**РЕФЕРАТ**

Пояснювальна записка складається з 8 розділів:

* вступ – визначає актуальність роботи, її цілі і головні задачі. Складається з 2 сторінок;
* призначення, постановка задачі, огляд аналогів і літератури – містить опис призначення роботи та методики вирішення поставленої задачі, приведені аналоги та огляд літератури. Складається з 16 сторінок;
* зовнішнє та логічне проектування – містить інженерно-технологічну постановку задачі, формалізацію та процес проектування бази даних. Складається з 13 сторінок;
* внутрішнє проектування – містить обґрунтування вибору парадигми проектування, мови програмування, проектування програмного інтерфейсу, архітектури та динаміки системи. Складається з 33 сторінок;
* відлагодження та тестування – містить опис обраних методів тестування та процесу відлагодження. Складається з 14 сторінок;
* охорона праці та безпека в надзвичайних ситуаціях – містить аналіз шкідливих та небезпечних факторів при роботі з ЕОМ, та заходи щодо їх мінімізації. Складається з 12 сторінок;
* висновки – складається з 2 сторінок;
* література – список використаних літературних джерел. Складається з 2 сторінок.

Технічне завдання та робочий проект містяться в додатках.

Кількість таблиць: 65

Кількість рисунків: 22

**ЗМІСТ**

[ВСТУП 7](#_Toc388960171)

[1 ПРИЗНАЧЕННЯ, ПОСТАНОВКА ЗАДАЧІ ТА ОГЛЯД АНАЛОГІВ І ЛІТЕРАТУРИ 9](#_Toc388960172)

[1.1 Призначення та область застосування 9](#_Toc388960173)

[1.2 Постановка задачі 9](#_Toc388960174)

[1.3 Огляд літератури 10](#_Toc388960175)

[1.4 Аналіз аналогів 19](#_Toc388960176)

[1.4.1 Характеристика системи ІРБІС 19](#_Toc388960177)

[1.4.2 Характеристика системи УФД/Бібліотека 21](#_Toc388960178)

[1.4.3 Характеристика системи Alfresco 22](#_Toc388960179)

[1.5 Аналіз розглянутих аналогів 23](#_Toc388960180)

[2 ЗОВНІШНЄ ТА ЛОГІЧНЕ ПРОЕКТУВАННЯ 25](#_Toc388960181)

[2.1 Зовнішнє проектування 25](#_Toc388960182)

[2.1.1 Опис функціональних характеристик 25](#_Toc388960183)

[2.1.2 Формалізація задач 26](#_Toc388960184)

[2.2 Аналіз баз даних 28](#_Toc388960185)

[2.2.1 Аналіз системи керування базами даних 29](#_Toc388960186)

[2.2.2 Сутності та їх представлення в базі даних 29](#_Toc388960187)

[2.2.3 ER-модель бази даних 30](#_Toc388960188)

[2.2.4 Аналіз фізичного проекту бази даних 32](#_Toc388960189)

[2.2.4 Нормалізація бази даних 36](#_Toc388960190)

[3 ВНУТРІШНЄ ПРОЕКТУВАННЯ 38](#_Toc388960191)

[3.1 Вибір парадигми програмування 38](#_Toc388960192)

[3.2 Проектування програмних інтерфейсів 39](#_Toc388960193)

[3.2.1 Проектування RESTful протоколу 41](#_Toc388960194)

[3.2.2 Протокол веб-сервісу 43](#_Toc388960195)

[3.3 Проектування архітектури системи 43](#_Toc388960196)

[3.3.1 Проектування генератору ідентифікаторів 54](#_Toc388960197)

[3.3.2 Проектування підсистеми розподілення прав 56](#_Toc388960198)

[3.3.3 Проектування підсистеми перетворення змісту документів 58](#_Toc388960199)

[3.3.4 Проектування підсистеми відкладеного виконання команд 59](#_Toc388960200)

[3.3.5 Проектування підсистеми доступу до вмісту 61](#_Toc388960201)

[3.4 Проектування динаміки системи 64](#_Toc388960202)

[3.5 Проектування системи на фізичному рівні 67](#_Toc388960203)

[3.6 Вибір мови програмування 69](#_Toc388960204)

[4 ВІДЛАГОДЖЕННЯ ТА ТЕСТУВАННЯ ПРОГРАМИ 71](#_Toc388960205)

[4.1 Вибір стратегії тестування 71](#_Toc388960206)

[4.2 Функції для тестування 73](#_Toc388960207)

[4.3 Розробка тестів методом «білого ящика» 75](#_Toc388960208)

[4.4 Розробка тестів методом «чорного ящика» 78](#_Toc388960209)

[4.5 Відлагодження програми 83](#_Toc388960210)

[5 ОХОРОНА ПРАЦІ 85](#_Toc388960211)

[5.1 Аналіз шкідливих та небезпечних виробничих факторів 85](#_Toc388960212)

[5.2 Проектні заходи 87](#_Toc388960213)

[5.3 Безпека праці при виконанні робіт на ПЕОМ 92](#_Toc388960214)

[5.3.1 Вимоги безпеки праці перед початком роботи на ПЕОМ 92](#_Toc388960215)

[5.3.2 Вимоги безпеки праці під час роботи на ПЕОМ 94](#_Toc388960216)

[5.3.3 Вимоги безпеки праці після закінчення роботи на ПЕОМ 95](#_Toc388960217)

[5.3.4 Вимоги безпеки праці в аварійних ситуаціях 95](#_Toc388960218)

[ВИСНОВКИ 97](#_Toc388960219)

[ЛІТЕРАТУРА 99](#_Toc388960220)

# ВСТУП

Програмний продукт «Модуль інтеграції даних систем аналізу показників бортових систем діагностування локомотивів» є сервісом, що надає програмні інтерфейси для доступу до діагностичних даних бортових систем локомотивів та проводить їх моніторинг.

Продукт є унікальним для ринку через те, що поєднує в собі функції сервісу для надання доступу до показників систем локомотивів та сервера збереження цих показників на безстроковий період.

Системи діагностування бортових систем локомотивів призначені для зменшення вірогідності виходу локомотива зі строю під час експлуатації.

На основі аналізу існуючих бортових систем діагностування локомотивів можна зробити висновок, що їх введення на сьогоднішній день дозволяє:

* підвищити безпеку руху;
* скоротити час виявлення несправності;
* виявити факт відмови контролюючого вузла.

Однією з таких систем є система діагностики «Магістраль-ВЛ11» призначена для автоматичного контролю параметрів технічного стану систем, вузлів і агрегатів магістрального електровоза постійного струму ВЛ11М/6 і забезпечення відображення процесів управління електровозом при експлуатації і в умовах стаціонарного обслуговування.

Розроблений модуль призначений для інтеграції із зазначеною системою, отримання даних бортових систем та збереження в реляційну базу даних. Надалі данні з реляційної бази даних використовуються для надання статистичних даних.

