення високої продуктивності, мінімального часу завантаження та плавної роботи додатку. Особлива увага повинна бути приділена адаптивному дизайну, щоб забезпечити правильне відображення додатка на різних пристроях і екранах різних розмірів;

2. Серверна частина (Backend**)** – серверна частина додатку відіграє важливу роль у забезпеченні ефективної та надійної роботи програмного забезпечення. Вона слугує як централізована система, яка обробляє та керує запитами користувачів, забезпечуючи швидкий доступ до даних та функціональності додатку.

На відміну від клієнтської частини, серверна сторона може працювати з великими обсягами даних та виконувати складні операції, що дозволяє підтримувати високу продуктивність та масштабованість системи. Вона також відповідає за забезпечення безпеки даних, контроль доступу до ресурсів та виконання бізнес-логіки додатку.

Іншою важливою функцією серверної частини є забезпечення взаємодії з різними іншими системами чи сервісами, такими як бази даних, зовнішні API або сервіси хмарного зберігання. Без належної серверної інфраструктури системи не зможе ефективно функціонувати, і може виникнути ризик втрати даних, низької продуктивності та надійності, а також обмежень у можливостях розширення та розвитку.

Цілі та завдання серверної частини додатку створені для забезпечення ефективного функціонування системи та задоволення потреб користувачів. Основною метою серверної частини є забезпечення стабільної роботи додатку, обробка запитів від клієнтської частини та забезпечення безпеки та конфіденційності даних.

Для досягнення цих цілей передбачено ряд завдань, включаючи розробку та підтримку серверних архітектур, забезпечення швидкодії та масштабованості системи, імплементацію backend логіки та інтеграцію з базами даних, а також впровадження механізмів аутентифікації, авторизації та захисту даних.

Крім того, серверна частина повинна забезпечити можливість моніторингу та аналізу даних для виявлення проблем та вдосконалення продукту, а також надійну роботу в умовах високої навантаженості та відмовостійкість системи. Реалізація цих завдань дозволить забезпечити оптимальний рівень сервісу та задоволення потреб користувачів;

3. База даних (БД) – важливий інструмент для організації, зберігання та управління великим обсягом інформації. БД є необхідною для ефективної роботи бізнесу та організацій у будь-якій галузі, що дозволяє зберігати дані у структурованому вигляді, що полегшує доступ до них і виконання різноманітних операцій. Це забезпечує збереження інформації у надійному та безпечному

середовищі, зменшує ризик втрати даних через випадкове видалення або пошкодження.

Крім того, база даних дозволяє ефективно виконувати операції з пошуком, фільтрацією та сортуванням даних, що сприяє швидкому та точному аналізу інформації. Вона також дозволяє забезпечити консистентність та цілісність даних шляхом встановлення правил та обмежень для їх обробки. База даних використовується для підтримки різноманітних операцій та процесів, таких як управління клієнтською базою, облік товарів та послуг, аналіз фінансових даних, автоматизація виробничих процесів та багато іншого. В цілому, база даних є ключовим компонентом сучасних систем управління та дозволяє забезпечити ефективну роботу організації на всіх рівнях.

Цілі та завдання бази даних формулюються з метою створення ефективної системи зберігання та управління інформацією з метою задоволення конкретних потреб користувачів. Головною метою бази даних є забезпечення доступу до інформації у відповідності до потреб користувачів, забезпечення її цілісності та безпеки, а також забезпечення швидкого та ефективного пошуку та використання даних. В рамках цього проекту передбачається створення бази даних, яка буде відповідати вимогам інформаційної системи, що покликана оптимізувати процеси обробки, зберігання та аналізу даних. Основними завданнями бази даних є забезпечення зручного та ефективного зберігання та організації інформації, забезпечення доступу до даних для авторизованих користувачів, реалізація механізмів захисту даних від несанкціонованого доступу та втрати, а також забезпечення можливості швидкого та зручного пошуку та аналізу даних;

4. Розподілена кеш-система та система обміну повідомленнями – розподілена кеш-система сприяє оптимізації швидкодії та продуктивності систем. Це досягається завдяки можливості зберігання кешованих даних ближче до кінцевих користувачів або середовища, де вони найчастіше використовуються [14]. Такий розподілений підхід дозволяє зменшити час доступу до даних та знизити навантаження на централізовані сервери.

Також, система обміну повідомленнями відіграє ключову роль у забезпеченні асинхронного та надійного обміну інформацією між компонентами системи. Замість прямого зв'язку між компонентами, який може бути недоступним або небажаним у деяких сценаріях, система обміну повідомленнями дозволяє відправляти та отримувати повідомлення через проміжну структуру (наприклад, чергу), забезпечуючи таким чином більшу резильєнтність та надійність системи в цілому.

Цілі та завдання розподіленої кеш-системи та системи обміну повідомленнями полягають у покращенні швидкодії, масштабованості та надійності системи шляхом ефективного управління даними та обміну інформацією між різними компонентами. Розподілена кеш-система спрямована на збереження та швидкий доступ до популярних або часто використовуваних даних, забезпечуючи високу продуктивність за рахунок зменшення навантаження на централізовані сервери. В той час як система обміну повідомленнями сприяє ефективному обміну даними між різними компонентами системи, полегшуючи спілкування та співпрацю між ними. Обидва рішення покликані забезпечити оптимальне функціонування розподіленої архітектури, знижуючи час доступу до ресурсів та підвищуючи загальну надійність системи;

5. Load Balancer та Reverse-Proxy – компоненти в мережевій архітектурі, спрямованими на поліпшення доступності, масштабованості та безпеки веб-додатків. Перш за все, Load Balancer розподіляє трафік між різними серверами, що зменшує навантаження на окремі системи та забезпечує стабільну працездатність. Використання Reverse-Proxy дозволяє приховати внутрішню інфраструктуру від зовнішніх користувачів, що підвищує рівень безпеки та забезпечує контроль доступу до веб-серверів. Крім того, вони забезпечують можливість масштабування системи за рахунок додавання нових серверів у пул, що дозволяє ефективно вирішувати зростаючі потреби в обробці трафіку. Узагальнюючи, Load Balancer та Reverse-Proxy сприяють покращенню продуктивності, доступності та безпеки веб-додатків, роблячи їх більш стійкими та ефективними в експлуатації [15].

Мета впровадження системи Load Balancer та Reverse-Proxy полягає у покращенні ефективності, масштабованості та надійності інфраструктури веб-сервісів. Забезпечення стабільної роботи, оптимізація розподілу навантаження між серверами, забезпечення безпеки та конфіденційності даних — це основні моменти, на яких спрямовані цілі та завдання використання Load Balancer та Reverse-Proxy.

Наявність Load Balancer дозволяє рівномірно розподіляти трафік між серверами, що зменшує ймовірність перевантаження та падіння продуктивності системи. Окрім цього, завдяки Load Balancer можливе автоматичне виявлення несправних серверів та їх виключення з обробки трафіку, що забезпечує безперебійну роботу системи та відновлення роботи після виникнення проблем.

Reverse-Proxy виконує функцію посередника між зовнішнім світом та внутрішньою мережею серверів. Він дозволяє захистити внутрішні сервери від прямого з'єднання з клієнтами, забезпечуючи додатковий рівень безпеки. Крім того, використання Reverse-Proxy дозволяє оптимізувати роботу серверів шляхом кешування статичного контенту та зниження навантаження на внутрішні сервери завдяки обробці запитів на рівні Reverse-Proxy;

6. Message broker – компонент в архітектурі розподілених систем, який забезпечує ефективний обмін повідомленнями між різними компонентами програмного забезпечення. Однією з основних причин використання Message broker є забезпечення асинхронного обміну даними між різними частинами системи. Це дозволяє компонентам системи працювати незалежно один від одного і забезпечує більшу гнучкість і масштабованість системи в цілому. Крім того, Message broker допомагає вирішувати проблеми, пов'язані з асинхронним обміном даними, такі як гарантія доставки повідомлень, керування пріоритетами та маршрутизація повідомлень. Використання Message broker також сприяє зменшенню зв'язку між компонентами системи, оскільки вони можуть спілкуватися через надійний посередник, а не безпосередньо між собою [16]. Це робить систему більш масштабованою, стабільною і легше супроводжуваною в майбутньому.

Метою Message broker є забезпечення ефективного обміну повідомленнями між різними компонентами системи. В основному, він виконує завдання реалізації асинхронного спілкування між різними компонентами системи, забезпечуючи надійність, масштабованість та інтеграцію. Це досягається за допомогою введення проміжного шару, який приймає, маршрутизує та обробляє повідомлення між відправником та отримувачем. Основними завданнями Message broker є: забезпечення гарантованої доставки повідомлень, маршрутизація повідомлень до відповідних отримувачів, підтримка різноманітних протоколів комунікації, таких як AMQP, MQTT, а також забезпечення можливості масштабування та високої доступності. Також важливим завданням є забезпечення безпеки та контролю доступу до повідомлень, щоб гарантувати конфіденційність та цілісність даних. Враховуючи ці цілі та завдання, Message broker є ключовою складовою для побудови розподілених систем, які вимагають ефективного та надійного обміну повідомленнями;

7. Grafana – потужний інструмент для візуалізації та моніторингу даних, який грає ключову роль у спрощенні процесу аналізу та розуміння великих обсягів інформації. Використання Grafana дозволяє з легкістю створювати графіки, діаграми та інші візуальні елементи на основі даних з різних джерел. Це робить його незамінним інструментом для моніторингу пристроїв, мереж, застосунків та інших систем.

Однією з ключових переваг Grafana є його здатність інтегруватися з різноманітними джерелами даних, такими як бази даних, сервіси моніторингу, інструменти збору логів тощо. Це дозволяє отримати повний обсяг інформації з різних джерел в одному місці, що полегшує аналіз та прийняття рішень.

Крім того, Grafana надає широкі можливості конфігурації та налаштування візуалізацій, що дозволяє користувачам створювати графіки та панелі, які найкраще відповідають їхнім потребам і вимогам [17]. Це робить Grafana дуже гнучким і адаптованим до різних сценаріїв використання.

Завдяки своїй простоті використання та потужному функціоналу, Grafana дозволяє швидко реагувати на зміни та проблеми в системі, а також ефективно аналізувати дані для покращення її ефективності та надійності. Таким чином, Grafana стає важливим інструментом для будь-якої організації, яка прагне забезпечити надійність та ефективність своїх систем.

Головною метою є створення інтерактивного та легкого у використанні інтерфейсу для візуалізації даних з різних джерел. Під цим пріоритетом мають бути розроблені такі функціональні можливості, як можливість побудови гнучких та динамічних графіків, створення адаптивних та інтерактивних панелей для відображення даних у реальному часі. Крім того, важливо забезпечити підтримку різних типів джерел даних та їх інтеграцію з Grafana для максимальної універсальності. Необхідно також врахувати аспекти безпеки, забезпечуючи зручні та надійні засоби аутентифікації та авторизації користувачів. Крім того, має бути забезпечена можливість моніторингу та аналізу роботи самого Grafana, зокрема, збір статистики щодо його використання та продуктивності для подальшої оптимізації та вдосконалення;

8. ELK Stack – складається з Elasticsearch, Logstash і Kibana, є потужним інструментом для збору, аналізу та візуалізації великих обсягів даних. Його використання дозволяє ефективно вирішувати низку завдань, пов'язаних з моніторингом, логуванням та аналізом даних. ELK Stack допомагає забезпечити стабільну та ефективну роботу систем, виявляти проблеми та відстежувати їх виникнення, а також забезпечує можливість реагувати на них вчасно. Він забезпечує цілісний погляд на дані, дозволяючи аналізувати їх з різних кутів та використовувати для прийняття стратегічних рішень. ELK Stack також сприяє підвищенню ефективності роботи команди, забезпечуючи її одночасний доступ до актуальних даних та інструментів для їх обробки. Враховуючи велику кількість корисних функцій, ELK Stack стає незамінним інструментом для будь-якої компанії, що має справу з обробкою великих обсягів даних і бажає ефективно використовувати їх для досягнення своїх цілей.

Метою впровадження ELK є створення потужної та ефективної системи для збору, аналізу та візуалізації журналів, що дозволить забезпечити надійний моніторинг та аналіз даних у реальному часі [18]. ELK дозволить забезпечити високу доступність та масштабованість, забезпечуючи оперативну реакцію на події, виявлення проблем та аналіз тенденцій для підтримки прийняття рішень. Реалізація ELK також спрямована на підвищення ефективності управління інфраструктурою та додатками, забезпечуючи можливість швидкої діагностики проблем та вирішення їх з мінімальними затримками;

2.2.1 Взаємодія між компонентами системи

Нижче описаний процес взаємодії між компонентами системи керування ефективністю виробничих активів:

1. Клієнт (React) надсилає запити через локальну мережу;
2. Reverse-Proxy приймає запити та перенаправляє їх до Load Balancer;
3. Load Balancer розподіляє запити між активними інстанціями Spring Boot;
4. Spring Boot виконує бізнес-логіку, взаємодіє з Hazelcast для доступу до кешованих даних та з TimescaleDB для зберігання та отримання даних. pgAdmin 4 використовується для управління базою даних;
5. Для асинхронних задач та обміну повідомленнями використовується RabbitMQ;
6. Grafana та ELK Stack забезпечують моніторинг та аналіз логів системи в реальному часі;

Цей процес також зображений на рисунку 2.4 у вигляді діаграми інтеграції систем та технологій для обробки даних та її візуалізації.

