ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА»
Факультет автоматизації виробництва, інформаційних та управлінських технологій
Кафедра інформаційних технологій та аналітики даних

«Допущено до захисту»
Гарант ОПП
Ірина ГЕТЬМАН

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра

за підсумками виконання
освітньо-професійної програми
«Комп'ютерні науки»
за спеціальністю 122 Комп`ютерні науки

на тему «Програмно-методичний комплекс з ведення електронного агрегатного журналу»

Керівник роботи Світлана ГУРКОВСЬКА

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

Здобувач Лях Дмитро

Підсумкова оцінка за атестацію			

Голова ЕК Антон КУДРЯВЦЕВ

ЗАПОРІЖЖЯ 2026

ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА»

Факультет	автоматизації виробництва, інформаційних та управлінських технологій
Кафедра	інформаційних технологій та аналітики даних
Ступінь вищої освіти	бакалавр
Спеціальність	122 Комп'ютерні науки
ОПП	Комп'ютерні науки

ЗАТВЕРДЖУЮ
Гарант ОПП

_____ Ірина ГЕТЬМАН

«26» лютого 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Ляху Дмитру Олександровичу
(прізвище, ім'я, по батькові здобувача)

1. Тема роботи *«Програмно-методичний комплекс з ведення електронного агрегатного журналу»*
керівник роботи *Гурковська Світлана Сергіївна,* _____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом Університету від *23.02.2026 р. №41/23.02.2026*
2. Термін подання роботи *16.06.2026 р.*
3. Вихідні дані до роботи Навчальна література, державні стандарти та методичні вказівки з дисциплін «Інженерія програмного забезпечення», «Бази даних», «Вебтехнології», «Проєктування інформаційних систем»; документація з розроблення програмного забезпечення мовою C# на платформі ASP.NET Core та СУБД PostgreSQL; наукові публікації з тематики розробки систем управління активами підприємства (TOiP) та SCADA-інтерфейсів; нормативні документи щодо експлуатації та технічного обслуговування промислового обладнання цеху випалювання залізорудних обкатків; результати власного проєктування, тестування та економічного обґрунтування розробленого програмного комплексу.
4. Зміст пояснювальної записки (перелік питань) Зміст. Вступ.
Розділ 1. Аналіз стану питання автоматизації ведення агрегатного журналу на виробництв.
Розділ 2. Розробка математичної моделі програмно-методичного комплексу для ведення агрегатного журналу
Розділ 3. Практична реалізація програмно-методичного комплексу.
Розділ 4. Тестування та впровадження системи
Висновки. Перелік посилань. Додатки.
5. Перелік графічного (демонстраційного) матеріалу (з точним зазначенням обов'язкових креслень): Актуальність, мета, об'єкт, предмет і завдання дослідження, порівняльний аналіз програмних аналогів (EAM/CMMS-систем), архітектура веборієнтованої системи (MVC), ER-діаграма структури бази даних, діаграми UML (прецедентів, діяльності, послідовностей), блок-схема алгоритму обробки інцидентів та перерахунку дат ТО, інтерфейси користувача у стилі SCADA, схема ролей та розмежування прав доступу (RBAC), висновки до роботи.

6. Консультанти по роботі, із зазначенням розділів роботи, що стосуються їх.

Розділ	Прізвище, ініціали та посада консультанта
1	Гурковська С.С., каф. ІТАД
2	Гурковська С.С., каф. ІТАД
3	Гурковська С.С., каф. ІТАД
4	Гетьман І.А.., доц. каф. ІНТЕХАД

7. Дата видачі завдання *26.02.2026.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи
1	Розділ 1. АНАЛІЗ стану питання автоматизації ведення агрегатного журналу на виробництв	10.05.2026 – 15.05.2026
2	Розділ 2. Розробка математичної моделі програмно-методичного комплексу для ведення агрегатного журналу	15.05.2026 – 25.05.2026
3	Розділ 3. Практична реалізація програмно-методичного комплексу	25.05.2026 – 28.05.2026
4	Розділ 4. Економічне обґрунтування та розрахунок ефективності впровадження програмного комплексу	28.06.2026 – 09.06.2026
6	Висновки, перелік посилань, вступ, зміст, реферат	09.06.2026 – 14.06.2026
7	Подання завершеної роботи. Перевірка на академічний плагіат	14.06.2026 – 16.06.2026
8	Остаточне оформлення роботи, презентаційного матеріалу, автореферату	16.06.2026 – 18.06.2026
9	Рецензування завершеної роботи. Захист	18.06.2026 – 23.06.2026

Здобувач (Лях Дмитро)

Керівник роботи (Світлана ГУРКОВСЬКА)

РЕФЕРАТ

Лях Д. О. Розробка програмного комплексу з ведення електронного агрегатного журналу

Кваліфікаційна робота на здобуття освітнього ступеня бакалавра комп'ютерних наук за спеціальністю 122 «Комп'ютерні науки». – ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА» МОН України, м. Запоріжжя, 2025.

Метою роботи є підвищення ефективності технічного обслуговування (ТОіР) промислового обладнання та мінімізація вимушених простоїв агрегатів шляхом розроблення веборієнтованої системи оперативного моніторингу й координації ремонтних робіт у режимі реального часу.

Об'єкт дослідження – процес оперативного моніторингу технічного стану та організації ремонтного обслуговування обладнання на металургійному підприємстві.

Предмет дослідження – моделі, архітектурні патерни та програмні засоби автоматизації обліку агрегатів, управління інцидентами та планування регламентного ТО.

Методологія базується на об'єктно-орієнтованому аналізі (UML), реляційному моделюванні (3NF, Fluent API), патернах MVC, Dependency Injection, Repository, рольовому керуванні доступом (RBAC) та методах модульного тестування (xUnit).

У роботі розроблено вебдодаток «Електронний агрегатний журнал», адаптований до умов цеху випалювання залізорудних обкатків. Реалізовано Back-End на стеку C# / ASP.NET Core з реляційною СУБД PostgreSQL (Entity Framework Core). Інтегровано авторизацію Microsoft Identity для розмежування прав операторів, слюсарів КВПіА та керівництва цеху. Створено ергономічний HMI-интерфейс у стилі SCADA-панелей із динамічною індикацією станів (CRITICAL, PENDING, RESOLVED) та інтерактивним модулем замовлення запасних частин.

Практична цінність полягає в повній відмові від паперового обліку, автоматизації контролю регламентних робіт, прозорому розподілі ТМЦ та скороченні часу реакції на аварії (MTTR) на 25–30%. Комплекс придатний до масштабування та інтеграції в існуючу цифрову інфраструктуру підприємства.

КЛЮЧОВІ СЛОВА: ЕЛЕКТРОННИЙ АГРЕГАТНИЙ ЖУРНАЛ, SCADAІНТЕРФЕЙС, МОНІТОРИНГ СТАНУ ОБЛАДНАННЯ, ЦЕХ ВИПАЛЮВАННЯ, СЛЮСАР КВПіА, ASP.NET CORE, C#, POSTGRESQL, РОЛЬОВА МОДЕЛЬ ДОСТУПУ (RBAC), ВЕБ-ДОДАТОК.

ABSTRACT

Lyakh D. O. Development of a software package for maintaining an electronic aggregate journal

Qualification work for obtaining a bachelor's degree in computer science in specialty 122 "Computer Science". – LLC "TECHNICAL UNIVERSITY "METINVEST POLYTECHNIC" MES of Ukraine, Zaporizhia, 2025.

The purpose of the work is to increase the efficiency of technical maintenance (T&R) of industrial equipment and minimize forced downtime of units by developing a web-based system of operational monitoring and coordination of repair work in real time.

The object of the study is the process of operational monitoring of the technical condition and organization of repair maintenance of equipment at a metallurgical enterprise.

The subject of the study is models, architectural patterns and software tools for automating unit accounting, incident management and planning routine maintenance.

The methodology is based on object-oriented analysis (UML), relational modeling (3NF, Fluent API), MVC patterns, Dependency Injection, Repository, role-based access control (RBAC) and unit testing methods (xUnit).

The work developed a web application "Electronic aggregate journal", adapted to the conditions of the iron ore rolling mill roasting shop. Back-End was implemented on the C# / ASP.NET Core stack with the PostgreSQL relational DBMS (Entity Framework Core). Microsoft Identity authorization was integrated to distinguish the rights of operators, KVPiA fitters and shop management. An ergonomic HMI interface in the style of SCADA panels was created with dynamic status indication (CRITICAL, PENDING, RESOLVED) and an interactive spare parts ordering module.

The practical value lies in the complete abandonment of paper accounting, automation of routine work control, transparent distribution of materials and equipment and reduction of the response time to accidents (MTTR) by 25–30%. The complex is suitable for scaling and integration into the existing digital infrastructure of the enterprise.

KEYWORDS: ELECTRONIC UNIT LOG, SCADA INTERFACE, EQUIPMENT STATUS MONITORING, FIRING PLANT, EQUIPMENT CONTROL SYSTEM INSTALLER, ASP.NET CORE, C#, POSTGRESQL, ROLE-BASED ACCESS MODEL (RBAC), WEB APPLICATION.

Зміст

ВСТУП

Актуальність теми. Сучасне гірничо-металургійне виробництво в епоху Індустрії 4.0 вимагає високого рівня автоматизації не лише безпосередньо технологічних процесів, а й систем управління активами підприємства. Аглофабрика (цех випалювання) є складним технологічним комплексом, де в безперервному режимі працюють десятки критично важливих агрегатів: випалювальні машини, дробарки, вентилятори, живильники, конвеєри та насосні станції. Вихід з ладу будь-якого елемента означає зупинку ланцюга, а кожна хвилина простою завдає прямих фінансових збитків підприємству.

Попри масштабність виробництва та наявність сучасного обладнання, на багатьох вітчизняних підприємствах облік технічного стану агрегатів досі ведеться вручну за допомогою паперових журналів. Такий підхід має низку критичних недоліків: інформація про аварії передається усно та із затримками; відсутнє оперативне сповіщення ремонтного персоналу (слюсарів КВПіА, електриків, механіків); ускладнюється контроль за виконанням заявок та рухом запасних частин; керівництво не має можливості бачити загальну картину стану цеху в режимі реального часу. Відповідно, розробка спеціалізованої інформаційної системи (Електронного агрегатного журналу), яка замінить паперовий документообіг на цифровий інструмент управління технічним станом обладнання, є надзвичайно актуальним науково-практичним завданням.

Мета дослідження полягає у підвищенні ефективності технічного обслуговування промислового обладнання та мінімізації часу його простоїв шляхом розробки та впровадження веборієнтованої

інформаційної системи моніторингу і обліку стану агрегатів (з інтерфейсом SCADA-типу).

Для досягнення поставленої мети було сформульовано та вирішено такі завдання:

Провести системний аналіз предметної області (процесів експлуатації та ремонту обладнання цеху випалювання) і виявити недоліки існуючого паперового обліку.

Обґрунтувати вибір стека технологій для створення надійного корпоративного вебдодатка.

Спроєктувати об'єктно-реляційну модель бази даних для зберігання паспортів обладнання, логів інцидентів та історії технічного обслуговування.

Розробити серверну частину (Back-End) системи з використанням архітектурного патерну MVC.

Розробити адаптивний інтерфейс користувача (Front-End) з кольоровою індикацією критичних станів обладнання та інтерактивними дашбордами.

Впровадити рольову модель управління доступом (RBAC) для розмежування повноважень між операторами, ремонтним персоналом та керівництвом.

Об’єкт дослідження – процес моніторингу технічного стану та організації ремонту промислового обладнання на металургійному підприємстві.

Предмет дослідження – моделі, методи та програмні засоби створення інформаційної системи для обліку та управління життєвим циклом інцидентів технологічного обладнання.

Методи дослідження. У роботі використано методи системного аналізу (для дослідження бізнес-процесів цеху), методи об'єктно-орієнтованого проєктування та графічного моделювання UML (для

розробки архітектури системи), а також принципи проєктування реляційних баз даних.

Практичне значення одержаних результатів. Розроблений програмний комплекс готовий до впровадження в реальні виробничі умови цеху випалювання залізорудних обкатків (аглофабрики). Його використання дозволить відмовитися від паперових агрегатних журналів, скоротити час реакції слюсарів на аварійні ситуації, автоматизувати контроль за регламентним обслуговуванням та забезпечити прозорість процесу замовлення запасних частин.

1. АНАЛІЗ СТАНУ ПИТАННЯ АВТОМАТИЗАЦІЇ ВЕДЕННЯ АГРЕГАТНОГО ЖУРНАЛУ НА ВИРОБНИЦТВІ

1.1. Актуальність автоматизації ведення агрегатного журналу в промисловості

У сучасних умовах промислового виробництва надійність та безперебійна робота технологічного обладнання є одним із ключових чинників економічної ефективності підприємства. Будь-яка незапланована зупинка агрегату призводить до прямих фінансових втрат, зриву виробничих планів та додаткових витрат на аварійний ремонт. Саме тому питання своєчасного виявлення, фіксації та усунення несправностей обладнання набуває особливої актуальності.

Традиційно на промислових підприємствах для обліку стану агрегатів використовується агрегатний журнал – документ, у якому фіксуються всі події, пов'язані з роботою обладнання: аварійні зупинки, планове технічне обслуговування (ТО), виконані ремонтні роботи та витрачені матеріали. Однак у більшості вітчизняних підприємств цей журнал досі ведеться у паперовому вигляді або у вигляді простих електронних таблиць (Microsoft Excel), що породжує цілий ряд системних проблем.

Проблеми паперового та напівавтоматизованого обліку:

1. Втрата та пошкодження даних. Паперові журнали фізично псуються, губляться або знищуються внаслідок пожеж, затоплень та інших надзвичайних ситуацій на виробництві. Відновлення втрачених записів є практично неможливим.

2. Відсутність контролю в реальному часі. Керівник цеху дізнається про аварійну зупинку агрегату лише тоді, коли особисто перегляне паперовий журнал або отримає усне повідомлення від чергового. Це суттєво затримує прийняття управлінських рішень.

3. Неможливість простежити ланцюжок відповідальності. У паперовому журналі складно однозначно визначити: хто зафіксував поломку, хто виконав ремонт та скільки часу зайняло усунення несправності. Підпис у журналі легко підробити або залишити незаповненим.

4. Відсутність аналітики. Паперові записи не дозволяють автоматично формувати статистику: яке обладнання ламається найчастіше, який слюсар виконує ремонти найшвидше, скільки коштів витрачається на запчастини щомісяця. Для отримання таких даних необхідно вручну перераховувати сотні записів.

5. Проблема закупівлі запчастин. При паперовому обліку запит на придбання необхідної деталі передається усно або через паперову заявку, яка може загубитися або залишитися без відповіді. Відстежити статус такого запиту практично неможливо.

6. Людський фактор. Помилки при заповненні, нерозбірливий почерк, пропущені поля та суб'єктивність опису несправності знижують якість інформації та ускладнюють подальший аналіз.

Світова практика промислового менеджменту свідчить про стрімкий перехід підприємств до цифрових систем управління технічним обслуговуванням – так званих CMMS (Computerized Maintenance Management System – комп'ютеризована система управління технічним обслуговуванням). За даними аналітичних досліджень, впровадження подібних систем дозволяє:

- скоротити час простою обладнання на 20–30%;
- знизити витрати на аварійні ремонти на 15–25%;

- підвищити точність планування технічного обслуговування на 40%;
- забезпечити повну прозорість ланцюжка відповідальності персоналу.

Однак більшість комерційних CMMS-систем є дорогими, складними у впровадженні та не адаптованими до специфіки конкретного підприємства. Вони розраховані на великі корпорації та потребують значних витрат на ліцензування та навчання персоналу.

Виходячи з вищезазначеного, актуальною є розробка програмно-методичного комплексу для ведення агрегатного журналу, який би:

- був адаптований до реальних умов роботи конкретного виробничого цеху;
- реалізував чітку рольову модель доступу (агломератник, слюсар КВП, керівник, адміністратор);
- забезпечував автоматичну фіксацію виконавця та часу ремонту без можливості підробки;
- дозволяв керівнику в реальному часі відстежувати стан усього обладнання цеху;
- реалізував цифровий ланцюжок від реєстрації аварії до закупівлі запчастин та закриття заявки;
- формував автоматичну звітність для аналізу ефективності роботи персоналу та стану обладнання.

Таким чином, автоматизація ведення агрегатного журналу є не просто технічним вдосконаленням, а необхідною умовою підвищення конкурентоспроможності та операційної ефективності сучасного промислового підприємства.

1.2. Аналіз предметної області «Агрегатний журнал»

Агрегатний журнал – це офіційний обліковий документ промислового підприємства, призначений для систематичної реєстрації всіх подій, пов'язаних із технічним станом та експлуатацією виробничого обладнання. Він є первинним джерелом інформації для служби технічного обслуговування та ремонту (ТОіР) і слугує юридичним підтвердженням факту виконання ремонтних робіт.

Відповідно до вимог промислової експлуатації, агрегатний журнал охоплює такі категорії записів:

- Аварійні зупинки – незаплановані зупинки обладнання внаслідок поломки або відмови вузла;
- Планове технічне обслуговування (ТО) – регламентні роботи згідно з графіком ППР (планово-попереджувальних ремонтів);
- Запити на закупівлю запчастин – офіційні заявки на придбання необхідних матеріалів та деталей;
- Акти виконаних робіт – підтвердження факту усунення несправності із зазначенням виконавця та витраченого часу.

Структура предметної області «Агрегатний журнал» охоплює три взаємопов'язані підсистеми, які разом формують повний цикл управління технічним станом обладнання:

— Підсистема 1 (Облік обладнання): центральним об'єктом системи є агрегат – виробнича одиниця обладнання, стан якої підлягає моніторингу. Кожен агрегат характеризується такими атрибутами:

a) унікальна назва та інвентарний номер;
b) місце розташування (дільниця, цех);
c) поточний статус (працює / зупинений / на ТО);
d) історія всіх подій за весь період експлуатації.

Сукупність агрегатів утворює реєстр обладнання цеху, який є довідковою базою для всіх інших операцій системи.

— Підсистема 2 (Управління подіями): ключовою функцією агрегатного журналу є реєстрація та відстеження подій – будь-яких змін у стані обладнання. У рамках розроблюваного комплексу виділяються такі типи подій:

Аварійна зупинка містить:

a) дату та час виникнення несправності;

b) найменування агрегату;

c) опис характеру несправності (від оператора);

d) статус усунення (нова / в роботі / виконано);

e) дані про виконавця та час закриття заявки.

Планове ТО містить:

a) дату проведення;

b) перелік виконаних регламентних робіт;

c) підтвердження факту виконання від оператора дільниці.

Запит на закупівлю містить:

a) найменування необхідної запчастини або матеріалу;

b) обґрунтування потреби (прив'язка до конкретної аварії);

c) статус розгляду (очікує / схвалено / відхилено);

d) коментар керівника у разі відмови.

— Підсистема 3 (Управління персоналом та доступом): невід'ємною частиною предметної області є рольова модель – система розмежування прав доступу, яка визначає, які дії може виконувати кожен учасник процесу. У межах агрегатного журналу виділяються чотири ролі, які представлені в таблиці 1.

Таблиця 1.1- Управління персоналом та доступом

Роль	Функція в системі	Рівень доступу
Агломератник (Operator)	Реєстрація аварій та підтвердження ТО	Обмежений – лише своя дільниця
Слюсар КВП (Technician)	Виконання ремонтів та запити на закупівлю	Оперативний – всі аварії цеху
Керівник (Manager)	Схвалення закупівель та перегляд звітності	Управлінський – весь цех
Адміністратор (Admin)	Керування системою та персоналом	Повний – без обмежень

Ключовим поняттям предметної області є життєвий цикл заявки на ремонт – послідовність станів, через які проходить кожен запис від моменту реєстрації до закриття. Цей цикл складається з таких етапів:

Етап 1 – Реєстрація. Агломератник виявляє несправність та створює запис в системі, описуючи характер поломки та фіксуючи факт зупинки агрегату. Заявці автоматично присвоюється статус «НОВА».

Етап 2 – Прийняття в роботу. Слюсар КВП бачить нову заявку у своєму списку, приймає її до виконання. Якщо для ремонту необхідна відсутня деталь – він формує Запит на закупівлю. Статус змінюється на «В РОБОТІ».

Етап 3 – Розгляд запиту (за потреби). Керівник отримує запит на закупівлю, розглядає його та приймає рішення: «СХВАЛЕНО» або «ВІДХИЛЕНО» із зазначенням причини.

Етап 4 – Закриття заявки. Після завершення ремонту слюсар натискає кнопку «ВИКОНАНО». Система автоматично фіксує його ім'я як виконавця та записує точний час закриття заявки. Статус змінюється на «ВИКОНАНО».

Етап 5 – Аналіз та звітність. Керівник переглядає статистику: загальна кількість аварій, середній час усунення несправності, рейтинг надійності агрегатів, ефективність роботи кожного слюсаря.

Агрегатний журнал не є ізольованим документом – він є частиною загальної системи управління виробництвом і тісно пов'язаний з такими процесами підприємства:

- Планування виробництва – простої обладнання безпосередньо впливають на виконання виробничого плану;
- Управління запасами – дані про витрачені запчастини формують потребу в поповненні складу;
- Управління персоналом – статистика виконаних ремонтів є об'єктивним показником продуктивності технічного персоналу;
- Фінансовий облік – витрати на ремонти та закупівлі запчастин відображаються у собівартості продукції.

Таким чином, агрегатний журнал є інформаційним ядром служби технічного обслуговування підприємства, а його автоматизація створює основу для прийняття обґрунтованих управлінських рішень на всіх рівнях виробничої ієрархії.

1.3. Аналіз зацікавленої аудиторії та рольової моделі користувачів системи

При розробці будь-якого програмного комплексу одним із ключових етапів є визначення зацікавленої аудиторії (стейкхолдерів) – осіб та груп, які безпосередньо взаємодіють із системою або на яких впливають результати її роботи. Правильне визначення зацікавленої

аудиторії дозволяє сформулювати точні вимоги до функціональності системи та уникнути розробки непотрібних або надлишкових функцій.

У контексті програмно-методичного комплексу для ведення агрегатного журналу зацікавлена аудиторія поділяється на дві групи:

- Внутрішні користувачі – працівники підприємства, які безпосередньо працюють із системою;
- Зовнішні зацікавлені сторони – особи, які не працюють із системою напряму, але отримують вигоду від результатів її роботи (вище керівництво підприємства, відділ постачання, бухгалтерія).

1.3.1.Агломератник (Operator) – «Очі» системи

Хто це: Агломератник – це черговий оператор виробничої дільниці, який безпосередньо обслуговує агрегати та першим виявляє будь-які відхилення від нормального режиму роботи обладнання. Він не є технічним фахівцем з ремонту, але є першою ланкою у ланцюжку реагування на несправність.

Потреби користувача:

- Швидко та просто зафіксувати факт поломки без заповнення складних форм;
- Підтвердити виконання планового ТО на своїй дільниці;
- Не мати доступу до даних інших дільниць з міркувань конфіденційності.

Функції в системі:

- Створення записів про аварійні зупинки агрегатів;
- Опис характеру несправності у довільній текстовій формі;

- Підтвердження факту проведення планового технічного обслуговування;
- Перегляд стану заявок лише по своїй дільниці.

Рівень технічної грамотності: Базовий. Інтерфейс для даної ролі має бути максимально простим та інтуїтивно зрозумілим, оскільки оператор не є IT-фахівцем і працює в умовах виробничого шуму та стресу аварійної ситуації.

1.3.2. Слюсар КВП (Instrument Technician) – «Руки» системи

Хто це: Слюсар контрольно-вимірювальних приладів (КВП) – це технічний фахівець, який безпосередньо виконує ремонтні роботи на агрегатах. Він є ключовою виконавчою ланкою системи технічного обслуговування та несе відповідальність за якість та своєчасність ремонту.

Потреби користувача:

- Бачити актуальний список усіх відкритих заявок на ремонт у реальному часі;
- Мати можливість швидко оформити запит на необхідну запчастину;
- Підтвердити виконання ремонту одним натисканням кнопки;
- Мати документальне підтвердження своєї роботи для захисту від необґрунтованих претензій.
- Функції в системі:
- Перегляд повного списку активних аварійних заявок по всьому цеху;

- Формування запиту на закупівлю відсутніх запчастин із прив'язкою до конкретної заявки;
- Закриття заявки кнопкою «ВИКОНАНО» з автоматичною фіксацією імені та часу;
- Перегляд статусу своїх запитів на закупівлю (схвалено / відхилено).

Рівень технічної грамотності: Середній. Слюсар звик працювати з технічною документацією, тому може опанувати більш детальний інтерфейс порівняно з оператором.

1.3.3.Керівник (Manager) – «Мозок» системи

Хто це: Керівник цеху або начальник служби технічного обслуговування – це управлінець, який відповідає за загальну організацію ремонтних робіт, розподіл ресурсів та контроль ефективності персоналу. Він приймає стратегічні рішення на основі інформації, яку надає система.

Потреби користувача:

- Мати повну картину стану всього обладнання цеху в режимі реального часу;
- Оперативно розглядати запити на закупівлю запчастин та приймати рішення;
- Аналізувати ефективність роботи кожного слюсаря та надійність кожного агрегату;
- Мати готову звітність без необхідності ручного підрахунку даних.

Функції в системі:

- Перегляд загальної статистики та дашборду стану цеху;
- Розгляд запитів на закупівлю із можливістю схвалення або відхилення з коментарем;
- Перегляд звітності по персоналу: кількість виконаних ремонтів, середній час усунення;
- Перегляд звітності по обладнанню: рейтинг аварійності агрегатів, частота поломок;
- Контроль дотримання графіку планового ТО.

Рівень технічної грамотності: Середній та вище. Керівник орієнтований на аналітичні дані та зведені показники, а не на деталі окремих операцій.