Для отримання подібного функціоналу від існуючих продуктів, потрібно було б придбати декілька продуктів, та провести роботу по їх інтеграції.

В першу чергу, продукт призначений для побудови систем аналізу показників бортових систем діагностування локомотивів, та використанні у зовнішніх програмних комплексах, що надають доступ до показників бортових систем локомотивів кінцевому користувачу. Загалом програмний продукт можна використовувати у будь якій системі яка потребує наявності показників бортових систем локомотивів.

# 1 ПРИЗНАЧЕННЯ, ПОСТАНОВКА ЗАДАЧІ ТА ОГЛЯД АНАЛОГІВ І ЛІТЕРАТУРИ

## 1.1 Призначення та область застосування

Програмний комплекс, що розробляється, призначений отримання та зберігання показників бортових систем локомотивів.

Функціональне призначення продукту полягає у інтеграції із системою «Магістраль-ВЛ11» для отримання статистичних даних показників бортових систем локомотивів, а також наданні сервісів для доступу до отриманих даних та даних про стан сховища. Надання сервісів дозволить створити програмний комплекс що може здійснювати пошук інформації за заданими критеріями та відображати її в зручному для користувача вигляді.

Завдяки універсальності та функціям, програма може бути розповсюджена та використана у організаціях, що потребують у своїй роботі наявності показників бортових систем локомотивів.

## 1.2 Постановка задачі

Необхідно розробити програмний комплекс, що надасть змогу отримувати данні із системи діагностики «Магістраль-ВЛ11» зберігати їх до реляційної бази даних а також надавати програмні інтерфейси для їх аналізу.

Результатом розробки повинен стати сервіс, що надає програмні інтерфейси для обслуговування сховища даних. Кінцевому користувачу повинні бути надані можливості для додавання інформації про показники бортових систем та інформації про локомотиви на яких ведуться виміри. Данні повинні зберігатися в реляційній базі даних. Програма повинна надати можливість для отримання статистичних даних для використання на клієнтській частині. Продукт повинен забезпечувати можливість клієнт-серверної взаємодії.

Продукт має відповідати наступним вимогам: відмово стійкість серверу, забезпечення гнучкої системи доступу до даних.

## 1.3 Огляд літератури

Під час виконання аналізу літератури розглядалися наступні питання: загальна характеристика бортових систем діагностування локомотивів, дослідження технічного стану локомотивів з використанням інформації бортових систем діагностування та проектування архітектури промислових додатків.

Посилення конкуренції між різними видами транспорту вимагає від локомотивного господарства підвищення експлуатаційної надійності локомотивного парку, скорочення часу простою локомотивів в ремонті і витрат на проведення ремонту. Підвищення експлуатаційної надійності повинно досягатись з мінімальними витратами, з точки зору технічного обслуговування це означає максимальне скорочення часу простою локомотивів і зменшення імовірності виходу локомотивів з ладу під час експлуатації. Одним з шляхів вирішення цієї задачі є впровадження на локомотивах засобів технічного діагностування.

Аналіз впровадження і використання систем діагностування на залізничному транспорті за кордоном, впровадження систем оптимізації технічного обслуговування транспортних засобів в інших видах транспорту дозволяє зробити висновок про високу ефективність використання систем діагностування при переході до системи ремонту локомотивів з урахуванням їх експлуатаційної надійності. Сучасний рівень розвитку технічних засобів дозволяє організувати спільну роботу систем управління, діагностування, системи контролю місця знаходження локомотива, систем бездротової передачі інформації на пункти технічного обслуговування локомотивів.

Найбільш перспективним напрямком розвитку діагностичних засобів локомотивів є бортові (вбудовані) системи діагностування локомотивів. Такі системи забезпечують найбільш повну реалізацію ресурсу вузлів і агрегатів локомотивів, попереджають аварійні відмови, знижують експлуатаційні витрати на утримання локомотива. Впровадження вбудованих систем діагностування здійснюється, у першу чергу, для тих вузлів і агрегатів, технічний стан яких впливає на безпеку руху поїздів.

Бортові системи мікропроцесорного діагностування уперше були застосовані в середині 1980-х рр. на тепловозах серії 60 відділення Е1 есtrо моdel (ЕМD) корпорації General Моtоrs і на тепловозах корпорації General Еlесtriс (GЕ). Пізніше фірма Rockwell Іnternational випустила систему аналізу й реєстрації технічного стану локомотивів (LARS), яку залізниця Burlington Northern (США) застосувала на 100 тепловозах SD40-2 і на 50 тепловозах ОР50 фірми ЕМD. Система LARS вимірює до 40 параметрів роботи (або відмов у роботі) локомотива й передає інформацію в пункт технічного обслуговування й ремонту тепловозів. Розглянуті системи діагностування автоматично передають по радіоканалах інформацію з локомотива в наступний пункт технічного обслуговування й ремонту, для того щоб його персонал міг підготуватися до виконання ремонту.

Перевагою бортових систем діагностування є контроль стану локомотива безпосередньо в процесі руху, що дозволяє виявляти причини виникнення й попереджати раптові відмови, виявляти сховані несправності, які можуть проявлятися тільки при деяких режимах роботи локомотива. Бортові системи діагностування підвищують безпека руху завдяки безперервному контролю найбільш відповідальних вузлів локомотива.

Бортові системи діагностування відносяться до нижнього рівня автоматизованих систем. Вони представляють інформацію для систем верхнього рівня (системи автоведения, системи автоматичного керування тяговим приводом, автоматизовані системи безпеки).

Недоліком бортових систем діагностування є їх висока вартість, що обумовлено складними умовами експлуатації рухомого складу, а також забезпеченням придатності вузлів і агрегатів до контролю. Використовувані мікропроцесорні пристрої повинні надійно працювати в умовах значних перепадів температур, вібрації, впливу електромагнітних полів, для чого необхідне використання обладнання спеціального виконання.

Аналіз сучасних систем діагностування вітчизняного і закордонного виробництва показав наявність широких можливостей в плані автоматизації вимірювань і попередньої обробки діагностичного сигналу. Розробка методики діагностування обладнання вимагає рішення ряду завдань, обумовлених безліччю прояву технічних станів механізму у діагностичному сигналі і складністю отримання діагностичної інформації. До числа таких завдань входить дослідження механізму і створення його діагностичної моделі на основі математичного моделювання, вибір діагностичних ознак на основі статистичної обробки діагностичних сигналів, побудова процедури діагностичного аналізу.

В даний час для оцінки поточного рівня надійності обладнання на практиці набули найбільшого поширення основні методи оперативної діагностики:

* метод математичного програмування;
* статистичний метод;
* методи теорії штучного інтелекту.

Методи математичного програмування застосовуються для вирішення багатьох екстремальних задач, з якими досить часто доводиться мати справу в дослідженні технічного стану. Рішення таких завдань зводиться до знаходження крайніх значень (максимуму або мінімуму) деяких функцій змінних величин. Найбільше застосування в дослідженні технічного стану знайшли наступні методи математичного програмування: лінійний, нелінійне, динамічний.