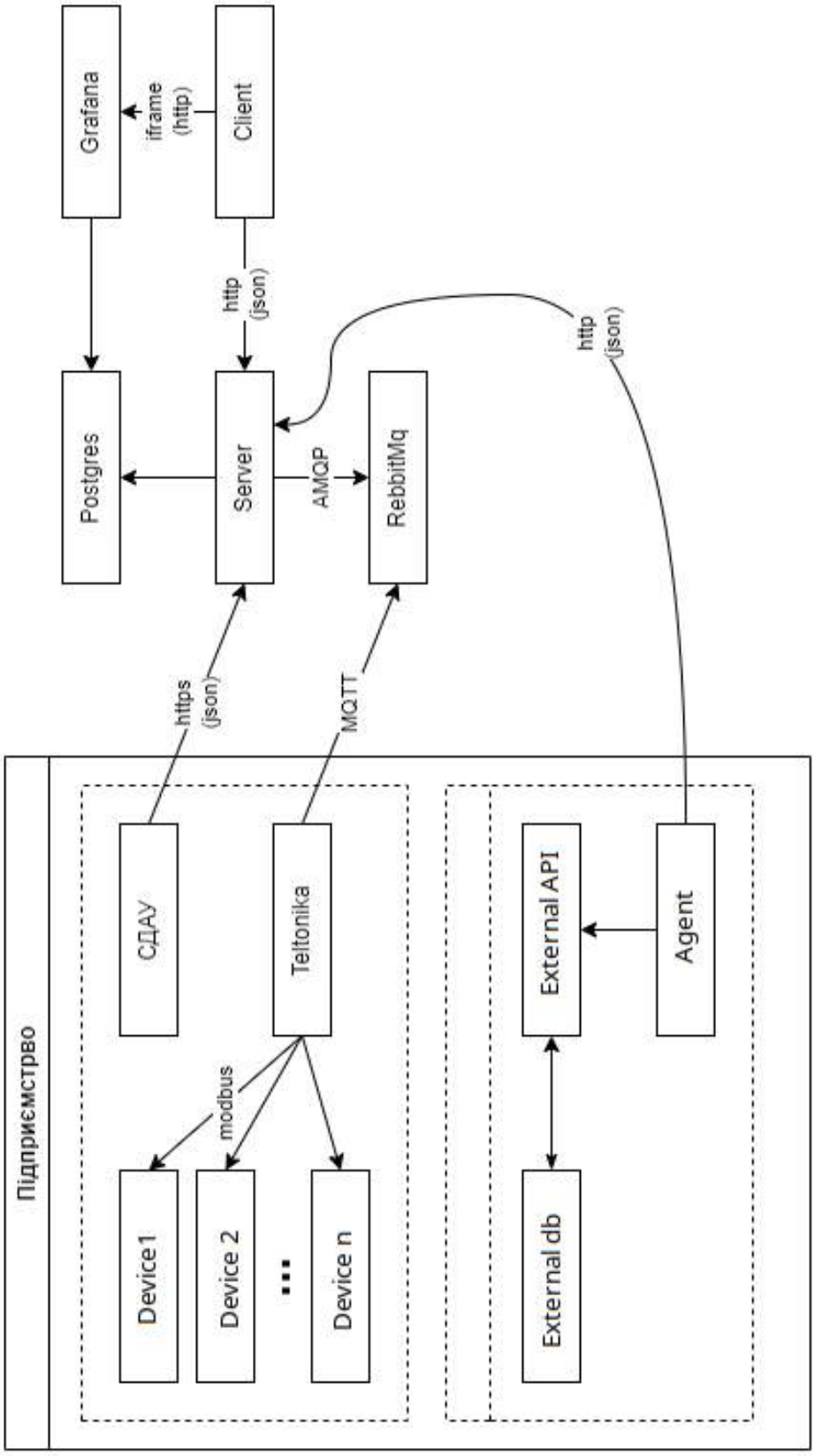


Рисунок 2.4 – Діаграма інтеграції систем та технологій для обробки даних та її візуалізації

2.3 Моделі даних системи

Модель даних передбачає, що основною одиницею є Підприємство, яке є центральним елементом, до якого можуть бути прив'язані різні сутності, забезпечуючи таким чином гнучке та масштабоване управління. Кожне підприємство може асоціюватися з множиною Користувачів, дозволяючи їм отримувати доступ до системи з урахуванням індивідуальних прав доступу. Це створює ієрархічну структуру управління, де кожен користувач має певний рівень доступу до даних та операцій, залежно від своєї ролі в організації.

До Підприємства також прив'язуються агентські додатки та шлюзи даних, які відіграють ключову роль у зборі, обробці та передачі даних з обладнання та інших джерел. Ці компоненти дозволяють інтегрувати різноманітне обладнання і системи в єдину екосистему, забезпечуючи ефективне управління даними та оперативний обмін інформацією.

Сутність обладнання є основним компонентом нашої системи, до якої можуть бути прив'язані як агентські додатки, так і шлюзи даних, забезпечуючи таким чином збір, обробку та аналіз даних з обладнання в реальному часі. Обладнання відправляє дані про свої поточні статуси, залученість у технологічні процеси, дозволяючи отримувати актуальну інформацію про виробничі процеси та ефективність використання ресурсів.

Додатково в моделі передбачено сутності, наприклад довідник по типам зерна, який є важливим для класифікації та аналізу зернових культур. Це дозволяє краще планувати закупівлі, зберігання та переробку зерна, оптимізуючи виробничі ланцюжки та мінімізуючи втрати. Також в моделі передбачена сутність для збереження енергометричних даних, яка є ключовою для моніторингу та оптимізації споживання енергоресурсів на підприємстві, дозволяючи знижувати витрати та підвищувати енергоефективність процесів. На рисунку 2.5, можна детально ознайомитись із ER-діаграмою бази даних.

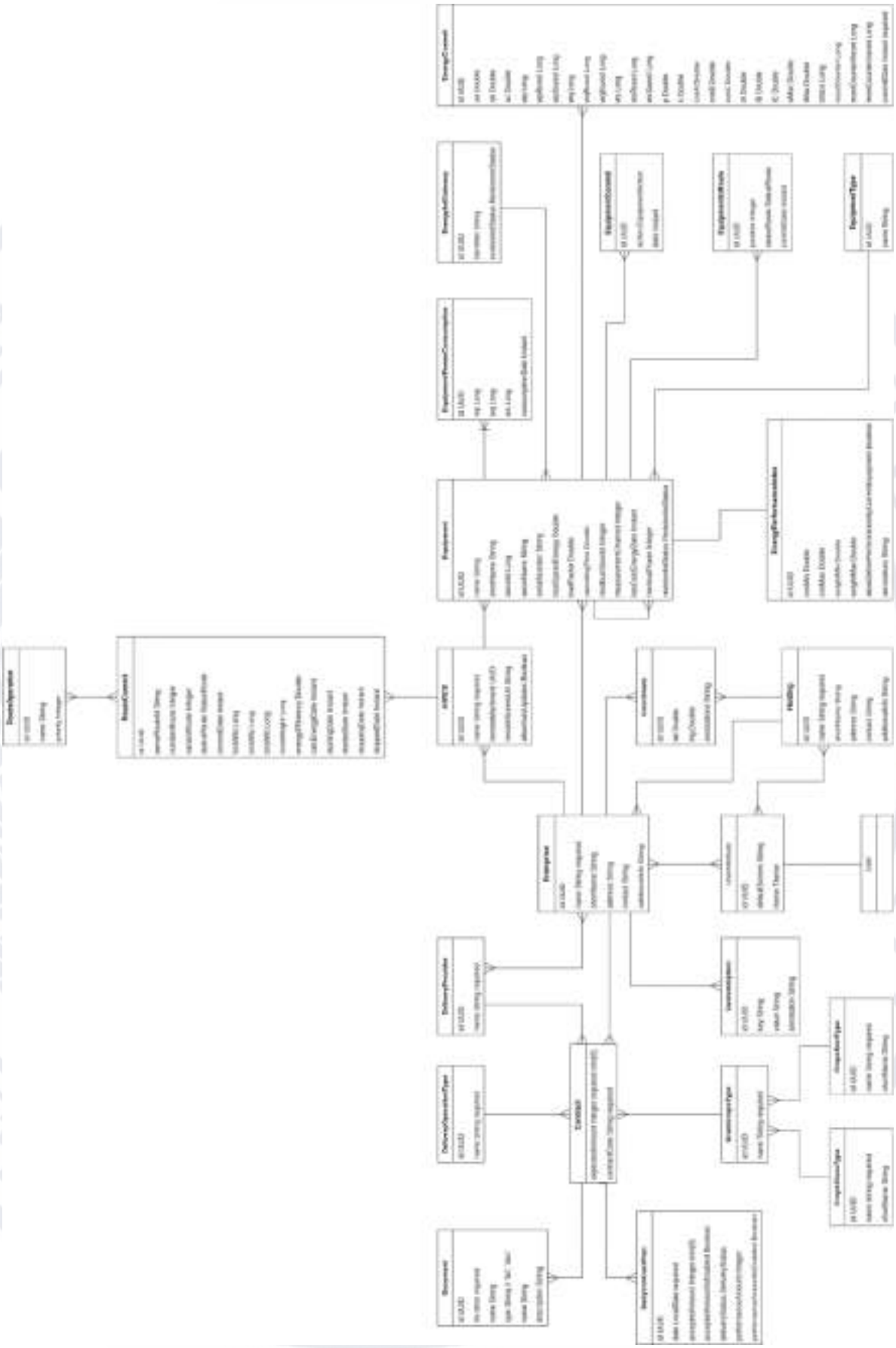


Рисунок 2.5 – ER-діаграма бази даних (Entity Relationships diagram) - відображення даних та відношень між ними

2.4 Забезпечення безпеки системи

Під цим розділом маються на увазі специфікації та вимоги, що стосуються заходів безпеки, які повинні бути виконані для забезпечення захисту конфіденційності, цілісності та доступності інформації та системи в цілому. Цей розділ визначає, які заходи безпеки потрібно реалізувати, щоб запобігти несанкціонованому доступу, зламам системи, втраті даних та іншим загрозам безпеці [19]:

1. Шифрування даних – всі дані, що передаються між користувачем та сервером, повинні бути зашифровані для захисту від несанкціонованого доступу. Повинні бути застосовані сучасні методи шифрування для збереження конфіденційності та цілісності даних.

2. Захист від атак – система повинна бути стійкою до різних видів атак, таких як перехоплення даних, введення шкідливих скриптів, тощо та забезпечувати використання заходів безпеки, таких як фаєрволи, системи виявлення вторгнень та інші для виявлення та блокування спроб несанкціонованого доступу.

3. Аудит та моніторинг – система повинна мати можливість ведення журналу подій для виявлення та аналізу потенційних загроз безпеці та моніторинг системи на предмет підозрілих активностей та вчасна реакція на них.

4. Управління сесіями – повинні бути використані безпечні методи управління сесіями для запобігання перехопленню сесії або атакам з фіксацією сесії.

5. Запобігання переповненню буфера – повинні бути вжиті заходи для запобігання атакам, що використовують переповнення буфера, особливо в програмному коді, написаному на мовах програмування, які не керують пам'яттю автоматично.

6. Реагування на інциденти безпеки – повинна бути розгорнута система моніторингу, яка відстежує події безпеки в реальному часі та має процедури для реагування на інциденти безпеки.

7. Регулярне оновлення програмного забезпечення – забезпечити регулярне оновлення всіх компонентів системи, включаючи операційні системи, сервери баз даних та додатки, забезпечує захист від відомих вразливостей.

8. Резервне копіювання та відновлення – забезпечити регулярне резервне копіювання даних та перевірка процедур відновлення для відновлення системи після збою або атаки.

9. Аутентифікація та авторизація – описати в специфікації вимоги та методи реалізації системи авторизації та аутентифікації на основі JSON Web Tokens (JWT) для веб-додатків, JWT надати безпечний спосіб відправлення інформації між двома сторонами.

2.4.1 Генерація JWT

При реалізації системи авторизації та аутентифікації на основі JWT важливо вибрати надійні бібліотеки та фреймворки, які підтримують стандарти безпеки та пропонують достатні можливості для контролю над процесом аутентифікації та авторизації [20].

Реалізація:

1. Імітер (iss) – ідентифікатор сервісу, який генерує токен.
2. Суб'єкт (sub) – ідентифікатор користувача.
3. Аудиторія (aud) – ідентифікатор сервісу, для якого призначений токен.
4. Час видачі (iat) – час створення токена.
5. Час закінчення дії (exp) – час, коли токен перестає бути дійсним.
6. Токен має бути підписаний використовуючи безпечний ключ.

Аутентифікація:

1. Користувач подає свої аутентифікаційні дані (наприклад, логін і пароль).
2. Система перевіряє ці дані та генерує JWT, якщо дані коректні.
3. JWT відправляється користувачу як доказ аутентифікації.