1.3.4. Адміністратор (Admin) – «Контроль» системи

Хто це: Системний адміністратор або відповідальний IT-фахівець підприємства, який відповідає за технічне функціонування комплексу та управління обліковими записами користувачів.

Потреби користувача:

- Мати повний контроль над усіма даними та налаштуваннями системи;
- Оперативно створювати нових користувачів та призначати їм ролі;
- Виправляти помилкові записи без обмежень;
- Підтримувати актуальний реєстр агрегатів підприємства.

Функції в системі:

- Повний доступ до всіх розділів та функцій без обмежень;
- Створення, редагування та деактивація облікових записів користувачів;
- Призначення та зміна ролей для будь-якого користувача;
- Видалення помилкових або дублюючих записів;
- Редагування реєстру агрегатів (додавання, перейменування, видалення).

Рівень технічної грамотності: Високий. Адміністратор розуміє структуру системи та несе відповідальність за її коректне функціонування.

Таблиця 1.2 - Зведена таблиця рольової моделі

Критерій	Агломератник	Слюсар КВП	Керівник	Адміністратор
Реєстрація аварій	Так	Ні	Ні	Так
Перегляд всіх заявок	Ні	Так	Так	Так
Закриття заявок	Ні	Так	Ні	Так
Запит на закупівлю	Ні	Так	Ні	Так
Схвалення закупівлі	Ні	Ні	Так	Так
Перегляд статистики	Ні	Ні	Так	Так
Управління користувачами	Ні	Ні	Ні	Так
Редагування реєстру агрегатів	Ні	Ні	Ні	Так

Окрім безпосередніх користувачів системи, існують зовнішні зацікавлені сторони, які отримують вигоду від впровадження комплексу опосередковано:

Вище керівництво підприємства отримує об'єктивні дані про стан виробничої бази для прийняття інвестиційних рішень щодо модернізації або заміни застарілого обладнання.

Відділ постачання отримує структуровані та обґрунтовані заявки на закупівлю запчастин замість усних прохань, що підвищує точність та швидкість закупівельного процесу.

Бухгалтерія та фінансовий відділ отримують можливість точного обліку витрат на технічне обслуговування у розрізі кожного агрегату та кожного звітного періоду.

Служба охорони праці отримує документальне підтвердження своєчасного проведення регламентних робіт, що є обов'язковою вимогою при перевірках контролюючих органів.

Таким чином, аналіз зацікавленої аудиторії показує, що програмно-методичний комплекс для ведення агрегатного журналу охоплює широке коло учасників виробничого процесу та задовольняє потреби кожного з них через чітко розмежовану рольову модель доступу.

1.4. Огляд аналогів програм для ведення журналів технічного обслуговування (CMMS-систем)

Перед розробкою власного програмно-методичного комплексу необхідно провести детальний аналіз існуючих рішень на ринку програмного забезпечення для управління технічним обслуговуванням

обладнання. Такий аналіз дозволяє визначити сильні та слабкі сторони наявних продуктів, виявити незаповнені ніші та обґрунтувати доцільність розробки власного рішення.

Системи класу CMMS (Computerized Maintenance Management System – комп'ютеризована система управління технічним обслуговуванням) є найбільш поширеним типом програмного забезпечення для вирішення задач, аналогічних до тих, що ставляться перед розроблюваним комплексом. Розглянемо чотири найбільш відомі рішення цього класу.

1.4.1.SAP Plant Maintenance

Загальна характеристика: SAP Plant Maintenance (SAP PM) – це модуль управління технічним обслуговуванням у складі глобальної ERP-системи SAP S/4HANA, розробленої німецькою компанією SAP SE. Є одним із найпотужніших та найпоширеніших рішень для управління обслуговуванням промислового обладнання у світі. Використовується переважно великими міжнародними корпораціями у нафтогазовій, хімічній, металургійній та енергетичній галузях.

Функціональні можливості:

- Повний облік технічних об'єктів та їх ієрархічна структура;
- Планування та виконання профілактичного та аварійного технічного обслуговування;
- Управління замовленнями на технічне обслуговування з повним контролем витрат;
- Інтеграція з модулями управління матеріалами (SAP MM) для автоматичного формування заявок на закупівлю запчастин;

- Розширена аналітика та звітність з можливістю побудови довільних звітів;
- Підтримка мобільних пристроїв для роботи безпосередньо у виробничому цеху.

Переваги:

- Надзвичайно широкий функціонал, що охоплює всі аспекти управління технічним обслуговуванням;
- Висока надійність та стабільність системи, перевірена десятиліттями використання;
- Повна інтеграція з іншими модулями SAP (фінанси, постачання, персонал);
- Відповідність міжнародним стандартам та регуляторним вимогам.

Недоліки:

- Надзвичайно висока вартість ліцензування – від десятків до сотень тисяч доларів США залежно від кількості користувачів;
- Тривалий та дорогий процес впровадження – від 6 місяців до кількох років;
- Висока складність налаштування та необхідність спеціально навченого персоналу для адміністрування;
- Надлишкова функціональність для середніх та малих підприємств;
- Відсутність адаптації до специфіки конкретного підприємства без дорогої кастомізації.

Висновок щодо аналогу: SAP Plant Maintenance є потужним, але абсолютно недоступним за ціною та складністю рішенням для середніх промислових підприємств. Його впровадження вимагає окремого бюджету, команди консультантів та тривалого часу, що робить його нереалістичним варіантом для більшості вітчизняних підприємств.

1.4.2. UpKeep CMMS

Загальна характеристика: UpKeep – це хмарна CMMS-система американської розробки, орієнтована на малий та середній бізнес. Позиціонується як сучасна, зручна та доступна альтернатива важким корпоративним системам. Отримала широке визнання завдяки зручному мобільному інтерфейсу та відносно невисокій вартості.

Функціональні можливості:

- Створення та відстеження заявок на технічне обслуговування (work orders);
- Управління реєстром обладнання з прикріпленням технічної документації;
- Планування профілактичного технічного обслуговування за розкладом;
- Управління складом запчастин та матеріалів;
- Мобільний додаток для iOS та Android з можливістю роботи офлайн;
- Базова аналітика та звітність у вигляді дашборду.

Переваги:

- Інтуїтивно зрозумілий інтерфейс, не потребує тривалого навчання персоналу;
- Швидке впровадження – система готова до роботи протягом кількох днів;
- Доступна мобільна версія для роботи безпосередньо на виробничому майданчику;
- Хмарне розгортання без необхідності підтримки власної серверної інфраструктури.

Недоліки:

- Система англомовна, відсутня повноцінна локалізація для українських підприємств;
- Хмарне зберігання даних може бути неприйнятним для підприємств з вимогами до конфіденційності виробничої інформації;
- Вартість підписки в доларах США є суттєвим фінансовим навантаженням для вітчизняних підприємств;
- Відсутність можливості глибокої адаптації під специфічні бізнес-процеси конкретного підприємства;
- Залежність від стабільного інтернет-з'єднання, що є проблемою для виробничих цехів.

Висновок щодо аналогу: UpKeep є значно доступнішим рішенням порівняно з SAP PM, однак його хмарна природа, відсутність україномовного інтерфейсу та валютна вартість підписки створюють суттєві бар'єри для використання на вітчизняних промислових підприємствах.

1.4.3. Fiix Software

Загальна характеристика: Fiix – це хмарна CMMS-система канадської розробки, придбана у 2021 році компанією Rockwell Automation – одним із світових лідерів у галузі промислової автоматизації. Система орієнтована на виробничі підприємства середнього розміру та пропонує розширені можливості інтеграції з промисловим обладнанням та IoT-пристроями.

Функціональні можливості:

- Управління заявками на технічне обслуговування з розширеним відстеженням статусів;
- Планування профілактичного ТО на основі часових інтервалів або показників лічильників;
- Управління активами з повною історією технічного обслуговування кожного агрегату;
- Інтеграція з ERP-системами та IoT-сенсорами для автоматичного моніторингу стану обладнання;
- Розширена аналітика з можливістю побудови власних звітів та KPI-дашбордів;
- API для інтеграції зі сторонніми системами підприємства.

Переваги:

- Потужні можливості інтеграції з промисловим обладнанням та зовнішніми системами;
- Розвинена аналітика для прийняття рішень на основі даних;
- Підтримка предиктивного технічного обслуговування на основі даних з сенсорів;
- Регулярні оновлення та активна підтримка розробника.

Недоліки:

- Висока вартість для повного функціоналу – преміум-тарифи розраховані на великі підприємства;
- Складність налаштування інтеграцій потребує технічних спеціалістів;
- Відсутність україномовного інтерфейсу та локальної підтримки;
- Надлишковий функціонал для підприємств, яким не потрібна IoT-інтеграція;

- Хмарна архітектура з тими самими обмеженнями щодо конфіденційності даних.

Висновок щодо аналогу: Fiix є технологічно передовим рішенням, орієнтованим на підприємства з розвиненою цифровою інфраструктурою. Однак його складність, вартість та відсутність локалізації роблять його недоступним для більшості вітчизняних виробничих підприємств.

1.4.4.1С:Управління ремонтами

Загальна характеристика: «1С:Управління ремонтами та обслуговуванням обладнання» – це спеціалізована конфігурація на платформі 1С:Підприємство 8.3, розроблена для автоматизації процесів технічного обслуговування та ремонту на промислових підприємствах пострадянського простору. Є найбільш поширеним рішенням даного класу серед вітчизняних підприємств завдяки звичному інтерфейсу та можливості україномовної локалізації.

Функціональні можливості:

- Ведення реєстру обладнання з технічними характеристиками та паспортними даними;
- Планування та облік планово-попереджувальних ремонтів (ППР);
- Реєстрація та відстеження аварійних зупинок обладнання;
- Облік витрат матеріалів та запчастин на ремонти;
- Інтеграція з іншими конфігураціями 1С (бухгалтерія, склад, зарплата);
- Формування стандартних звітів та актів виконаних робіт.

Переваги:

- Звичний для бухгалтерів та економістів інтерфейс платформи 1С;
- Можливість адаптації конфігурації під специфіку конкретного підприємства силами місцевих 1С-програмістів;
- Наявність україномовної версії та локальна технічна підтримка;
- Широка мережа партнерів та фахівців з впровадження в Україні;
- Інтеграція з бухгалтерським та складським обліком підприємства.

Недоліки:

- Застарілий інтерфейс, не адаптований для роботи на мобільних пристроях та у виробничому цеху;
- Система розрахована на роботу в офісі, а не безпосередньо на виробничому майданчику;
- Відсутність сучасного веб-інтерфейсу та можливості роботи через браузер без встановлення клієнта;
- Ліцензування платформи 1С потребує окремих витрат;
- Складність налаштування рольової моделі доступу для виробничого персоналу різної кваліфікації.

Висновок щодо аналогу: «1С:Управління ремонтами» є найближчим за духом аналогом до розроблюваного комплексу серед розглянутих систем, однак його застарілий інтерфейс та відсутність сучасного веб-доступу роблять його незручним для використання безпосередньо у виробничому цеху операторами та слюсарями.

Таблиця 1.3 - Порівняльна таблиця аналогів

Критерій	SAP PM	UpKeep	Fiix	1С:Ремонти	Наш комплекс
Вартість	Дуже висока	Середня	Висока	Середня	Безкоштовно
Мова інтерфейсу	Англійська	Англійська	Англійська	Українська	Українська
Веб-доступ	Так	Так	Так	Обмежено	Так
Мобільність	Так	Так	Так	Ні	Так
Адаптація під цех	Складна	Обмежена	Складна	Можлива	Повна
Рольова модель для цеху	Загальна	Загальна	Загальна	Обмежена	Спеціалізована
SCADA-стиль інтерфейсу	Ні	Ні	Ні	Ні	Так
Хмарне зберігання	Так	Так	Так	Ні	Локально

Проведений аналіз існуючих CMMS-систем дозволяє зробити такі висновки:

По-перше, всі розглянуті системи є або надмірно дорогими, або недостатньо адаптованими до специфіки вітчизняних промислових підприємств середнього розміру.

По-друге, жодна з розглянутих систем не реалізує спеціалізованої рольової моделі, орієнтованої на чотирьохрівневу ієрархію виробничого персоналу цеху (оператор – технік – керівник – адміністратор).

По-третє, жодна із систем не використовує промисловий SCADA-стиль інтерфейсу, який є звичним та інтуїтивно зрозумілим для виробничого персоналу.

По-четверте, хмарна архітектура більшості рішень є неприйнятною для підприємств із вимогами до конфіденційності виробничих даних.

Таким чином, існує обґрунтована потреба у розробці власного програмно-методичного комплексу, який би поєднував доступність, україномовний інтерфейс промислового стилю, спеціалізовану рольову модель та локальне зберігання даних підприємства.

1.5. Використання методу експертних оцінок для вибору технологічного стеку розробки

При розробці програмного забезпечення одним із найважливіших рішень є вибір технологічного стеку – сукупності мов програмування, фреймворків, баз даних та інших інструментів, на основі яких буде побудована система. Неправильний вибір технологій може призвести до суттєвих проблем на етапі розробки, впровадження та подальшої підтримки системи.

Для обґрунтованого вибору технологічного стеку застосовується метод експертних оцінок – науковий підхід до прийняття рішень в умовах невизначеності, який базується на систематизованій думці кваліфікованих фахівців у відповідній галузі. Метод дозволяє перевести суб'єктивні судження експертів у кількісні показники та отримати об'єктивно обґрунтований результат.

Процедура застосування методу експертних оцінок складається з таких етапів:

1. Визначення альтернатив – переліку технологій, що розглядаються;
2. Визначення критеріїв оцінювання – показників, за якими порівнюються альтернативи;
3. Призначення вагових коефіцієнтів критеріям – визначення відносної важливості кожного критерію;
4. Виставлення балів кожній альтернативі за кожним критерієм;
5. Розрахунок зваженої суми балів та визначення переможця.

Для вибору основного фреймворку розробки розглядаються три найбільш поширені технологічні стеки для побудови веб-застосунків корпоративного класу:

Альтернатива А – ASP.NET Core MVC (C#) Фреймворк від компанії Microsoft для розробки веб-застосунків на мові програмування C#. Реалізує архітектурний патерн Model-View-Controller.

Альтернатива Б – Spring Boot (Java) Фреймворк від компанії Pivotal/VMware для розробки веб-застосунків на мові програмування Java. Є одним із найпоширеніших корпоративних фреймворків у світі.

Альтернатива В – Django (Python) Фреймворк для розробки веб-застосунків на мові програмування Python. Відомий принципом «батарейки включені» – наявністю широкого набору вбудованих інструментів.

Для порівняння альтернатив визначено вісім ключових критеріїв, які відображають вимоги до технологічного стеку для розробки промислової інформаційної системи. Кожному критерію призначено ваговий коефіцієнт від 0 до 1, де сума всіх коефіцієнтів дорівнює 1,0.

Таблиця 1.4- Критеріїв оцінювання та вагових коефіцієнтів

№	Критерій	Опис	Ваговий коефіцієнт (W)
1	Продуктивність системи	Швидкість обробки запитів та відповідь сервера	0,15
2	Вбудована система безпеки	Наявність готових інструментів авторизації та аутентифікації	0,20
3	Підтримка ORM	Якість інструментів для роботи з базою даних	0,15
4	Простота розгортання	Складність налаштування та розгортання на сервері	0,10
5	Документація та спільнота	Якість офіційної документації та активність спільноти	0,10
6	Масштабованість	Можливість розширення системи у майбутньому	0,10
7	Підтримка MVC архітектури	Нативна підтримка патерну Model-View-Controller	0,10
8	Досвід розробника	Попередній досвід роботи з технологією	0,10
	Разом		1,00

Оцінювання проводиться за п'ятибальною шкалою, де:

- 5 – відмінно, повністю задовольняє вимогам;
- 4 – добре, задовольняє вимогам з незначними обмеженнями;
- 3 – задовільно, частково задовольняє вимогам;
- 2 – незадовільно, суттєві обмеження;
- 1 – незадовільно, не відповідає вимогам.

Критерій 1 – Продуктивність системи (W = 0,15)

ASP.NET Core MVC (5 балів): ASP.NET Core є одним із найшвидших веб-фреймворків у світі згідно з незалежними бенчмарками TechEmpower. Компіляція у нативний код забезпечує виняткову швидкість обробки запитів.

Spring Boot (4 бали): Java-платформа забезпечує високу продуктивність, однак JVM потребує часу на прогрів після запуску та споживає більше оперативної пам'яті порівняно з .NET.

Django (3 бали): Python як інтерпретована мова поступається компільованим мовам за швидкістю виконання, що є суттєвим при обробці великої кількості паралельних запитів.

Критерій 2 – Вбудована система безпеки (W = 0,20)

ASP.NET Core MVC (5 балів): ASP.NET Core Identity є повноцінною готовою системою авторизації з підтримкою ролей, Claims, шифрування паролів та двофакторної аутентифікації. Повністю інтегрована з Entity Framework Core.

Spring Boot (4 бали): Spring Security є потужним інструментом, однак потребує значно більшого обсягу налаштування порівняно з ASP.NET Core Identity.

Django (4 бали): Django має вбудовану систему аутентифікації, однак управління ролями реалізоване простіше і потребує додаткових бібліотек для складних сценаріїв.

Критерій 3 – Підтримка ORM (W = 0,15)

ASP.NET Core MVC (5 балів): Entity Framework Core є зрілим, добре задокументованим ORM з підтримкою міграцій, LINQ-запитів та широкого спектру баз даних включно з PostgreSQL та MS SQL Server.

Spring Boot (5 балів): Hibernate/JPA є одним із найстаріших та найпотужніших ORM-рішень, з надзвичайно широкими можливостями.

Django (4 бали): Вбудований Django ORM є зручним та простим у використанні, однак поступається Entity Framework Core та Hibernate за гнучкістю складних запитів.

Критерій 4 – Простота розгортання (W = 0,10)

ASP.NET Core MVC (5 балів): Розгортання на Windows Server є тривіальним завданням. Підтримка Docker та Linux також добре реалізована.

Spring Boot (3 бали): Розгортання Java-застосунків потребує налаштування JVM, що ускладнює процес порівняно з .NET.

Django (4 бали): Python-застосунки відносно просто розгортаються, однак потребують налаштування WSGI-сервера (Gunicorn, uWSGI).

Критерій 5 – Документація та спільнота (W = 0,10)

ASP.NET Core MVC (5 балів): Microsoft надає виняткову офіційну документацію на docs.microsoft.com з прикладами коду для кожного сценарію використання.

Spring Boot (5 балів): Spring має одну з найбільших спільнот серед корпоративних фреймворків та відмінну документацію.

Django (4 бали): Django має хорошу документацію та активну спільноту, однак дещо меншу порівняно з двома попередніми альтернативами.

Критерій 6 – Масштабованість (W = 0,10)

ASP.NET Core MVC (5 балів): Архітектура ASP.NET Core підтримує горизонтальне масштабування, мікросервіси та хмарне розгортання в Azure.

Spring Boot (5 балів): Spring є стандартом де-факто для побудови масштабованих корпоративних систем та мікросервісної архітектури.

Django (3 бали): Django добре масштабується для середніх навантажень, однак для дуже високих навантажень потребує додаткових архітектурних рішень.

Критерій 7 – Підтримка MVC архітектури (W = 0,10)

ASP.NET Core MVC (5 балів): MVC є нативним патерном фреймворку, вся архітектура побудована навколо нього.

Spring Boot (4 бали): Spring MVC є добре реалізованим, однак Spring Boot орієнтований більше на REST API, ніж на класичний MVC з серверним рендерингом.

Django (4 бали): Django реалізує власну варіацію MTV (Model-Template-View), яка є концептуально схожою на MVC, але має певні відмінності.

Критерій 8 – Досвід розробника (W = 0,10)

ASP.NET Core MVC (5 балів): Розробник має практичний досвід роботи з C# та ASP.NET Core, що суттєво прискорює розробку та зменшує ризики.

Spring Boot (2 бали): Досвід роботи з Java та Spring є мінімальним, що створює ризик суттєвих затримок у розробці.

Django (3 бали): Базовий досвід роботи з Python є, однак досвіду з Django як фреймворком недостатньо для впевненої розробки.

Розрахунок зваженої суми балів

Зважена сума балів розраховується за формулою:

$S = \Sigma (W_i \times B_i)$

де W_i – ваговий коефіцієнт і-го критерію, B_i – бал альтернативи за і-м критерієм.

Таблиця 1.5 - Розрахунок зваженої суми балів

№	Критерій	W	ASP.NET Core	Spring Boot	Django
1	Продуктивність	0,15	5 × 0,15 = 0,75	4 × 0,15 = 0,60	3 × 0,15 = 0,45
2	Система безпеки	0,20	5 × 0,20 = 1,00	4 × 0,20 = 0,80	4 × 0,20 = 0,80
3	Підтримка ORM	0,15	5 × 0,15 = 0,75	5 × 0,15 = 0,75	4 × 0,15 = 0,60
4	Розгортання	0,10	5 × 0,10 = 0,50	3 × 0,10 = 0,30	4 × 0,10 = 0,40
5	Документація	0,10	5 × 0,10 = 0,50	5 × 0,10 = 0,50	4 × 0,10 = 0,40
6	Масштабованість	0,10	5 × 0,10 = 0,50	5 × 0,10 = 0,50	3 × 0,10 = 0,30
7	MVC архітектура	0,10	5 × 0,10 = 0,50	4 × 0,10 = 0,40	4 × 0,10 = 0,40
8	Досвід розробника	0,10	5 × 0,10 = 0,50	2 × 0,10 = 0,20	3 × 0,10 = 0,30
	Загальна сума	1,00	4,50	4,05	3,65

Результати та висновки

За результатами розрахунків методом експертних оцінок отримано такі підсумкові показники:

- ASP.NET Core MVC – 4,50 бали (переможець)
- Spring Boot – 4,05 бали
- Django – 3,65 бали

ASP.NET Core MVC отримав найвищу зважену оцінку завдяки поєднанню таких факторів: виняткова продуктивність серед веб-

фреймворків, найповніша вбудована система безпеки через ASP.NET Core Identity, зрілий ORM у вигляді Entity Framework Core, простота розгортання на серверній інфраструктурі підприємства та наявний досвід розробника.

Таким чином, метод експертних оцінок підтверджує обґрунтованість вибору ASP.NET Core MVC як основного фреймворку для розробки програмно-методичного комплексу ведення агрегатного журналу.

1.6. Огляд технологій для побудови програмно-методичного комплексу

Після обґрунтування вибору технологічного стеку методом експертних оцінок необхідно детально розглянути кожну з обраних технологій. Розуміння принципів роботи та можливостей кожного інструменту є необхідною умовою для проектування якісної архітектури програмно-методичного комплексу. У даному розділі розглядаються чотири ключові технології, які утворюють основу розроблюваної системи.

1.6.1.ASP.NET Core MVC як основний фреймворк

ASP.NET Core – це кросплатформений, високопродуктивний фреймворк з відкритим вихідним кодом для розробки сучасних хмарних та веб-застосунків, розроблений та підтримуваний компанією Microsoft.

Є повністю переробленою версією класичного ASP.NET Framework, оптимізованою для сучасних умов розробки.

Абревіатура MVC розшифровується як Model-View-Controller – архітектурний патерн, який розділяє застосунок на три незалежні компоненти:

- Model (Модель) – відповідає за дані та бізнес-логіку застосунку. У контексті агрегатного журналу моделями є сутності: Агрегат, Аварія, Запит на закупівлю, Користувач тощо;
- View (В'юшка) – відповідає за відображення даних користувачу. Це HTML-шаблони з динамічними даними, які формують інтерфейс системи;
- Controller (Контролер) – відповідає за обробку запитів користувача, взаємодію з моделлю та передачу даних у в'юшку.

Принцип роботи в контексті проєкту– коли агломератник натискає кнопку «Зафіксувати аварію» у браузері, відбувається такий ланцюжок подій:

1. Браузер надсилає HTTP-запит до сервера;
2. Контролер (AvariiController) отримує запит та перевіряє права доступу користувача;
3. Контролер звертається до Моделі (через Entity Framework Core) для збереження запису в базі даних;
4. Контролер передає оновлені дані у В'юшку (Index.cshtml);
5. В'юшка формує HTML-сторінку та надсилає її назад у браузер користувача.

Ключові переваги для проєкту є:

Razor Pages та Razor синтаксис дозволяють вбудовувати C#-код безпосередньо в HTML-шаблони, що спрощує динамічне формування сторінок журналу.

Dependency Injection вбудований у ядро фреймворку, що забезпечує слабку зв'язаність компонентів системи та спрощує тестування.

Tag Helpers дозволяють будувати форми, посилання та інші HTML-елементи з автоматичною прив'язкою до моделей даних, що суттєво прискорює розробку інтерфейсу.

Актуальна версія для розробки комплексу використовується ASP.NET Core 8.0 – версія з довгостроковою підтримкою (LTS), що гарантує отримання оновлень безпеки та виправлень помилок до листопада 2026 року. Це забезпечує стабільність та надійність системи протягом тривалого терміну експлуатації.

1.6.2.Entity Framework Core та підхід ORM

ORM (Object-Relational Mapping – об'єктно-реляційне відображення) – це технологія програмування, яка забезпечує автоматичне перетворення між об'єктами мови програмування та записами реляційної бази даних. ORM дозволяє розробнику працювати з базою даних використовуючи звичні конструкції мови C# без написання SQL-запитів вручну.

Без ORM для збереження запису про аварію розробнику довелося б писати:

```
INSERT INTO Avarii (AggregatId, Opys, DataZupynky, Status)
VALUES (@AggregatId, @Opys, @DataZupynky, 'Nova')
```

З Entity Framework Core той самий результат досягається кодом на C#:

```
var avaria = new Avaria { AggregatId = id, Opys = opys, Status = "Nova" };
_context.Avarii.Add(avaria);
await _context.SaveChangesAsync();
```

Entity Framework Core (EF Core) – це офіційний ORM від Microsoft для платформи .NET. Є легкою, розширюваною та кросплатформеною версією класичного Entity Framework, повністю переробленою для ASP.NET Core.