Метод математичного програмування має наступні переваги:

* можливість вибору оптимального варіанту з значної кількості альтернатив;
* висока оперативність отримання результатів рішення за рахунок застосування ЕОМ;
* можливість розв’язання великого класу задач.

Проте цей метод має недоліки:

* метод достатньо трудомісткий і вимагає великого об'єму
* розрахунків;
* необхідність опису альтернативних розв’язків і основних обмежень у вигляді математичних виразів.

Статистичний метод – є основою дослідження технічного стану об’єкта.

Статистичний метод включає наступні методи:

* методи математичної статистики;
* методи теорії імовірності;
* методи статистичного імітаційного моделювання.

Недоліки статистичних методів:

* результати застосування статистичних методів достовірні лише з певною вірогідністю, що задається дослідниками перед початком обробки статистичних даних;
* отримання результатів статистичними методами вимагає обробки великого об'єму статистичних даних;
* важко забезпечити необхідну точність при визначенні статистичних даних із-за недоліків вимірювальної техніки і суб'єктивності дій дослідника.

Переваги методу:

* дозволяє отримувати результати навіть в тих випадках, коли не відомий аналітичний зв'язок між параметрами системи і результатом її функціонування;
* дозволяє описувати і будувати моделі систем практично будь-якої складності.

В порівнянні з традиційними методами дослідження технічного стану нейронні мережі володіють наступними перевагами:

* нейронні мережі не залежать від властивостей вхідних даних, для них не існує вимоги до певного типу розподілу початкових даних, або вимоги до лінійності цільових функцій;
* використання нейронних мережі не вимагає спеціальних знань та спеціальної підготовки для її практичного застосування;
* нейронні мережі здатні моделювати залежності у разі великої кількості змінних.

Недоліки нейронних мереж:

* складність побудови архітектури мережі для конкретного завдання. Для переважної більшості реальних завдань не розроблено стандартних схем нейронних мереж;
* складність інтерпретації результатів навчання;
* значення параметрів елементів нейронних мережі майже завжди неможливо пояснити в термінах розв’язуваного завдання.

Розроблюваний програмний комплекс потребує аналізу сучасних тенденцій в області розробки програмного забезпечення. Сучасні промислові додатки є окремою галуззю розробки прикладного програмного забезпечення, та мають свої особливості в дизайні архітектури[3].

При побудові корпоративних систем, часто має сенс орієнтуватися на можливості для апаратного масштабування, а не потужності або навіть ефективності обчислень. Масштабованість дає можливість підвищення продуктивності, якщо це потрібно, та дозволяє це легше зробити. Часто проектування дозволяє зробити складні речі, які покращують ефективність на конкретному обладнанні, коли насправді, дешевше було б купити додаткове обладнання.

Розшарування є одним з найбільш поширених методів, які розробники програмного забезпечення використовують для спрощення складного програмного забезпечення. Концепція розшарування наведена на рисунку 1.1.

Можна виділити 3 основні шари системи:

* представлення (presentation) – надання послуг, відображення інформації (наприклад, в вікнах або HTML), обробка користувацького вводу;
* галузь (domain) – шар включає в себе реалізацію логіки, що має безпосереднє відношення до задачі програмного забезпечення;
* джерело даних ( data source) – зв'язок з базами даних, системами обміну повідомленнями, менеджерами транзакцій та іншими пакетами.

Переваги розшарування додатків:

* можливість працювати з окремим шаром системи, як незалежною сутністю, без необхідності розбиратися в супутніх компонентах;
* можливість реалізувати декілька варіантів одного шару, та легко змінити використовувану реалізацію. Наприклад, FTP-клієнт не залежить від середовища передачі, і може працювати як із Ethernet-мережами, так і GPRS;
* мінімізація залежностей між компонентами системи. Зміна залежного компоненту не потребує модифікації системи в цілому;
* додаткові рівні абстракції позитивно впливають на комплексне програмне забезпечення.

Недоліки розшарування:

* шари інкапсулюють деякі речі добре, але не всі. Таким чином, іноді необхідні каскадні зміни у всіх шарах, наприклад якщо необхідно відображення нового поля в інтерфейсі користувача, то буде необхідно додати його на всіх шарах абстракцій;
* додаткові шари можуть негативно впливати на ефективність. Як правило, на кожному з шарів необхідно трансформувати значення із одного представлення в інше.

## 1.4 Аналіз аналогів

На сьогоднішній день, аналогів які призначені для тих самих цілей що проект та надають подібний функціонал не знайдено. Це пов’язане з тим, що зазначений програмний комплекс поєднує в собі можливості як завантаження та збереження аналітичних даних так і надання програмних інтерфейсів для їх аналізу, тому, в цьому розділі будуть розглянуті найбільш використовувані систем діагностування локомотивів.

### 1.4.1 Характеристика системи «Магістраль-ВЛ11».

### Система діагностики “Магістраль–ВЛ11” є вимірювально обчислювальним комплексом, призначеним для автоматичного контролю параметрів технічного стану систем, вузлів і агрегатів магістрального електровоза постійного струму ВЛ11М/6 і забезпечення відображення процесів управління електровозом при експлуатації і в умовах стаціонарного обслуговування.

Система забезпечує:

* вимір аналогових і дискретних параметрів:
* температури БПТР;
* температури на вході і виході тягових двигунів ;
* температури букс;
* температура навколишнього середовища ;
* положення запобіжників і автоматів захисту в ланцюгах живлення.

Основні технічні дані системи:

* кількість аналогових вимірювальних каналів - 26;
* кількість дискретних вимірювальних каналів - 54;
* частота опитування датчиків (не більше), Гц - 1;
* місткість пам'яті для реєстрації аварійних ситуацій, Мб - 128;
* напруга живлення, В - 50;
* споживана потужність по ланцюгу 50 В (не більше), Вт - 126;
* пусковий струм по ланцюгу 50 В (не більше), А - 15;
* напрацювання на відмову, тис. годин - 6;
* час підготовки системи до роботи, сек - 60.

### 1.4.2 Характеристика системи FIRE

Система FIRE (Functionally Integrated Railroad Electronics), забезпечує контроль технічного стану локомотива в реальному часі та оперативне реагування на отриману інформацію.

Система забезпечує:

* контроль технічного стану локомотиву в реальному часі;
* оперативне реагування на отриману інформацію;
* отримує 800 сигналів з локомотиву, що сигналізують які визначають його технічний стан;
* присутня можливість отримати рекомендацію по швидкому усуненню несправності, якщо це можливо;
* прогнозування подальшого розвитку несправності;
* можливість зберігання архівної інформації, для отримання інформації про стан приладів за більш великий проміжок часу.