Авторизація:

1. При спробі доступу до захищеного ресурсу користувач відправляє JWT разом з запитом.
2. Система перевіряє JWT (валідність, час життя та інші претензії).
3. Якщо токен дійсний, система надає доступ до запитаного ресурсу.

Заходи безпеки:

1. Використання HTTPS для захисту даних при передачі.
2. Строге управління ключами для підпису JWT.
3. Обмеження часу життя токена для зменшення ризику використання вкрадених токенів.
4. Перевірка на наявність CSRF (Cross-Site Request Forgery) атак.;
5. Опціонально, використання JWE (JSON Web Encryption) для шифрування токенів.

Ця специфікація встановлює основні вимоги та керівні принципи для створення безпечної та ефективної системи авторизації та аутентифікації на основі JWT, але конкретні деталі реалізації можуть варіюватись в залежності від конкретних потреб і умов використання.

2.4.2 Використання HTTPS

Вимоги для використання HTTPS включає в себе ретельне планування та врахування різноманітних аспектів безпеки, сумісності та виконання. HTTPS (Hypertext Transfer Protocol Secure) є розширеним варіантом HTTP, який захищає передачу даних між клієнтом і сервером за допомогою шифрування. На рисунку 2.6 детально розглянутий процес аутентифікації користувача між браузером та сервером за допомогою JWT, а також основні компоненти HTTPS.

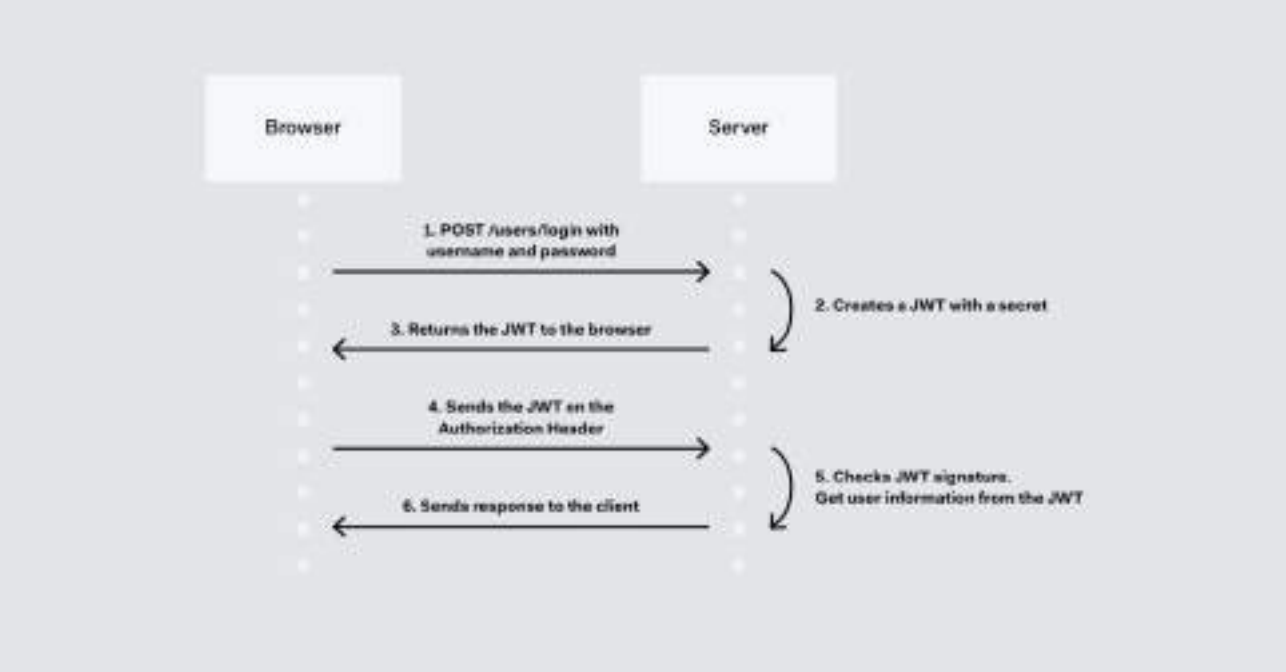


Рисунок 2.6 – Процес аутентифікації користувача між браузером та сервером за допомогою JWT, а також основні компоненти HTTPS

1. SSL/TLS Сертифікати – використання SSL (Secure Sockets Layer) або TLS (Transport Layer Security) сертифікатів для шифрування з'єднань та верифікації ідентичності сервера. Сертифікати повинні бути видані надійним сертифікаційним органом (CA).
2. Ключові параметри шифрування:
   - сильне шифрування – використання сучасних алгоритмів шифрування, таких як AES (Advanced Encryption Standard) з ключем щонайменше 256 біт;
   - версії протоколу TLS – рекомендується використовувати TLS версії 1.2 або вище для забезпечення найкращого рівня безпеки;
   - список шифрів (Cipher Suite) – налаштування сервера для підтримки безпечного набору шифрів, що запобігає атакам, пов'язаним з вразливими шифрами;
3. HSTS (HTTP Strict Transport Security) – конфігурація сервера для використання HSTS, що примушує клієнтські браузери використовувати безпечні з'єднання при доступі до сайту;

Вимоги до Клієнта і Сервера:

1. Верифікація сертифікатів – клієнти мають перевіряти сертифікати серверів перед встановленням з'єднань для запобігання атакам «людина посередині».
2. Оновлення програмного забезпечення – система повинна забезпечувати Регулярні оновлення програмного забезпечення для клієнтів і серверів з метою виправлення вразливостей у протоколах шифрування та інших компонентах безпеки.
3. Конфіденційність і цілісність даних – система повинна гарантувати , що дані, передані через HTTPS, захищені від перехоплення, модифікації або втручання третіх сторін.

Додаткові сервіси:

1. Оновлення сертифікатів – мають бути передбачені процеси для своєчасного оновлення сертифікатів перед їхнім закінченням терміну дії.
2. Моніторинг і логування – повинні бути реалізовані системи моніторингу та логування для виявлення та аналізу спроб несанкціонованого доступу або інших безпекових інцидентів.
3. Оцінка вразливостей – проведення оцінок вразливостей та тестування на проникнення для ідентифікації та усунення потенційних слабких місць має проводитись на регулярній основі.
4. Сумісність – забезпечити сумісність HTTPS з різними клієнтськими пристроями та браузерами, забезпечуючи широку доступність та надійність з'єднань.

Використання CSRF (Cross-Site Request Forgery) токенів є ключовим елементом захисту веб-додатків від атак, спрямованих на несанкціоноване виконання запитів від імені користувача. Для ефективного захисту від таких атак, необхідно ретельно спланувати роботу із CSRF токенами між клієнтом, сервером та додатковими сервісами.

Загальні принципи:

1. Генерація та валідація токенів – сервер має генерувати унікальний CSRF токен для кожної сесії користувача або форми, і валідувати цей токен при кожному запиті, який змінює стан даних (POST, PUT, DELETE).
2. Використання секретності – CSRF токени повинні бути таємними і непередбачуваними, що забезпечує захист від можливості їх вгадування атакуючими.
3. Обмеження дії токенів – токени мають бути обмежені до конкретної сесії користувача та/або URL, для яких вони були створені.

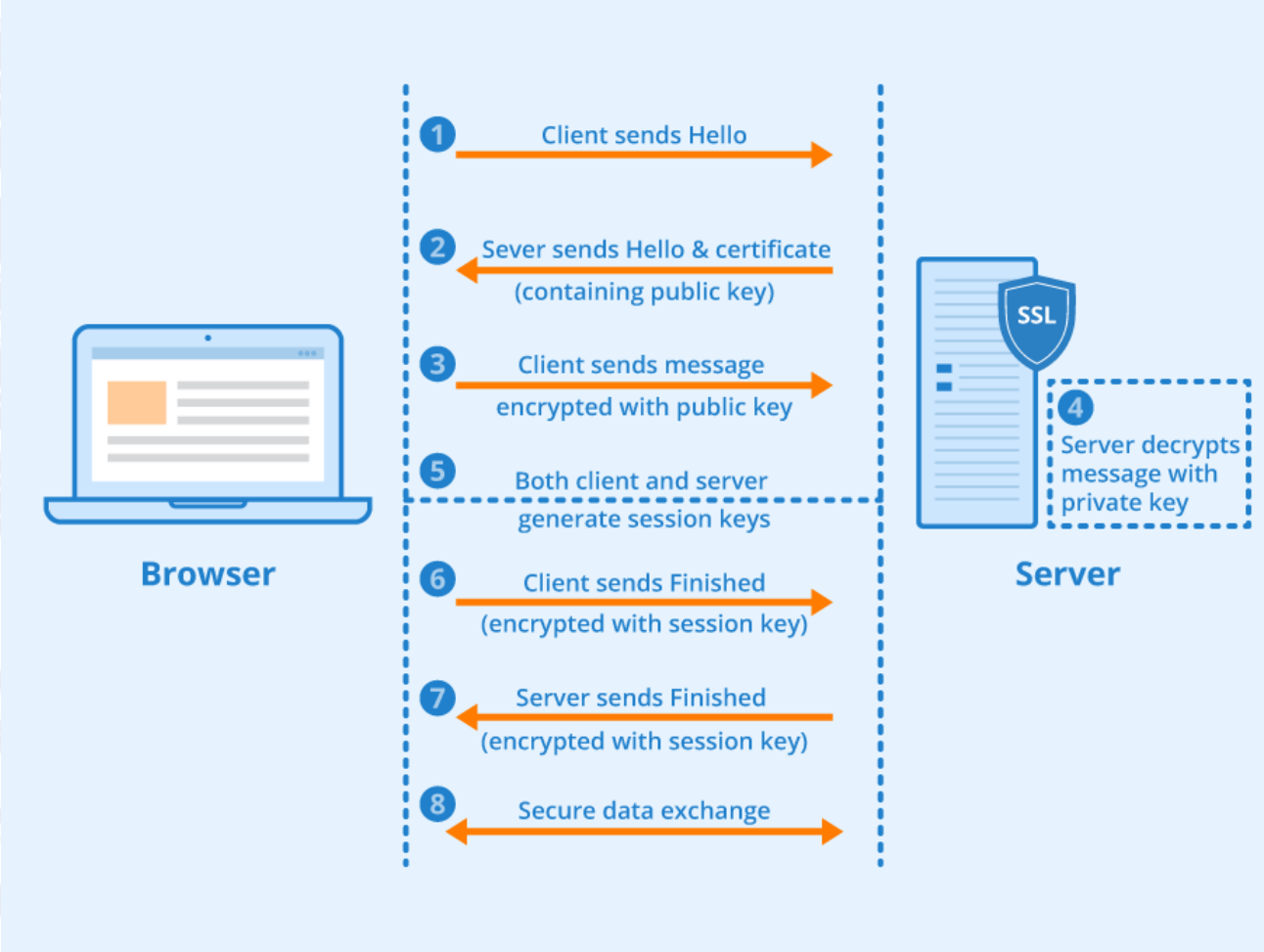


Рисунок 2.7 – Пояснення процесу забезпечення безпеки процесу аутентифікації та авторизації

1. Клієнт:
   - зберігання токенів – клієнтська частина (наприклад, веб-браузер) повинна зберігати CSRF токени таким чином, щоб вони були доступні для включення в майбутні запити, але при цьому були захищені від доступу через скрипти (наприклад, використання HTTP Only cookies);
   - включення токенів у запити – при кожному запиті, що змінює стан даних, клієнт повинен додавати CSRF токен, отриманий від сервера, в заголовки запиту або тіло повідомлення.

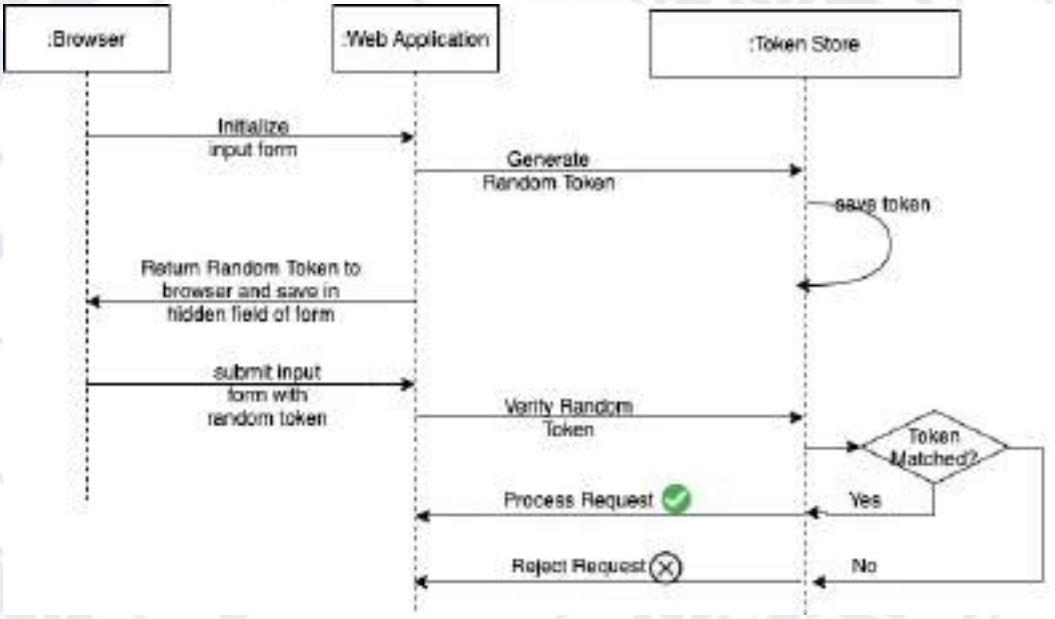


Рисунок 2.8 – Діаграма процесу обробки токену в системі

2. Сервер:
   - генерація токенів: при кожній новій сесії або запиті, що вимагає захисту, сервер повинен генерувати новий CSRF токен і надсилати його клієнту;
   - перевірка токенів – сервер має перевіряти наявність і валідність CSRF токену при обробці кожного запиту, що змінює стан. Якщо токен відсутній або невалідний, запит має бути відхилений;

- обробка помилок – в разі виявлення невалідного CSRF токену, сервер повинен повідомити клієнта про помилку, надавши інформацію про необхідність повторної аутентифікації або оновлення сторінки.