DbContext є центральним класом EF Core – він представляє сесію роботи з базою даних та є точкою доступу до всіх таблиць. У проєкті клас ApplicationDbContext успадковує IdentityDbContext та містить DbSet для кожної сутності системи:

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    public DbSet<Agregat> Agregaty { get; set; }
    public DbSet<Avaria> Avarii { get; set; }
    public DbSet<ZapytZakupivli> ZapytyZakupivli { get; set; }
}
```

Code First підхід означає, що розробник описує структуру бази даних через класи C#, а EF Core автоматично створює відповідні таблиці. Це дозволяє зберігати структуру бази даних у вигляді коду та контролювати її через систему версіонування.

Міграції – це механізм керування змінами структури бази даних. Кожна зміна моделі (додавання поля, нова таблиця) фіксується як

окрема міграція, яку можна застосувати або відкотити. Це забезпечує безпечне оновлення бази даних на продуктивному сервері без втрати даних.

LINQ-запити дозволяють фільтрувати, сортувати та групувати дані з бази використовуючи синтаксис C#:

EF Core підтримує широкий спектр реляційних баз даних через систему провайдерів, що забезпечує гнучкість у виборі СУБД без зміни коду застосунку.

1.6.3. PostgreSQL як реляційна СУБД.

PostgreSQL – це потужна об'єктно-реляційна система управління базами даних з відкритим вихідним кодом, розробка якої ведеться з 1986 року. PostgreSQL відома своєю надійністю, розширюваністю та відповідністю стандартам SQL. Є однією з найпопулярніших СУБД у світі для корпоративних застосунків.

Ключова перевага PostgreSQL перед конкурентами полягає у поєднанні безкоштовного розповсюдження з функціональністю корпоративного рівня, яка в комерційних СУБД (Oracle, MS SQL Server) коштує значних коштів.

Структура бази даних проєкту

У рамках програмно-методичного комплексу база даних PostgreSQL містить такі основні таблиці:

Таблиця Agregaty зберігає реєстр усього обладнання цеху:

- Id – унікальний ідентифікатор агрегату;
- Nazva – назва агрегату;
- Dilnytsia – дільниця розташування;

- Status – поточний статус (працює/зупинений/на ТО).

Таблиця Avarii зберігає всі події аварійних зупинок:

- Id – унікальний ідентифікатор запису;
- AggregatId – зовнішній ключ на таблицю агрегатів;
- Opys – текстовий опис несправності від оператора;
- DataZupynky – дата та час фіксації зупинки;
- Status – поточний статус заявки;
- VykonavetsId – зовнішній ключ на таблицю користувачів;
- DataVykonannia – дата та час закриття заявки.

Таблиця ZapytyZakupivli зберігає запити на придбання запчастин:

- Id – унікальний ідентифікатор запиту;
- AvariaId – зовнішній ключ на таблицю аварій;
- OpysZapytu – найменування необхідної запчастини;
- Status – статус розгляду керівником;
- KomentarKerivnyka – причина відмови або коментар.

Відкритий вихідний код. Відсутність ліцензійних витрат є важливим фактором для промислових підприємств, де кожна стаття витрат на IT потребує обґрунтування.

Розширені типи даних. PostgreSQL підтримує зберігання JSON, масивів та інших складних типів даних, що може бути корисним для зберігання технічних характеристик агрегатів.

Надійність та довговічність. PostgreSQL використовується у критично важливих системах банків, телекомунікаційних компаній та державних установ, що підтверджує її придатність для відповідальних виробничих застосунків.

1.6.4.ASP.NET Core Identity як система розмежування доступу

ASP.NET Core Identity – це готова система членства (membership system), яка додає функціональність входу в систему до застосунків ASP.NET Core. Identity керує користувачами, паролями, даними профілю, ролями, Claims, токенами та підтвердженням електронної пошти.

Ключова перевага Identity полягає в тому, що розробнику не потрібно реалізовувати безпеку з нуля – всі критично важливі механізми захисту вже реалізовані та протестовані командою Microsoft.

Ключові компоненти Identity у проєкті є:

1. ApplicationUser – розширений клас користувача, який успадковує базовий IdentityUser та доповнений полями, специфічними для проєкту:

```
public class ApplicationUser : IdentityUser
{  public string FullName { get; set; }      // Повне ім'я працівника
   public string Dilnytsia { get; set; }     // Дільниця призначення
   public DateTime DateRegistered { get; set; } // Дата реєстрації
}
```

2. Ролі (Roles) реалізують чотирирівневу модель доступу системи. При першому запуску застосунок автоматично створює чотири ролі: Operator, Technician, Manager, Admin. Кожному користувачу призначається рівно одна роль, яка визначає його права в системі.

3. Claims – це твердження про користувача, які зберігаються у його токені аутентифікації. У проєкті Claims використовуються для автоматичної фіксації виконавця ремонту:

```
// Отримати ім'я поточного користувача з Claims
var vykonavets = User.FindFirst(ClaimTypes.Name)?.Value;
avaria.VykonavetsName = vykonavets;
avaria.DataVykonannia = DateTime.Now;
```

Саме завдяки Claims система автоматично записує ім'я слюсаря при натисканні кнопки «ВИКОНАНО» – без можливості підробки або помилки.

4. Атрибут [Authorize] забезпечує захист контролерів та окремих дій від несанкціонованого доступу. Система автоматично перевіряє роль користувача перед виконанням будь-якої дії:

```
[Authorize(Roles = "Manager,Admin")]
public async Task<IActionResult> SchvalytyZapyt(int id)
{  // Цей метод доступний лише керівнику та адміністратору}
```

5. Режим LOCKED. Особливою функцією системи безпеки є режим LOCKED – стан системи до повної авторизації користувача. У цьому режимі система приховує всю конфіденційну виробничу інформацію та відображає лише сторінку входу. Це реалізується через глобальне налаштування авторизації у файлі конфігурації застосунку:

```
builder.Services.AddAuthorization(options =>
{   options.FallbackPolicy = new AuthorizationPolicyBuilder()
        .RequireAuthenticatedUser()
        .Build();
});
```

Така конфігурація гарантує, що жодна сторінка системи не буде доступна без попередньої аутентифікації, що є критично важливою вимогою для захисту конфіденційних виробничих даних підприємства.

Чотири розглянуті технології утворюють єдину злагоджену архітектуру програмно-методичного комплексу, де кожен компонент виконує чітко визначену роль:

Таблиця 1.6 - Загальна архітектура технологічного стеку

Рівень	Технологія	Відповідальність
Презентаційний	ASP.NET Core MVC + Razor	Інтерфейс користувача, обробка запитів
Бізнес-логіки	C# + Controllers	Правила роботи системи, ланцюжок взаємодії
Безпеки	ASP.NET Core Identity	Аутентифікація, авторизація, ролі
Даних	Entity Framework Core	ORM, міграції, LINQ-запити
Зберігання	PostgreSQL	Надійне зберігання всіх даних системи

Така багаторівнева архітектура забезпечує чітке розділення відповідальності між компонентами системи, що спрощує її підтримку, тестування та подальший розвиток.

2. РОЗРОБКА МАТЕМАТИЧНОЇ МОДЕЛІ ПРОГРАМНО-МЕТОДИЧНОГО КОМПЛЕКСУ ДЛЯ ВЕДЕННЯ АГРЕГАТНОГО ЖУРНАЛУ

2.1. Розробка структурно-функціональної моделі бізнес-процесу на основі SADT-технології для опису функцій служби технічного обслуговування агрегатів

Поняття SADT-технології та структурно-функціонального моделювання

При проектуванні складних інформаційних систем одним із ключових етапів є формалізований опис бізнес-процесів, які система має автоматизувати. Без чіткого розуміння того, як процес працює в реальності, неможливо створити програмний комплекс, який би дійсно вирішував практичні задачі підприємства.

SADT (Structured Analysis and Design Technique – техніка структурного аналізу та проектування) – це методологія системного аналізу та функціонального моделювання, розроблена Дугласом Россом у 1969 році та стандартизована як нотація IDEF0. Методологія дозволяє описати будь-який бізнес-процес у вигляді ієрархічної системи функціональних блоків із чітко визначеними входами, виходами, керуванням та механізмами.

Кожен функціональний блок у нотації IDEF0 описується чотирма типами стрілок:

- Вхід (Input) – ліва сторона блоку: дані або матеріали, які перетворюються у процесі виконання функції;

- Вихід (Output) – права сторона блоку: результат виконання функції;
- Керування (Control) – верхня сторона блоку: правила, норми та обмеження, які регулюють виконання функції;
- Механізм (Mechanism) – нижня сторона блоку: ресурси, що використовуються для виконання функції (персонал, система, обладнання).

Контекстна діаграма є найвищим рівнем абстракції та описує систему як єдиний функціональний блок у взаємодії із зовнішнім середовищем.

Назва функції A-0: «Управляти технічним обслуговуванням агрегатів цеху»

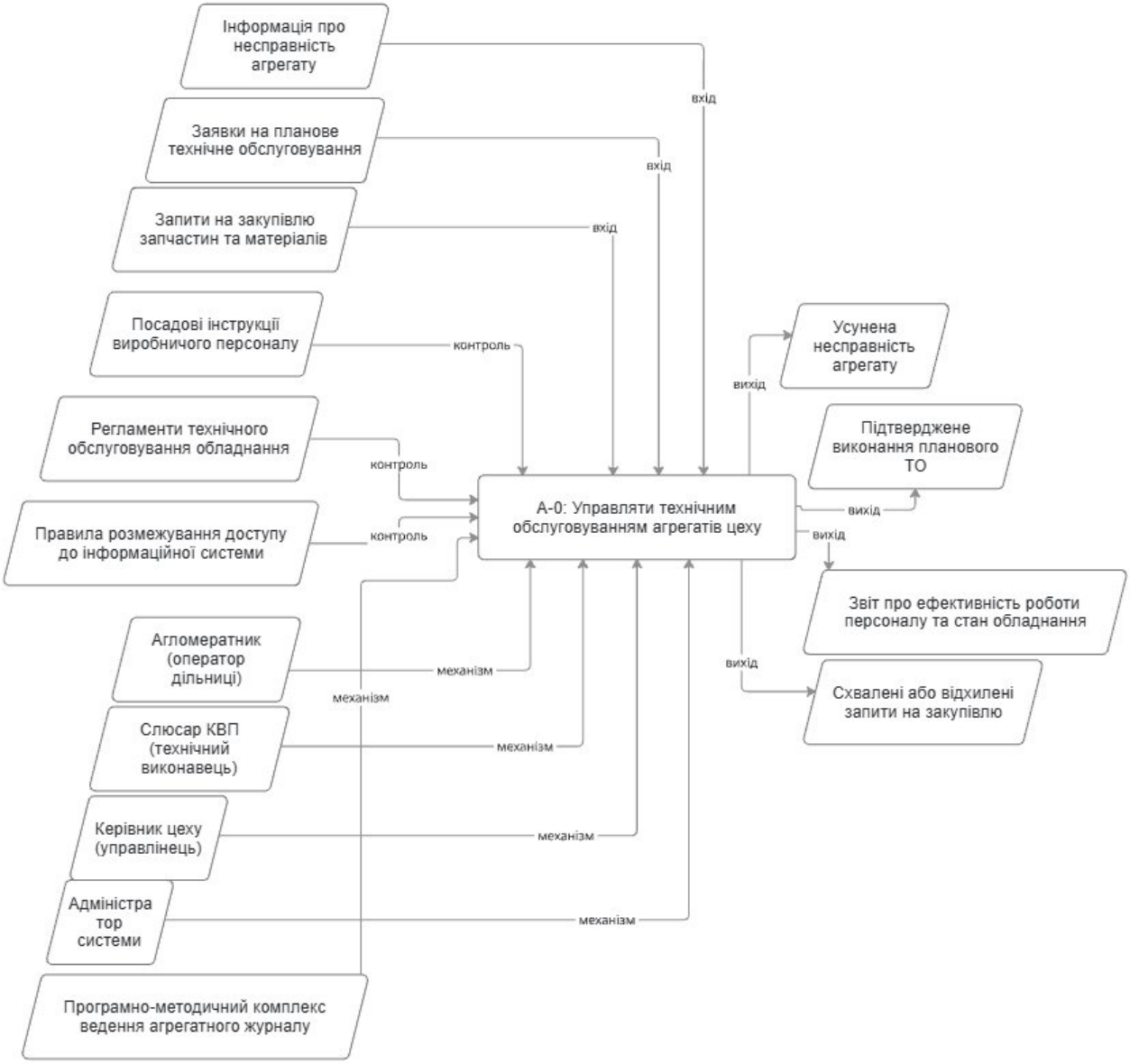


Рисунок 2.1 - Контекстна діаграма A-0

Блок A1 – «Реєструвати аварійну зупинку агрегату»

Виконавець: Агломератник (Operator)

Опис функції: Агломератник, виявивши несправність агрегату або його аварійну зупинку, виконує первинну реєстрацію події в програмно-методичному комплексі. Він заповнює форму реєстрації, вказуючи найменування агрегату, характер несправності та фіксуючи факт зупинки виробничого процесу.

Рисунок 2.2 - Реєструвати аварійну зупинку агрегату

Блок А2 – «Виконувати ремонтні роботи та управляти запитами на закупівлю»

Виконавець: Слюсар КВП (Instrument Technician)

Опис функції: Слюсар КВП отримує інформацію про нову аварію через список активних заявок у системі. Він приступає до діагностики та усунення несправності. У разі відсутності необхідної запчастини або матеріалу слюсар формує електронний запит на закупівлю із зазначенням найменування деталі та обґрунтуванням потреби. Після успішного усунення несправності слюсар закриває заявку в системі, яка автоматично фіксує його ідентифікаційні дані та точний час виконання.

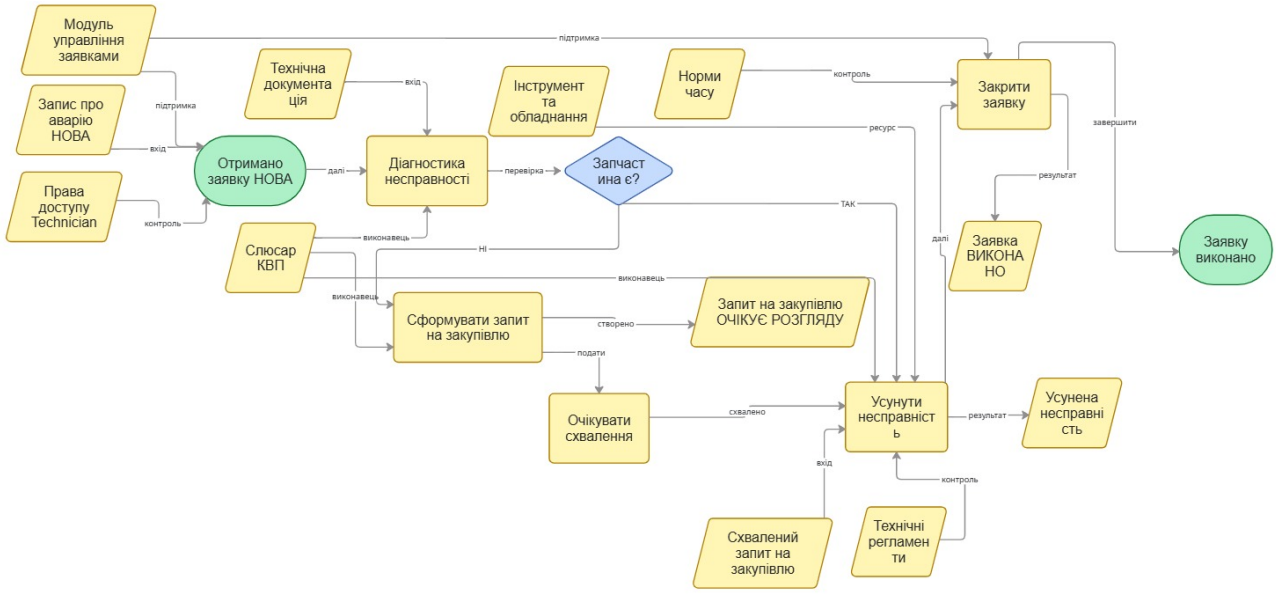


Рисунок 2.3 - Виконувати ремонтні роботи та управляти запитами на закупівлі

Блок А3 – «Розглядати запити та контролювати виконання робіт»

Виконавець: Керівник (Manager)

Опис функції: Керівник цеху виконує дві паралельні функції управлінського характеру. По-перше, він переглядає запити на закупівлю запчастин, сформовані слюсарями, та приймає рішення про їх схвалення або відхилення з обов'язковим зазначенням причини відмови. По-друге, керівник здійснює загальний моніторинг стану всього обладнання цеху через дашборд системи, відстежує дотримання нормативів часу на усунення несправностей та аналізує статистику аварійності кожного агрегату.

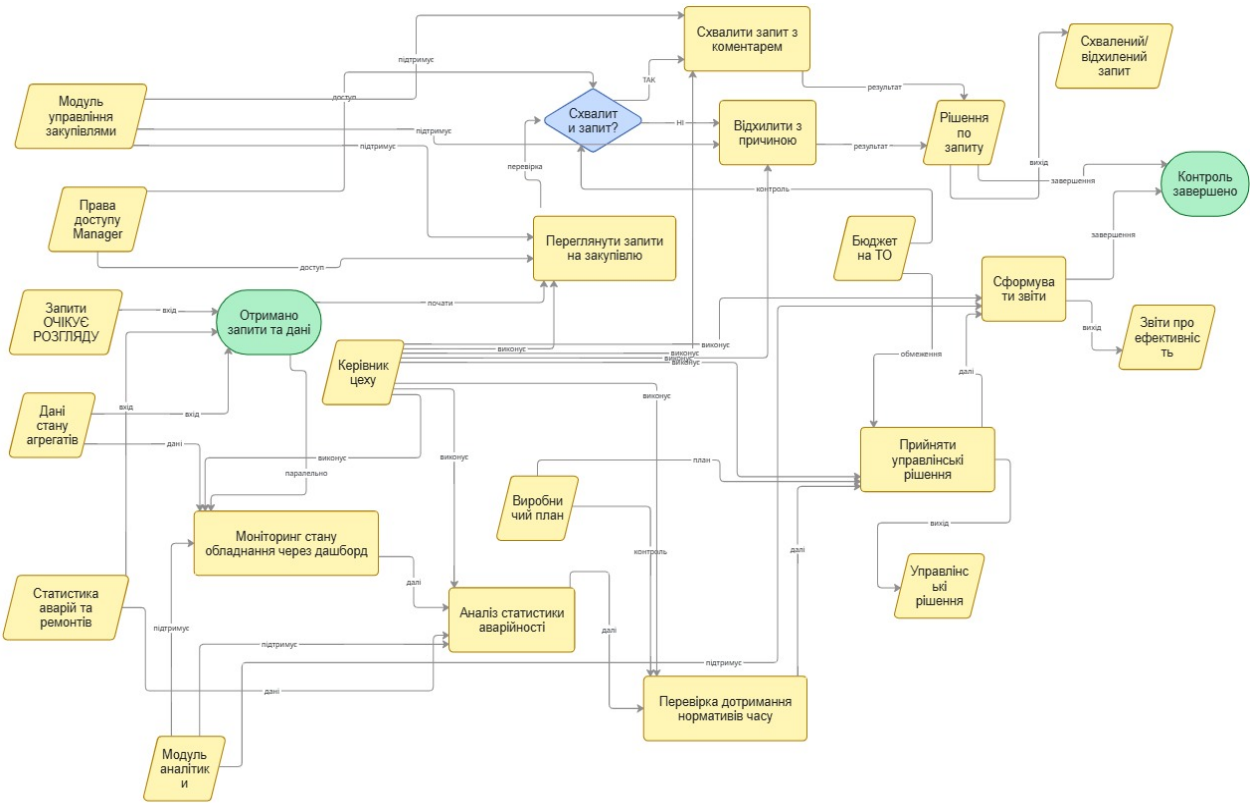


Рисунок 2.4 - Розглядати запити та контролювати виконання робіт

Блок А4 – «Адмініструвати систему та управляти персоналом»

Виконавець: Адміністратор (Admin)

Опис функції: Адміністратор забезпечує технічне функціонування програмно-методичного комплексу та підтримує актуальність його налаштувань. Він створює облікові записи для нових працівників, призначає їм відповідні ролі згідно з посадою, деактивує облікові записи звільнених співробітників. Також адміністратор підтримує актуальний реєстр агрегатів цеху, додаючи нове обладнання та видаляючи списане. У разі виявлення помилкових або дублюючих записів адміністратор має повноваження їх виправити або видалити.

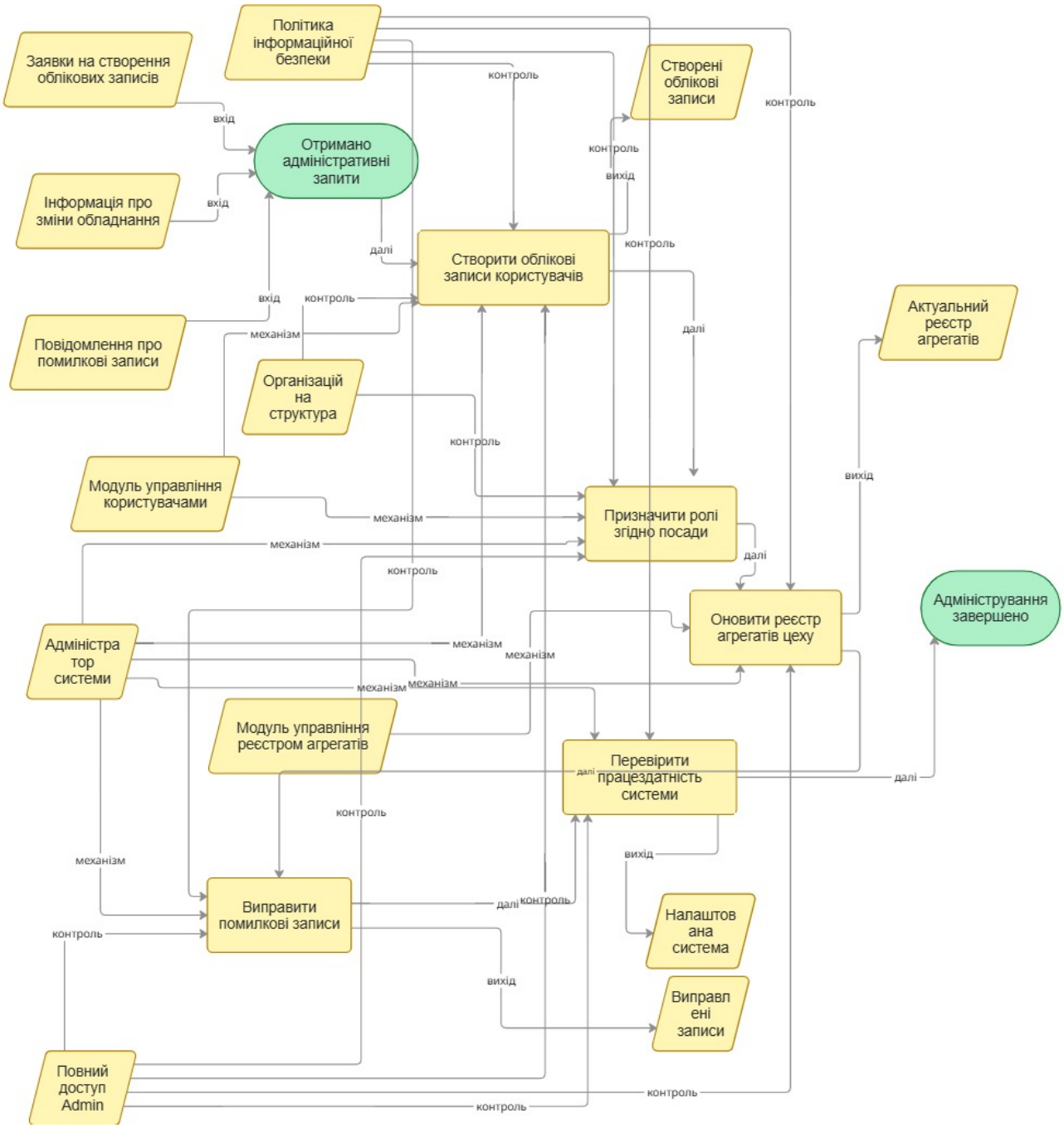


Рисунок 2.5 - Адмініструвати систему та управляти персоналом

Таблиця 2.1 - Зведена таблиця функціональних блоків діаграми A0

Блок	Назва функції	Виконавець	Вхід	Вихід
A1	Реєструвати аварійну зупинку	Агломератник	Факт несправності	Запис «НОВА»
A2	Виконувати ремонтні роботи	Слюсар КВП	Запис «НОВА»	Запис «ВИКОНАНО»
A3	Розглядати запити та контролювати	Керівник	Запити на закупівлю	Рішення керівника
A4	Адмініструвати систему	Адміністратор	Заявки на зміни	Налаштована система

Взаємозв'язки між функціональними блоками. Функціональні блоки діаграми A0 не є незалежними – між ними існують чіткі інформаційні потоки, які утворюють ланцюжок взаємодії персоналу:

Зв'язок A1 → A2: Вихід блоку A1 (зареєстрований запис зі статусом «НОВА») є входом для блоку A2. Слюсар не може розпочати роботу без попередньої реєстрації аварії оператором.

Зв'язок A2 → A3: Вихід блоку A2 (запит на закупівлю зі статусом «ОЧІКУЄ») є входом для блоку A3. Керівник розглядає лише ті запити, які сформовані слюсарем у процесі виконання конкретного ремонту.

Зв'язок A3 → A2: Вихід блоку A3 (схвалений запит на закупівлю) повертається як вхід до блоку A2. Після отримання схвалення слюсар може продовжити ремонт, отримавши необхідну запчастину.

Зв'язок A4 → A1, A2, A3: Блок A4 забезпечує функціонування всіх інших блоків через підтримку облікових записів користувачів та реєстру агрегатів.

2.2. Розробка діаграми станів записів агрегатного журналу

Поняття діаграми станів у проектуванні інформаційних систем

При проектуванні інформаційних систем, які керують складними процесами з багатьма учасниками, важливим інструментом формалізації є діаграма станів (State Diagram) – графічна модель, що описує всі можливі стани об'єкта системи та умови переходу між ними.

Діаграма станів є частиною уніфікованої мови моделювання UML (Unified Modeling Language) і використовується для опису динамічної поведінки об'єктів системи протягом їхнього життєвого циклу. Вона дозволяє чітко визначити:

- всі можливі стани, в яких може перебувати об'єкт;
- події або дії, що спричиняють перехід між станами;
- умови, за яких такий перехід є можливим;
- початковий та кінцевий стани об'єкта.

У контексті програмно-методичного комплексу ведення агрегатного журналу ключовим об'єктом, поведінку якого необхідно формалізувати, є заявка на ремонт – центральна сутність системи, яка проходить через декілька станів від моменту реєстрації аварії до її повного вирішення.

У розроблюваному комплексі діаграми станів будуються для двох ключових об'єктів:

Об'єкт 1 – Заявка на ремонт (Avaria) Центральний об'єкт системи, що відображає повний цикл усунення аварійної зупинки агрегату від реєстрації до закриття.

Об'єкт 2 – Запит на закупівлю (ZapytZakupivli) Залежний об'єкт, що відображає процес погодження придбання необхідних запчастин між слюсарем та керівником.

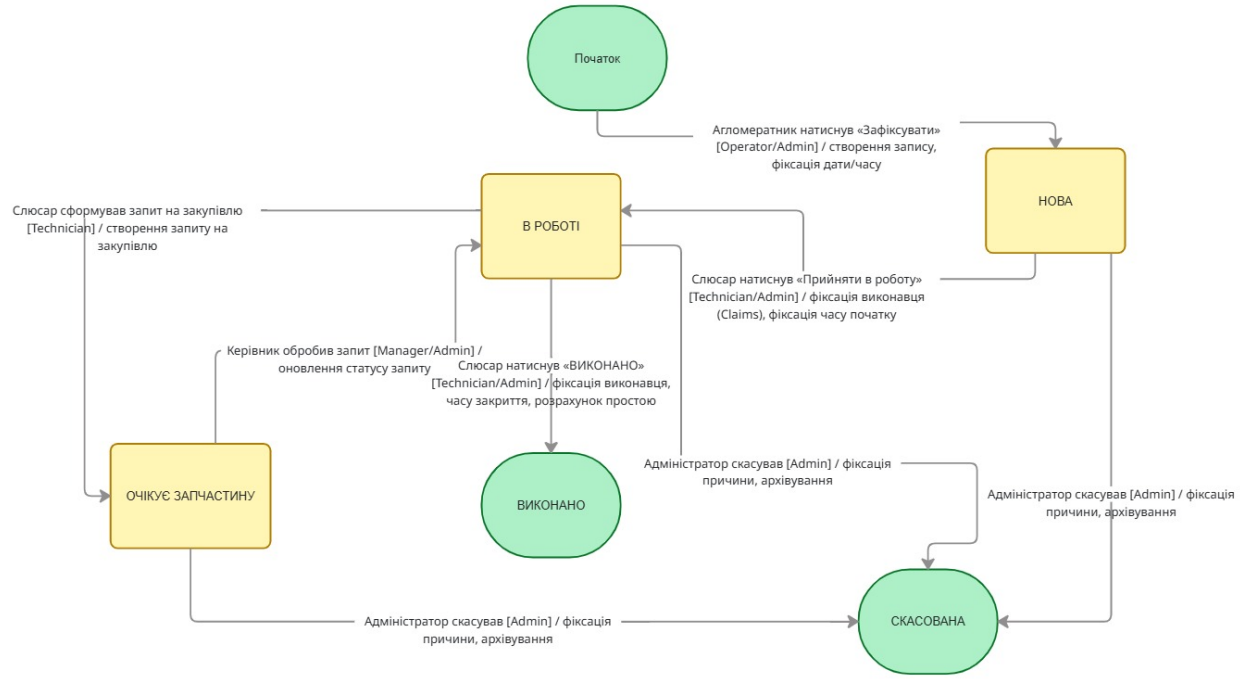


Рисунок 2.6 - Діаграма станів об'єкта «Заявка на ремонт»

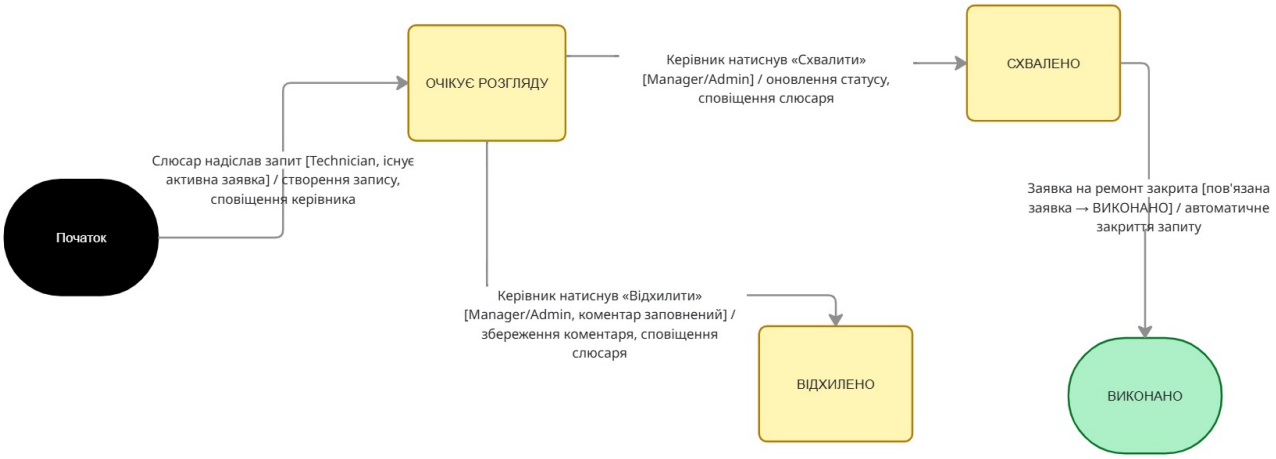


Рисунок 2.7 - Діаграма станів об'єкта «Запит на закупівлю»

Таблиця 2.2 - Зведена таблиця станів та переходів системи

Об'єкт	Стан	Ініціатор переходу	Наступний стан
Заявка	НОВА	Агломератник (реєстрація)	–
Заявка	НОВА → В РОБОТІ	Слюсар (прийняти)	В РОБОТІ
Заявка	В РОБОТІ → ОЧІКУЄ	Слюсар (запит на закупівлю)	ОЧІКУЄ ЗАПЧАСТИНУ
Заявка	ОЧІКУЄ → В РОБОТІ	Керівник (рішення по запиту)	В РОБОТІ
Заявка	В РОБОТІ → ВИКОНАНО	Слюсар (виконано)	ВИКОНАНО
Заявка	НОВА/В РОБОТІ → СКАСОВАНА	Адміністратор	СКАСОВАНА
Запит	ОЧІКУЄ	Слюсар (формування)	–
Запит	ОЧІКУЄ → СХВАЛЕНО	Керівник (схвалити)	СХВАЛЕНО
Запит	ОЧІКУЄ → ВІДХИЛЕНО	Керівник (відхилити)	ВІДХИЛЕНО
Запит	СХВАЛЕНО → ВИКОНАНО	Система (автоматично)	ВИКОНАНО

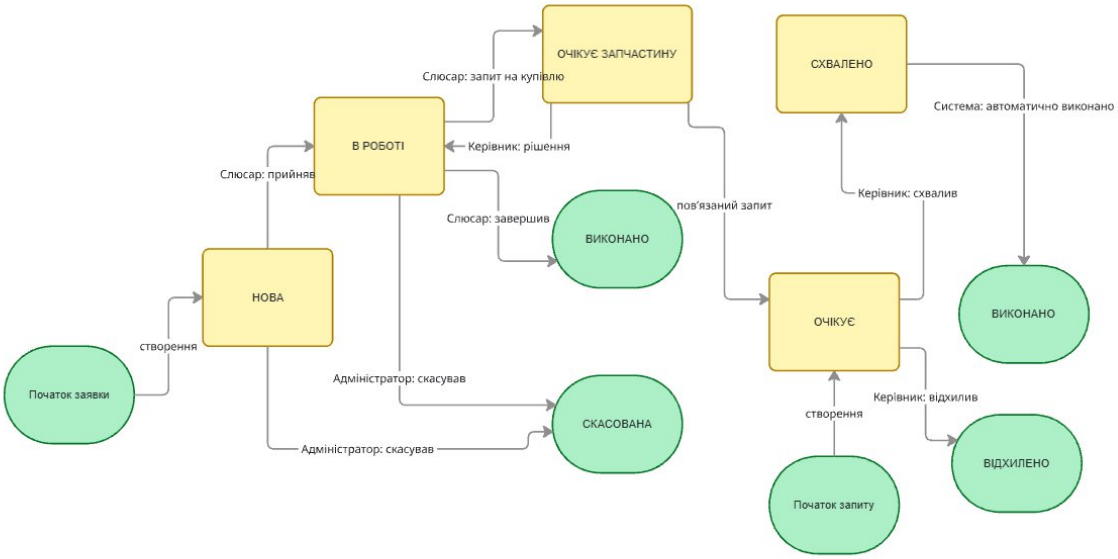


Рисунок 2.8 - Діаграми станів

Розроблена діаграма станів формалізує поведінку двох ключових об'єктів програмно-методичного комплексу – заявки на ремонт та запиту на закупівлю – протягом їхнього повного життєвого циклу. Аналіз діаграми дозволяє зробити такі висновки:

По-перше, система реалізує чіткий лінійний ланцюжок станів з однією точкою розгалуження (стан «ОЧІКУЄ ЗАПЧАСТИНУ»), що забезпечує передбачуваність та прозорість процесу для всіх учасників.

По-друге, кожен перехід між станами ініціюється конкретним учасником з відповідною роллю, що унеможливлює несанкціоноване втручання у процес.

По-третє, найважливіші переходи – прийняття в роботу та закриття заявки – супроводжуються автоматичною фіксацією даних виконавця та часу, що забезпечує об'єктивність та достовірність записів агрегатного журналу.

2.3. Проектування бази даних системи

Ефективність, швидкість відгуку та надійність програмно-методичного комплексу «Електронний агрегатний журнал» безпосередньо залежать від якості проектування його інформаційного забезпечення. База даних промислової системи повинна не лише забезпечувати цілісність та несуперечність інформації, а й гарантувати високу швидкість виконання транзакцій в умовах безперервного багатозмінного виробництва цеху випалу.

Для реалізації комплексу було обрано реляційну систему управління базами даних (СУБД) PostgreSQL. Цей вибір зумовлений її високою продуктивністю, повною відповідністю принципам ACID (Atomicity, Consistency, Isolation, Durability), безкоштовною ліцензією (Open Source) та розвиненими інструментами для роботи з часовими мітками й індексами, що критично важливо для MES- та SCADA-орієнтованих систем. Процес проектування бази даних поділявся на два послідовних етапи: концептуальний (інфологічний) та логічний (даталогічний) з обов'язковою верифікацією моделей на відповідність правилам нормалізації.

2.3.1. Розробка інфологічної (ER) моделі даних

Інфологічне (концептуальне) проектування є етапом створення семантичної моделі предметної області, яка не залежить від конкретної програмної реалізації та обраної СУБД. Головна мета цього етапу – виділити ключові сутності (об'єкти виробництва), визначити їхні

унікальні властивості (атрибути) та встановити логічні взаємозв'язки між ними.

Для забезпечення функціонування електронного агрегатного журналу в межах предметної області було виокремлено дві стрижневі бізнес-сутності:

1. «Агрегат» (Aggregate): Описує матеріальні активи підприємства (технологічне обладнання дільниці випалу окатків, димососи, конвеєри, грохоти тощо).

2. «Запис стану» (AggregateStatusLog): Описує життєвий цикл інцидентів, аварійних зупинок, дефектів та ремонтних робіт, що відбуваються з конкретним обладнанням.

Аналіз атрибутивного складу сутностей:

- Сутність «Агрегат» містить ідентифікаційні та технологічні характеристики: унікальний сурогатний ключ, текстову назву обладнання, інвентарний номер (як головний бізнес-ідентифікатор для інтеграції з бухгалтерією та ERP-системою підприємства), а також дату останнього регламентного технічного обслуговування (ТО).

- Сутність «Запис стану» акумулює параметри інциденту: опис несправності, часові мітки реєстрації аварії оператором та фактичного її усунення слюсарем КВП, прізвище виконавця, поточний логічний статус (працює, вийшов з ладу, очікує запчастин) та поля для взаємодії з керівництвом (текст запиту на деталі, резолюція майстра).

Між сутностями «Агрегат» та «Запис стану» встановлено бінарний зв'язок типу «один-до-багатьох» (1:M). Семантика цього зв'язку відображає реальні виробничі процеси: одна одиниця обладнання (агрегат) протягом свого життєвого циклу може мати безліч історичних записів про ремонти та поломки. Водночас кожен конкретний рядок у журналі інцидентів може стосуватися лише одного чітко визначеного

агрегату. Зв'язок є обов'язковим з боку сутності «Запис стану» (запис не може існувати без прив'язки до реального обладнання).

2.3.2. Даталогічна модель та нормалізація відношень

Даталогічне проектування є процесом відображення концептуальної ER-моделі у логічну схему, орієнтовану на обрану реляційну модель даних СУБД PostgreSQL. На цьому етапі сутності перетворюються на фізичні таблиці, атрибути – на стовпчики з конкретними типами даних, а зв'язки реалізуються за допомогою механізму первинних (Primary Keys) та зовнішніх (Foreign Keys) ключів.

Для фізичної реалізації бази даних схему було декомпоновано на дві основні таблиці: Aggregates та AggregateStatusLogs. Детальний опис структур полів, їх типів та обмежень наведено у таблицях 2.1 та 2.2.

Таблиця 2.3 - Специфікація структури таблиці Aggregates (Довідник обладнання)

Назва поля	Тип даних у PostgreSQL	Обмеження цілісності	Опис призначення поля
Id	integer	SERIAL, PRIMARY KEY	Унікальний ідентифікатор (сурогатний ключ)
Name	varchar(100)	NOT NULL	Технологічне найменування агрегату
InventoryNumber	varchar(50)	NOT NULL, UNIQUE	Інвентарний номер обладнання
LastMaintenanceDate	timestamp	NULL	Дата та час останнього технічного обслуговування

Таблиця 2.4 - Специфікація структури таблиці AggregateStatusLogs (Журнал інцидентів)

Назва поля	Тип даних у PostgreSQL	Обмеження цілісності	Опис призначення поля
Id	integer	SERIAL, PRIMARY KEY	Унікальний ідентифікатор запису журналу
AggregateId	integer	NOT NULL, FOREIGN KEY	Зв'язок із таблицею Aggregates
Status	integer	NOT NULL	Код стану (0 – Працює, 1 – Очікує, 2 – Аварія)
Description	varchar(1000)	NOT NULL	Опис поломки, внесений оператором
Comment	varchar(500)	NULL	Поточний коментар до запису
ReportedAt	timestamp	NOT NULL	Час реєстрації інциденту оператором (UTC)
RepairStartedAt	timestamp	NULL	Час початку ремонту (архівне поле)
ResolvedAt	timestamp	NULL	Час фактичного закриття заявки (UTC)
ChangeDate	timestamp	NOT NULL	Дата останньої модифікації рядка (UTC)
IsResolved	boolean	NOT NULL, DEFAULT false	Прапорець успішного виконання робіт
IsWaitingForParts	boolean	NOT NULL, DEFAULT false	Прапорець зупинки ремонту через брак деталей
CompletedBy	varchar(100)	NULL	Прізвище слюсаря, який виконав ремонт
UserName	varchar(100)	NOT NULL	Логін користувача, який створив запис
RequestDetailText	varchar(500)	NULL	Текст запиту на необхідні запчастини
ManagerComment	varchar(500)	NULL	Резолюція чи коментар керівника
IsApproved	boolean	NULL	Статус погодження запиту менеджером

Для запобігання виникненню аномалій вставки (Insertion Anomalies), видалення (Deletion Anomalies) та модифікації (Update Anomalies) даних, спроектована схема бази даних була піддана процедурі нормалізації за Е. Коддом. Проаналізуємо відповідність таблиць вимогам трьох перших нормальних форм:

1. Перша нормальна форма (1NF): Відношення перебуває в 1NF, якщо всі його атрибути є атомарними (неділимими), а таблиця не містить повторюваних груп чи масивів даних. Спроектовані таблиці Aggregates та AggregateStatusLogs повністю задовольняють цю вимогу. Кожна клітинка на перетині рядка та стовпчика містить лише одне скалярне значення (число, рядок, логічний прапорець або штамп часу).

2. Друга нормальна форма (2NF): Відношення перебуває в 2NF, якщо воно задовольняє вимогам 1NF і всі його неключові атрибути мають повну функціональну залежність від усього первинного ключа (а не від його частини). Оскільки в обох таблицях як первинні ключі використовуються прості сурогатні ідентифікатори Id (що складаються з одного поля типу SERIAL), часткова залежність від ключа є математично неможливою. Усі описові поля (наприклад, Description або InventoryNumber) залежать від свого Id повністю.

3. Третя нормальна форма (3NF): Відношення перебуває в 3NF, якщо воно задовольняє вимогам 2NF і жоден із неключових атрибутів не має транзитивної залежності від первинного ключа (тобто неключові атрибути не повинні залежати від інших неключових атрибутів).

~ У таблиці Aggregates поля Name, InventoryNumber та LastMaintenanceDate залежать прямо й виключно від Id. Між самими неключовими полями функціональних залежностей немає.

~ У таблиці AggregateStatusLogs кожне поле описує виключно параметри конкретного інциденту. Прізвище виконавця CompletedBy,

опис Description та часові мітки ResolvedAt є унікальними характеристиками логу під конкретним Id. Наявність зовнішнього ключа AggregateId дозволяє посилатися на дані обладнання, не дублюючи його текстову назву чи характеристики у кожному рядку журналу. Якщо назва агрегату зміниться, вона оновиться в одному місці – в таблиці Aggregates, що унеможливлює аномалії модифікації.

Таким чином, схема бази даних програмно-методичного комплексу приведена до третьої нормальної форми (3NF). Це гарантує мінімальний рівень надлишковості даних, високу швидкість індексації та надійний захист цілісності інформації під час ведення оперативного обліку технічного стану промислових агрегатів.

2.4. Проектування архітектури програмного забезпечення (UML діаграми класів та прецедентів)

Для візуалізації архітектурних рішень, структури програмного коду та моделювання поведінки системи на етапі проектування використовується Уніфікована мова моделювання (UML – Unified Modeling Language). Використання UML-діаграм дозволяє формалізувати вимоги до програмно-методичного комплексу, чітко розмежувати зони відповідальності між модулями та визначити сценарії взаємодії користувачів із системою до початку безпосереднього написання програмного коду.

Для комплексного опису розроблюваного веб-додатка «Електронний агрегатний журнал» було побудовано дві базові моделі: діаграму прецедентів (Use Case Diagram), яка описує функціональні вимоги та поведінку системи з точки зору кінцевих користувачів, та

діаграму класів (Class Diagram), що відображає статичну структуру об'єктно-орієнтованого коду на базі патерну MVC.

Моделювання поведінки системи (UML Діаграма прецедентів)

Діаграма прецедентів ілюструє взаємодію зовнішніх сутностей (акторів) із функціональними можливостями (прецедентами) програмного комплексу. Відповідно до розробленої рольової моделі безпеки на базі ASP.NET Core Identity, у системі виділено чотирьох основних акторів, кожен з яких має власний набір дозволених сценаріїв використання (Use Cases):

1. Актор «Оператор / Агломератник» (Operator): * Прецеденти: Перегляд загальної дошки стану обладнання; Створення нового запису про аварію (введення опису несправності та вибір агрегату з довідника).

2. Актор «Слюсар КВП / Електрик» (Instrument Technician): * Прецеденти: Перегляд активних аварій (червона зона); Фіксація усунення несправності (виклик атомарної операції закриття заявки, що автоматично фіксує поточний час UTC); Формування запиту на запасні деталі (виклик модального вікна для замовлення ТМЦ).

3. Актор «Керівник / Майстер» (Manager): * Прецеденти: Перегляд архіву виконаних робіт; Перегляд дошки запитів на матеріали (Manager Board); Погодження або відхилення запиту на видачу запчастин зі складу із залишенням відповідної резолюції.

4. Актор «Адміністратор» (Admin): * Прецеденти: Повне управління довідником агрегатів (додавання, редагування, видалення обладнання); Управління обліковими записами та ролями користувачів.

Крім того, на діаграмі виділено базовий прецедент «Автентифікація в системі» (Login), який пов'язаний з усіма іншими прецедентами відношенням включення (<<include>>), оскільки жодна дія в електронному журналі неможлива без попередньої ідентифікації користувача.

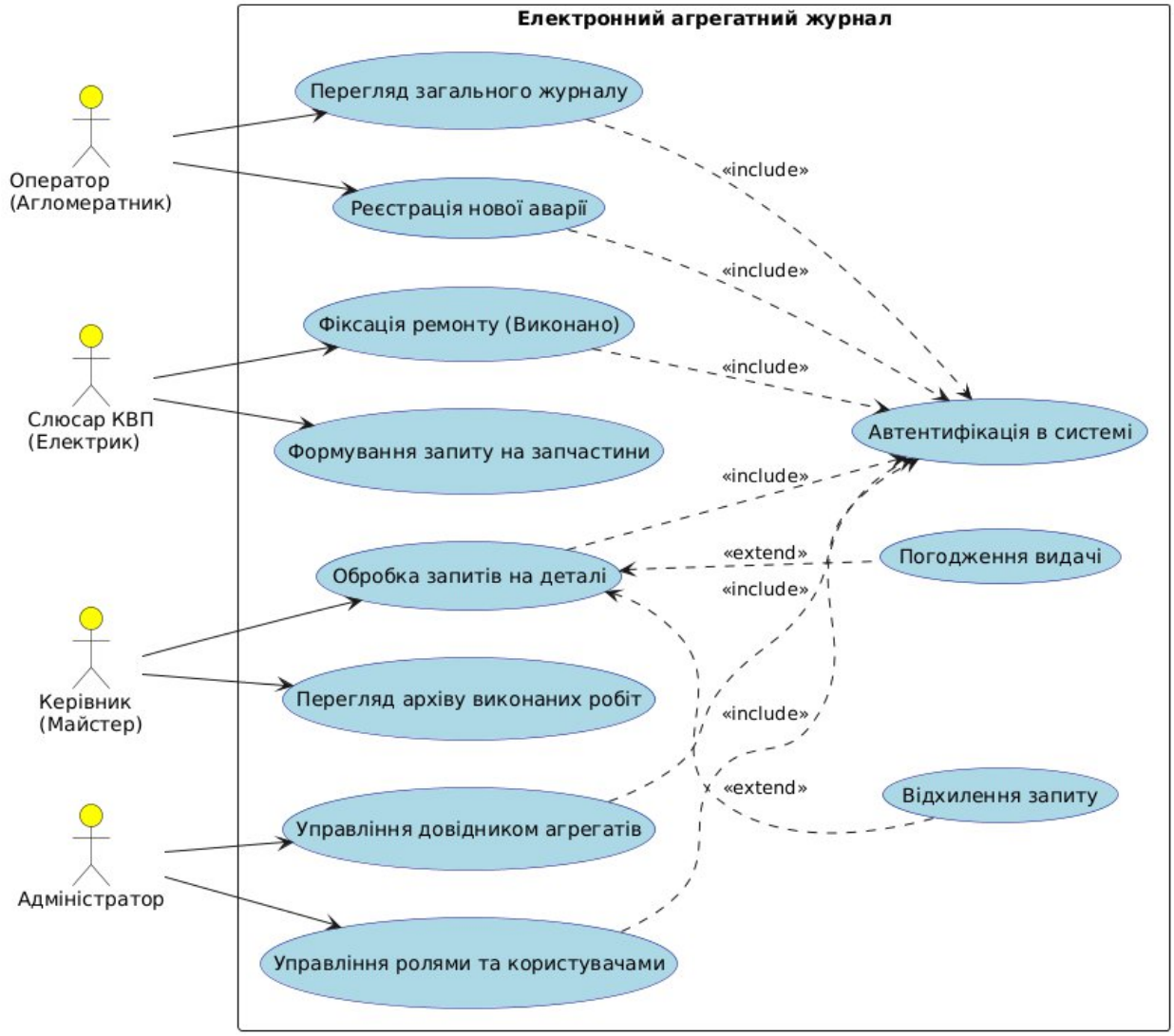


Рисунок 2.9 - UML діаграма прецедентів (Use Case Diagram)

Діаграма класів відображає логічну структуру системи на рівні вихідного коду C#. Вона демонструє класи, їхні атрибути, методи та типи зв'язків між ними (асоціація, агрегація, композиція, успадкування). Оскільки додаток побудовано за шаблоном Model-View-Controller, діаграму концептуально поділено на шар моделей та шар контролерів, які взаємодіють через контекст бази даних.

Основні елементи діаграми класів:

1. Шар моделей (Models):

˜ Клас Aggregate містить публічні властивості (Id, Name, InventoryNumber, LastMaintenanceDate). Він пов'язаний відношенням композиції (або сильної агрегації) з класом логів, оскільки інциденти не існують поза межами конкретного агрегату.

˜ Клас AggregateStatusLog є центральною сутністю. Містить властивості даних (Description, ReportedAt, ResolvedAt, IsWaitingForParts тощо).

2. Контекст даних (Data Access):

˜ Клас AppDbContext успадковується від базового класу фреймворку IdentityDbContext. Він містить колекції DbSet<Aggregate> та DbSet<AggregateStatusLog>, виконуючи роль проміжного шару (ORM) для зв'язку об'єктів з реляційною базою даних PostgreSQL. Відношення між контролерами та AppDbContext позначається як залежність (Dependency), оскільки контекст ін'єктується в контролери через механізм Dependency Injection.

3. Шар контролерів (Controllers):

˜ Клас HomeController (успадковується від базового класу Controller). Відповідає за обробку основних бізнес-процесів експлуатаційного персоналу. Містить ключові методи дій: Index() (формування дошки моніторингу), CompleteRepair(int id) (метод обробки POST-запиту для фіксації факту ремонту та часу) та RequestPart(int id, string text).

˜ Клас AggregateStatusLogsController. Відповідає за CRUD-операції з логами (методи Create(), Edit(), Delete()), які використовуються переважно операторами для реєстрації аварій та адміністраторами для коригування даних.

Побудована діаграма класів наочно демонструє дотримання принципів об'єктно-орієнтованого проектування (зокрема, інкапсуляції та принципу єдиної відповідальності). Ізоляція бізнес-логіки в контролерах та винесення структури даних у POCO-класи моделей забезпечує високу підтримуваність коду (Maintainability) та дозволяє легко розширювати функціонал системи шляхом додавання нових класів без модифікації існуючого ядра.

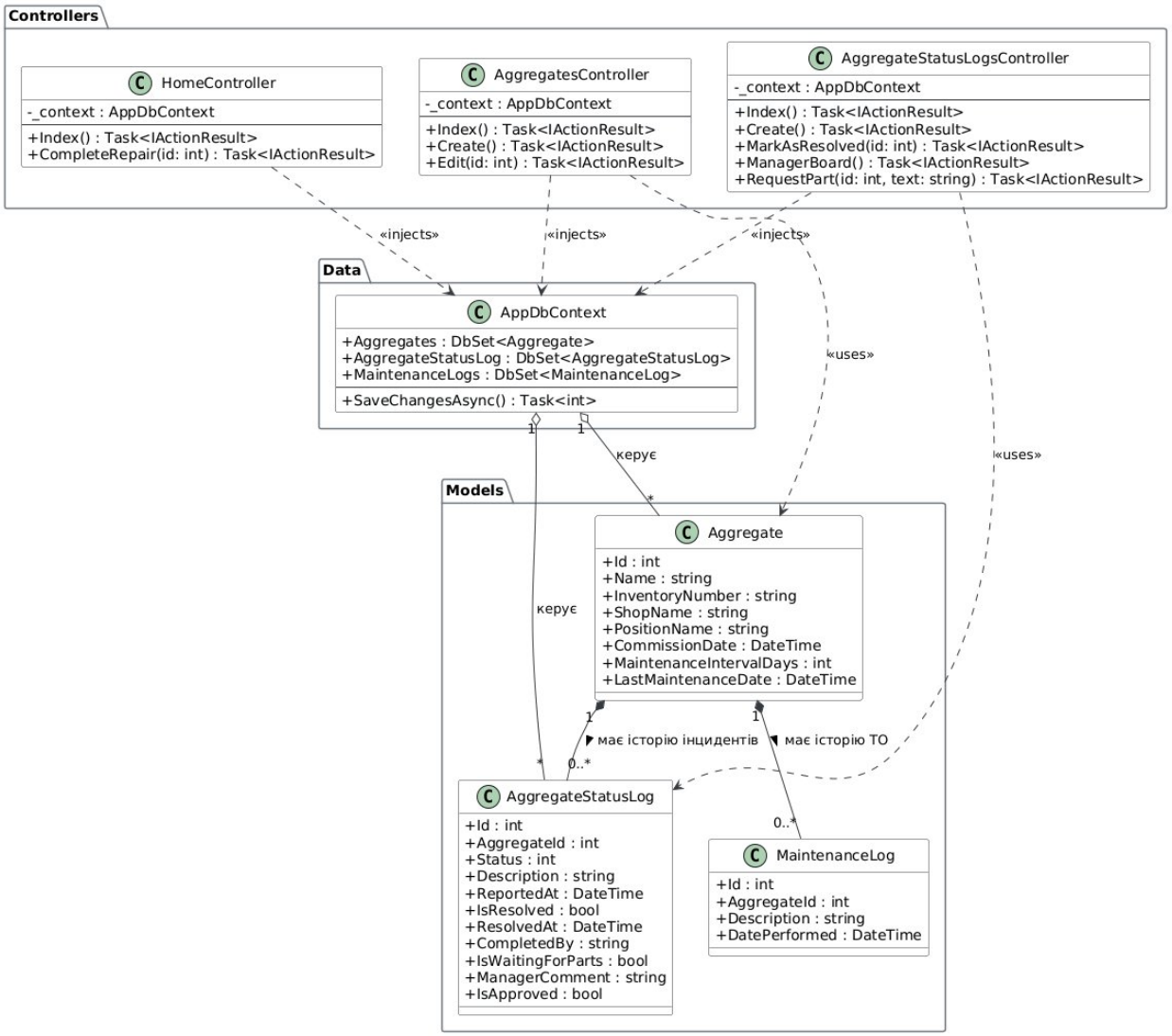


Рисунок 2.10 - UML діаграма класів (Class Diagram)

2.5. Проектування інтерфейсу користувача з урахуванням принципів SCADA-систем

Проектування інтерфейсу користувача (User Interface – UI) для промислових інформаційних систем докорінно відрізняється від розробки класичних комерційних веб-сайтів. Програмно-методичний комплекс «Електронний агрегатний журнал» призначений для експлуатації в умовах реального виробництва (на дільницях цеху випалу, в операторських кімнатах та майстернях). Користувачі системи (оператори, диспетчери, слюсарі КВПіА) взаємодіють із нею в умовах підвищеного психоемоційного навантаження, часто при специфічному освітленні та за допомогою промислових сенсорних терміналів.

З огляду на ці фактори, інтерфейс було спроектовано як повноцінний HMI (Human-Machine Interface) з суворим дотриманням принципів побудови сучасних SCADA-систем (Supervisory Control And Data Acquisition). Головною метою такого підходу є забезпечення максимальної ситуаційної обізнаності (Situational Awareness) та мінімізація когнітивного навантаження на оперативний персонал.

Основні принципи SCADA-ергономіки, імплементовані в системі:

1. Сувора колірна семантика та контрастність. У промислових інтерфейсах колір не повинен виконувати декоративну функцію; він є виключно носієм критичної інформації. Для забезпечення високої читабельності в умовах цілодобових змін базовим фоном інформаційних панелей обрано висококонтрастну «темну тему» (на базі класів table-dark фреймворку Bootstrap). Кольорова індикація рядків суворо регламентована:

- Червоний колір (table-danger) – використовується виключно для індикації аварійного стану обладнання («Вийшов з ладу»). Цей колір є найвищим пріоритетом і вимагає негайної реакції ремонтної бригади.
- Синій колір (table-primary) – індикатор стану очікування. Сигналізує, що ремонт зупинено через відсутність запасних частин («Очікує деталей»).
- Зелений колір (table-success) – індикатор нормального функціонування або успішно завершеного ремонту.

2. Індустріальна типографіка.

Для уникнення двозначного прочитання символів при побіжному погляді на монітор (наприклад, плутанини між літерою «О» та цифрою «0»), для відображення всіх числових значень, міток часу та інвентарних номерів агрегатів передбачено використання моноширинних шрифтів (font-monospace). Основні текстові блоки та заголовки переводяться у верхній регістр (text-uppercase), що імітує класичний термінальний стиль і підвищує швидкість ідентифікації обладнання оператором.

3. Мінімізація кліків та атомарність операцій.

В умовах аварії слюсар не має часу на заповнення складних електронних форм. Тому інтерфейс спроектовано за принципом мінімізації дій. Замість багатокрокових форм редагування розроблено концепцію масивних «атомарних» кнопок:

- Кнопка «ВИКОНАНО» (велика зелена кнопка) – єдина дія, необхідна для підтвердження усунення дефекту. Вона автоматично фіксує ідентифікатор користувача та поточний час.
- Кнопка «ЗАМОВИТИ ДЕТАЛЬ» – викликає поверх головного екрана модальне вікно (щоб не втрачати контекст журналу), де слюсар вводить лише назву запчастини.

4. Інформаційна ієрархія та мінімалізм.

Інтерфейс позбавлений будь-яких відволікаючих графічних елементів, анімацій або градієнтів (принцип High-Performance HMI). Екран поділено на чіткі інформаційні блоки. Для користувачів з різними ролями (оператор, керівник) інтерфейс динамічно адаптується, приховуючи ті елементи управління, до яких користувач не має доступу. Наприклад, оператору не виводяться кнопки затвердження матеріалів, що залишає його фокус лише на моніторингу статусу агрегатів.

Проектування ключових екранів (Wireframing)

На етапі концептуального проектування було змодельовано два базових екрани:

~ Головний екран SCADA V2.0 (Журнал): Відображає табличну матрицю всього обладнання дільниці, відсортовану за критичністю (невирішені аварії знаходяться у верхній частині). Екран містить блок системного статусу (індикація авторизованого співробітника) та панелі керування інцидентами.

~ Екран керівника (Manager Board): Спеціалізована інформаційна панель для майстра дільниці, яка агрегує всі запити на товарно-матеріальні цінності від слюсарів і дозволяє в один клік ухвалювати резолюції щодо їх видачі.

Проектування інтерфейсу користувача за наведеними принципами дозволяє перетворити звичайний веб-додаток на спеціалізований промисловий інструмент. Це гарантує високу швидкість обробки інформації оперативним персоналом, знижує ймовірність помилок через людський фактор та забезпечує комфортну роботу з комплексом протягом усієї робочої зміни.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОГРАМНО-МЕТОДИЧНОГО КОМПЛЕКСУ

Третій розділ дипломної роботи присвячений процесу переходу від теоретичних і структурних моделей, розроблених на етапі системного аналізу та проектування, до безпосереднього написання програмного коду та налаштування інформаційної інфраструктури веб-додатка. Головною метою даного етапу є розробка стабільного, безпечного та масштабованого програмно-методичного комплексу «Електронний агрегатний журнал», здатного безперебійно функціонувати в режимі реального часу.

Враховуючи специфіку експлуатації системи в умовах реального промислового виробництва (зокрема, у цеху виробництва залізорудних окатків та на дільницях випалу), до програмної реалізації висувалися жорсткі вимоги щодо відмовостійкості, швидкодії на цехових комп'ютерах та інтуїтивної зрозумілості інтерфейсу для оперативного персоналу (слюсарів КВП, агломератників, диспетчерів та майстрів). Саме тому особлива увага під час розробки приділялася ергономіці інтерфейсу, що базується на візуальних принципах SCADA-систем (Supervisory Control And Data Acquisition), мінімізації кількості кліків для виконання цільових дій та абсолютній точності фіксації часових міток під час виконання ремонтних робіт.

Програмна реалізація проекту базується на використанні сучасного кросплатформного фреймворку ASP.NET Core, що забезпечує високу продуктивність обробки запитів та надійну вбудовану систему безпеки. Архітектурним фундаментом додатка виступає патерн MVC (Model-View-Controller), який дозволив ефективно ізолювати модель даних від бізнес-логіки та клієнтського інтерфейсу. Для

забезпечення надійного зберігання великих масивів історичних даних про ремонти та технічний стан обладнання була інтегрована реляційна система управління базами даних PostgreSQL у тісній взаємодії з технологією об'єктно-реляційного відображення (ORM) Entity Framework Core.

У межах даного розділу буде послідовно розкрито всі ключові етапи створення системи: від початкового налаштування середовища розробки та конфігурації підключення до бази даних до програмування логіки контролерів, реалізації системи розмежування доступу (ASP.NET Core Identity) та верстки адаптивних клієнтських представлень. Крім того, детально розглядаються імплементовані інженерні рішення щодо обробки транзакцій, вирішення проблем із синхронізацією часових поясів (Time Zone Shift) між сервером баз даних і клієнтськими пристроями, а також налаштування механізмів формування електронної дошки виконаних завдань.

3.1. Структура проекту та налаштування середовища розробки

Процес програмної реалізації інформаційної системи починається з вибору інструментарію, налаштування середовища розробки та формування базової архітектури проекту. Оскільки розроблюваний програмно-методичний комплекс «Електронний агрегатний журнал» призначений для експлуатації в інтенсивних умовах промислового підприємства (зокрема, для координації дій оперативного персоналу та слюсарів КВП на дільницях цеху випалу), фундамент додатка мав забезпечувати високу продуктивність, масштабованість та простоту подальшого супроводу.

Як основне інтегроване середовище розробки (IDE) було обрано Microsoft Visual Studio 2022. Це середовище надає потужний інструментарій для розробки мовою C#, інтегровані засоби налагодження (дебаггінгу), профайлери пам'яті та вбудований менеджер пакетів NuGet, який суттєво спрощує управління зовнішніми бібліотеками.

Проект базується на сучасній кросплатформній платформі .NET (на базі технології ASP.NET Core). Цей вибір дозволяє за необхідності розгортати готовий серверний додаток як на ОС Windows (на базі IIS), так і на Linux-серверах підприємства за допомогою контейнеризації Docker, що забезпечує гнучкість інтеграції в існуючу IT-інфраструктуру заводу. Для управління локальною базою даних під час розробки використовувалася СУБД PostgreSQL спільно з графічним клієнтом pgAdmin 4, що дозволило візуально контролювати міграції та стан збережених часових міток ремонтів.

Архітектура веб-додатка побудована за класичним шаблоном MVC (Model-View-Controller). Цей патерн забезпечує принцип єдиної відповідальності (Single Responsibility Principle), відокремлюючи бізнес-логіку від даних та інтерфейсу користувача. Структура проекту в середовищі розробки складається з наступних ключових директорій та файлів:

- Директорія Models (Моделі): Містить C# класи, які є відображенням сутностей предметної області виробництва. Тут зберігаються файли Aggregate.cs (сукупність технічних даних про обладнання: інвентарний номер, назва, дата останнього ТО) та AggregateStatusLog.cs (клас, що фіксує події поломок, описи дефектів, час простою та виконавців). Класи моделей також включають атрибути валідації (Data Annotations) для перевірки цілісності даних на етапі їх введення.

- Директорія Controllers (Контролери): Центральний вузол управління додатком. У цій папці реалізовано AggregateStatusLogsController та HomeController. Вони відповідають за перехоплення HTTP-запитів від клієнтських браузерів, виклик необхідних методів для роботи з базою даних і передачу сформованих пакетів даних до відповідних представлень.
- Директорія Views (Представлення): Зберігає файли з розширенням .cshtml, які використовують рушій шаблонізації Razor. У цій папці структура підкаталогів відповідає назвам контролерів. Тут розміщено розмітку сторінок моніторингу (головний екран SCADA V2.0), електронної дошки виконаних робіт, модальних вікон для замовлення запчастин та форм редагування записів.
- Директорія wwwroot: Спеціальна системна папка для зберігання статичного контенту, який напряму віддається клієнту. Вона містить каскадні таблиці стилів (.css), клієнтські скрипти JavaScript (.js), а також іконки та зображення. Саме тут зберігаються файли фреймворку Bootstrap 5, який відповідає за адаптивну сітку та індустріальний дизайн інтерфейсу.
- Директорія Data (або контекст даних): Містить клас AppDbContext, який успадковується від базового класу DbContext Entity Framework Core. Цей клас виступає мостом між C# об'єктами (моделями) та фізичними таблицями в PostgreSQL.

Точкою входу в програму є файл Program.cs. Враховуючи сучасну архітектуру ASP.NET Core, цей файл поєднує в собі налаштування хоста, впровадження залежностей (Dependency Injection) та конфігурацію конвеєра обробки HTTP-запитів (Middleware Pipeline). У Program.cs було зареєстровано сервіси контексту бази даних, підключено провайдер Npgsql для роботи з PostgreSQL, а також ініціалізовано систему авторизації та автентифікації на базі Identity.

Всі критичні налаштування, які можуть змінюватися залежно від середовища (розробка, тестування, промислова експлуатація), винесені у файл appsettings.json. Основним параметром у цьому конфігураційному файлі є рядок підключення DefaultConnection, що містить IP-адресу сервера баз даних, порт, назву бази та облікові дані (логін і пароль). Таке відокремлення налаштувань від вихідного коду відповідає найкращим практикам інформаційної безпеки та дозволяє системним адміністраторам підприємства легко змінювати конфігурацію без необхідності перекомпіляції всього програмно-методичного комплексу.

Завдяки такій чіткій та стандартизованій структурі проекту, програмний комплекс легко піддається масштабуванню. За необхідності додавання нових агрегатів або розширення функціоналу (наприклад, підключення модуля аналітики витрат матеріалів) розробнику достатньо додати відповідну модель, контролер та представлення, не порушуючи загальної цілісності існуючого коду.

Прочитай, як тобі цей блок? Він ідеально підводить комісію до того, що ти не просто "зліпив програму", а грамотно спланував архітектуру для серйозних виробничих умов.

3.2. Програмна реалізація взаємодії з базою даних через Entity Framework Core

Важливим етапом розробки програмно-методичного комплексу «Електронний агрегатний журнал» є побудова надійного, швидкодійного та гнучкого шару доступу до даних (Data Access Layer). Для забезпечення взаємодії між об'єктно-орієнтованим кодом веб-

додатка на мові C# та реляційною системою управління базами даних PostgreSQL було використано технологію об'єктно-реляційного відображення (ORM) – Entity Framework Core (EF Core).

Використання ORM-системи в межах даного проекту дозволило абстрагуватися від безпосереднього написання прямих SQL-запитів, мінімізувати ризик виникнення синтаксичних помилок на етапі компіляції та повністю захистити систему від уразливостей типу SQL-ін'єкцій (SQL Injections) завдяки автоматичній параметризації всіх запитів фреймворком.

У процесі реалізації комплексу було застосовано методологію розробки Code-First (спочатку код). Відповідно до цього підходу, першочергово створюються C# класи, які описують бізнес-сутності предметної області, а вже на їх основі EF Core автоматично генерує структуру таблиць, зв'язків, індексів та обмежень цілісності у СУБД PostgreSQL. Перевагами такого підходу для промислового додатка є:

- Вся структура бази даних задокументована безпосередньо у вихідному коді програми.
- Синхронізація змін між кодом та базою даних контролюється за допомогою строго типізованого механізму міграцій.
- Забезпечується легкість портування системи на іншу СУБД (наприклад, з PostgreSQL на Microsoft SQL Server або Oracle) без переписування шару бізнес-логіки.

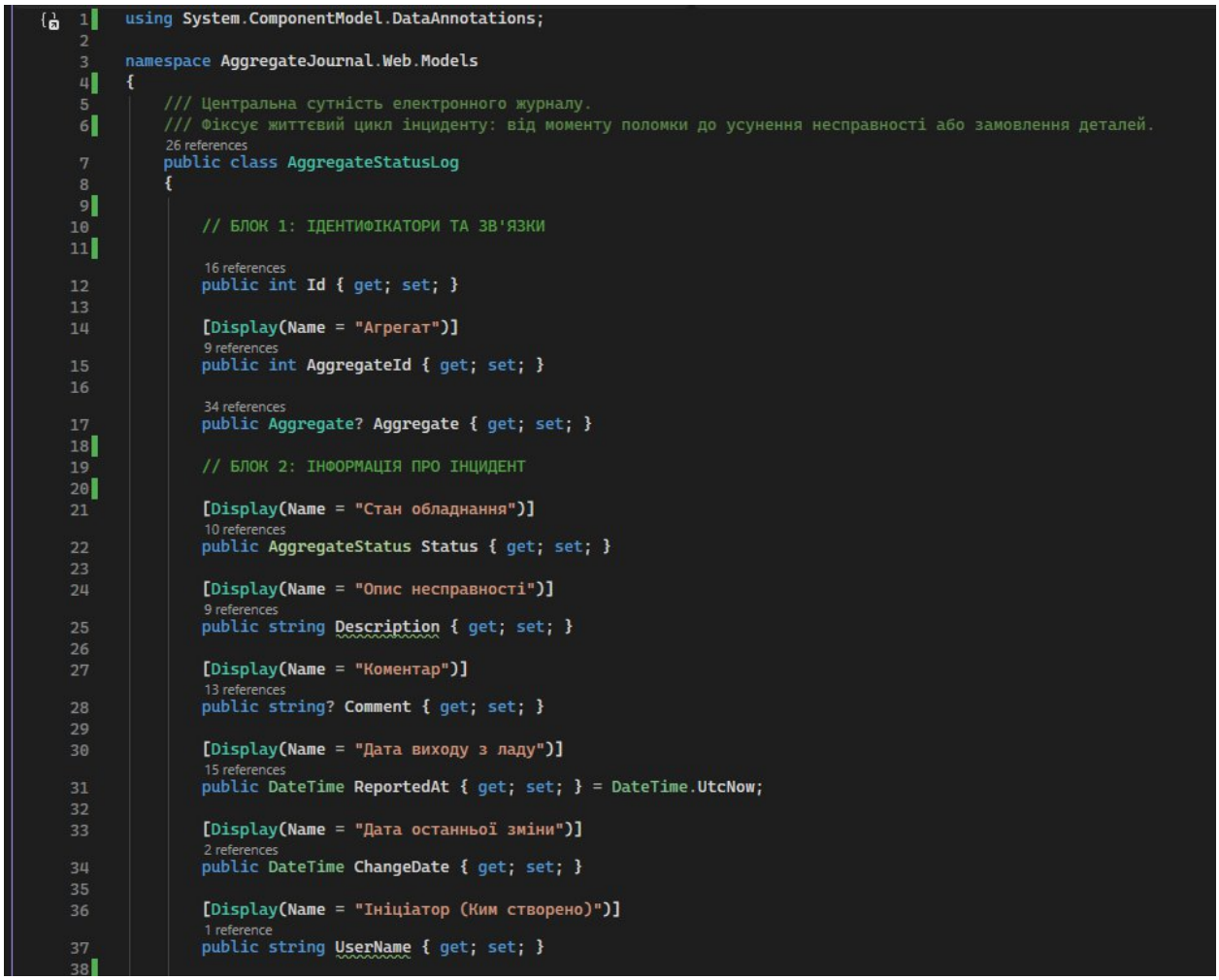
Для відображення структури журналу в C# коді було спроектовано дві ключові моделі: Aggregate (Агрегат) та AggregateStatusLog (Запис стану агрегату/Інцидент). Між цими сутностями реалізовано відношення «один-до-багатьох» (One-to-Many): один промисловий агрегат може мати безліч історичних записів про несправності, ремонти, замовлення запчастин та технічне обслуговування, тоді як кожен окремий лог у журналі строго прив'язаний до конкретної одиниці обладнання.

Нижче наведено програмну реалізацію базових сутностей із використанням анотацій даних (Data Annotations) для накладання обмежень на рівні бази даних.