### 1.4.3 Характеристика системи "Магістраль" ДЕ1м

Проаналізувавши систему "Магістраль" ДЕ1м були виділені наступні характеристики:

* статистичні данні зчитуються і зберігаються на стаціонарному комп’ютері;
* перегляд даних у зручному форматі;
* статистика відмов вузлів за заданий період часу;
* друк даних у зручному форматі;
* данні по кожному локомотиву зберігаються в окремій базі;
* база даних має не реляційну структуру та представляє собою бінарний файл.

## 1.5 Аналіз розглянутих аналогів

Серед розглянутих аналогів, найбільш функціональним є система FIRE (Functionally Integrated Railroad Electronics). Це пов'язане з орієнтованістю на велику кількість моделей локомотивів.

З іншого боку, система діагностики «Магістраль–ВЛ11» є системою, працездатність якої перевірена часом у локомотивному господарстві Укрзалізниці.

Ні в одному з аналогів не передбачається можливості ручного керування завантаженням даних до сховища, та наявності web-інтерфейсу для аналізу статистичних даних показників локомотивів.

# 2 ЗОВНІШНЄ ТА ЛОГІЧНЕ ПРОЕКТУВАННЯ

## 2.1 Зовнішнє проектування

### 2.1.1 Опис функціональних характеристик

Дана робота призначена для надання програмних інтерфейсів отримання статистичних даних діагностування локомотивів. Виходячи з постановки задачі, приведена характеристика функціональних вимог до програми, що розробляється.

Програмний продукт повинен надавати такі можливості:

* інтеграція з системою діагностики «Магістраль-ВЛ11» та перенесення інформації до реляційної бази даних;
* надання сервісу для отримання локомотивів що присутні у системі;
* надання сервісу для додавання та видалення інформації про локомотиви;
* надання сервісу для отримання датчиків за якими робляться виміри;
* наданні сервісів для доступу до діагностичних даних бортових систем;
* надання сервісів для перегляду стану бази даних.

Вимоги до інтеграції з системою діагностики «Магістраль-ВЛ11»:

* данні з системи «Магістраль-ВЛ11» повинні надаватися у вигляді бінарного файлу;
* данні можуть завантажуватися як одним так і групою файлів;
* якщо при декодуванні файлу виникла помилка при отриманні показника, відповідне значення в базі даних повинно містити значення null.

Список локомотивів повинен мати наступні данні:

* номер локомотиву;
* назву локомотиву.

Сервіс для додавання та видалення інформації про локомотиви повинен:

* видаляти інформацію по обраному локомотиву з бази даних (номер, назва);
* додавати до бази даних інформацію про локомотив(номер, назва).

Сервіс для отримання датчиків повинен повертати наступну інформацію:

* назва датчика;
* опис датчика;
* одиниці вимірювання.

Сервіс з надання діагностичних даних повинен надавати данні за такими критеріями:

* всі статистичні данні що містяться в базі даних;
* всі статистичні данні до заданої дати;
* всі статистичні данні після заданої дати;
* всі статистичні данні за заданий проміжок часу;
* всі статистичні данні до заданої дати для заданого локомотиву;
* всі статистичні данні після заданої дати для заданого локомотиву;
* всі статистичні данні за заданий проміжок часу для заданого локомотиву.

Сервіс для перегляду стану бази даних повинен надавати данні за такими критеріями:

* фізичний розмір бази даних;
* кількість локомотивів;
* кількість датчиків;
* кількість статистичних даних.

### 2.1.2 Формалізація задач

Формалізація задачі на рівні зовнішнього проектування представлена у вигляді діаграми варіантів використання (Use Case Diagram) [6].

Система, що проектується представлена у вигляді декількох діаграм. Таке рішення дозволило відокремити функції та можливості користувачів із різним рівнем привілеїв. Діаграми варіантів використання системи були розділені на діаграму використання для звичайного користувача системи та користувача із правами адміністратора. На діаграмах користувач представлений у вигляді одного актора (Actor), що взаємодіє з системою за допомогою варіантів використання (Use Case). У якості актора виступає користувач системи. На діаграмі використані наступні типи відношень між варіантами використання та актором:

* відношення асоціації – відображає зв’язок між актором та варіантом використання. Відображається лінією зі стрілкою між актором і варіантом використання;
* відношення розширення – варіант використання розширює базову послідовність дій і додає власну, позначається пунктирною стрілкою з поміткою «extend», тобто такий варіант використання є опціональним, що може розширити базовий варіант;
* відношення включення – показує, що варіант використання включається в базову послідовність, позначається пунктирною стрілкою з поміткою «include», тобто є обов’язковою частиною варіанту використання.

Звичайному користувачу доступні такі варіанти використання (рис: 2.1):

* отримання інформації про локомотиви що присутні у системі;
* отримання інформації про вузли за якими робляться виміри;
* отримання діагностичних даних бортових систем;
* отримання діагностичних даних за проміжок часу;
* отримання діагностичних даних відносно локомотиву;
* отримання усіх діагностичних даних;
* отримання інформації про оновлення даних;
* отримання інформації про оновлення даних за проміжок часу;
* отримання інформації про стан бази даних.



Рисунок 2.1 – Діаграма варіантів використання для користувача системи

Для адміністратора системи доступні такі варіанти використання (рис. 2.2):

* керування інформацією про локомотиви;
  + додавання інформації;
  + видалення інформації;
* керування інформацією про користувачів;
  + додавання нового користувача;
  + видалення користувача;
* додавання статистичних даних;
  + шляхом додавання одного файлу з даними;
  + шляхом додавання декількох файлів з даними;



Рисунок 2.2 – Діаграма варіантів використання для адміністратора систему

## 2.2 Аналіз баз даних

Програмний комплекс, що розробляється, буде використовувати базу даних «LocoStatistic», що призначена для збереження інформації про статистичні данні, локомотиви, вузли за якими ведуться виміри, структури файлу що надається системою «Магістраль-ВЛ11», структуру запису в файлі та інформацію про користувачів.

### 2.2.1 Аналіз системи керування базами даних

У якості серверу баз даних було обрано MySQL – вільно розповсюджувану систему керування реляційними базами даних з відкритим кодом. Це зумовлено підтримкою великої кількості платформ на яких може працювати сервер, його технічними характеристиками та ціною.

MySQL – компактний багатопоточний сервер баз. Характеризується великою швидкістю, стійкістю і простотою використання.

Можливості сервера MySQL:

* простота у встановленні та використанні;
* підтримується необмежена кількість користувачів, що одночасно працюють із БД;
* кількість рядків у таблицях може досягати 50 млн;
* висока швидкість виконання команд;
* наявність простої і ефективної системи безпеки.

MySQL вважається гарним рішенням для малих і середніх застосувань. Сирцеві коди сервера компілюються на багатьох платформах. Найповніше можливості сервера виявляються в UNIX-системах, де є підтримка багатопоточності, що підвищує продуктивність системи в цілому.