3. Додаткові сервіси:
   - передача токенів – якщо для інтеграції з додатковими сервісами необхідно передавати CSRF токени, це має відбуватися через захищені канали з використанням сильного шифрування;
   - оновлення токенів – сервіси, які приймають CSRF токени, повинні підтримувати механізм їх оновлення у випадку, коли токен стає недійсним (наприклад, через завершення сесії).

   Додаткові сервіси:

   1. Оновлення токенів – система має забезпечити регулярне оновлення CSRF токенів під час сесії для ускладнення атак.
   2. Двофакторна перевірка – мають бути застосовані додаткові механізми аутентифікації разом із CSRF захистом для підвищення безпеки.
   3. Сувора політика походження – має бути розгорнута політика безпеки для перевірки походження запитів (наприклад, за допомогою заголовків CORS або SameSite для cookies), щоб додатково захистити від CSRF та інших видів атак.

   Загальні Вимоги:

   1. Налаштування та управління – забезпечити гручкі інструменти для налаштування та управління через CLI або веб-інтерфейс.
   2. Моніторинг та логування – інтегрувати систему з системами моніторингу та агрегації логів.
   3. Безпека – реалізувати політики безпеки, включаючи WAF (Web Application Firewall) для захисту від веб-атак.

   Специфікація Load Balancer.

Load balancer (LB) та reverse proxy (RP) є ключовими компонентами в сучасних високодоступних веб-архітектурах. Вони забезпечують

масштабування, безпеку та ефективність обробки клієнтських запитів до веб-сервісів.

1. Масштабування – забезпечити розподіл навантаження між кількома серверами для забезпечення горизонтального масштабування.

2. Висока доступність – забезпечити неперервності сервісу шляхом розподілу запитів між серверами, які функціонують. На рисунку 2.9 зображена діаграма, яка відображає систему балансування навантаження в комп'ютерних мережах.

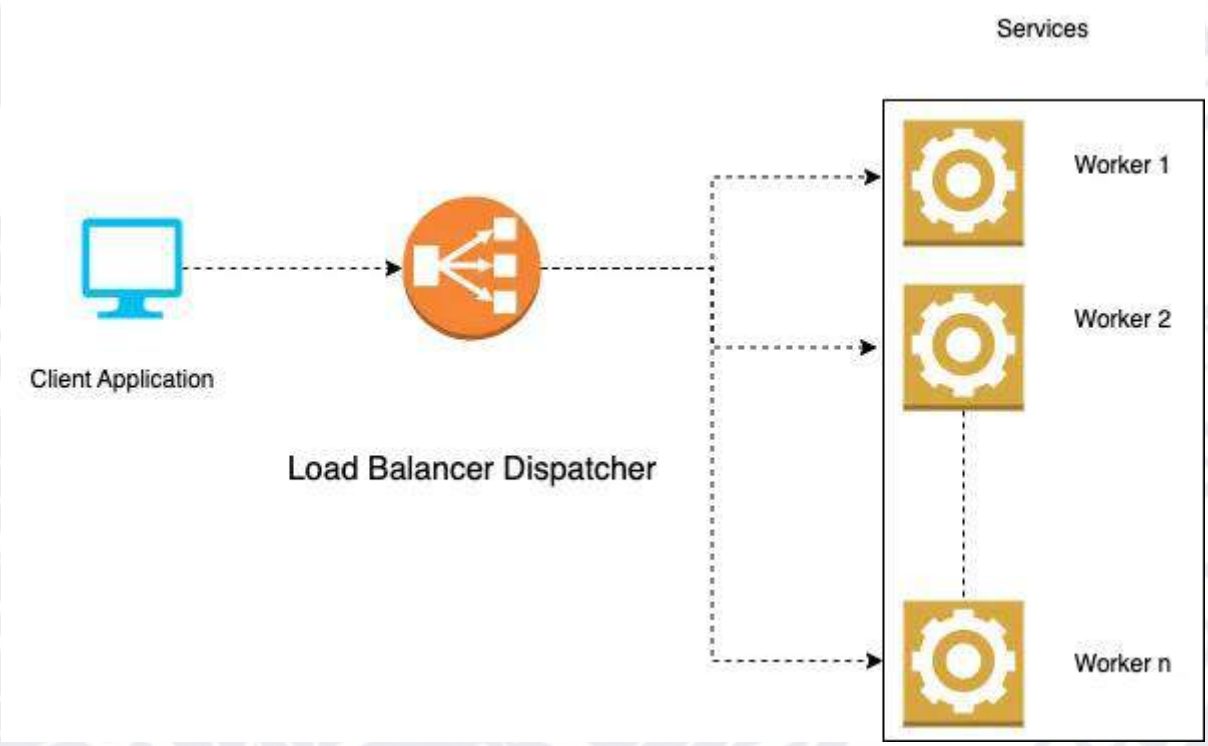


Рисунок 2.9 – Система балансування навантаження в комп'ютерних мережах

3. Протоколи – підтримка HTTP/HTTPS, TCP, UDP.

4. Стратегії балансування – Round Robin, Least Connections, IP Hash, Custom Rules.

5. Здоров'я серверів – перевіряти статус у серверів через health checks.

6. SSL Termination – здійснювати дешифрування SSL на стороні LB для зниження навантаження на сервери.

7. Session Persistence – забезпечити сталість сесії користувача на одному сервері, якщо це необхідно.

Специфікація Reverse Proxy має бути забезпечена шляхом виконання наступних заходів:

1. Безпека – приховування деталей внутрішньої інфраструктури від зовнішнього світу.

2. Кешування – зберігання копій статичного вмісту для швидшого доступу;

3. SSL Offloading – розшифровка SSL-запитів на RP для оптимізації навантаження.

4. Підтримка протоколів – HTTP/HTTPS, WebSocket.

5. Модифікація запитів/відповідей – додавання, видалення або зміна заголовків.

6. Аутентифікація та авторизація – можливість інтегрувати системи контролю доступу.

7. Кешування – налаштування політик кешування для оптимізації відповідей.

8. Логування та моніторинг – детальне логування запитів та відповідей для аналізу та моніторингу.

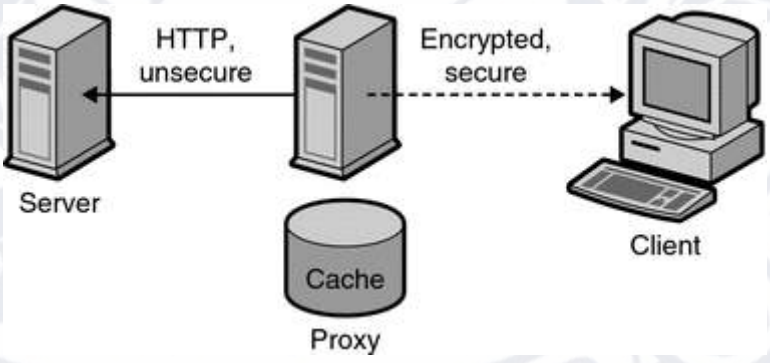


Рисунок 2.10 – Спрощена діаграма, комунікації між сервером, проксі та клієнтом в мережі

2.5 Вибір інструментів розробки

Вибір інструментів розробки для створення системи керування ефективністю виробничих активів «SAKURA» має базуватись на необхідності забезпечення високої продуктивності, надійності, масштабованості та зручності використання системи. Важливим фактором було також використання сучасних та перевірених технологій, які підтримують активну розробку та мають широку спільноту підтримки. Нижче наведено детальне обґрунтування вибору основних інструментів розробки.

Front-end розробка. Для побудови динамічного користувацького інтерфейсу було обрано бібліотеку React, яка є однією з найбільш популярних та підтримуваних JavaScript-бібліотек. Використання React забезпечує швидке оновлення інтерфейсу без необхідності перезавантаження сторінки, що підвищує зручність користувача. Доповненням до React став TypeScript, що надає строгий контроль типів, зменшуючи кількість потенційних помилок у коді та покращуючи його підтримку. На рисунку 2.11 зображений лістинг коду TypeScript tsconfig.

Рисунок 2.11 – Лістинг коду TypeScript tsconfig

Redux-Toolkit був обраний для керування станом додатка, оскільки він спрощує роботу з Redux завдяки готовим рішенням для створення сховища стану, редюсерів та асинхронних дій. Разом з Redux Thunk, який полегшує реалізацію асинхронних дій, ці інструменти забезпечують ефективне управління станом додатка.

React Router був обраний для реалізації маршрутизації у додатку, що дозволяє створювати динамічні маршрути та забезпечує зручність навігації для користувачів. Axios був використаний для виконання HTTP-запитів, що забезпечує легку та ефективну комунікацію між клієнтом та сервером.

Для спрощення розробки інтерфейсу та інтеграції сучасних UI-компонентів, таких як Material UI та Reactstrap, було обрано React-Jhipster. Цей набір інструментів надає генерацію інтерфейсів, функціонал перекладу (i18n), валідацію, пагінацію та інші корисні компоненти. Leaflet та React Leaflet забезпечили інтерактивні карти для відображення географічних даних, а React Hook Form полегшив управління формами в додатку.

Joi була обрана для валідації даних, що забезпечує правильність та цілісність інформації у додатку. React-toastify допоміг у створенні сповіщень, надаючи зручний інтерфейс для інформування користувачів про події у системі. React-Select та React-fontawesome забезпечили створення та стилізацію селектів та іконок відповідно, а Swiper дозволив реалізувати динамічні слайдери. Day.js була використана для зручного виконання операцій з датами та часом.

HTML, CSS та SCSS використовувалися для створення структури та стилізації веб-сторінок. Використання SCSS додало додаткові можливості до звичайного CSS, спрощуючи розробку та підтримку стилів у проекті.

Back-end розробка. Для серверної частини додатку було обрано Java як основну мову програмування завдяки її високій продуктивності та надійності. Spring Boot забезпечив швидку розробку та легку інтеграцію різних компонентів системи. Hibernate був використаний для об'єктно-реляційного відображення (ORM), що полегшило взаємодію з базою даних.

Для управління збіркою проектів та залежностями було обрано Maven, а Docker та Docker-compose забезпечили легкість впровадження та масштабування додатку. RabbitMQ забезпечив асинхронну взаємодію між компонентами системи, а ELK Stack (Elasticsearch, Logstash, Kibana) надав можливості для пошуку, аналізу та візуалізації даних в реальному часі.

Timescale був використаний для зберігання та аналізу даних у часових рядах, а Hazelcast для розподіленого кешування та управління інмеморі даними. MapStruct дозволив автоматизувати мапінг між Java-бінами, зменшуючи кількість шаблонного коду.

Для інтеграційного тестування було обрано TestContainers, що дозволяє створювати легкі та відтворювані середовища за допомогою Docker контейнерів. H2 Database використовувалася для розробки та тестування. Checkstyle забезпечив перевірку вихідного коду Java на відповідність стандартам форматування, а Liquibase — управління версіями баз даних.

Інші інструменти:

Для керування залежностями та ресурсами проекту використовувався Npm, а для перевірки стилю та форматування коду — ESLint та Prettier. EditorConfig допоміг підтримувати єдиний стиль коду в рамках проекту, а Husky автоматизував виконання скриптів git hooks. Git забезпечив систему контролю версій для відстеження змін у коді та спільної роботи над проектом. Grafana використовувалася для моніторингу та аналізу даних з різноманітних джерел. Окрім цього, також для забезпечення транслювання робочих місць операторів на підприємствах, було обрано інструмент MeshCentral [21]. Це безкоштовне програмне забезпечення для віддаленого керування комп’ютером із відкритим вихідним кодом. Оскільки сервер MeshCentral написаний на NodeJS, його можна встановити на багатьох операційних системах, включаючи Windows, Linux.

РОЗДІЛ 3 РОЗРОБКА СИСТЕМИ КЕРУВАННЯ ЕФЕКТИВНІСТЮ ВИРОБНИЧИХ АКТИВІВ

3.1 Розгортання середовища розробки та інтеграція інструментів

Відповідно до принципів розробки та створеної архітектури для цієї системи, в першу чергу я здійснив розгортання середовищ розробки IntelliJ IDEA та Visual Studio, та створив репозиторій в GitLab.