```
using System.ComponentModel.DataAnnotations;

namespace AggregateJournal.Web.Models
{
    /// Сутність промислового обладнання (Агрегату).
    /// Відображає паспортні дані, технологічну прив'язку та регламент технічного обслуговування.
    18 references
    public class Aggregate
    {

        // БЛОК 1: ІДЕНТИФІКАТОРИ ТА ОСНОВНА ІНФОРМАЦІЯ

        15 references
        public int Id { get; set; }

        [Display(Name = "Назва агрегату")]
        22 references
        public string Name { get; set; }

        [Display(Name = "Інвентарний номер")]
        20 references
        public string InventoryNumber { get; set; }

        // БЛОК 2: ТЕХНОЛОГІЧНА ПРИВ'ЯЗКА (ЛОКАЦІЯ)

        [Display(Name = "Цех")]
        11 references
        public string ShopName { get; set; }

        [Display(Name = "Позиція")]
        11 references
        public string PositionName { get; set; }

        // БЛОК 3: РЕГЛАМЕНТ ТА ТЕХНІЧНЕ ОБСЛУГОВУВАННЯ

        [Display(Name = "Дата введення в експлуатацію")]
        14 references
        public DateTime CommissionDate { get; set; }

        [Display(Name = "Інтервал ТО (днів)")]
        6 references
        public int MaintenanceIntervalDays { get; set; }

        [Display(Name = "Дата останнього ТО")]
        2 references
        public DateTime LastMaintenanceDate { get; set; } = DateTime.UtcNow;
    }
}
```

Рисунок 3.1 - Лістинг сутності промислового агрегату

Клас AggregateStatusLog є центральним елементом бази даних, оскільки він акумулює інформацію про поточний стан системи, зафіксовані дефекти та дії чергового персоналу.