### 2.2.2 Сутності та їх представлення в базі даних

Для отримання статистичних даних були розроблені наступні сутності:

* структура файлу (FILE\_STRUCTURE\_INFO) – опис структури файлу: індекс початку записів у файлі, довжина запису, тип файлу;
* структура запису (RECORD\_STRUCTURE\_INFO) – опис структури запису, включає назву вузла що закодовано, індекс у записі, коефіцієнт, регламентує зв'язок між структурою файлу та структурою запису.

Для зберігання статистичних даних були розроблені наступні сутності:

* датчик (SENSOR) – інформація про датчики за якими ведеться вимір статистичної інформації, включає назву, опис, одиниці вимірювання, група до якої належить датчик;
* локомотив (LOCO) – інформація про локомотив: номер та назва;
* данні по локомотиву (LOCO\_DATA) – статистичні данні, включає дату та час коли були додані данні та показники датчиків, регламентує зв'язок з локомотивом.

Для збереження інформації про користувачів системи були розроблені наступні сутності:

* користувач (USER) – зберігає логін, пароль, ім'я та роль користувача у системі.

Для запобігання повторного завантаження файлу, та дублювання даних були розроблені наступні сутності:

* інформація про файл (FILE\_INFO) – зберігає інформацію про файл що було завантажено до системи, містить назву, розмір та дату та час завантаження.

### 2.2.3 ER-модель бази даних

Під час первинного проектування моделі бази даних, за спроектованими сутностями, для формалізації зв'язків між ними було побудовано ER-моделі бази даних.

ER-модель (Entity Relation, зв'язок між сутностями)[7] бази даних – модель даних, яка дозволяє описувати концептуальні схеми за допомогою узагальнених конструкцій блоків. ER-модель - це мета-модель даних, тобто засіб опису моделей даних.

Нижче наведена ER-модель бази «LocoStatistic» наступних сутностей:

* + зберігання статистичної інформації (рис. 2.3);
* отримання статистичної інформації (рис. 2.4);
* зберігання інформації про користувачів (рис. 2.5);
* запобігання повторного завантаження файлу (рис. 2.6).



Рисунок 2.3 – ER модель бази LocoStatistic сутностей зберігання статистичної інформації



Рисунок 2.4 – ER модель бази LocoStatistic сутностей отримання статистичної інформації



Рисунок 2.6 – ER модель бази LocoStatistic сутностей зберігання інформації про користувачів



Рисунок 2.5 – ER модель бази LocoStatistic сутностей запобігання повторного завантаження файлу

### 2.2.4 Аналіз фізичного проекту бази даних

Для того щоб забезпечити збереження спроектованих сутностей в базі даних, необхідно виконати побудову фізичної моделі даних на основі логічних сутностей. На відміну від логічної моделі, яка складається з сутностей, фізична модель враховує особливості розміщення даних в реляційній базі даних.

Нижче наведені таблиці бази даних «LocoStatistic» (таблиці 2.1-2.7) з переліком полів та їх типів.

Таблиця 2.1 – Характеристика таблиці LOCO

|  |  |
| --- | --- |
| LOCO | |
| Назва поля | Тип |
| ID | char(20) |
| TITLE\_LOCO | varchar(512) |

Таблиця 2.2 – Характеристика таблиці SENSOR

|  |  |
| --- | --- |
| SENSOR | |
| Назва поля | Тип |
| NAME | char(50) |
| DESCRIPTION | varchar(1024) |
| UNIT\_OF\_MEASURE | varchar(10) |
| UNIT\_NAME | varchar(10) |

Таблиця 2.3 – Характеристика таблиці LOCO\_DATA

|  |  |
| --- | --- |
| LOCO\_DATA | |
| Назва поля | Тип |
| ID\_LOCO\_DATA | long |
| ID\_LOCO | char(10) |
| RECORD\_TIME\_LOCO\_DATA | timestamp |
| NAPR\_ACCUM\_BATR\_SEC\_A | tinyint |
| NAPR\_ACCUM\_BATR\_SEC\_B | tinyint |
| TOK\_ACCUM\_BATR\_SEC\_A | tinyint |
| TOK\_ACCUM\_BATR\_SEC\_B | tinyint |
| TOK\_VOZ\_TED\_SEC\_A | smallint |
| TOK\_VOZ\_TED\_SEC\_B | smallint |
| TOK\_YAK\_TED12\_SEC\_A | smallint |
| TOK\_YAK\_TED12\_SEC\_B | smallint |
| TOK\_YAK\_TED34\_SEC\_A | smallint |
| TOK\_YAK\_TED34\_SEC\_B | smallint |
| NAPR\_CONT\_SET\_SEC\_A | smallint |
| NAPR\_CONT\_SET\_SEC\_B | smallint |
| TEMPR\_BUKS1\_STOR\_MASH\_SEC\_A | tinyint |
| TEMPR\_BUKS1\_STOR\_POM\_MASH\_SEC\_A | tinyint |
| TEMPR\_BUKS2\_STOR\_MASH\_SEC\_A | tinyint |
| TEMPR\_BUKS2\_STOR\_POM\_MASH\_SEC\_A | tinyint |
| TEMPR\_BUKS3\_STOR\_MASH\_SEC\_A | tinyint |
| TEMPR\_BUKS3\_STOR\_POM\_MASH\_SEC\_A | tinyint |
| TEMPR\_BUKS4\_STOR\_MASH\_SEC\_A | tinyint |
| TEMPR\_BUKS4\_STOR\_POM\_MASH\_SEC\_A | tinyint |
| TEMPR\_BUKS4\_STOR\_POM\_MASH\_SEC\_B | tinyint |
| TEMPR\_BUKS4\_STOR\_MASH\_SEC\_B | tinyint |
| TEMPR\_BUKS3\_STOR\_POM\_MASH\_SEC\_B | tinyint |
| TEMPR\_BUKS3\_STOR\_MASH\_SEC\_B | tinyint |
| TEMPR\_BUKS2\_STOR\_POM\_MASH\_SEC\_B | tinyint |
| TEMPR\_BUKS2\_STOR\_MASH\_SEC\_B | tinyint |
| TEMPR\_BUKS1\_STOR\_POM\_MASH\_SEC\_B | tinyint |
| TEMPR\_BUKS1\_STOR\_MASH\_SEC\_B | tinyint |
| TEMPR\_VHOD\_TED\_SEC\_A1 | tinyint |
| TEMPR\_VHOD\_TED\_SEC\_A2 | tinyint |
| TEMPR\_VHOD\_TED\_SEC\_B1 | tinyint |
| TEMPR\_VHOD\_TED\_SEC\_B2 | tinyint |
| TEMPR\_VIHOD\_TED1\_SEC\_A | tinyint |
| TEMPR\_VIHOD\_TED2\_SEC\_A | tinyint |
| TEMPR\_VIHOD\_TED3\_SEC\_A | tinyint |
| TEMPR\_VIHOD\_TED4\_SEC\_A | tinyint |
| TEMPR\_VIHOD\_TED1\_SEC\_B | tinyint |
| TEMPR\_VIHOD\_TED2\_SEC\_B | tinyint |
| TEMPR\_VIHOD\_TED3\_SEC\_B | tinyint |
| TEMPR\_VIHOD\_TED4\_SEC\_B | tinyint |
| TEMPR\_BPTR1\_SEC\_A | smallint |
| TEMPR\_BPTR2\_SEC\_A | smallint |
| TEMPR\_BPTR1\_SEC\_B | smallint |
| TEMPR\_BPTR2\_SEC\_B | smallint |