Останню версію IntelliJ IDEA, було завантажено з офіційного сайту JetBrains. Після чого, слідуючи інструкціям майстра встановлення, та завершення інсталяції, я здійснив початкове налаштування середовище та повторив аналогічну процедуру із середовищем Visual Studio, завантаживши його з офіційного сайту Microsoft.

Далі, зареєструвавшись на офіційному сайті GitLab, на панелі управління створюємо новий проект та налаштовуємо його у відповідності до раніше зазначених потреб. Після створення проекту, необхідно скопіювати URL репозиторію для його подальшого використання.

Після успішного встановлення середовищ розробки та налаштування GitLab, необхідно в терміналі здійснити ініціалізацію усіх необхідних пакетів згаданих в другому розділі, при побудові архітектури системи. Для цього, було використано вказаний на рисунку 3.1 перелік команд в терміналі.

Команда «npm install» є однією з найчастіше використовуваних команд npm і є ключовою для управління залежностями у проектах. Вона використовується для встановлення пакетів (бібліотек або модулів) з npm (Node Package Manager) до проекту.

```
>_ npx create-react-app my-app --template typescript cd my-app
>_ npm install @reduxjs/toolkit react-redux redux-thunk
>_ npm install react-router-dom
>_ npm install axios
>_ npm install generator-jhipster-react
>_ npm install leaflet react-leaflet
>_ npm install react-hook-form
>_ npm install joi
>_ npm install react-toastify
>_ npm install react-select
>_ npm install @fortawesome/react-fontawesome
>_ npm install swiper
>_ npm install dayjs
>_ npm install node-sass
>_ npm install @mui/material @emotion/react @emotion/styled
>_ npm install material-react-table
>_ npm install reactstrap
>_ npm install @testing-library/react jest
>_ npm install webpack webpack-cli webpack-dev-server
>_ npm install eslint prettier\
>_ npm install husky
```

Рисунок 3.1 – Встановлення необхідних пакетів для розробки

Після успішного встановлення необхідних пакетів, необхідно імпортувати всі бібліотеки додавши їх у java-класи. На рисунку 3.2 наведено код, який включає імпорти для зазначених бібліотек в класі-контролері та на рисунку 3.3 в конфігураційному класі Spring Boot.

```
package com.example.myproject;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.http.ResponseEntity;

// React and TypeScript are used on the client-side and are not imported here
// Redux Toolkit, Redux Thunk, React Router, Axios, React-Jhipster, Leaflet, React Leaflet, React Hook Form, Joi, React-toastify, React-Select, Fo

// Server-side imports
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@SpringBootApplication
@RestController
public class SakuraServer {

    public static void main(String[] args) {
        SpringApplication.run(SakuraServer.class, args);
    }

    @RequestMapping(value = "SAKURA", method = RequestMethod.GET)
    public ResponseEntity<String> hello() {
        return ResponseEntity.ok("SAKURA");
    }
}
```

Рисунок 3.2 – Імпорт бібліотек в класі-контролері

```
package com.example.myproject.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;

// Hibernate
import org.springframework.orm.jpa.JpaTransactionManager;
import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;
import org.springframework.transaction.PlatformTransactionManager;
import org.springframework.transaction.annotation.EnableTransactionManagement;

// MapStruct
import org.mapstruct.Mapper;
import org.mapstruct.factory.Mappers;

// Liquibase
import liquibase.integration.spring.SpringLiquibase;

// Docker
// Docker specific imports are not needed for application code but for Dockerfile and docker-compose.yml

// RabbitMQ
import org.springframework.amqp.core.Queue;
import org.springframework.amqp.rabbit.annotation.RabbitListener;
import org.springframework.amqp.rabbit.connection.ConnectionFactory;
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.amqp.rabbit.listener.SimpleMessageListenerContainer;
import org.springframework.amqp.rabbit.listener.adapter.MessageListenerAdapter;

// Etc stock imports (usually for logging)
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@Configuration
@EnableTransactionManagement
public class AppConfig {

    private static final Logger logger = LoggerFactory.getLogger(AppConfig.class);

    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }

    @Bean
    public PlatformTransactionManager transactionManager(LocalContainerEntityManagerFactoryBean entityManagerFactory) {
        JpaTransactionManager transactionManager = new JpaTransactionManager();
        transactionManager.setEntityManagerFactory(entityManagerFactory.getObject());
        return transactionManager;
    }

    @Bean
    public SpringLiquibase liquibase() {
        SpringLiquibase liquibase = new SpringLiquibase();
        liquibase.setChangeLog("classpath:db/changelog/db.changelog-master.xml");
        return liquibase;
    }
```

Рисунок 3.3 – Імпорт бібліотек в конфігураційному класі Spring Boot

Після успішного імпорту, необхідно перевірити роботу Git-репозиторію, перейшовши у корневий каталог проекту та виконавши команду для ініціалізації Git-репозиторію «git init». Після цього, додавши усі файли проекту до індексу Git, зробивши початковий коміт із повідомленням «git commit -m "Initial commit», я додав раніше скопійовану URL адресу репозиторію та використав команду «git remote add origin https://gitlab.com/sakura-apm/server/» після чого запушив зміни.

```
>_ git init
>_ git commit -m "Initial commit"
>_ git remote add origin https://gitlab.com/sakura-apm/server/»
>_ git push -u origin master
```

Рисунок 3.4 – Кроки перевірки роботи Git-репозиторію в терміналі

3.2 Визначення сутностей та зв'язків у базі даних

Після успішної підготовки середовища розробки та усіх необхідних інструментів, в першу чергу необхідно здійснити визначення сутностей (entity) та їх зв’язків у Java з використанням JPA (Java Persistence API) та анотацій для створення сутностей.

На рисунку 3.5 та 3.6 зображена частина коду, де здійснюється створення сутності «Contract», що має багато до одного зв’язки з стуністю «Enterptise», «DeliveryProvider», «DeliveryOperationType», «GrainCropsType» та один до багатьох з «Document», згідно створеної ER-діаграма бази даних на рисунку 2.5.

```
@Entity
public class Contract {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private UUID id;

    @NotNull
    @Min(value = 0)
    private Integer expectedAmount;

    @NotNull
    private String contractCode;

    @ManyToOne
    @JoinColumn(name = "enterprise_id")
    private Enterprise enterprise;

    @ManyToOne
    @JoinColumn(name = "deliveryProvider_id")
    private DeliveryProvider deliveryProvider;

    @ManyToOne
    @JoinColumn(name = "deliveryOperationType_id")
    private DeliveryOperationType deliveryOperationType;

    @ManyToOne
    @JoinColumn(name = "grainCropsType_id")
    private GrainCropsType grainCropsType;

    @OneToMany(mappedBy = "contract")
    private Set<Document> documents;

    // Getters and Setters
}
```

Рисунок 3.5 – Перша частина коду створення сутності «Contract»

```
42
43  @Entity
44  public class Enterprise {
45      @Id
46      @GeneratedValue(strategy = GenerationType.AUTO)
47      private UUID id;
48
49      @NotNull
50      private String name;
51      private String shortName;
52      private String address;
53      private String additionalInfo;
54
55      // Getters and Setters
56  }
57
58  @Entity
59  public class DeliveryProvider {
60      @Id
61      @GeneratedValue(strategy = GenerationType.AUTO)
62      private UUID id;
63
64      @NotNull
65      private String name;
66
67      // Getters and Setters
68  }
69
70  @Entity
71  public class DeliveryOperationType {
72      @Id
73      @GeneratedValue(strategy = GenerationType.AUTO)
74      private UUID id;
```

Рисунок 3.5 – Перша частина коду створення сутності «Contract»

(продовження)

```
76      @NotNull
77      private String name;
78
79      // Getters and Setters
80  }
81
82  @Entity
83  public class GrainCropsType {
84      @Id
85      @GeneratedValue(strategy = GenerationType.AUTO)
86      private UUID id;
87
88      @NotNull
89      private String name;
90
91      // Getters and Setters
92  }
93
94  @Entity
95  public class Document {
96      @Id
97      @GeneratedValue(strategy = GenerationType.AUTO)
98      private UUID id;
99
100     private byte[] file;
101     private String name;
102     private String type;
103     private String description;
104
105     @ManyToOne
106     @JoinColumn(name = "contract_id")
107     private Contract contract;
108
109     // Getters and Setters
```

Рисунок 3.6 – Друга частина коду створення сутності «Contract»

Аналогічним чином, було створено решту сутностей та зв’язків між ними. Окрім цього, для автоматичного створення та управління схемою бази даних можна використовувати «spring.jpa.hibernate.ddl-auto=update» у файлі конфігурації Spring Boot. Це дозволить Hibernate автоматично генерувати та оновлювати схему бази даних на основі визначених сутностей.

```
# application.properties
spring.datasource.url=jdbc:postgresql://localhost:5432/your_db
spring.datasource.username=your_username
spring.datasource.password=your_password
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Рисунок 3.7 – Файл конфігурації Spring Boot

Ця конфігурація налаштовує підключення до PostgreSQL бази даних та автоматично оновлює її схему при зміні сутностей.

3.3 Реалізація точки входу в React застосунок

Наступним кроком, нам необхідно реалізувати точку входу в React застосунок для побудови усієї веб-сторінки системи. Це необхідно, для того щоб правильно налаштувати і ініціалізувати веб-застосунок та забезпечити необхідну функціональність для стабільної роботи, включаючи обробку стану, локалізацію, тему, обробку помилок та налаштування мережевих запитів. Це робить систему більш масштабованою, гнучкою та підтримуваною. На рисунку 3.8 зображено лістинг коду який показує, як імпортуються необхідні модулі та компоненти, налаштовуються локалізація та інтерцептори, а також рендериться головний компонент додатку у кореневий елемент DOM.

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Provider } from 'react-redux';
import setupAxiosInterceptors from '@config/axios-interceptor';
import { loadIcons } from '@config/icon-loader';
import { registerLocale } from '@config/translation';
import { clearAuthentication } from '@redux/reducers/authentication/actions';
import store from '@redux/store';
import ErrorBoundary from '@shared/error/error-boundary';
import { ThemeProvider } from './config/context/theme-context';
import AppComponent from './app';

import { bindActionCreators } from 'redux';

registerLocale(store);

const actions = bindActionCreators({ clearAuthentication }, store.dispatch);
setupAxiosInterceptors(() => actions.clearAuthentication('login.error.unauthorized'));

loadIcons();

const rootEl = document.getElementById('root');

const render = Component =>
  // eslint-disable-next-line react/no-render-return-value
  ReactDOM.render(
    <ErrorBoundary>
      <Provider store={store}>
        <ThemeProvider>
          <Component />
        </ThemeProvider>
      </Provider>
    </ErrorBoundary>,
    rootEl
  );

render(AppComponent);

```

Рисунок 3.8 – Лістинг коду точки входу в React застосунок

Таким чином, цей код забезпечує правильну ініціалізацію та побудову веб-застосунку, роблячи його готовим до роботи з усіма необхідними функціональними можливостями.

3.4 Авторизація в системі

Для реалізації авторизації в системі, необхідно створити компонент форми входу (LoginForm) для React застосунку з використанням TypeScript. Спочатку

було створено компонент форми авторизації, який включає поля для введення логіну користувача, пароля та опціонального прапорця «запам’ятати мене». За допомогою бібліотеки «react-hook-form» керується станом форми і валідацією даних. Потім я реалізував компонент, який використовує функцію «handleSubmit» з «userForm» для обробки події відправки форми. При відпраці форми функція «submitHandler», яка обробляє введені дані та передає їх до функції «login». Функція «login» в свою чергу викликає «handleLogin», передаючи введені дані (ім'я користувача, пароль, прапорець "запам'ятати мене").

Якщо при авторизації виникає помилка, вона передається в пропс «loginError». Якщо «loginError» існує, в інтерфейсі відображається повідомлення про помилку, яке інформує користувача про неправильні облікові дані або інші проблеми при вході.

```
21  const LoginForm: React.FC<IProps> = ({ loginError, handleLogin }) => {
22    const login = ({ username, password, rememberMe }: ILoginProps) => {
23      handleLogin(username, password, rememberMe);
24    };
25
26    const {
27      handleSubmit,
28      register,
29      formState: { errors, touchedFields },
30    } = useForm({
31      mode: 'onTouched',
32    });
33
34    const submitHandler = (e): void => {
35      handleSubmit(login)(e);
36    };
37
38    return (
39      <div className="home-main-div">
40        <div classNome="d-flex flex-column m-auto">
41          <div className="home-wrapper-div">
42            <Row classNome="justify-content-center mb-3">
43              <BrandIcon icon="login" />
44            </Row>
45            <Row>
46              <p classNome="home-title">
47                <Translate contentKey="home.title">Welcome to JHIPSTER</Translate>
48              </p>
49            </Row>
50            <StarpForm onSubmit={submitHandler} classNome="d-flex flex-column align-items-center">
51              <Col>
52                {loginError ? (
53                  <Alert color="danger" data-cy="loginError">
54                    <Translate contentKey="login.messages.error.authentication">
55                      <strong>Failed to sign in!</strong> Please check your credentials and try again.
```

Рисунок 3.9 – Компонент з відображенням сторінки авторизації та функція login

Щоб дана сторінка правильно функціонувала, необхідно здійснити інтеграцію з backend частиною. Функція «handleLogin» я інтегрував з бекендом через API-запити для перевірки облікових даних користувача. Якщо облікові дані вірні, користувач авторизується і отримує доступ до захищених ресурсів системи. У разі успішної авторизації може встановлюватися токен автентифікації в локальному сховищі браузера (наприклад, localStorage або sessionStorage), щоб зберігати стан авторизації між сеансами. На рисунку 3.9 зображена частина лістингу коду реалізації даного компоненту та функції login.