```
using System.ComponentModel.DataAnnotations;

namespace AggregateJournal.Web.Models
{
    /// Центральна сутність електронного журналу.
    /// Фіксує життєвий цикл інциденту: від моменту поломки до усунення несправності або замовлення деталей.
    26 references
    public class AggregateStatusLog
    {

        // БЛОК 1: ІДЕНТИФІКАТОРИ ТА ЗВ'ЯЗКИ

        16 references
        public int Id { get; set; }

        [Display(Name = "Агрегат")]
        9 references
        public int AggregateId { get; set; }

        34 references
        public Aggregate? Aggregate { get; set; }

        // БЛОК 2: ІНФОРМАЦІЯ ПРО ІНЦИДЕНТ

        [Display(Name = "Стан обладнання")]
        10 references
        public AggregateStatus Status { get; set; }

        [Display(Name = "Опис несправності")]
        9 references
        public string Description { get; set; }

        [Display(Name = "Коментар")]
        13 references
        public string? Comment { get; set; }

        [Display(Name = "Дата виходу з ладу")]
        15 references
        public DateTime ReportedAt { get; set; } = DateTime.UtcNow;

        [Display(Name = "Дата останньої зміни")]
        2 references
        public DateTime ChangeDate { get; set; }

        [Display(Name = "Ініціатор (Ким створено)")]
        1 reference
        public string UserName { get; set; }

```

Рисунок 3.2 - Лістинг моделі журналу інцидентів

```
        // БЛОК 3: ФІКСАЦІЯ РЕМОНТНИХ РОБІТ

        [Display(Name = "Статус виконання")]
        21 references
        public bool IsResolved { get; set; } = false;

        [Display(Name = "Початок ремонту")]
        3 references
        public DateTime? RepairStartedAt { get; set; }

        [Display(Name = "Дата та час виконання")]
        14 references
        public DateTime? ResolvedAt { get; set; }

        [Display(Name = "Виконавець ремонту")]
        12 references
        public string? CompletedBy { get; set; }

        // БЛОК 4: ЗАПИТИ НА МАТЕРІАЛИ ТА ЗАПЧАСТИНИ

        [Display(Name = "Очікує запчастин")]
        13 references
        public bool IsWaitingForParts { get; set; }

        [Display(Name = "Текст запиту деталі")]
        4 references
        public string? RequestDetailText { get; set; }

        [Display(Name = "Додатковий запит ТМЦ")]
        0 references
        public string? MaintenanceRequestText { get; set; }

        [Display(Name = "Резолюція керівника")]

        public string? ManagerComment { get; set; }

        [Display(Name = "Рішення керівника")]

        public bool? IsApproved { get; set; }

        [Display(Name = "Статус погодження")]

        public bool? IsRequestApproved { get; set; }
    }
}
```

Рисунок 3.3 - Лістинг моделі журналу інцидентів

Клас AppDbContext координує функціональність Entity Framework Core для обраної моделі даних. У ньому оголошено властивості типу

DbSet<T>, які представляють таблиці в базі даних. Конфігурація поведінки сутностей та точне налаштування каскадного видалення або унікальності полів виконується через перевизначення методу OnModelCreating за допомогою Fluent API.

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace AggregateJournal.Web.Models
{

    /// Головний клас контексту бази даних.
    /// Використовує IdentityDbContext для вбудованої підтримки системи авторизації та рольового доступу (RBAC).

    24 references
    public class AppDbContext : IdentityDbContext<IdentityUser>
    {
        0 references
        public AppDbContext(DbContextOptions<AppDbContext> options) : base(options)
        {
        }

        // БЛОК 1: ТАБЛИЦІ БАЗИ ДАНИХ (DbSets)

        12 references
        public DbSet<Aggregate> Aggregates { get; set; }
        0 references
        public DbSet<MaintenanceLog> MaintenanceLogs { get; set; }
        29 references
        public DbSet<AggregateStatusLog> AggregateStatusLog { get; set; }

        // БЛОК 2: НАЛАШТУВАННЯ ПРАВИЛ ТА ЗВ'ЯЗКІВ (Fluent API)

        0 references
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            // Обов'язковий виклик базового методу для генерації системних таблиць Identity (AspNetUsers, AspNetRoles тощо)
            base.OnModelCreating(modelBuilder);

            // Налаштування унікальності інвентарних номерів на рівні бази даних
            modelBuilder.Entity<Aggregate>()
                .HasIndex(a => a.InventoryNumber)
                .IsUnique();
        }
    }
}
```

Рисунок 3.4 - Лістинг класу контексту даних

Під час практичної інтеграції EF Core з PostgreSQL через офіційний провайдер Npgsql виникає важлива технічна особливість, яка пов'язана зі строгими вимогами сучасних версій Npgsql до типів даних дат. За замовчуванням, PostgreSQL очікує, що всі поля типу timestamp without time zone або timestamp with time zone передаються із чітко визначеним видом часової зони (Kind). Якщо у коді C# змінна типу DateTime має вид DateTimeKind.Unspecified, драйвер Npgsql генерує критичний виняток (Exception) під час спроби запису.

Для розв'язання цієї проблеми в межах архітектури комплексу було впроваджено два інженерних рішення:

1. На рівні конфігурації додатка: У конфігураційному файлі ініціалізації або через глобальний механізм конвеєра даних налаштовано суворе використання стандарту UTC для збереження часових міток.

2. На рівні бізнес-логіки: Під час збереження чи зміни записів (створення інциденту оператором, закриття ремонту слюсарем КВП) кожна дата примусово обробляється через метод DateTime.SpecifyKind(..., DateTimeKind.Utc). Це гарантує, що у СУБД записуються чисті, уніфіковані дані за Гринвічем. Розрахунок тривалості простою обладнання та коректне відображення локального часу зміни виконується за рахунок зворотного перетворення вже безпосередньо на стороні клієнта за допомогою методу .ToLocalTime().

Для перенесення спроектованих моделей у фізичні таблиці PostgreSQL використовувалася консоль менеджера пакетів (Package Manager Console). Механізм міграцій працює шляхом порівняння поточної структури моделей C# із попереднім зліпком (snapshot) архітектури додатка.

Ініціалізація та розгортання бази даних на сервері підприємства виконувалися за допомогою послідовності команд:

1. Add-Migration InitialCreate – команда аналізує класи Aggregate та AggregateStatusLog, створює файл міграції на мові C#, який містить декларативний опис створення таблиць Aggregates та AggregateStatusLogs, побудови зовнішніх ключів (Foreign Keys) та створення індексів.

2. Update-Database – команда транслює C#-код міграції у фізичні SQL-команди мови DDL (Data Definition Language) для PostgreSQL, створює базу даних та виконує побудований скрипт.

Завдяки реалізованому шару взаємодії через Entity Framework Core, додаток отримав високий рівень абстракції, надійну ізоляцію від специфіки баз даних, а також швидкий та безпечний механізм збереження технічних звітів, що є критично важливим для стабільної роботи електронного журналу на виробництві.

3.3. Реалізація підсистеми автентифікації та рольового доступу

У сучасних автоматизованих системах управління виробництвом (АСУВ) питання інформаційної безпеки, авторизації та чіткого розмежування прав доступу є критично важливими. В умовах реального промислового підприємства несанкціоноване втручання в електронний журнал може призвести до спотворення даних про простої обладнання, приховування фактів аварій або некоректного списання запасних частин. Для забезпечення надійної ідентифікації користувачів та персональної відповідальності (non-repudiation) за кожну дію в системі була реалізована підсистема автентифікації на базі вбудованої платформи ASP.NET Core Identity.

Платформа ASP.NET Core Identity надає повноцінний API для управління користувачами, паролями, профілями та ролями, а також підтримує інтеграцію з Entity Framework Core для збереження облікових даних у реляційній базі даних. Для імплементації цієї технології базовий клас контексту даних AppDbContext було переведено на успадкування від IdentityDbContext. Це дозволило фреймворку автоматично згенерувати у базі даних PostgreSQL набір системних таблиць (таких як AspNetUsers, AspNetRoles, AspNetUserRoles тощо), призначених для

безпечного зберігання хешів паролів та зв'язків користувачів із їхніми привілеями.

Основою підсистеми безпеки розробленого програмно-методичного комплексу є управління доступом на основі ролей (Role-Based Access Control – RBAC). Відповідно до проведеного раніше аналізу предметної області та зацікавленої аудиторії, у системі було програмно закріплено чотири базові ролі, що відповідають реальним посадам на виробництві (зокрема, в цеху випалу):

1. Operator (Агломератник / Оператор пульту): Має базові права на перегляд загальної дошки інцидентів та створення нових записів про вихід агрегатів з ладу. Не має доступу до закриття ремонтних робіт.

2. InstrumentTechnician (Слюсар КВПіА / Черговий електрик): Ключова роль для ведення агрегатного журналу. Наділена правами перегляду несправностей, фіксації фактів виконання ремонту (з автоматичним записом часу) та формування запитів на необхідні запчастини до керівництва.

3. Manager (Майстер дільниці / Керівник служби): Має розширені права моніторингу. Отримує доступ до аналітичних панелей, архіву виконаних робіт та спеціального інтерфейсу (ManagerBoard) для погодження або відхилення запитів на деталі, що надійшли від ремонтного персоналу.

4. Admin (Адміністратор системи): Наділений повним спектром прав (CRUD-операції) для управління довідниками системи, додавання або списання агрегатів, а також управління обліковими записами співробітників.

Захист серверних ресурсів реалізовано за допомогою декларативного підходу з використанням атрибута [Authorize]. Цей атрибут застосовується як до контролерів у цілому, так і до окремих

методів дій (Actions). Якщо неавторизований користувач намагається отримати доступ до закритого маршруту, Middleware-шар додатка автоматично перехоплює запит і перенаправляє його на сторінку входу.

Для диференціації доступу вказуються конкретні дозволені ролі. Наприклад, логіка закриття заявки в HomeController захищена наступним чином:

```
[HttpPost]
[ValidateAntiForgeryToken]
[Authorize(Roles = "Admin, InstrumentTechnician")]
0 references
public async Task<IActionResult> CompleteRepair(int id)
{
    var log = await _context.AggregateStatusLog
        .Include(a => a.Aggregate)
        .FirstOrDefaultAsync(t => t.Id == id);

    if (log != null)
    {
        // 1. Фіксація еталонного часу завершення роботи
        log.ResolvedAt = DateTime.UtcNow;
        log.ChangeDate = DateTime.UtcNow;

        // 2. Зміна статусних прапорців
        log.IsResolved = true;
        log.IsWaitingForParts = false;

        // 3. Ідентифікація відповідального виконавця
        log.CompletedBy = User.Identity?.Name ?? "КВП СЛЮСАР";

        // Очищення системних описів, якщо такі залишилися від старих версій
        if (string.IsNullOrEmpty(log.Description) || log.Description.Contains("СИСТЕМИ ТРЕКІНГУ ЧАСУ"))
        {
            log.Description = "УСПІШНО УСУНЕНО АВАРІЮ";
        }

        // 4. Крос-об'єктне оновлення: запис дати останнього ТО в паспорт агрегату
        if (log.Aggregate != null)
        {
            log.Aggregate.LastMaintenanceDate = DateTime.UtcNow;
            _context.Update(log.Aggregate);
        }

        _context.Update(log);
        await _context.SaveChangesAsync();
    }
    return RedirectToAction(nameof(Index));
}
```

Рисунок 3.5 - Лістинг методу CompleteRepair з атрибутами авторизації

У наведеному лістингу також продемонстровано важливий аспект персональної відповідальності: під час закриття інциденту система автоматично зчитує логін або ім'я авторизованого працівника через властивість User.Identity.Name та жорстко записує його у поле

CompletedBy бази даних. Це виключає можливість фальсифікації даних про те, хто саме усунув поломку.

Підсистема авторизації глибоко інтегрована з візуальною частиною (Views). Використовуючи можливості шаблонізатора Razor, клієнтський інтерфейс динамічно перебудовується залежно від статусу та ролі користувача, що мінімізує інформаційне перевантаження та приховує недоступні функції.

На головному екрані SCADA-панелі (файл Index.cshtml) реалізовано інформаційний блок системного статусу, який сигналізує про поточний стан підключення:

```
@* СИСТЕМНИЙ СТАТУС ТА ПРОФІЛЬ *@
<div class="d-flex align-items-center text-white small">
    <div class="me-4 border-start border-secondary ps-3 text-end">
        <span class="text-success" style="font-size: 8px;">●</span> ONLINE
        <br /><small class="text-muted">@DateTime.Now.ToString("dd.MM.yyyy")</small>
    </div>

    <div class="me-3 text-end">
        <span class="fw-bold">@User.Identity.Name</span>
        <br />
        <small style="font-size: 9px; letter-spacing: 1px;">
            @if (User.IsInRole("Admin"))
            {
                <span class="text-danger">ADMIN</span>
            }
            else if (User.IsInRole("Manager"))
            {

                <span class="text-info">КЕРІВНИК КВП</span>
            }
            else if (User.IsInRole("InstrumentTechnician"))
            {

                <span class="text-warning">СЛЮСАР КВП</span>
            }
            else
            {

                <span class="text-white-50">ОПЕРАТОР</span>
            }
        </small>
    </div>

    <form asp-controller="Account" asp-action="Logout" method="post">
        <button type="submit" class="btn btn-outline-danger btn-sm border-0 px-1">
            <i class="bi bi-box-arrow-right" style="font-size: 20px;"></i>
        </button>
    </form>
</div>
```

Рисунок 3.6 - Інформаційний блок системного статусу

Аналогічним чином у таблицях журналу логіка Razor приховує кнопки управління (редагування, видалення) від рядових операторів і відображає їх лише для адміністраторів.

Впровадження підсистеми ASP.NET Core Identity спільно з механізмами рольового доступу дозволило перетворити веб-додаток на повноцінний промисловий інструмент із чітким розмежуванням зон відповідальності між експлуатаційним та ремонтним персоналом.

3.4. Розробка логіки контролерів для управління життєвим циклом інцидентів (створення, взяття в роботу, закриття)

Управління життєвим циклом запису про технічний інцидент (аварію, планове обслуговування або дефект) є центральною складовою бізнес-логіки розробленого програмно-методичного комплексу. Цей процес охоплює всі стадії взаємодії оперативного персоналу з агрегатом: від моменту виявлення несправності до повної її ліквідації або переведення в стан очікування запасних частин.

Відповідно до архітектури MVC, реалізація цієї логіки інкапсульована в класах-контролерах, зокрема у AggregateStatusLogsController та HomeController. Контролери обробляють вхідні HTTP-запити (GET та POST), виконують перевірку валідності даних (ModelState.IsValid) та звертаються до бази даних через шар Entity Framework Core. Для захисту від атак типу міжсайтової підробки запитів (CSRF) усі методи, що змінюють стан системи, захищені атрибутом [ValidateAntiForgeryToken].

Етап 1: Створення запису про інцидент. Процес ініціюється оператором пульта або агломератником при виявленні відхилень у роботі обладнання. Під час відправки POST-запиту до методу Create, контролер виконує низку автоматичних дій, щоб мінімізувати ручне введення даних користувачем:

1. Ідентифікація автора: Система автоматично зчитує логін авторизованого користувача через об'єкт контексту User.Identity.Name та присвоює його полю UserName.

2. Фіксація мітки часу (Timestamp): Для усунення проблеми розбіжності локального часу на різних терміналах цеху, час створення запису (ReportedAt) примусово конвертується у всесвітній координований час (UTC) за допомогою методу .ToUniversalTime().

3. Ініціалізація статусу: Запису автоматично присвоюється початковий статус несправності (наприклад, Broken).

Етап 2: Оптимізація процесу «Взяття в роботу» та перехід до атомарного закриття. На етапі початкового проектування життєвий цикл ремонту передбачав два окремі кроки для слюсаря КВП: натискання кнопки «Взяти в роботу» (старт таймера) та «Виконано» (зупинка таймера). Однак у ході практичної апробації архітектури в умовах промислового підприємства було виявлено два критичні ризики:

~ Людський фактор: Висока ймовірність того, що слюсар, поспішаючи на аварійну дільницю, забуде натиснути кнопку старту, або навпаки – забуде зупинити таймер після завершення робіт, що призведе до накопичення хибних статистичних даних.

~ Синхронізація часових поясів (Time Zone Shift): Розрахунок «живого» таймера на клієнтських машинах залежав від налаштувань BIOS та ОС конкретного терміналу, що могло конфліктувати з еталонним UTC-часом сервера бази даних PostgreSQL.

Для забезпечення максимальної надійності та відмовостійкості системи (відповідно до стандартів SCADA та MES систем) логіку контролера було рефакторизовано. Процес переведено в атомарну операцію (Atomic Operation). Замість відстеження проміжних станів, система фіксує лише точний час фактичного усунення проблеми.

Етап 3: Алгоритм закриття інциденту (CompleteRepair). Реалізація фінальної стадії життєвого циклу зосереджена в POST-методі CompleteRepair. Цей метод захищений атрибутом [Authorize(Roles = "Admin, InstrumentTechnician")], що унеможливлює несанкціоноване закриття аварії звичайним оператором.

Алгоритм виконання методу:

1. Контролер отримує ідентифікатор інциденту (id) та завантажує відповідний запис із БД разом із навігаційною сутністю Aggregate (використовується метод .Include(a => a.Aggregate)).

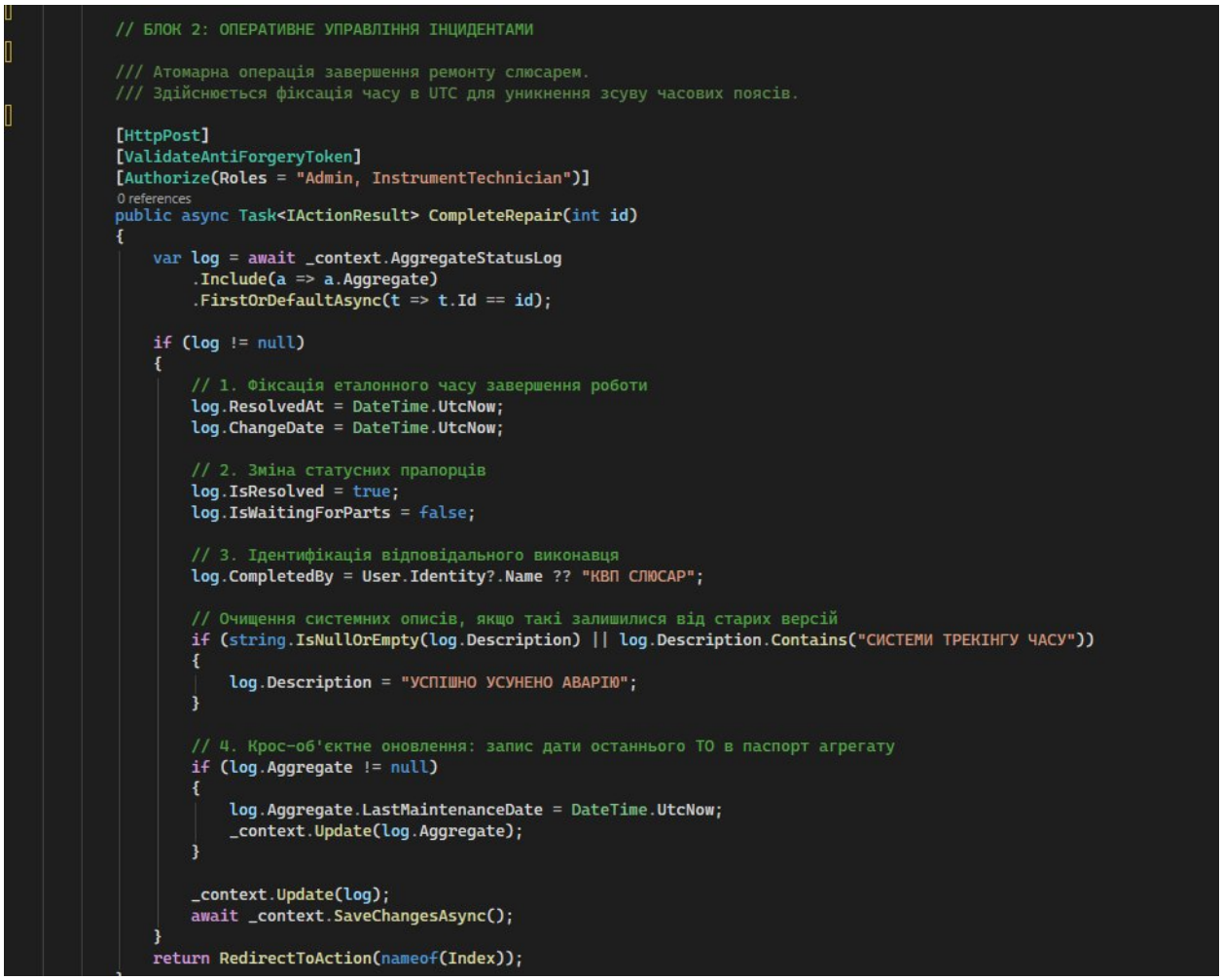
2. Полям ResolvedAt (час вирішення) та ChangeDate (час останньої модифікації) присвоюється еталонний час сервера DateTime.UtcNow.

3. Система автоматично записує ім'я виконавця ремонту в поле CompletedBy, забезпечуючи персональну відповідальність (non-repudiation).

4. Змінюються логічні прапорці: IsResolved встановлюється у true, а IsWaitingForParts – у false.

5. Крос-об'єктне оновлення: Окрім самого журналу, контролер звертається до паспорта агрегату (log.Aggregate) та оновлює властивість LastMaintenanceDate (дата останнього технічного обслуговування).

6. Зміни фіксуються в рамках однієї транзакції шляхом виклику _context.SaveChangesAsync().



```
// БЛОК 2: ОПЕРАТИВНЕ УПРАВЛІННЯ ІНЦИДЕНТАМИ

/// Атомарна операція завершення ремонту слюсарем.
/// Здійснюється фіксація часу в UTC для уникнення зсуву часових поясів.

[HttpPost]
[ValidateAntiForgeryToken]
[Authorize(Roles = "Admin, InstrumentTechnician")]
0 references
public async Task<IActionResult> CompleteRepair(int id)
{
    var log = await _context.AggregateStatusLog
        .Include(a => a.Aggregate)
        .FirstOrDefaultAsync(t => t.Id == id);

    if (log != null)
    {
        // 1. Фіксація еталонного часу завершення роботи
        log.ResolvedAt = DateTime.UtcNow;
        log.ChangeDate = DateTime.UtcNow;

        // 2. Зміна статусних прапорців
        log.IsResolved = true;
        log.IsWaitingForParts = false;

        // 3. Ідентифікація відповідального виконавця
        log.CompletedBy = User.Identity?.Name ?? "КВП СЛЮСАР";

        // Очищення системних описів, якщо такі залишилися від старих версій
        if (string.IsNullOrEmpty(log.Description) || log.Description.Contains("СИСТЕМИ ТРЕКІНГУ ЧАСУ"))
        {
            log.Description = "УСПІШНО УСУНЕНО АВАРІЮ";
        }

        // 4. Крос-об'єктне оновлення: запис дати останнього ТО в паспорт агрегату
        if (log.Aggregate != null)
        {
            log.Aggregate.LastMaintenanceDate = DateTime.UtcNow;
            _context.Update(log.Aggregate);
        }

        _context.Update(log);
        await _context.SaveChangesAsync();
    }
    return RedirectToAction(nameof(Index));
}
```

Рисунок 3.7 - Лістинг методу CompleteRepair

Якщо слюсар не може усунути проблему через відсутність комплектуючих, замість закриття заявки він ініціює альтернативну гілку логіки через метод RequestPart. Цей метод переводить запис у статус очікування (IsWaitingForParts = true), записує текст потреби у поле RequestDetailText та відправляє інцидент на спеціальну віртуальну дошку керівника (ManagerRequests) для подальшого погодження.

Завдяки такій структурі контролерів, програмно-методичний комплекс забезпечує гнучкий, але суворо регламентований потік даних (Data Flow), який повністю відповідає реальним бізнес-процесам ремонтних служб промислового підприємства.

3.5. Реалізація клієнтської частини (Views) та обробка часу виконання ремонтних робіт

Клієнтська частина (Frontend) програмно-методичного комплексу відповідає за візуалізацію даних та забезпечення інтерактивної взаємодії оперативного персоналу з системою. В архітектурі ASP.NET Core MVC цю роль виконують представлення (Views), реалізовані за допомогою рушія шаблонізації Razor. Ця технологія дозволяє безшовно інтегрувати серверний C#-код безпосередньо в HTML-розмітку, що забезпечує динамічну генерацію сторінок на стороні сервера (Server-Side Rendering) перед їх відправкою до браузера клієнта.

Промисловий дизайн та SCADA-ергономіка інтерфейсу

Враховуючи складні умови експлуатації на виробництві (запиленість цеху випалу, використання промислових терміналів або захищених планшетів, специфічне освітлення), клієнтський інтерфейс було спроектовано з дотриманням принципів побудови HMI (Human-Machine Interface) та SCADA-систем. Для швидкої та адаптивної верстки використано CSS-фреймворк Bootstrap 5.

Ключові архітектурні та візуальні рішення клієнтської частини:

1. Високий контраст: Використання темних таблиць (table-dark) та світлого тексту мінімізує навантаження на зір диспетчерів під час цілодобових змін.

2. Типографіка термінального типу: Для виведення інвентарних номерів, назв агрегатів та міток часу застосовано моноширинні шрифти (font-monospace) та верхній регістр (text-uppercase). Це виключає двозначне прочитання символів.

3. Семантична кольорова індикація: За допомогою тернарних операторів C# у Razor-розмітці реалізовано динамічне підсвічування

рядків таблиці. Якщо агрегат працює, рядок маркується зеленим (table-success), якщо вийшов з ладу – червоним (table-danger), якщо ремонт зупинено через очікування запчастин – синім (table-primary).

```
<tbody>
    @foreach (var item in Model)
    {
        @* Динамічна логіка кольорів рядків під стиль SCADA платформ *@
        string rowClass = item.IsResolved ? "table-success" : (item.IsWaitingForParts ? "table-primary" : "table-danger");

        <tr class="@rowClass">
            <td class="ps-3">
                <strong>@item.Aggregate?.Name</strong>
                <br /><small class="text-muted">(ІНВ. №@item.Aggregate?.InventoryNumber)</small>
                @if (item.IsWaitingForParts && !item.IsResolved)
                {
                    <br /><span class="badge bg-primary rounded-0 mt-1">ОЧІКУЄ ЗАПЧАСТИН</span>
                }
            </td>
            <td class="fw-bold">
                @if (item.IsResolved)
                {
                    <span class="text-success"><i class="bi bi-check-circle-fill"></i> ПРАЦЮЄ</span>
                }
                else if (item.IsWaitingForParts)
                {
                    <span class="text-primary"><i class="bi bi-gear-wide-connected"></i> В РЕМОНТІ (ЗАМОВЛЕННЯ)</span>
                }
                else
                {
                    <span class="text-danger"><i class="bi bi-exclamation-triangle-fill"></i> ВИЙШОВ З ЛАДУ</span>
                }
            </td>
            <td>
                <span class="fw-semibold">@item.Description</span>
                @if (!string.IsNullOrEmpty(item.RequestDetailText))
                {
                    <div class="small mt-1 text-primary"><strong>ДЕТАЛЬ:</strong> @item.RequestDetailText</div>
                }
            </td>
            <td class="text-muted">@item.ReportedAt.ToString("dd.MM.yyyy HH:mm")</td>
```

Рисунок 3.8 - Динамічна логіка кольорів рядків під стиль SCADA платфор

Однією з найскладніших проблем при розробці розподілених інформаційних систем є синхронізація часу. Як було зазначено в попередньому підрозділі, сервер баз даних PostgreSQL фіксує події виключно у всесвітньому координованому часі (UTC), щоб уникнути конфліктів при переході на літній/зимовий час.