Таблиця 2.4 – Характеристика таблиці FILE\_INFO

|  |  |
| --- | --- |
| FILE\_INFO | |
| Назва поля | Тип |
| FILE\_NAME | varchar(50) |
| FILE\_SIZE | integer |
| LOADING\_DATE | timestamp |

Таблиця 2.5 – Характеристика таблиці FILE\_STRUCTURE\_INFO

|  |  |
| --- | --- |
| FILE\_STRUCTURE\_INFO | |
| Назва поля | Тип |
| ID | long |
| RECORD\_START\_INDEX | smallint |
| RECORD\_LENGTH | smallint |
| TYPE | varchar(10) |

Таблиця 2.6 – Характеристика таблиці RECORD\_STRUCTURE\_INFO

|  |  |
| --- | --- |
| RECORD\_STRUCTURE\_INFO | |
| Назва поля | Тип |
| NAME | char(50) |
| INDEX | smallint |
| DELTA | numeric(9,6) |
| FSI\_TYPE | varchar(10) |

Таблиця 2.7 – Характеристика таблиці USER

|  |  |
| --- | --- |
| USER | |
| Назва поля | Тип |
| LOGIN | char(20) |
| PASSWORD | char(32) |
| NAME | char(100) |
| ROLE | char(20) |

### 2.2.4 Нормалізація бази даних

Після проектування бази даних було виконаю етап її нормалізації за трьома нормальними формами:

* Перша нормальна форма:
* кожна таблиця повинна мати основний ключ: мінімальний набір колонок, які ідентифікують запис;
* уникнення повторень груп (категорії даних, що можуть зустрічатись різну кількість разів в різних записах) правильно визначаючи не ключові атрибути;
* атомарність: кожен атрибут повинен мати лише одне значення, а не множину значень;
* Друга нормальна форма:
* схема бази даних повинна відповідати вимогам першої нормальної форми;
* дані, що повторно з'являються в декількох рядках виносяться в окремі таблиці;
* Третя нормальна форма:
* схема бази даних повинна відповідати всім вимогам другої нормальної форми;
* будь-яке поле, що залежить від основного ключа та від будь-якого іншого поля, має виноситись в окрему таблицю.

Для забезпечення 1НФ для таблиці SENSOR поле INFO було розділене на 2 окремі поля DESCRIPTION та UNIT\_OF\_MEASURE (див. рис. 2.6).



Рисунок 2.6 – Зміни в моделі для забезпечення НФ 1

Для того, щоб база даних відповідала (2НФ) інформацію про локомотив до якого відносяться статистичні данні було винесено до таблиці LOCO (рис. 2.7).



Рисунок 2.7 – Зміни в моделі для забезпечення НФ 2

# 3 ВНУТРІШНЄ ПРОЕКТУВАННЯ

## 3.1 Вибір парадигми програмування

Парадигма програмування визначає те, як програміст розглядає роботу програми. Наприклад, в об'єктно-орієнтованому програмуванні програміст розглядає програму як множину взаємодіючих об'єктів, в той час як у функціональному програмуванні програму можна представити як послідовність обчислення функцій без станів.

Парадигми програмування відображають різні аспекти діяльності програмістів по розробці програм. Одні парадигми розвиваються незалежно, інші з'являються в результаті поєднання різних концептуальних ідей. На сьогоднішній час існує велика кількість парадигм, серед яких можна виділити чотири парадигми, які відіграють основну роль у всіх напрямках і стилях сучасного програмування і називаються основними. Це – процедурна, функціональна, алгебраїчна і логічна. Вони відрізняються допустимими засобами представлення даних і алгоритмів у програмах, моделями обчислень, і в поєднанні з іншими ідеями, використовуються для утворення всієї множини парадигм.

Парадигма програмування в одних випадках «природно» визначається самою задачею, наприклад, об'єктно-орієнтований підхід при створенні пакетів графічних програм, в інших – її вибір може бути не таким очевидним і потребує значної роботи щодо обґрунтування.

Задача з розробки модулю інтеграції даних систем аналізу показників бортових систем діагностування локомотивів що поставлена в рамках дипломного проекту відноситься до задач що добре вкладається в концепцію об’єктно-орієнтованого програмування.

Об'єктно-орієнтоване програмування – одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою. Основу ООП складають три основні концепції: інкапсуляція, успадкування та поліморфізм. Одною з переваг ООП є краща модульність програмного забезпечення (тисячу функцій процедурної мови, в ООП можна замінити кількома десятками класів із своїми методами). Сьогодні багато мов програмування або підтримують ООП або ж є цілком об'єктно-орієнтованими. В даний час кількість прикладних мов програмування, що реалізують об'єктно – орієнтовану парадигму, є найбільшим по відношенню до інших парадигм.

На відміну від традиційних поглядів, коли програму розглядали як набір підпрограм, або як перелік інструкцій комп'ютеру, ООП програми можна вважати сукупністю об'єктів. Відповідно до парадигми об'єктно-орієнтованого програмування, кожний об'єкт здатний отримувати повідомлення, обробляти дані, та надсилати повідомлення іншим об'єктам. Кожен об'єкт – своєрідний незалежний автомат з окремим призначенням та відповідальністю[2].

Основні переваги ООП для практичного програмування наступні[8]:

* наявність добре узгодженої мови проектування;
* наявність якісного середовища проектування;
* можливість керування якістю об‘єктно-орієнтованого проекту;
* можливість розробляти великі програмні проекти, розбиваючи великі задачі на невеликі, незалежні і легко доступні до огляду частини, що дозволяє залучати до розробки велику кількість виконавців;
* широкі можливості з повторного використання коду (більш спеціалізовані функції можуть бути написані за рахунок додавання частин, що роблять їх унікальними; загальна частина успадковується).

Об’єктно-орієнтований підхід дозволяє використати ORM(Object-relational mapping, Об'єктно-реляційна проекція) підхід до організації роботи із БД, що підвищує швидкість розробки, та зрозумілість реалізації, через те, що програмна модель оперує сутностями що зберігає модель збереження.

Також ООП надає можливість використовувати компонентно-орієнтований підхід до архітектури додатку, що дозволяє розділити програму на повністю незалежні модулі, комунікація між якими виконується лише через інтерфейси, для зменшення зв’язності кодової бази та простоти внесення змін.