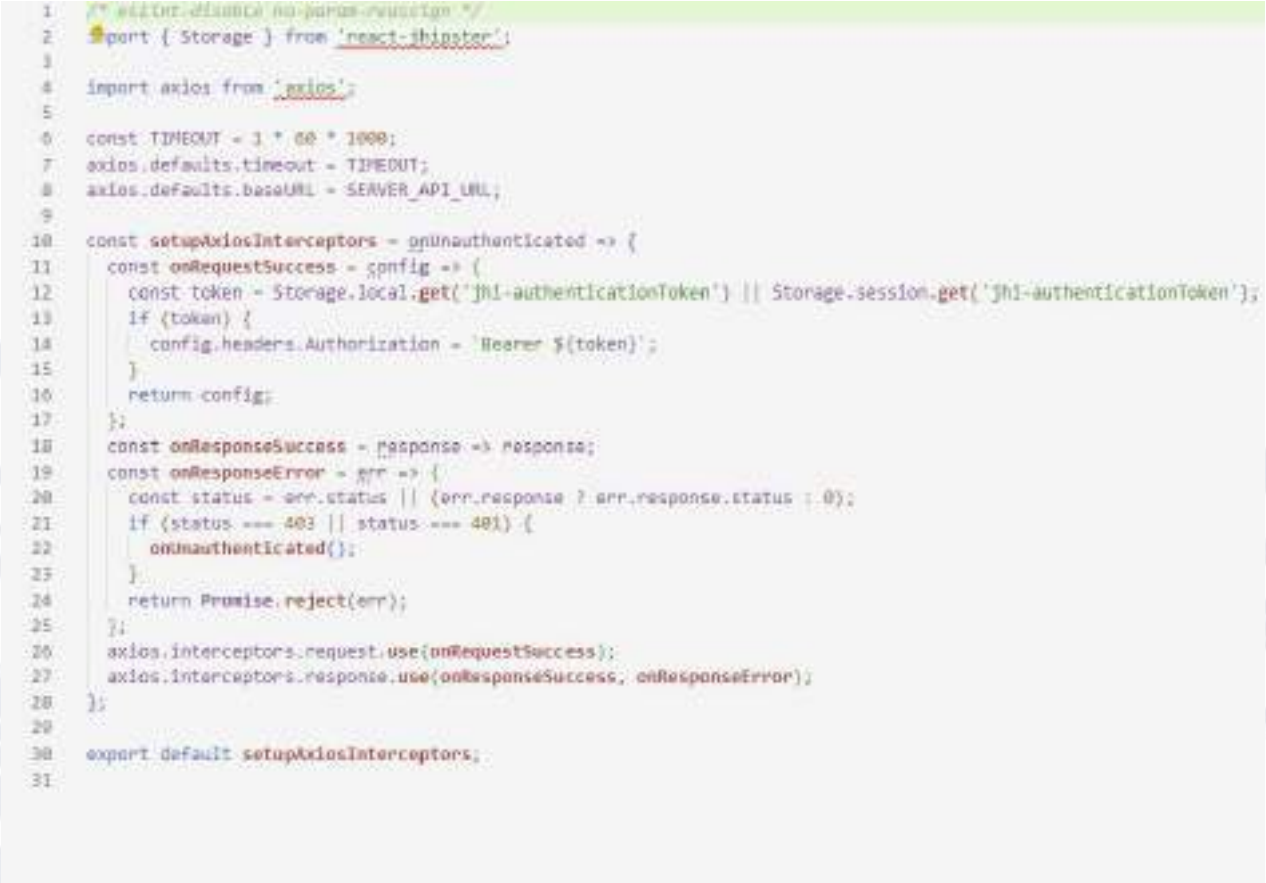
Частина коду на рисунку 3.10 відповідає за налаштування перехоплювачів (interceptors) для бібліотеки Axios, яка використовується для здійснення HTTP-запитів у React застосунку. Перехоплювачі дозволяють додавати JWT токен до кожного запиту і обробляти певні відповіді від сервера, такі як помилки авторизації.

Коли користувач намагається здійснити запит до сервера, перехоплювач запиту (onRequestSuccess) додає JWT токен з локального або сесійного сховища в заголовок Authorization. Це забезпечує автентифікацію на сервері для захищених ресурсів.

Якщо сервер повертає відповідь з помилкою авторизації (статус 401 або 403), перехоплювач відповіді (onResponseError) викликає функцію onUnauthenticated, яка може обробити цю помилку, наприклад, шляхом перенаправлення користувача на сторінку входу або оновленням токену за допомогою refresh токену.

Таким чином, ці перехоплювачі допомагають централізовано керувати автентифікацією запитів та обробкою помилок, забезпечуючи безпеку та зручність користування додатком.

Дана реалізація забезпечує надійну систему авторизації, яка надає безпечний і зручний доступ користувачів до системи, обробляючи введені дані, перевіряючи їх на сервері та відображаючи відповідні повідомлення про помилки.



```
/* eslint-disable no-param-reassign */
import { Storage } from 'react-jhipster';

import axios from 'axios';

const TIMEOUT = 1 * 60 * 1000;
axios.defaults.timeout = TIMEOUT;
axios.defaults.baseURL = SEAVER_API_URL;

const setupAxiosInterceptors = onUnauthenticated => {
  const onRequestSuccess = config => {
    const token = Storage.local.get('jhi-authenticationToken') || Storage.session.get('jhi-authenticationToken');
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  };
  const onResponseSuccess = response => response;
  const onResponseError = err => {
    const status = err.status || (err.response ? err.response.status : 0);
    if (status === 403 || status === 401) {
      onUnauthenticated();
    }
    return Promise.reject(err);
  };
  axios.interceptors.request.use(onRequestSuccess);
  axios.interceptors.response.use(onResponseSuccess, onResponseError);
};

export default setupAxiosInterceptors;
```

Рисунок 3.10 – Приклад налаштування axios interceptors

3.5 Реалізація решти сторінок системи

Для успішної реалізації інших сторінок системи, необхідно написати компоненти React. На прикладі «Operational Dashboard», реалізуємо компонент, який відповідатиме за відображення списку договорів та відкриття модального вікна для додавання нового договору. Лістинг частини коду, зображений на рисунку 3.11.

Спершу я імпортував залежності, React і «useState» для створення функціонального компонента та управління станом. «Translate» з «react-jhipster» для підтримки багатомовності. Два дочірні компоненти: «AddNewModal» для модального вікна і «ListOfAgreements» для відображення списку договорів. Створюємо компонент «Agreements», що є функціональним компонентом React та використаємо хук «useState» для створення стану «isModalOpen», який визначає, чи відкрите модальне вікно. Початкове значення — false.

Після чого, я створив функцію для перемикання стану модального вікна, «toggle», що змінює стан «isModalOpen» на протилежне значення, відкриваючи або закриваючи модальне вікно.

Таким чином, компонент «Agreements» забезпечує організовану та гнучку структуру для управління договорів у системі.

Рисунок 3.11 – Компонент ініціалізації договорів та їх списку

Наступним, був створений компонент React ListOfAgreements, який відповідає за відображення списку договорів та управління їхнім завантаженням з сервера. Лістинг даного коду, зображений на рисунку 3.12.

Даний компонент має кілька ключових завдань:

1. Отримання та відображення даних: Компонент відповідає за завантаження договорів з сервера та їх відображення. Це важливо для надання користувачам актуальної інформації.
2. Управління станом: Використання хуків «useState», «useEffect», «useAppDispatch», «useAppSelector» дозволяє ефективно управляти станом компонента, обробляти помилки та відображати індикатор завантаження.

3. Модульність: Інкапсуляція логіки завантаження та відображення договорів у окремий компонент робить код більш організованим, підтримуваним та повторно використовуваним.
4. Реактивність: Використання «useEffect» для завантаження даних при зміні «enterpriceId» забезпечує реактивну поведінку компонента, що дозволяє автоматично оновлювати інформацію при зміні підприємства.
5. Інтернаціоналізація: Використання компонента «Translate» дозволяє легко підтримувати багатомовність додатку, що є важливим для міжнародних користувачів.

Рисунок 3.12 – Компоненту з відображенням списку усіх договорів

Таким чином, компонент ListOfAgreements забезпечує ефективне та зручне

управління списком договорів, що покращує користувацький досвід та підтримку додатку.

Підв'язка backend частини до frontend компонентів здійснюється через Redux, який відповідає за управління станом та виконання асинхронних запитів до сервера, що дозволяє компонентам React отримувати та відображати актуальні дані.

Ідентичним до вищевказаного способу було реалізовано й інші компоненти, об’єднавши їх в єдину систему. Для кожного з них був створений відповідний Redux slice для управління станом, асинхронні дії для отримання та відправлення даних на backend, а також селектори для отримання необхідних даних зі store. Усі компоненти було підключено до Redux store через «Provider», що забезпечило єдиний підхід до управління станом додатку.

Переваги такого підходу:

- модульність – компоненти легко підтримувати та розширювати завдяки чіткій структурі та розподілу відповідальностей;
- повторне використання – загальний підхід дозволяє легко повторно використовувати код для створення нових компонентів;
- централізоване управління станом – використання Redux забезпечує єдине джерело правди для стану додатку, що спрощує відстеження та управління даними;
- реактивність – компоненти автоматично оновлюються при зміні стану, що покращує взаємодію з користувачем та зменшує ймовірність помилок;

Таким чином, вся система складається з компонентів, що реалізовані та підв'язані за аналогічним принципом, що забезпечує узгодженість та надійність системи.

3.6 Огляд функціоналу системи «SAKURA»

Сторінка Авторизації. Розроблена система управління ефективністю

виробничих активів пропонує користувачам інтерактивний та зручний інтерфейс для доступу до основних функцій. На першому етапі користувач потрапляє на сторінку авторизації, де потрібно ввести логін і пароль для входу в систему. Є можливість активувати функцію автоматичного входу, яка зберігає токен доступу для подальших сеансів. Вигляд сторінки авторизації, можна детально розглянути на рисунку 3.13.

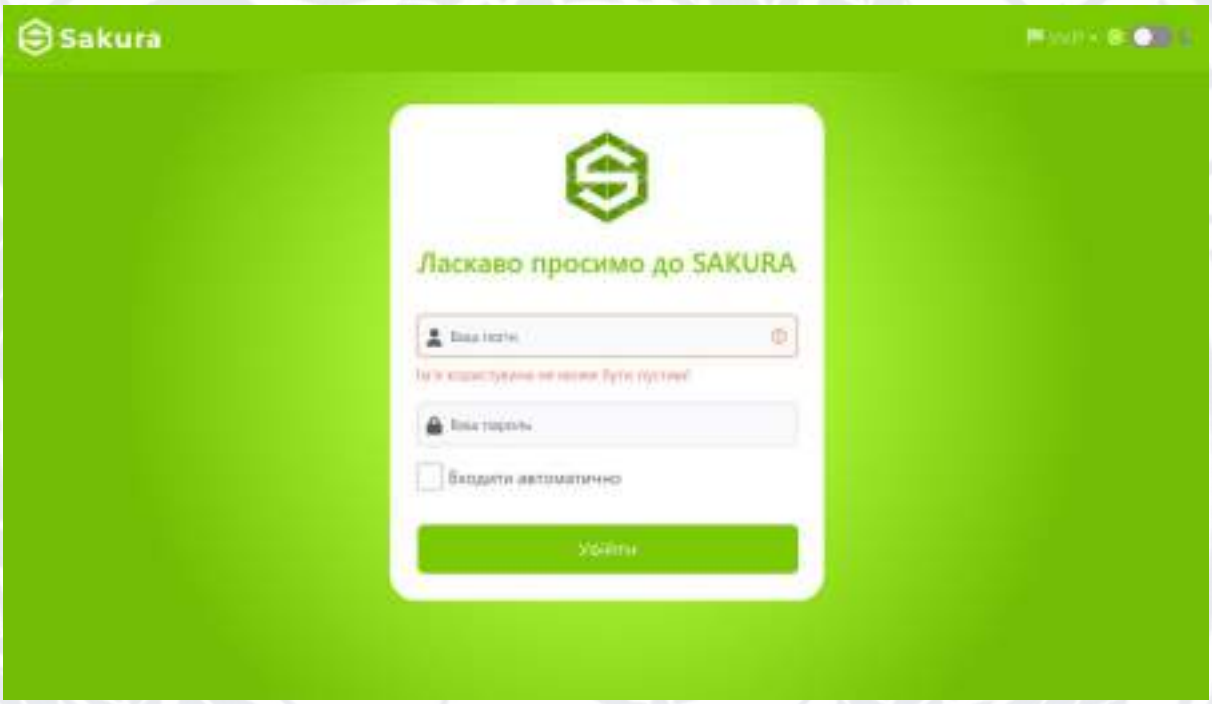


Рисунок 3.13 – Сторінка авторизації системи SAKURA

На даній сторінці, системою передбачена валідація, за рахунок якої здійснюється перевірка, чи введений логін та пароль користувача в відповідних полях. Ця валідація забезпечує базову перевірку введених даних, забезпечуючи мінімальні вимоги для успішного входу до системи, гарантуючи, що всі необхідні дані заповнені перед відправкою запиту.