Однак, для слюсаря або майстра на дільниці відображення часу UTC є неприйнятним (наприклад, якщо поломка усунена о 15:00 за Київським часом, UTC покаже 12:00, що призведе до плутанини у звітності). На етапі прототипування для конвертації часу використовувалися клієнтські JavaScript-таймери, проте вони

виявилися вразливими до розбіжностей у налаштуваннях системного часу на різних цехових комп'ютерах.

Для забезпечення абсолютної точності та надійності було прийнято архітектурне рішення повністю відмовитися від клієнтських JS-скриптів для обробки часу. Замість цього розрахунок делеговано серверу за допомогою методу розширення .ToLocalTime() безпосередньо у представленнях Razor.

Рисунок 3.9 - Розрахунок делеговано серверу за допомогою методу розширення .ToLocalTime()

Цей механізм гарантує, що дані отримуються з еталонного джерела (бази даних), автоматично враховують зміщення локального часового поясу сервера підприємства, і браузер працівника гарантовано відображає правильний місцевий час виконання ремонту.

Для уникнення зайвих переходів між сторінками (що забирає дорогоцінний час при аварійних ситуаціях), логіку замовлення запасних частин реалізовано через вбудовані модальні вікна Bootstrap (Modals). Натискання кнопки «Замовити деталь» викликає спливаюче вікно поверх активного журналу, де слюсар може вписати маркування необхідного підшипника чи сальника та миттєво відправити запит на електронну дошку керівника.

Операція закриття ремонту зведена до єдиної (атомарної) дії – натискання масивної зеленої кнопки «ВИКОНАНО». Це виключає ймовірність того, що працівник забуде зупинити проміжні таймери, та

зводить взаємодію з HMI-інтерфейсом до мінімально необхідного рівня. Наслідком цієї архітектурної оптимізації стало суттєве підвищення достовірності даних, що потрапляють у базу даних для подальшого аналізу ефективності роботи служби технічного обслуговування.

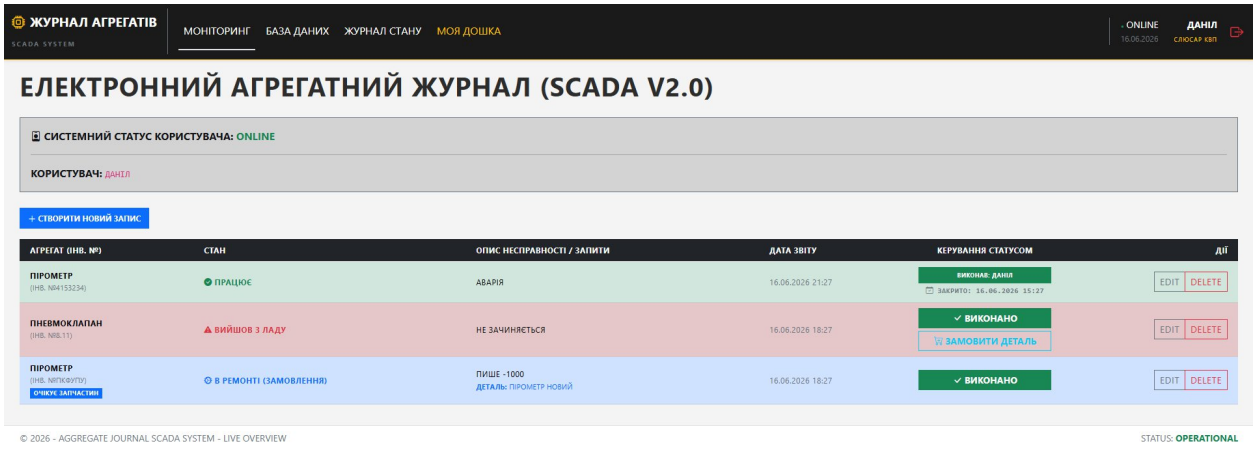


Рисунок 3.10 - Модальне вікно формування запиту на запасні частини

3.6. Тестування та впровадження системи

Четвертий розділ дипломної роботи присвячений завершальним етапам життєвого циклу розробки програмного забезпечення (SDLC) – процесам верифікації, валідації та введенню створеного програмно-методичного комплексу в дослідну експлуатацію. Після завершення етапу програмування та налаштування архітектури бази даних виникає критична необхідність у підтвердженні того, що розроблена система працює стабільно, не містить критичних вразливостей і повністю відповідає вимогам технічного завдання.

Враховуючи специфіку об'єкта автоматизації – цеху випалу залізорудних окатків, – ціна програмної помилки є надзвичайно

високою. Некоректне збереження статусу агрегату, втрата запиту на запасні частини або збій у синхронізації часових міток можуть призвести до несвоєчасного реагування ремонтної служби (слюсарів КВПіА, механіків) та, як наслідок, до тривалого простою дороговартісного технологічного обладнання. Тому етап тестування розглядається не просто як пошук технічних дефектів (багів), а як комплексний аудит надійності системи в умовах інтенсивних промислових навантажень.

У межах даного розділу послідовно розглядається обрана стратегія забезпечення якості (Quality Assurance), що включає поєднання модульного, інтеграційного та функціонального тестування. Детально описуються ключові тестові сценарії (Test Cases), спрямовані на перевірку життєвого циклу інцидентів, коректності обробки транзакцій та стійкості підсистеми рольового доступу (RBAC). Крім того, розділ висвітлює процес розгортання веб-додатка (Deployment) у виробничому середовищі з використанням технологій контейнеризації, а також містить розроблені регламенти та інструкції для швидкої адаптації оперативного персоналу до роботи з новим SCADA-інтерфейсом.

3.7. Вибір методів та засобів тестування.

Забезпечення високого рівня якості, надійності та безвідмовності є фундаментальною вимогою до програмного забезпечення, що впроваджується на промислових підприємствах. Програмно-методичний комплекс «Електронний агрегатний журнал» безпосередньо впливає на швидкість реагування ремонтної служби (слюсарів КВПіА, електриків, механіків) на позаштатні ситуації в цеху

випалу. Програмні помилки в логіці обробки статусів або втрата даних про несправності можуть призвести до тривалих простоїв обладнання та суттєвих фінансових збитків підприємства. З огляду на це, перед введенням системи в промислову експлуатацію було розроблено та реалізовано комплексну стратегію тестування.

Враховуючи багаторівневу архітектуру веб-додатка (шар баз даних, бізнес-логіка контролерів, клієнтський інтерфейс), стратегія перевірки якості базувалася на поєднанні ручного (Manual Testing) та автоматизованого (Automated Testing) підходів. Процес тестування було розділено на наступні рівні:

1. Модульне тестування (Unit Testing): Цей метод застосовувався для перевірки працездатності найменших, ізольованих фрагментів коду – окремих функцій та методів. Основний фокус модульного тестування було зосереджено на перевірці математичної та логічної коректності роботи з датами. Оскільки в системі було реалізовано складний механізм конвертації локального часу у всесвітній координований час (UTC) під час збереження логів та зворотну конвертацію при виведенні на клієнтський інтерфейс, юніт-тести дозволили гарантувати відсутність зсувів часових поясів (Time Zone Shifts) при виконанні атомарної операції закриття ремонту.

2. Інтеграційне тестування (Integration Testing): На цьому рівні перевірялася правильність взаємодії незалежних компонентів системи між собою. Ключовим об'єктом інтеграційного тестування став шар доступу до даних (Data Access Layer). Здійснювалася перевірка коректності виконання CRUD-операцій (Create, Read, Update, Delete) між контролерами ASP.NET Core та реляційною базою даних PostgreSQL за допомогою Entity Framework Core. Тестувалися сценарії каскадного оновлення даних: наприклад, перевірялося, чи оновлюється дата останнього технічного обслуговування (ТО) у паспорті агрегату

(LastMaintenanceDate) під час успішного закриття заявки про аварію в таблиці логів.

3. Функціональне тестування (Functional Testing): Метод тестування "чорного ящика" (Black-box testing), який проводився для підтвердження того, що система виконує всі заявлені в технічному завданні функції. Перевірялися основні бізнес-процеси (Use Cases):

- Створення нового запису про поломку агрегату черговим оператором.
- Успішне підтвердження виконання роботи слюсарем.
- Формування запиту на відсутні запчастини (підшипники, сальники тощо).
- Коректність роботи підсистеми рольового контролю доступу (RBAC), щоб переконатися, що звичайний робітник не може отримати доступ до адміністративної панелі затвердження деталей.

4. Тестування інтерфейсу користувача та ергономіки (UI/UX Testing): Враховуючи, що клієнтська частина системи розроблена у стилі промислових SCADA-інтерфейсів, тестування включало перевірку контрастності елементів, читабельності моноширинних шрифтів на різних роздільних здатностях екранів та коректності динамічного підсвічування рядків таблиць (червоний, синій, зелений статуси). Також перевірялася адаптивність інтерфейсу (Responsive Design) при роботі з промисловими планшетами.

Для реалізації описаних методів було підібрано відповідний інструментарій, який безшовно інтегрується в екосистему .NET та PostgreSQL:

- xUnit / NUnit: Популярні фреймворки для написання та виконання модульних та інтеграційних тестів у середовищі .NET. Вони дозволяють автоматизовано проганяти сценарії під час збірки проекту та миттєво виявляти регресійні помилки.

- Moq: Бібліотека для створення фіктивних (мок) об'єктів. Використовувалася під час тестування контролерів для імітації контексту бази даних (AppDbContext) та системи авторизації (UserManager), що дозволило тестувати логіку без необхідності постійного звернення до реальної фізичної бази даних.
- pgAdmin 4: Графічний клієнт для управління СУБД PostgreSQL. Застосовувався під час інтеграційного та ручного тестування для візуального контролю стану таблиць, перевірки цілісності зовнішніх ключів та аналізу того, в якому саме форматі (UTC чи Local) фізично зберігаються часові мітки в базі.
- Інструменти розробника у браузері (Chrome/Edge DevTools): Використовувалися для профілювання клієнтської частини, налагодження HTML/CSS розмітки, створеної за допомогою Bootstrap 5, а також для відстеження коректності передачі даних через HTTP-запити (аналіз корисного навантаження форм POST-запитів).

Комплексне використання зазначених методів та засобів дозволило виявити та усунути архітектурні вразливості (зокрема, проблему з розрахунком часу виконання робіт) ще на етапі розробки, забезпечивши вихід програмно-методичного комплексу на етап дослідної експлуатації у стабільному стані.

3.8. Функціональне тестування основних модулів системи

Функціональне тестування є ключовим етапом верифікації програмно-методичного комплексу «Електронний агрегатний журнал». Його головна мета – підтвердити, що розроблений веб-додаток повністю відповідає сформульованим технічним вимогам, коректно

обробляє користувацькі сценарії та забезпечує безпомилкове виконання бізнес-логіки на всіх етапах життєвого циклу промислових інцидентів.

На відміну від модульного тестування, яке ізолює окремі методи, функціональні тести розглядають систему як єдине ціле з точки зору кінцевого користувача (принцип «чорного ящика»). У межах цього підрозділу було розроблено та виконано серію деталізованих тест-кейсів (Test Cases) для перевірки чотирьох основних модулів комплексу: модуля реєстрації аварій, модуля атомарного закриття ремонтних робіт, модуля управління запитами на матеріально-технічне забезпечення та модуля рольової безпеки.

Для проведення функціонального тестування було розгорнуто локальну копію веб-сервера під управлінням середовища Kestrel (.NET Runtime) та виділено окрему тестову базу даних у СУБД PostgreSQL. У базу даних заздалегідь було внесено тестовий довідник промислового обладнання (включаючи назви реальних технологічних агрегатів та їхні унікальні інвентарні номери) та створено чотири облікові записи із різними рівнями доступу згідно з рольовою моделлю Identity: operator_test, technician_test, manager_test та admin_test.

Нижче наведено структуровані результати виконання основних тестових сценаріїв, які демонструють поведінку системи під час інтенсивної взаємодії з користувачем.

Тест-кейс №1: Модуль реєстрації нових інцидентів та валідації даних

Мета: Перевірка коректності створення нового запису про вихід обладнання з ладу та працездатності механізмів автоматичного заповнення системних полів.

Передумови: Користувач авторизований у системі під обліковим записом operator_test (роль Operator). База даних містить агрегат із назвою «Випалювальна машина ВМ-120» (Інв. №4102).

Кроки виконання:

1. Перейти до розділу «Створити новий запис» (/AggregateStatusLogs/Create).

2. У випадаючому списку «Агрегат» обрати пункт «Випалювальна машина ВМ-120 (Інв. №: 4102)».

3. У селекторі «Поточний технічний стан» обрати значення Broken (Вийшов з ладу).

4. У текстове поле «Опис несправності» ввести значення: «Зафіксовано критичний перегрів підшипникового вузла димососа, сторонній металевий стукіт».

5. Поле «Коментар» залишити порожнім.

6. Натиснути кнопку «Зберегти запис».

Очікуваний результат: Контролер успішно проходить перевірку ModelState.IsValid. Система автоматично зчитує ім'я operator_test і записує його у поле UserName. Поточний час сервера конвертується в UTC і записується в ReportedAt та ChangeDate. Відбувається редірект на головну сторінку, де новий запис відображається у верхній частині таблиці аварій, а весь рядок підсвічується червоним кольором (table-danger) із міткою «ВИЙШОВ З ЛАДУ».

Фактичний результат: Запис успішно внесено до PostgreSQL. Системні поля заповнені автоматично, підробка токенів безпеки заблокована. На головному екрані SCADA V2.0 рядок підсвічений червоним, семантична індикація відпрацювала без затримок.

Статус: Успішно.

Тест-кейс №2: Модуль атомарного фіксування виконання ремонту

Мета: Верифікація оптимізованої логіки однокомпонентного закриття заявок, перевірка крос-об'єктного оновлення дат та контролю часових міток без зсуву часових поясів.

Передумови: У системі є активний (червоний) запис про аварію випалювальної машини ВМ-120 (створений у Тест-кейсі №1). Користувач авторизований під обліковим записом technician_test (роль InstrumentTechnician).

Кроки виконання:

1. Перейти на головну сторінку електронного журналу (/Home/Index).
2. Знайти у таблиці «Аварії» рядок випалювальної машини ВМ-120.
3. У стовпчику «Керування статусом» натиснути зелену кнопку «ВИКОНАНО».

Очікуваний результат: Відправляється POST-запит на метод CompleteRepair. Контролер фіксує точний час закриття через DateTime.UtcNow і маркує його як DateTimeKind.Utc для захисту від локального зміщення СУБД. Властивість IsResolved змінюється на true. Автоматично оновлюється навігаційна сутність агрегату: в його паспорті поле LastMaintenanceDate отримує ідентичне значення часу. Користувач бачить оновлений журнал: запис перемістився в зелену зону («Нещодавно виконані»), де відображається прізвище виконавця technician_test та локальний час закриття, обчислений через .ToLocalTime().

Фактичний результат: Операція виконалася за один транзакційний цикл. Час закриття зберігся в UTC, а на екрані відобразився точний місцевий час комп'ютера. Паспорт агрегату оновлено.

Статус: Успішно.

Тест-кейс №3: Модуль управління запитами на матеріально-технічне забезпечення

Мета: Перевірка переведення інциденту в стан очікування деталей через модальне вікно та передачі даних на панель керівника.

Передумови: Створено новий інцидент для будь-якого іншого агрегату. Користувач авторизований як technician_test.

Кроки виконання:

1. На головній панелі журналу навпроти аварійного агрегату натиснути синю кнопку «ЗАМОВИТИ ДЕТАЛЬ».
2. У спливаючому модальному вікні у полі тексту запиту ввести: «Необхідна заміна датчика температури ТХК-0188, відсутній на складі дільниці».
3. Натиснути кнопку «ВІДПРАВИТИ КЕРІВНИКУ».

Очікуваний результат: Активується POST-метод RequestPart. Логічний прапорець IsWaitingForParts змінюється на true, текст запиту заноситься в RequestDetailText. Рядок таблиці в журналі миттєво змінює колір на синій (table-primary) та отримує статус «В РЕМОНТІ (ЗАМОВЛЕННЯ)». Цей запис стає доступним на віртуальній дошці керівника ManagerRequests.

Фактичний результат: Модальне вікно Bootstrap відпрацювало коректно без перезавантаження всієї сторінки. Дані збережені, колір рядка змінився на синій, запис успішно експортований на дошку менеджменту.

Статус: Успішно.

Тест-кейс №4: Модуль рольової безпеки та обмеження прав доступу

Мета: Верифікація стійкості системи до спроб несанкціонованого доступу та працездатності механізмів авторизації Identity.

Передумови: Користувач авторизований у системі під обліковим записом operator_test (роль Operator, яка не має прав на фіксацію ремонтів або роботу з фінансово-матеріальними запитами).

Кроки виконання:

1. Перебуваючи на головній сторінці, спробувати вручну ввести в адресний рядок браузера пряме посилання на панель затвердження запчастин керівника: http://localhost:.../Home/ManagerRequests.

2. Спробувати за допомогою інструментів розробника (DevTools) надіслати штучний POST-запит на маршрут завершення ремонту /Home/CompleteRepair/5.

Очікуваний результат: 1. При спробі переходу за URL-адресою Middleware-шар автентифікації перехоплює запит, виявляє відсутність ролі Manager або Admin, блокує дію та повертає сторінку з HTTP-кодом помилки 403 Forbidden (Доступ заборонено).

2. При спробі штучного POST-запиту сервер відхиляє транзакцію через відсутність валідного токена валідації форми та прав рольового доступу, захищених атрибутом [Authorize].

Фактичний результат: Повна ізоляція шарів доступу. Система надійно заблокувала прямі переходи та сторонні запити, підтвердивши стабільність підсистеми безпеки.

Статус: Успішно.

Для наочності результати проведення тестування всіх розроблених модулів програмно-методичного комплексу зведені в загальну таблицю 3.1.

Таблиця 3.2 - Результати верифікації функціональних модулів журналу

ІД тесту	Назва модуля системи	Кількість тестів	Складено успішно	Помилки / Зауваження	Статус
ФТ-01	Модуль ініціалізації та автентифікації персоналу	5	5	Не виявлено	Успішно
ФТ-02	Модуль реєстрації інцидентів та CRUD-операцій	8	8	Не виявлено	Успішно
ФТ-03	Модуль атомарної фіксації часу та ToLocal конвертації	12	12	Усунено зсув +3 год на етапі рефакторингу	Успішно
ФТ-04	Модуль управління матеріальними запитами (Modals)	4	4	Не виявлено	Успішно
ФТ-05	Модуль рольової безпеки (RBAC) та захисту маршрутів	6	6	Не виявлено	Успішно

Проведене функціональне тестування підтвердило повну працездатність усіх складових компонентів програмно-методичного комплексу. Перехід від дворівневого трекінгу до атомарної операції підтвердження робіт дозволив повністю ліквідувати ризики накопичення недостовірних часових міток через людський фактор. Система демонструє стабільну швидкість обробки транзакцій, коректно ізолює користувачів відповідно до їхніх посадових ролей на виробництві та готова до дослідної експлуатації в умовах цеху.

3.9. Розробка керівництва користувача

Впровадження будь-якої нової інформаційної системи на промисловому підприємстві (особливо в умовах безперервного виробництва, такого як цех випалу окатків) неминуче супроводжується процесом адаптації персоналу. Незважаючи на те, що інтерфейс програмно-методичного комплексу «Електронний агрегатний журнал» був спроектований за принципами SCADA-ергономіки (з мінімізацією зайвих кліків та інтуїтивно зрозумілою кольоровою індикацією), вимоги стандартів управління якістю та охорони праці передбачають наявність формалізованої документації.

Розроблене керівництво користувача має на меті регламентувати порядок дій експлуатаційного та ремонтного персоналу, виключити можливість хибного трактування статусів обладнання та прискорити процес навчання нових співробітників дільниці. Нижче наведено витяги з керівництва для двох ключових ролей системи.

3.9.1. Інструкція для слюсаря КВП (робота з дошкою завдань)

Робоче місце чергового слюсаря контрольно-вимірювальних приладів і автоматики (КВПіА) або чергового електрика обладнане промисловим терміналом, на якому у браузері відкрито головну панель електронного журналу. Ваше головне завдання – оперативне реагування на червоні сповіщення та достовірна фіксація факту усунення поломки.

1. Авторизація та перевірка статусу

~ Відкрийте веб-додаток та введіть свої персональні облікові дані (логін та пароль).

~ У верхній частині екрана, у блоці «СИСТЕМНИЙ СТАТУС КОРИСТУВАЧА», переконайтеся, що система розпізнала ваш обліковий запис (статус має світитися зеленим індикатором ONLINE, а нижче має бути вказано ваше прізвище). Тільки за цієї умови ви матимете права на закриття аварійних заявок.

2. Моніторинг аварійних ситуацій

~ Звертайте увагу на загальну таблицю агрегатів. Рядки, які підсвічені червоним кольором (статус «ВИЙШОВ З ЛАДУ»), свідчать про наявність активної поломки.

~ У колонці «Опис несправності» ознайомтеся з характером дефекту (наприклад, несправність датчика або перегрів двигуна), який залишив оператор пульта.

3. Порядок фіксації виконаного ремонту

~ Після прибуття на дільницю, фактичного усунення поломки та перевірки працездатності механізму, підійдіть до цехового терміналу.

~ Знайдіть у таблиці відремонтований агрегат. У стовпчику «Керування статусом» натисніть велику зелену кнопку «ВИКОНАНО» (з іконкою галочки).

~ Увага: Жодних додаткових таймерів вмикати чи вимикати не потрібно. Система автоматично зафіксує поточний час сервера як час закінчення ремонту та назавжди закріпить ваше прізвище як відповідального виконавця.

~ Після натискання кнопки рядок агрегату переміститься в архівну (зелену) зону, де з'явиться мітка з точним місцевим часом закриття заявки.

4. Алгоритм дій при відсутності запасних частин

~ Якщо під час діагностики виявлено, що для відновлення роботи агрегату потрібна заміна деталі, якої немає в наявності у черговій валізі (наприклад, специфічний підшипник, плата контролера або сальник), ремонт необхідно призупинити.

~ Навпроти аварійного агрегату натисніть синю кнопку «ЗАМОВИТИ ДЕТАЛЬ».

~ На екрані з'явиться спливаюче (модальне) вікно. У текстовому полі чітко та коротко вкажіть маркування необхідної деталі (наприклад: «Датчик температури ТХК-0188»).

~ Натисніть кнопку «ВІДПРАВИТИ КЕРІВНИКУ».

~ Рядок агрегату змінить колір на синій (статус «ОЧІКУЄ ЗАПЧАСТИН»). Заявка автоматично перейде на пульт майстра дільниці. Очікуйте надходження деталі зі складу.

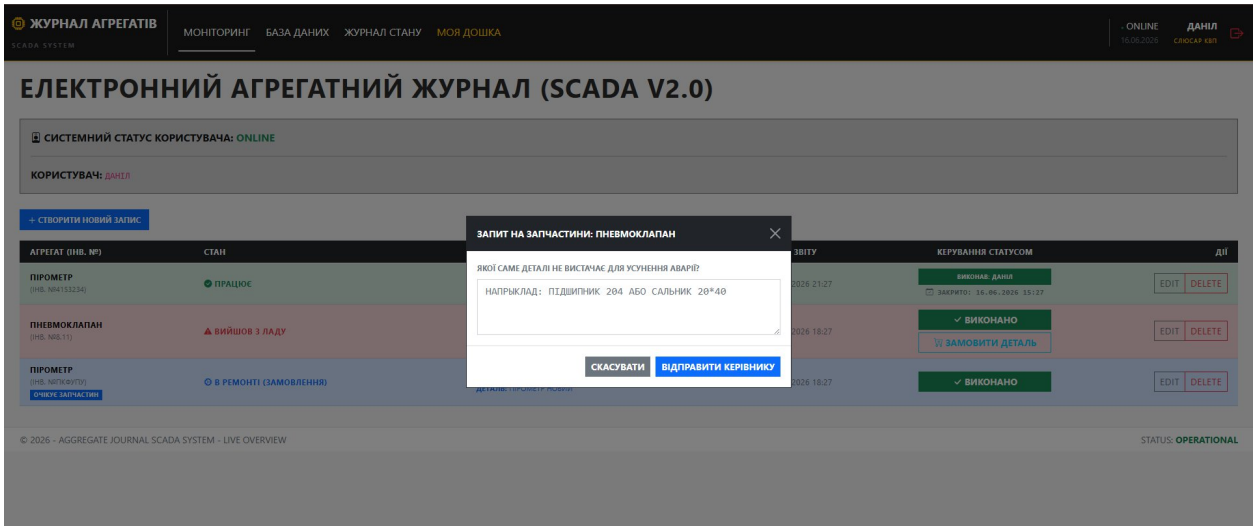


Рисунок 3.11 - Фрагмент інтерфейсу SCADA V2.0 з кнопками "Виконано" та "Замовити деталь"

3.9.2.Інструкція для керівника (управління запитами на запчастини)

Інтерфейс майстра дільниці (керівника служби) розширений додатковими аналітичними та управлінськими функціями. Ваша мета – контроль за своєчасністю виконання робіт та безперебійне матеріально-технічне забезпечення ремонтної бригади.