## 3.2 Проектування програмних інтерфейсів

Розроблюваний комплекс відноситься до класу додатків, що надають зовнішньому програмному забезпеченню інтерфейси за якими воно зможе взаємодіяти із програмним комплексом.

Інтерфейс взаємодії «додаток-додаток» називається API (application programming interface, прикладний програмний інтерфейс).

API – набір визначень взаємодії різнотипного програмного забезпечення. API – це зазвичай метод абстракції між низькорівневим та високорівневим програмним забезпеченням.

API визначає функціональність, яку надає програма (модуль, бібліотека), при цьому інтерфейс API дозволяє абстрагуватися від того, як саме ця функціональність реалізована.

Одним з найпоширеніших призначень API є надання набору широко використовуваних функцій, наприклад для малювання вікна чи іконок на екрані. Програмісти використовують переваги API у функціональності, таким чином їм не доводиться розробляти все з нуля. API є абстрактним поняттям – програмне забезпечення, що пропонує деякий API, часто називають реалізацією даного API. У багатьох випадках API є частиною набору розробки програмного забезпечення, водночас, набір розробки може включати як API, так і інші інструменти/апаратне забезпечення, отже ці два терміни не є взаємозамінювані.

При проектуванні мережевого API необхідно вирішити 2 проблеми: метод передачі даних, та протокол взаємодії.

При прийнятті рішення про вибір механізму мережевого транспорту було проаналізовано декілька підходів до реалізації мережевої взаємодії між компонентами. Деякі з них:

* COM RPC – видалений виклик процедур в межах COM-компонентів;
* CORBA – інтерфейс взаємодії розподілених систем, що написані різними мовами та працюють у різних вузлах мережі;
* .NET Remoting/Java RMI – інтерфейси мережевої передачі об'єктів для платформ .NET та Java;
* веб-сервіси;
* мережеві сокети.

Із існуючих підходів, на даний момент найзручнішим є використання веб-сервісів, через наявність готових засобів для роботи із трафіком що передається за протоколом HTTP. Використання протоколу HTTP дозволяє використати із розробленим додатком вже існуючі засоби для моніторингу, маршрутизації та забезпечення відмовостійкості, що використовуються для звичайних веб сайтів. Також використання веб-технологій дозволяє використовувати протокол HTTPS (HyperText Transfer Protocol Secure, безпечний протокол передачі гіпертексту), що надає можливість для захисту трафіку засобами шифрування.

Іншою причиною для використання HTTP є те, що він не обмежує користувача мовою програмування – клієнти для роботи із HTTP існують для усіх більш-менш розповсюджених мов програмування.

Існує декілька протоколів для реалізації веб-сервісів:

* XML-RPC – використовує XML для кодування своїх повідомлень і HTTP в якості транспортного механізму. Всі посилання ідуть на один URI, запит кодує ім’я процедури та параметри;
* SOAP – протокол обміну структурованими повідомленнями в розподіленої обчислювальної середовищі. Всі посилання ідуть на один URI, запит кодує ім’я процедури та параметри, формат виклику є стандартизованим, та описується WSDL-схемою;
* REST – REpresentive State Transfer (передача стану представлення). Метод за яким URI повністю кодує команду, а дані запиту лише зберігають корисну інформацію для команди.

Реалізація протоколу за архітектурою REST є найбільш зручною для кінцевого користувача, тому що URI ресурсів однозначно ідентифікують запит, та спрощують роботу із сервісом. Іншою перевагою REST є «природня» інтеграція із об'єктно-орієнтованою парадигмою програмування що обрана для реалізації програмного комплексу.

Таким чином, програмний комплекс буде надавати список ресурсів за яким користувачі сервісу зможуть взаємодіяти за допомогою звичайних HTTP-запитів, уточнюючи тип запиту за допомогою HTTP-методів, та передаваючи сервісу однотипні об'єкти.

Для сучасних веб технологій, найбільш характерним для передачі даних є формати XML та JSON.

XML – запропонований консорціумом World Wide Web (W3C) стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками. Є спрощеною підмножиною мови розмітки SGML. XML документ складається із текстових знаків, і придатний до читання людиною.

Стандарт XML визначає набір базових лексичних та синтаксичних правил для побудови мови описання інформації шляхом застосування простих тегів. Цей формат достатньо гнучкий для того, аби бути придатним для застосування в різних галузях. Запропонований стандарт визначає метамову, на основі якої, шляхом запровадження обмежень на структуру та зміст документів визначаються специфічні, предметно-орієнтовані мови розмітки даних. Ці обмеження описуються мовами схем, таких як XML Schema (XSD), DTD або RELAX NG.

JSON це легкий формат обміну даними. JSON базується на тексті, і може бути з легкістю прочитаним людиною. Формат дозволяє описувати об'єкти та інші структури даних. Цей формат головним чином використовується для передачі структурованої інформації через мережу. Його головне призначення у написанні веб-програм, а саме при використанні технології AJAX. JSON виступає як заміна XML під час асинхронної передачі структурованої інформації між клієнтом та сервером. При цьому перевагою JSON перед XML є те, що він дозволяє складні структури в атрибутах та займає менше місця.

За рахунок своєї лаконічності в порівнянні з XML, формат JSON було обрано форматом повідомлень при роботі модулю.

### 3.2.1 Проектування RESTful протоколу

REST (Representational state transfer) - це стиль архітектури програмного забезпечення для розподілених систем, таких як World Wide Web, який, як правило, використовується для побудови веб-служб. Системи, що підтримують REST, називаються RESTful-системами.

У загальному випадку REST є дуже простим інтерфейсом управління інформацією без використання якихось додаткових внутрішніх прошарків. Кожна одиниця інформації однозначно визначається глобальним ідентифікатором, таким як URL. Кожна URL в свою чергу має строго заданий формат.

Відсутність додаткових внутрішніх прошарків означає передачу даних в тому ж вигляді, що й самі дані.

Кожна одиниця інформації однозначно визначається URL - це означає, що URL по суті є первинним ключем для одиниці даних. Тобто наприклад третя книга з книжкової полиці буде мати вигляд / book / 3, а 35 сторінка в цій книзі - / book/3/page/35. Звідси і виходить строго заданий формат. Причому абсолютно не має значення, в якому форматі знаходяться - це може бути і HTML, і відсканована копія у вигляді jpeg-файлу, і документ Microsoft Word.

Як відбувається управління інформацією сервісу - цілком і повністю ґрунтується на протоколі передачі даних. Найбільш поширеним протоколом є HTTP. Для HTTP дія над даними здійснюється за допомогою методів: GET (отримати), PUT (додати, замінити), POST (додати, змінити, видалити), DELETE (видалити). Таким чином, дії CRUD (Create-Read-Updtae-Delete) можуть виконуватися як з усіма 4-ма методами, так і лише за допомогою GET і POST.