Сторінка підприємств. Після успішної авторизації відкривається сторінка з переліком підключених робочих місць на підприємствах, що належать холдингу. За допомогою технології MeshCentral, що була обрана раніше, ця

сторінка в режимі реального часу відображає все, що відбувається на робочих місцях операторів ключових пунктів керування.

Рисунок 3.14 – Логотип open source програмного забезпечення MeshCentral

Рисунок 3.15 – Сторінка «Техпроцес» в системі SAKURA

MeshCentral надає значні переваги для управління IT-інфраструктурою. Централізоване керування дозволяє адміністраторам ефективно моніторити та налаштовувати всі підключені пристрої, незалежно від їхньої фізичної локації.

Віддалений доступ забезпечує оперативне вирішення технічних проблем та підтримку віддалених співробітників. Моніторинг у реальному часі допомагає швидко виявляти та вирішувати проблеми, забезпечуючи стабільну роботу системи. На рисунку 3.15 зображена сторінка «Техпроцес» із моніторингом робочих місць операторів підприємств.

Користувач також може перемикатися на сторінку мапа, щоб переглянути місцезнаходження всіх підприємств холдингу.

Сторінка оперативний дашборд. Дана сторінка в свою чергу надає повний контроль підприємства на одному екрані, планування і керування роботою підприємства, оперативне реагування на зміни ситуації, а також контроль і керування ефективністю.

Операційний дашборд елеватора, забезпечує керівника підприємства ключовою інформацією на одному екрані. Це дозволяє швидко оцінити поточну ситуацію на елеваторі без значних витрат часу та ефективно планувати роботу. Дашборд складається з шести блоків: витяг з річного плану заготівлі зерна, силосна дошка, енергетичні показники, надходження, добовий план роботи та видаток.

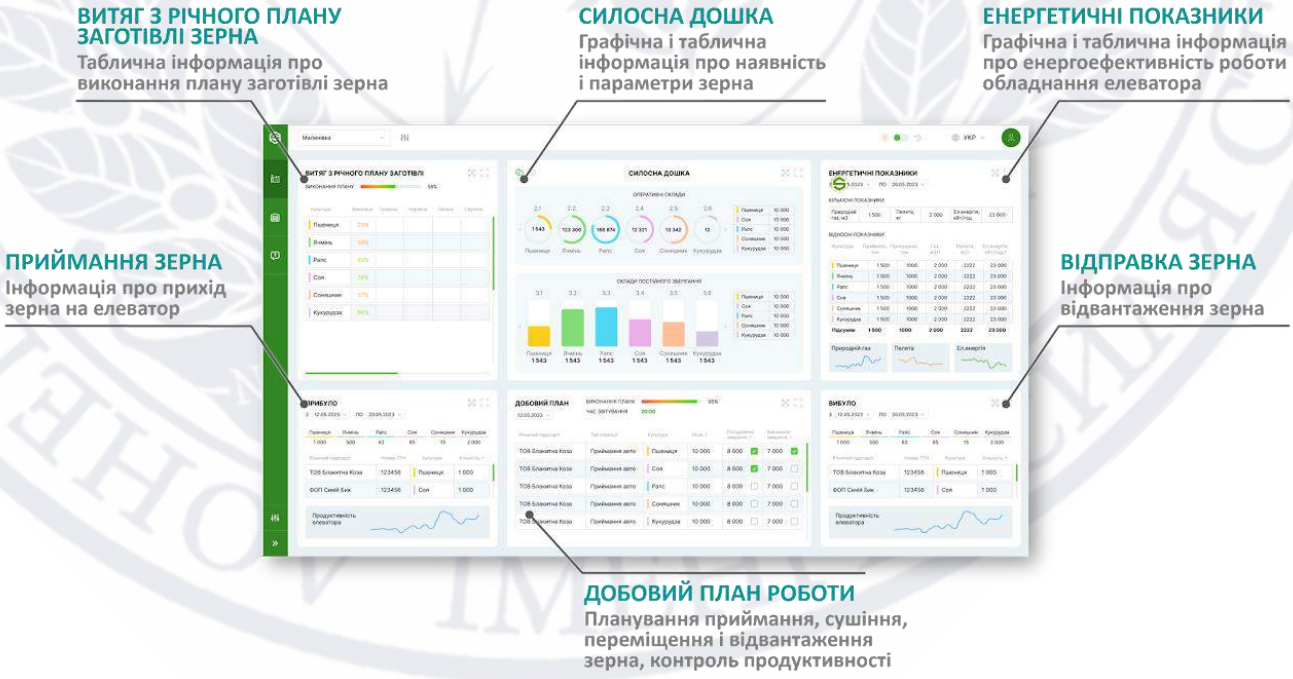


Рисунок 3.16 – Діаграма з поясненням блоків на сторінці «Операційний Дашборд» в системі SAKURA

Окрім вище перерахованого функціоналу, система також має змогу отримати більш детальну інформацію по кожному блоку інформації, вибрати мову інтерфейсу та кольорову тему.

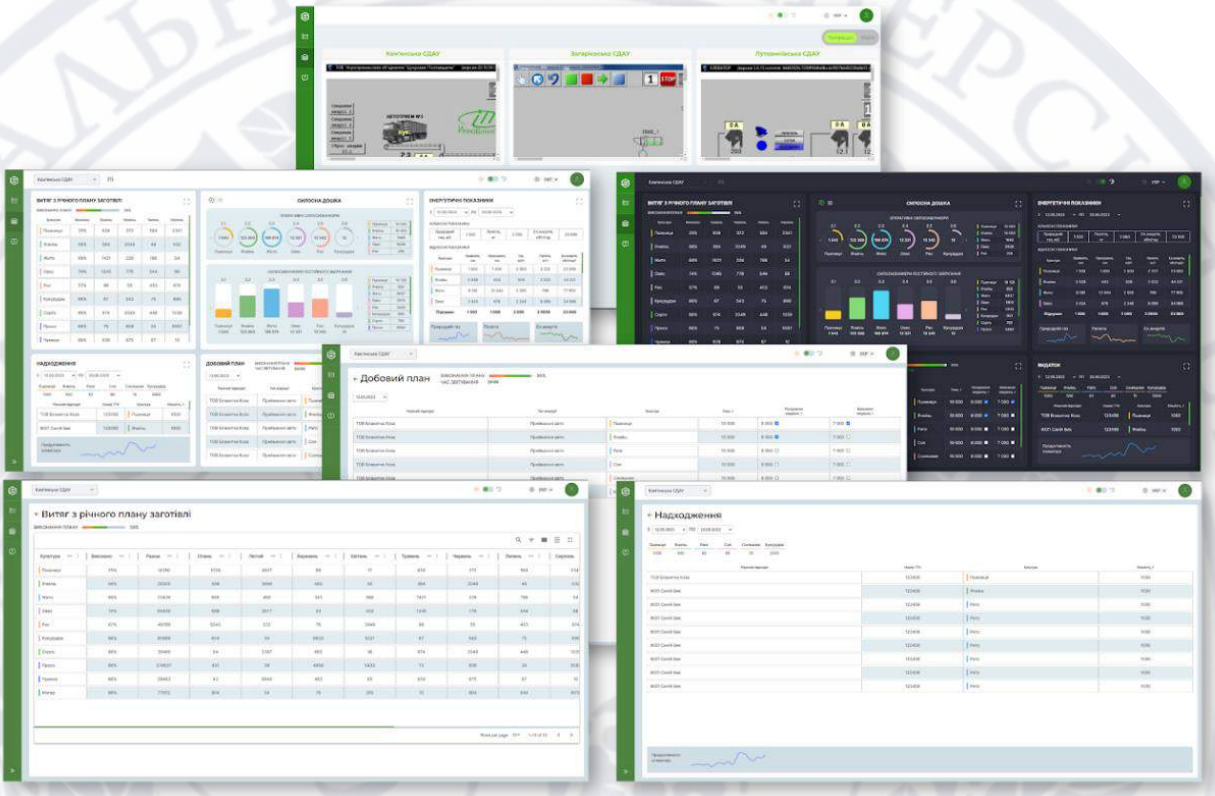


Рисунок 3.17 – Розширений вигляд блоків дашборду та різні кольорові теми

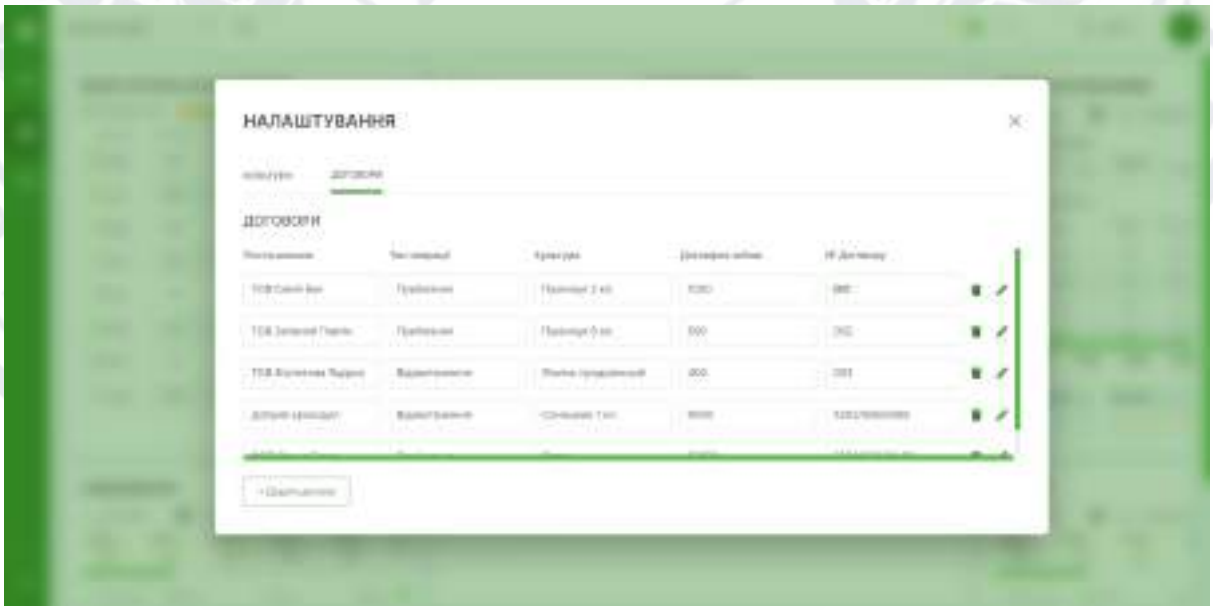


Рисунок 3.18 – Вкладка «Договори» в налаштуваннях

Для переключення на інше підприємство, користувач може скористатися випадаючим списком у верхньому лівому куті сторінки, вибравши потрібне підприємство, щоб оновити дані на дашборді. Створення, видалення або редагування договорів здійснюється через вкладку «Договори» в налаштуваннях, де відображається таблиця існуючих договорів з відповідними можливостями.

Сторінка енергоефективність. Ця сторінка надає доступ до чотирьох вкладок: техпроцес, обладнання, енергомоніторинг та загальні показники. Вкладка техпроцесу дозволяє моніторити параметри енергоспоживання СДАУ, поділяючи екран на робоче місце оператора та графік даних. Вкладка обладнання надає параметри енергії для кожного пристрою з відповідними датчиками. Енергомоніторинг показує середню інформацію щодо енергоспоживання всього холдингу з можливістю перегляду даних за різні періоди.