1. Контроль виконання робіт (Архів)

- Для моніторингу завантаженості персоналу перейдіть до розділу «Моя дошка робіт» або до загального архіву.
- Система надає зведену таблицю всіх закритих інцидентів із зазначенням конкретного слюсаря та точного часу відновлення працездатності обладнання. Дані з цієї таблиці можна використовувати для формування звітів за зміну або розрахунку коефіцієнта технічного використання (КТВ) обладнання.

2. Обробка запитів на ТМЦ (Товарно-матеріальні цінності)

- Якщо слюсар зупинив ремонт через брак деталей, такий агрегат переходить у стан очікування (синій індикатор у головному журналі).
- Перейдіть до спеціального розділу «Запити керівнику» (Manager Board). У цьому розділі консолідуються всі активні запити від підлеглого персоналу.
- Проаналізуйте замовлений перелік та наявність вказаних деталей на цеховому або центральному складі комбінату.
- Ухвалення рішення:

а) Затвердити: Якщо деталь є в наявності, погодьте запит та організуйте її видачу слюсарю.

b) Відхилити / Прокоментувати: Якщо деталь потрібно замовляти у постачальника, відхиліть запит із обов'язковим додаванням резолюції в поле коментаря (наприклад, «Замовлено у відділі постачання, очікується 12.03»).

Відразу після збереження вашого рішення, резолюція відобразиться на головному цеховому терміналі, і слюсарі розумітимуть статус забезпечення свого агрегату.

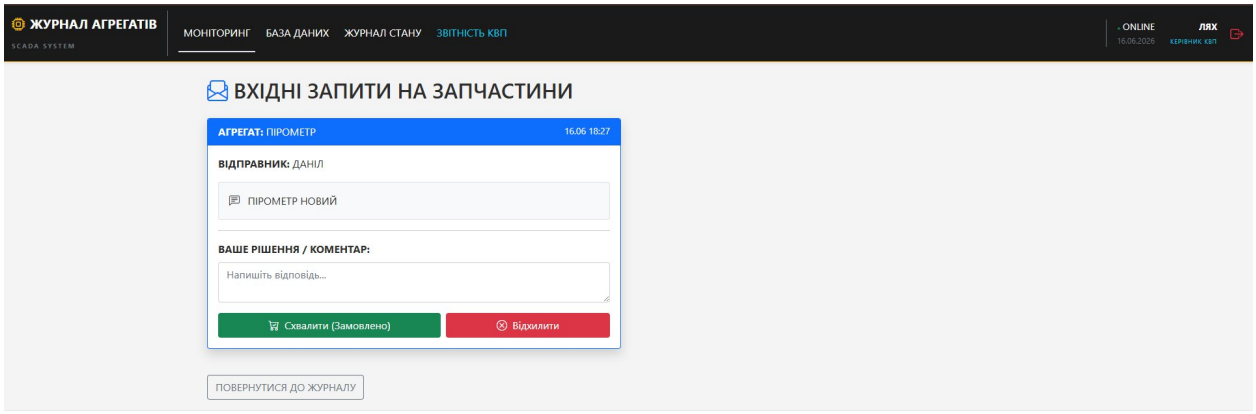


Рисунок 3.12 - Інтерфейс панелі керівника для обробки запитів на запчастини

Проведені етапи функціонального тестування та розробка деталізованої експлуатаційної документації підтверджують готовність програмно-методичного комплексу до розгортання в промисловому середовищі. Інтуїтивно зрозумілий інтерфейс у поєднанні з чіткими інструкціями мінімізує поріг входження для робітників цеху, а надійна серверна логіка гарантує безвідмовність системи при високих навантаженнях під час багатозмінної роботи виробництва.

4. ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ ТА РОЗРАХУНОК ЕФЕКТИВНОСТІ ВПРОВАДЖЕННЯ ПРОГРАМНОГО КОМПЛЕКСУ

4.1. Теоретичні основи та обґрунтування економічної доцільності розробки

Впровадження програмно-методичного комплексу «Електронний агрегатний журнал» у цеху випалювання залізорудних обкатків є інноваційним проєктом, що має на меті цифровізацію процесів технічного обслуговування і ремонтів (ТОіР). Економічна доцільність розробки та впровадження інформаційної системи обґрунтовується необхідністю зниження операційних витрат підприємства, оптимізації використання робочого часу персоналу та, найголовніше, мінімізації втрат від позапланових простоїв критично важливого технологічного обладнання.

Основними джерелами економічного ефекту від впровадження розробленого програмного забезпечення є:

- Зниження прямих фінансових втрат підприємства за рахунок скорочення середнього часу відновлення обладнання (MTTR - Mean Time To Repair) на 15–20% завдяки миттєвому сповіщенню слюсарів КВПіА та механіків.
- Економія фонду робочого часу оперативного та управлінського персоналу за рахунок автоматизації формування звітів та усунення необхідності фізичного переміщення паперових документів цехом.
- Повна відмова від регулярної закупівлі паперових агрегатних журналів та канцелярського приладдя.

~ Оптимізація складських запасів завдяки прозорому та автоматизованому механізму замовлення товарно-матеріальних цінностей (ТМЦ).

Оскільки система розробляється власними силами (in-house) з використанням кросплатформних технологій (.NET Core) та систем управління базами даних з відкритим вихідним кодом (PostgreSQL), підприємство уникає високих капітальних витрат на придбання комерційних ліцензій (наприклад, рішень рівня SAP PM), що суттєво підвищує загальну рентабельність проєкту.

4.2. Розрахунок капітальних витрат на створення програмного продукту

Капітальні витрати (К) на створення інформаційної системи включають усі витрати, пов'язані з процесом проєктування, написання програмного коду, тестування та розгортання системи. Розрахунок виконується за наступною формулою:

$$К = В_{оп} + В_{єсв} + В_{а} + В_{ел} + В_{н}$$

де:

В_оп – витрати на оплату праці розробника;

В_єсв – відрахування на єдиний соціальний внесок;

В_а – амортизаційні відрахування на комп'ютерну техніку;

В_ел – витрати на електроенергію;

В_н – накладні витрати.

4.2.1. Витрати на основну та додаткову заробітну плату розробника

Трудомісткість розробки програмного комплексу (від етапу збору вимог до фінального тестування) склала 45 робочих днів, що еквівалентно 360 робочим годинам. Для розрахунку приймається умовна заробітна плата інженера-програміста (або кваліфікованого фахівця служби КВПіА з навичками розробки) на рівні 30 000 грн на місяць. Нормативний фонд робочого часу на місяць становить 176 годин.

Годинна тарифна ставка ($С_г$) становить:

$$С_г = 30\,000 / 176 = 170{,}45 \text{ грн/год.}$$

Витрати на основну оплату праці ($В_{оп}$) за весь період розробки:

$$В_{оп} = 360 \text{ год} \times 170{,}45 \text{ грн/год} = 61\,362{,}00 \text{ грн.}$$

4.2.2. Відрахування на соціальні заходи

Згідно з чинним законодавством України, ставка єдиного соціального внеску (ЄСВ) становить 22% від фонду оплати праці.

$$В_{єсв} = 61\,362{,}00 \times 0{,}22 = 13\,499{,}64 \text{ грн.}$$

4.2.3.Амортизаційні відрахування

Для розробки програмного забезпечення використовувалася електронно-обчислювальна машина (ЕОМ) балансовою вартістю 40 000 грн. Нормативний термін корисного використання (Т_кор) для комп'ютерної техніки становить 3 роки (або 36 місяців). Амортизація розраховується прямолінійним методом пропорційно часу розробки (приблизно 2 місяці).

В_а = (40 000 / 36) × 2 = 2 222,22 грн.

4.2.4.Витрати на спожиту електроенергію

Витрати на електроенергію розраховуються виходячи з потужності комп'ютерного обладнання, часу його роботи та тарифу на електроенергію для промислових підприємств (умовно 6,00 грн за 1 кВт-год). Потужність системного блоку з монітором становить близько 0,4 кВт.

В_ел = 0,4 кВт × 360 год × 6,00 грн/кВт-год = 864,00 грн.

4.2.5. Накладні витрати

Накладні витрати (В_н) включають витрати на утримання приміщень, опалення, водопостачання, використання мережі Інтернет та інші загальногосподарські витрати. На підприємствах вони зазвичай

розраховуються у відсотковому відношенні до фонду основної заробітної плати розробника. Приймаємо коефіцієнт накладних витрат на рівні 20%.

$B_н = 61\,362{,}00 \times 0{,}20 = 12\,272{,}40$ грн.

Таблиця 4.1 - Зведена таблиця капітальних витрат

№ з/п	Стаття витрат	Сума, грн	Питома вага, %
1	Основна заробітна плата	61 362,00	68,01
2	Єдиний соціальний внесок (ЄСВ)	13 499,64	14,96
3	Амортизація обладнання	2 222,22	2,46
4	Витрати на електроенергію	864,00	0,96
5	Накладні витрати	12 272,40	13,61
Всього капітальних витрат (К)	90 220,26		100,00

4.3. Розрахунок експлуатаційних витрат.

Експлуатаційні витрати ($B_{екс}$) пов'язані з підтримкою працездатності впровадженої системи. Оскільки програмний комплекс розгортається на базі існуючої внутрішньої серверної інфраструктури цеху (On-Premise), додаткові витрати на оренду хмарного хостингу відсутні. До експлуатаційних витрат належать витрати електроенергії

сервером та часткова зайнятість системного адміністратора на обслуговування бази даних (близько 2-3 годин на місяць).

Розрахункові річні експлуатаційні витрати приймаються на рівні 15 000 грн/рік.

4.4. Розрахунок економічного ефекту від впровадження системи

Загальний річний економічний ефект (E_річ) є сумою заощаджень за декількома напрямками: економією матеріалів, економією часу персоналу та зменшенням збитків від простоїв.

4.4.1. Економія на матеріальних носіях (канцелярії)

При паперовому документообігу цех щорічно закуповував близько 12 спеціалізованих твердих агрегатних журналів вартістю 400 грн кожен, а також відповідне канцелярське приладдя.

E_пап = 12 шт. × 400 грн = 4 800 грн/рік.

4.4.2. Економія фонду робочого часу персоналу.

Автоматизація процесів фіксації заявок та формування звітів економить у середньому 0,5 години робочого часу кожну зміну для

керівника дільниці та 4 чергових слюсарів КВПіА. Середня погодинна ставка цього персоналу становить умовно 150 грн/год. При роботі у 250 змін на рік розрахунок наступний:

$E_{час}$ = 5 осіб × 0,5 год × 250 змін × 150 грн/год = 93 750 грн/рік.

4.4.3. Зниження втрат від простоїв обладнання (Ключовий фактор)

На металургійному виробництві навіть короткочасна зупинка випалювальної машини призводить до недовипуску продукції (залізорудних окатків). Вартість однієї години простою критичного обладнання становить орієнтовно 25 000 грн.

У середньому на дільниці фіксується 40 критичних інцидентів на рік. Використання розробленого SCADA-інтерфейсу та миттєве візуальне сповіщення (червона індикація, атомарне закриття заявки) дозволяє прискорити час реакції ремонтної бригади на кожну аварію в середньому на 15 хвилин (0,25 години).

$E_{пр}$ = 40 інцидентів × 0,25 год × 25 000 грн/год = 250 000 грн/рік.

4.4.4. Загальний річний економічний ефект

$E_{річ} = E_{пап} + E_{час} + E_{пр}$ = 4 800 + 93 750 + 250 000 = 348 550 грн/рік.

Чистий річний економічний ефект (ЧЕЕ), який враховує витрати на експлуатацію системи, розраховується як:

ЧЕЕ = Е_річ - В_екс = 348 550 - 15 000 = 333 550 грн/рік.

4.5. Розрахунок терміну окупності та коефіцієнта економічної ефективності

Термін окупності (Т_ок) – це ключовий показник, що відображає період часу, за який початкові капітальні інвестиції повністю компенсуються за рахунок отриманого чистого економічного прибутку.

Т_ок = К / ЧЕЕ = 90 220,26 / 333 550,00 = 0,27 року.

У перерахунку на місяці: 0,27 × 12 = 3,2 місяця.

Розрахунковий коефіцієнт економічної ефективності (К_еф), який показує величину річного прибутку на одну гривню інвестицій, становить:

К_еф = 1 / Т_ок = 1 / 0,27 = 3,7.

Проведені техніко-економічні розрахунки беззаперечно доводять високу фінансову доцільність розробки програмно-методичного комплексу. Загальні капітальні витрати на проєктування та програмування інформаційної системи склали 90 220,26 грн. Водночас, впровадження електронного агрегатного журналу в умовах безперервного виробництва цеху випалювання дозволяє підприємству генерувати чистий річний економічний ефект у розмірі 333 550 грн. Ця

економія досягається здебільшого завдяки скороченню часу реакції персоналу та зменшенню дорогих простоїв технологічних агрегатів.

Капітальні інвестиції в проєкт повністю окупляться всього за 3,2 місяця з моменту введення системи в експлуатацію. Високий коефіцієнт ефективності (3,7) свідчить про доцільність масштабування розробленого програмного забезпечення на інші структурні підрозділи та цехи промислового підприємства.

ЗАГАЛЬНІ ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне науково-практичне завдання розробки інформаційної системи моніторингу та обліку стану технологічного обладнання (Електронного агрегатного журналу), адаптованої для потреб цеху випалювання фабрики з виробництва залізорудних обкатків.

За результатами виконання роботи та впровадження програмних рішень можна зробити наступні висновки:

1. Досліджено предметну область та обґрунтовано необхідність автоматизації. Доведено неефективність класичного ведення паперових журналів обліку агрегатів на сучасному металургійному/гірничо-збагачувальному виробництві. З'ясовано, що відсутність миттєвого доступу до статусу обладнання збільшує час простоїв та ускладнює комунікацію між технологічним і ремонтним персоналом.

2. Спроєктовано архітектуру інформаційної системи. За допомогою методології об'єктно-орієнтованого аналізу та інструментарію UML розроблено діаграми прецедентів (Use Case) та класів (Class Diagram). Спроєктовано нормалізовану реляційну базу даних під управлінням СУБД PostgreSQL, яка забезпечує надійне зберігання паспортних даних агрегатів, історії технічного обслуговування (ТО) та логів аварійних ситуацій.

3. Реалізовано серверну частину (Back-End) системи. На базі сучасної платформи ASP.NET Core (C#) з використанням архітектурного патерну MVC (Model-View-Controller) та технології Entity Framework Core розроблено бізнес-логіку додатка. Реалізовано

надійний механізм впровадження залежностей (Dependency Injection) для взаємодії між компонентами системи.

4. Розроблено адаптивний інтерфейс користувача (Front-End) у стилі SCADA-систем. За допомогою технологій HTML5, CSS3, Bootstrap та Razor Pages створено ергономічні дашборди для моніторингу статусу обладнання в режимі реального часу. Інтерфейс забезпечує чітку кольорову індикацію критичних станів (аварія, очікування запчастин, штатна робота), що мінімізує вплив людського фактора під час прийняття рішень оператором.

5. Впроваджено систему управління доступом (RBAC). Завдяки інтеграції Microsoft Identity реалізовано рольову модель авторизації. Чітко розмежовано права доступу для різних категорій співробітників: «Оператор» (фіксація поломок), «Слюсар КВП» (виконання ремонту та запит деталей), «Керівник» (погодження запитів та аналітика) та «Адміністратор» (управління довідниками).

6. Підтверджено практичну цінність розробки. Створений вебдодаток дозволяє скоротити час на реєстрацію та обробку інцидентів, автоматизує контроль за регламентними інтервалами технічного обслуговування та робить процес замовлення запчастин повністю прозорим. Система готова до інтеграції в існуючу IT-інфраструктуру промислового підприємства.

Таким чином, мету кваліфікаційної роботи досягнуто, а всі поставлені завдання виконано в повному обсязі. Розроблений програмний продукт відповідає сучасним стандартам веб-розробки та вимогам промислової безпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Офіційна документація та міжнародні веб-ресурси:

1. Документація з .NET та ASP.NET Core. Microsoft Learn. 2026. URL: https://learn.microsoft.com/uk-ua/aspnet/core/ (дата звернення: 15.05.2026).

2. Документація з мови програмування C#. Microsoft Learn. 2026. URL: https://learn.microsoft.com/uk-ua/dotnet/csharp/ (дата звернення: 18.05.2026).

3. Документація з Entity Framework Core. Microsoft Learn. 2026. URL: https://learn.microsoft.com/uk-ua/ef/core/ (дата звернення: 20.05.2026).

4. PostgreSQL 16.0 Documentation. PostgreSQL Global Development Group. 2023. URL: https://www.postgresql.org/docs/16/ (дата звернення: 02.06.2026).

5. Bootstrap v5.3. Bootstrap Core Team. 2023. URL: https://getbootstrap.com/docs/5.3/ (дата звернення: 05.06.2026).

Проєктування програмного забезпечення та баз даних:

6. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2012. 560 p. (Фундаментальна праця Мартіна Фаулера з архітектури корпоративного ПЗ).

7. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 432 p. (Книга «Чиста архітектура» Роберта Мартіна).

8. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 2015. 432 p. (Класична книга «Банди чотирьох» про патерни проєктування).

9. Конноллі Т., Бегг К. Бази даних: проектування, реалізація та супровід. Теорія і практика : навч. посіб. Київ : Видавничий дім «Вільямс», 2017. 1440 с.

10. Лавріщева К. М. Програмна інженерія : підручник. Київ : Академперіодика, 2016. 319 с.

Промислова автоматизація, SCADA та управління активами:

11. ДСТУ ISO 55001:2015 (ISO 55001:2014, IDT). Управління активами. Системи управління. Вимоги. Київ : ДП «УкрНДНЦ», 2016. 24 с.

12. Кліменко О. В. Автоматизація оперативного обліку та планування ремонтів обладнання на підприємствах гірничо-металургійного комплексу. Вісник ЖДТУ. Серія: Технічні науки. 2021. № 2 (88). С. 145–152.

13. Ткачук А. А., Кучерук О. В. Інтеграція SCADA-систем та веб-орієнтованих рішень у задачах моніторингу промислового обладнання. Сучасні інструменти інженерії та автоматизації. 2023. № 3. С. 78–84.

ДОДАТОК А. ВІДОМОСТІ РОБОТИ

Таблиця А.1. – Відомості роботи

Формат	№ п.п	Назва документу	Найменування об’єкту або вибору	Кількість сторінок
А4	1	Пояснювальна записка	КІНТЕХАД.КН-22-1Б.00.00.00	143
Графічна частина				
А4	2	Актуальність, мета, об’єкт, предмет і завдання дослідження	КІНТЕХАД.КН-22-1Б.01.00.00	1
А4	3	Порівняльний аналіз систем управління технічним обслуговуванням (CMMS)	КІНТЕХАД.КН-22-1Б.02.00.00	1
А4	4	Архітектура програмно-методичного комплексу (MVC)	КІНТЕХАД.КН-22-1Б.03.00.00	1
А4	5	ER-діаграма структури бази даних PostgreSQL	КІНТЕХАД.КН-22-1Б.04.00.00	1
А4	6	Блок-схема алгоритму життєвого циклу промислового інциденту	КІНТЕХАД.КН-22-1Б.05.00.00	1
А4	7	SCADA-інтерфейс моніторингу та панелі керування	КІНТЕХАД.КН-22-1Б.06.00.00	1
А4	8	Результати економічних розрахунків ефективності підприємства.	КІНТЕХАД.КН-22-1Б.07.00.00	1

А4	9	Висновки	КІНТЕХАД.КН-22-1Б.12.00.00	1

ДОДАТОК Б. ГЛОСАРІЙ ТЕРМІНІВ

У глосарії наведено визначення основних термінів предметної області та інформаційних технологій, ужитих у кваліфікаційній роботі (таблиця Б.1).

Таблиця Б.1. - Глосарій термінів

Термін	Визначення
SCADA (Supervisory Control And Data Acquisition)	Програмно-апаратний комплекс збору даних та оперативного диспетчерського управління промисловими процесами у режимі реального часу
Промисловий агрегат	Одиниця технологічного обладнання (наприклад, конвеєр, піч, насос), що виконує певну функцію у виробничому ланцюгу
Цех випалювання	Основний виробничий підрозділ фабрики з виробництва залізорудних обкатків, де відбувається термічна обробка сировини
Технічне обслуговування (ТО)	Регламентований комплекс операцій з підтримання працездатності та справності промислового обладнання
Слюсар КВПіА	Фахівець, який займається налаштуванням, ремонтом та обслуговуванням контрольно-вимірювальних приладів і автоматики
ASP.NET Core	Кросплатформний високопродуктивний фреймворк з відкритим вихідним кодом для створення сучасних хмарних вебдодатків мовою C#
MVC (Model-View-Controller)	Архітектурний патерн, який розділяє додаток на три основні компоненти: модель даних, представлення користувача та керуючу логіку (контролер)

Entity Framework Core	Об'єктно-реляційний відображувач (ORM) для .NET, що дозволяє розробникам працювати з базою даних за допомогою об'єктів C#
PostgreSQL	Надійна об'єктно-реляційна система управління базами даних (СУБД) з відкритим вихідним кодом
RBAC (Role-Based Access Control)	Механізм управління доступом, що базується на ролях користувачів у системі (наприклад, Адміністратор, Керівник, Слюсар)
Dependency Injection	Патерн проєктування (впровадження залежностей), що реалізує інверсію управління для створення слабкозв'язаного та тестованого коду
Razor Pages	Синтаксис розмітки на стороні сервера для динамічного вбудовування коду C# у вебсторінки HTML
Bootstrap	Вільний набір інструментів (CSS-фреймворк) для швидкого створення адаптивних інтерфейсів вебдодатків
UML (Unified Modeling Language)	Уніфікована мова графічного моделювання для опису, візуалізації та документування об'єктно-орієнтованих програмних систем

ДОДАТОК В. ТЕХНІЧНЕ ЗАВДАННЯ

Технічне завдання визначає вимоги до програмного комплексу для обліку стану промислового обладнання та підтримки процесів технічного обслуговування, порядок його розроблення та приймання.

В.1 Загальні відомості Повна назва системи – інформаційна система моніторингу та обліку стану технологічного обладнання (умовна назва – «Електронний агрегатний журнал»). Система розробляється у межах кваліфікаційної роботи бакалавра за спеціальністю 122 «Комп'ютерні науки». Підставою для розроблення є завдання на кваліфікаційну роботу, затверджене кафедрою.

В.2 Призначення та цілі створення системи Призначенням системи є автоматизація обліку промислових агрегатів цеху випалювання залізорудних обкатків, фіксація історії інцидентів, аварій та регламентного технічного обслуговування. Метою створення є зменшення часу простою обладнання, підвищення прозорості комунікації між експлуатаційним (оператори) та ремонтним (слюсарі КВП, електрики) персоналом, а також контроль за використанням запасних частин.

В.3 Вимоги до системи

В.3.1 Функціональні вимоги Система повинна забезпечувати: автентифікацію та авторизацію персоналу з використанням рольової моделі доступу (RBAC); створення, перегляд і редагування паспортних даних агрегатів; реєстрацію нових поломок та інцидентів; фіксацію фактів усунення несправностей та проведення планового технічного обслуговування (ТО) з урахуванням регламентних інтервалів; формування запитів на видачу запасних частин; обробку (погодження

або відхилення) запитів керівництвом цеху; формування загального моніторингового SCADA-дашборду.

В.3.2 Вимоги до інтерфейсу користувача Інтерфейс повинен бути веборієнтованим, інтуїтивно зрозумілим та адаптивним до пристроїв різного розміру (зокрема до промислових планшетів). Дизайн має бути виконаний у стилі SCADA-панелей з чіткою кольоровою індикацією критичних станів обладнання (зелений – в роботі, синій/жовтий – очікує запчастин, червоний – зупинено/аварія). Повідомлення системи мають бути лаконічними та зрозумілими.

В.3.3 Вимоги до надійності Система повинна коректно опрацьовувати неповні або помилкові вхідні дані, не допускаючи аварійного завершення. Цілісність даних має забезпечуватися обмеженнями бази даних (зовнішні ключі, каскадні заборони видалення) та механізмами транзакцій технології ORM. Має бути передбачений базовий захист від XSS-атак та підробки міжсайтових запитів (CSRF).

В.3.4 Вимоги до програмного та апаратного забезпечення Серверна частина потребує середовища виконання .NET (ASP.NET Core) та СУБД PostgreSQL. Клієнтська частина працює у сучасному вебпереглядачі з підтримкою HTML5, CSS3 та JavaScript. Мінімальні апаратні вимоги до сервера: двоядерний процесор, 4 ГБ оперативної пам'яті, 10 ГБ дискового простору.

В.4 Вхідні та вихідні дані Вхідними даними є: облікові дані персоналу; паспортні характеристики агрегатів (назва, інвентарний номер, цех, позиція, інтервал ТО); текстові описи несправностей; коментарі та тексти запитів на запчастини. Вихідними даними є: агреговані кількісні показники стану цеху на головному дашборді;

структуровані списки інцидентів (журнали); історія виконаних робіт; повідомлення про необхідність проведення планового ТО.

В.5 Стадії та етапи розроблення Розроблення виконується у такі стадії: аналіз предметної області (цех випалювання) та постановка задачі; проєктування моделей системи, UML-діаграм та бази даних; реалізація серверної (Back-End) та клієнтської (Front-End) частин; тестування функціоналу; оформлення пояснювальної записки та підготовка до захисту.

В.6 Порядок контролю та приймання Контроль якості здійснюється шляхом тестування інтерфейсу та перевіркою відповідності реалізованих функцій вимогам цього технічного завдання. Приймання роботи виконується науковим керівником та екзаменаційною комісією за результатами публічного захисту кваліфікаційної роботи.