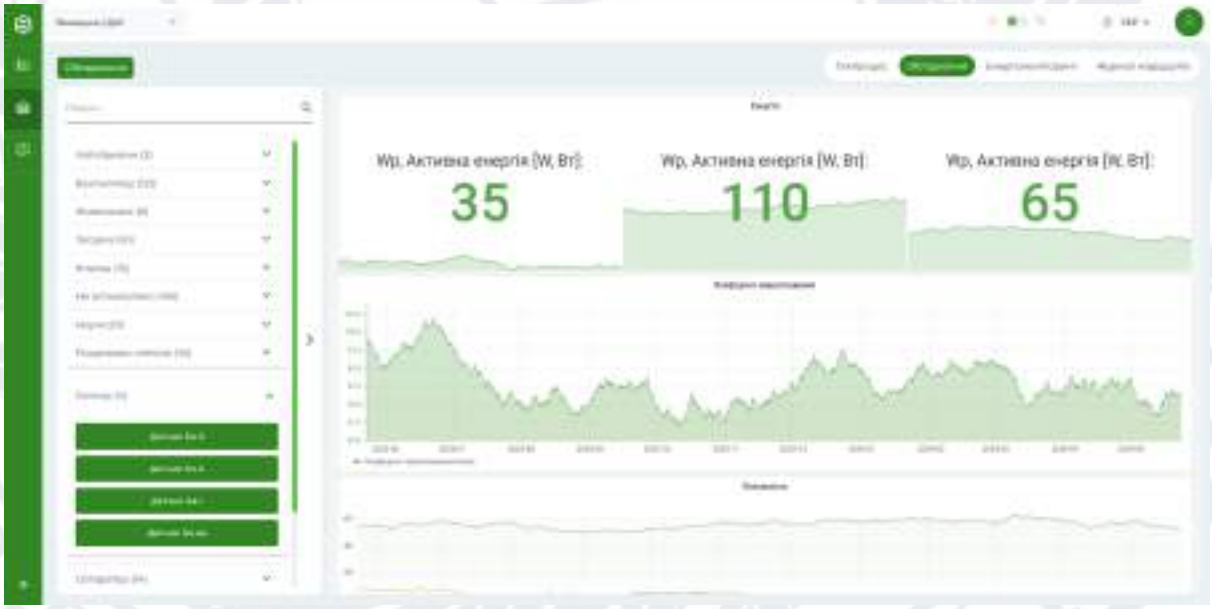


Рисунок 3.19 – Вкладка «Обладнання» на сторінці «Енергоефективність»

Таким чином, система забезпечує широкий функціонал для ефективного управління виробничими активами, що допомагає підприємствам підвищувати продуктивність, знижувати витрати та забезпечувати стабільний розвиток.

ВИСНОВКИ

У ході виконання дипломної роботи була досягнута основна мета — розробка системи керування ефективністю виробничих активів «SAKURA». Ця система дозволяє здійснювати моніторинг, планування та управління ресурсами в режимі реального часу, що є критично важливим для сучасних підприємств в умовах високої конкуренції та необхідності оптимізації виробничих процесів.

Розробка системи включала детальний аналіз предметної галузі, що дозволило чітко визначити вимоги до функціональності та архітектури «SAKURA». Завдяки використанню сучасних інформаційних технологій, таких як Інтернет речей (IoT), хмарні сервіси та аналітика великих даних, вдалося створити ефективну систему, яка значно підвищує рівень автоматизації та продуктивності підприємств.

Реалізовані функціональні можливості системи включають моніторинг робочих місць, операційний дашборд для планування та організації завдань, а також енергомоніторинг обладнання. Це забезпечує керівництву підприємства оперативний доступ до ключової інформації, що дозволяє швидко приймати обґрунтовані рішення щодо управління виробничими активами.

Теоретичне значення отриманих результатів полягає у розробці концептуальних основ EAM систем та принципів інтеграції сучасних IT технологій у процеси управління активами. Отже, виконання даної дипломної роботи, привело до успішного проектування та реалізації системи «SAKURA», що відповідає сучасним вимогам управління виробничими активами. Отримані результати свідчать про доцільність впровадження таких систем для забезпечення конкурентоспроможності підприємств у сучасних умовах.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. Enterprise Asset Management, EAM [Електронний ресурс] / it.ua – Режим доступу до ресурсу: https://www.it.ua/knowledge-base/technology-innovation/enterprise-asset-management-eam
2. Індустрія 4.0 в Україні [Електронний ресурс] / industry4-0-ukraine.com.ua – Режим доступу до ресурсу: https://industry4-0-ukraine.com.ua/2021/12/09/apm-4-0-keruvannya-efektivnistyu/
3. Корпоративна інформаційна система [Електронний ресурс] / uk.wikipedia.org – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Корпоративна_інформаційна_система\
4. CRM-системи як складова сучасних технологій автоматизації управління підприємством. / Мусієнко А. В. [Електронний ресурс] / ir.kneu.edu.ua – Режим доступу до ресурсу: https://ir.kneu.edu.ua/server/api/core/bitstreams/901dfe18-6130-4e7b-b0cd-31d04d4461d9/content.
5. Maximo application suite // IBM. *IBM in Deutschland, Österreich und der Schweiz*. [Електронний ресурс] / ibm.com – Режим доступу до ресурсу: https://www.ibm.com/products/maximo
6. Maximise asset health and performance with SAP [Електронний ресурс] / sap.com – Режим доступу до ресурсу: https://www.sap.com/ukraine/products/scm/asset-management-eam.html
7. Infor EAM is now part of Hexagon! [Електронний ресурс] / infor.com – Режим доступу до ресурсу: https://www.infor.com/hexagon-eam.
8. Oracle Enterprise Asset Management Overview [Електронний ресурс] / docs.oracle.com – Режим доступу до ресурсу: https://docs.oracle.com/cd/E18727_01/doc.121/e13670/T259967T259970.htm
9. Что такое Asset Performance Management и зачем его применяют передовые промышленные предприятия? / Chernyshov A. [Електронний ресурс] /

linkedin.com – Режим доступу до ресурсу: https://www.linkedin.com/pulse/что-такое-asset-performance-management-и-зачем-его-andrey-chernyshov-mtetf/.

10. What is the Internet of Things (IoT)? [Електронний ресурс] / aws.amazon.com – Режим доступу до ресурсу: https://aws.amazon.com/what-is/iot/
11. Client-Server Architecture. / Özsu M. T. [Електронний ресурс] / link.springer.com – Режим доступу до ресурсу: https://link.springer.com/referenceworkentry/10.1007/978-1-4899-7993-3_664-2
12. Layered (N-Layer) Architecture with SOLID Design Principles / Ozkaya M. [Електронний ресурс] / medium.com – Режим доступу до ресурсу: https://medium.com/design-microservices-architecture-with-patterns/layered-n-layer-architecture-with-solid-design-principles-15967a518ff1.
13. ReactJS App Architecture [Електронний ресурс] / aws.amazon.com – Режим доступу до ресурсу: https://medium.com/@Xourse/reactjs-app-architecture-7a33d7ae9834
14. Організація комп'ютерних систем із розподіленою пам'яттю [Електронний ресурс] / studfile.net – Режим доступу до ресурсу: https://studfile.net/preview/3907471/page:179/.
15. Load Balancer та Reverse Proxy у мікросервісній архітектурі [Електронний ресурс] / habr.com – Режим доступу до ресурсу: https://habr.com/ru/companies/otus/articles/741136/
16. Message broker per service / Vlad D. [Електронний ресурс] / habr.com – Режим доступу до ресурсу: https://habr.com/ru/articles/774636/
17. Grafana documentation / Grafana [Електронний ресурс] / grafana.com – Режим доступу до ресурсу: https://grafana.com/docs/grafana/latest/
18. What is the ELK Stack? [Електронний ресурс] / aws.amazon.com – Режим доступу до ресурсу: https://aws.amazon.com/what-is/elk-stack/
19. ‘ALL IN’ Cybersecurity. [Електронний ресурс] / glesec.com – Режим доступу до ресурсу: https://www.glesec.com/
20. Безпека та імплементація аутентифікації та авторизації в веб-додатках за допомогою JWT та HTTPS протоколу. / Фоучек В.О., Лухверчик С.А., Богач

І.В. // Матеріали Всеукраїнської науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2024)» 2024», Вінниця, 2024. [Електронний ресурс]. Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/view/21044

21. MeshCentral Docs [Електронний ресурс] / github.com – Режим доступу до ресурсу: https://github.com/Ylianst/MeshCentral

ДОДАТОК А

Прототип операційного дашборду системи SAKURA

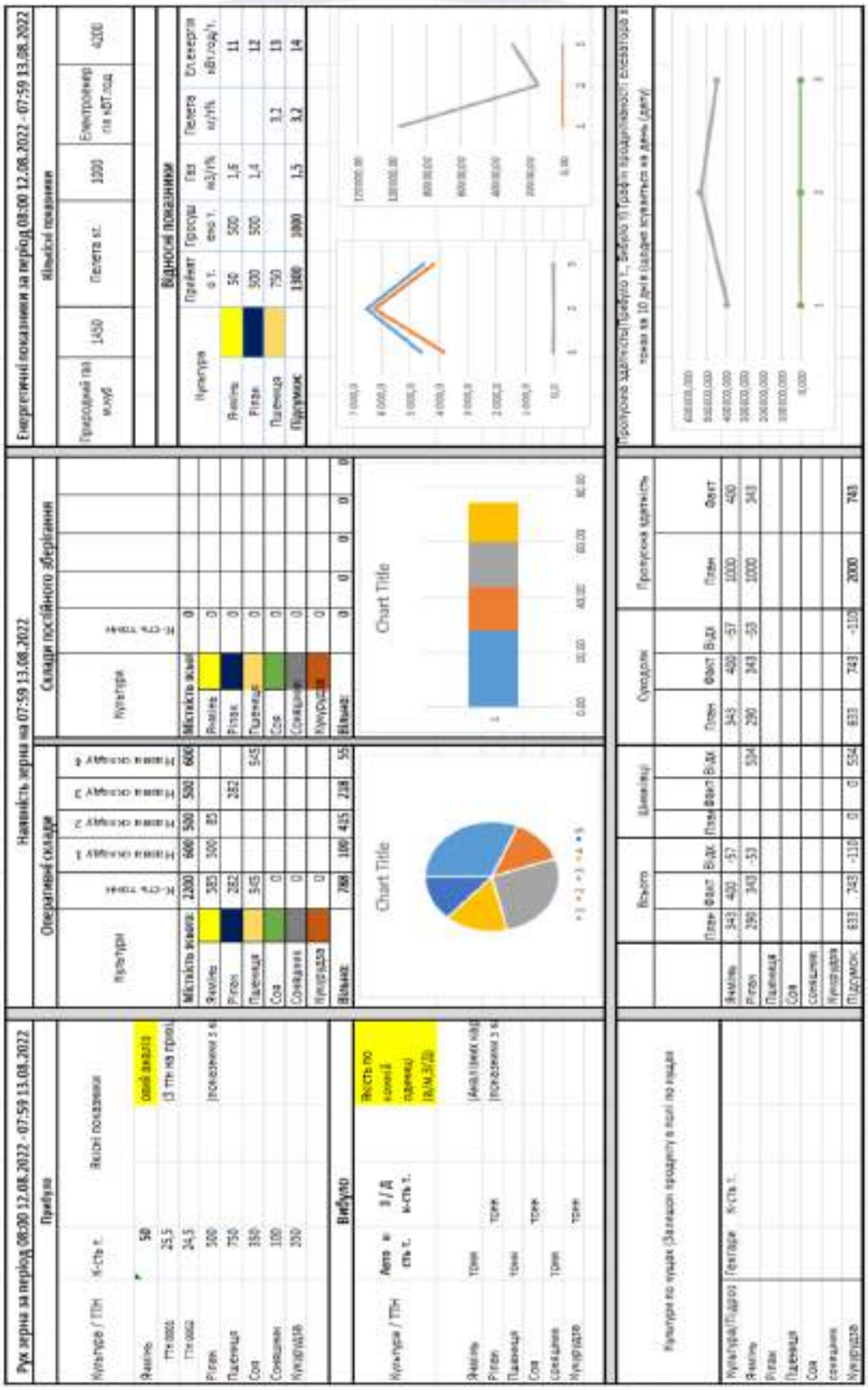


Рисунок А.1 – Перший рисунок додатка

ДОДАТОК Б

Декларація щодо унікальності текстів роботи

та невикористання матеріалів інших авторів без посилань

ДЕКЛАРАЦІЯ

про дотримання академічної доброчесності

Я, _____

_____

Повністю вказується ПІБ та статус (посада для працівників, освітня (освітньо-наукова) програма – для здобувачів вищої освіти)

що нижче підписалась/підписався, розуміючи та підтримуючи загальновизнані засади справедливості, доброчесності та законності,

ЗОБОВ'ЯЗУЮСЬ:

дотримуватися принципів та правил академічної доброчесності, що визначені законодавством України, локальними нормативними актами Донецького національного університету імені Василя Стуса, положеннями, правилами, умовами, визначеними іншими суб'єктами, та не допускати їх порушення.

ПІДТВЕРДЖУЮ:

що мені відомі положення статті 42 Закону України «Про освіту»;

що у даній роботі не представляла/представляв чиїсь роботи повністю або частково як свої власні. Там, де я скористалася/скористався працею інших, я зробила/зробив відповідні посилання на джерела інформації;

що дана робота не передавалась іншим особам і подається вперше, не порушує авторських та суміжних прав закріплених статтями 21-25 Закону України «Про авторське право та суміжні права», а дані та інформація не отримувались в недозволений спосіб.

УСВІДОМЛЮЮ:

що ця робота може бути перевірена університетом на плагіат або інші порушення академічної доброчесності, в тому числі з використанням спеціалізованих сервісів;

що у разі порушення академічної доброчесності, до мене можуть бути застосовані процедури, передбачені законодавством України та Кодексом академічної доброчесності та корпоративної етики Донецького національного університету імені Василя Стуса, іншими локальними нормативними актами університету, та я можу бути притягнута/притягнутий до академічної відповідальності.

_____ _____

(дата) (підпис)