Як видно, REST архітектура є дуже простою у плані використання. По запиту що надійшов відразу можна визначити, що він робить, не розбираючись в форматах (на відміну від SOAP, XML-RPC). Дані передаються без застосування додаткових шарів, тому REST вважається менш ресурсоємним, оскільки не треба розбирати запит щоб зрозуміти що він повинен зробити і не треба переводити дані з одного формату в іншій.

Таблиця 3.1 – Призначення веб-методів в REST

|  |  |  |  |
| --- | --- | --- | --- |
| Метод | Ресурс | Тіло запиту | Призначення |
| GET | /resources | - | отримання всіх існуючих ресурсів |
| GET | /resources/1 | - | отримання ресурсу з ідентифікатором «1» |
| PUT | /resources | ресурс | створення нового ресурсу |
| POST | /resources/1 | ресурс | оновлення значення ресурсу «1» |
| DELETE | /resources/1 | - | видалення ресурсу з ідентифікатором «1» |

Для ідентифікації вхідного формату запиту буде використовуватися HTTP-заголовок «Content-Type», що є стандартним заголовком HTTP:

* для JSON: application/json.

Важливою частиною протоколу є обробка помилок. Формат відповіді про помилку має бути стандартизованим, та не залежати від запиту.

## 3.3 Проектування архітектури системи

Для первинної класифікації був обраний метод оснований на неформальному описі програми – метод CRC-карток [9]. CRC-картки містять наступну інформацію:

* базовий клас;
* похідні класи;
* відповідальність класу;
* зв’язки з іншими класами.

У якості зв’язків показані класи, які використовуються описаним класом.

Були розроблені CRC-картки для класів програми (табл. 3.2 – 3.35).

Таблиця 3.2 – CRC-картка для класу AboutController

|  |  |
| --- | --- |
| AboutController | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Обробка запитів щодо інформації про розробників |  |

Таблиця 3.3 – CRC-картка для класу DashboardController

|  |  |
| --- | --- |
| DashboardController | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Обробка запитів до вузлів |  |

Таблиця 3.4 – CRC-картка для класу DataManagmentController

|  |  |
| --- | --- |
| DataManagmentController | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Обробка запитів до статистичних даних | LocomotiveService, SensorService |

Таблиця 3.5 – CRC-картка для класу DbInfoController

|  |  |
| --- | --- |
| DbInfoController | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Обробка запитів про стан бази даних | LocomotiveService, StatisticService,  SensorService |

Таблиця 3.6 – CRC-картка для класу DownloadDataController

|  |  |
| --- | --- |
| DownloadDataController | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Обробка запитів щодо завантаження даних до системи | LocomotiveService, FileStructureInfoDao,  DataLoadingService, FileCheckInService |

Таблиця 3.7 – CRC-картка для класу LocomotiveController

|  |  |
| --- | --- |
| LocomotiveController | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Обробка запитів до локомотивів | LocomotiveService |

Таблиця 3.8 – CRC-картка для класу UsersController

|  |  |
| --- | --- |
| UsersController | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Обробка запитів до користувачів | UserAuthManager |

Таблиця 3.9 – CRC-картка для класу UserManagerController

|  |  |
| --- | --- |
| UserManagerController | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Обробка запитів щодо керування користувачами | UserAuthManager |

Таблиця 3.10 – CRC-картка для класу CustomAuthenticationProvider

|  |  |
| --- | --- |
| CustomAuthenticationProvider | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Аунтефікація користувача | UserAuthManager |

Таблиця 3.11 – CRC-картка для класу DataLoadingServiceImpl

|  |  |
| --- | --- |
| DataLoadingServiceImpl | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Дешифрування файлу з даними | FileStructureInfoDao, RecordStructureInfoDao |

Таблиця 3.12 – CRC-картка для класу FileCheckInServiceImpl

|  |  |
| --- | --- |
| FileCheckInServiceImpl | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Перевірка на повторне завантаження даних | FileInfoDao |

Таблиця 3.13 – CRC-картка для класу LocomotiveServiceImpl

|  |  |
| --- | --- |
| LocomotiveServiceImpl | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Робота з керування даними про локомотиви | LocomotiveDao |

Таблиця 3.14 – CRC-картка для класу SensorServiceImpl

|  |  |
| --- | --- |
| SensorServiceImpl | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Робота з керування даними про датчики | SensorDao |

Таблиця 3.15 – CRC-картка для класу StatisticServiceImpl

|  |  |
| --- | --- |
| StatisticServiceImpl | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Робота з керування статистичними даними | DataDao, LocomotiveService |

Таблиця 3.16 – CRC-картка для класу UserAuthManagerImpl

|  |  |
| --- | --- |
| UserAuthManagerImpl | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Автентифікація користувача |  |

Таблиця 3.17 – CRC-картка для класу DataDaoImpl

|  |  |
| --- | --- |
| DataDaoImpl | |
| Базовий класс | Похідні класи |
| JdbcDaoSupport | - |
| Відповідальність | Зв’язки |
| Отримання та збереження статистичних даних з бази даних |  |

Таблиця 3.18 – CRC-картка для класу FileInfoDaoImpl

|  |  |
| --- | --- |
| FileInfoDaoImpl | |
| Базовий класс | Похідні класи |
| JdbcDaoSupport | - |
| Відповідальність | Зв’язки |
| Збереження та отримання інформації про файл з даними з бази даних |  |

Таблиця 3.19 – CRC-картка для класу FileStructureInfoDaoImpl

|  |  |
| --- | --- |
| FileStructureInfoDaoImpl | |
| Базовий класс | Похідні класи |
| JdbcDaoSupport | - |
| Відповідальність | Зв’язки |
| Отримання інформації про структуру файлу з даними |  |

Таблиця 3.20 – CRC-картка для класу RecordStructureInfoDaoImpl

|  |  |
| --- | --- |
| RecordStructureInfoDaoImpl | |
| Базовий класс | Похідні класи |
| JdbcDaoSupport | - |
| Відповідальність | Зв’язки |
| Отримання інформації про структуру запису у файлі з даними |  |

Таблиця 3.21 – CRC-картка для класу SensorDaoImpl

|  |  |
| --- | --- |
| SensorDaoImpl | |
| Базовий класс | Похідні класи |
| JdbcDaoSupport | - |
| Відповідальність | Зв’язки |
| Зберігання та отримання інформації про датчики за бази даних |  |

Таблиця 3.22 – CRC-картка для класу UserDaoImpl

|  |  |
| --- | --- |
| UserDaoImpl | |
| Базовий класс | Похідні класи |
| JdbcDaoSupport | - |
| Відповідальність | Зв’язки |
| Редагування інформації  про користувача в базі даних |  |

Таблиця 3.23 – CRC-картка для класу FileInfo

|  |  |
| --- | --- |
| FileInfo | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Сутність що представляє таблицю в базі даних |  |

Таблиця 3.24 – CRC-картка для класу FileStructureInfo

|  |  |
| --- | --- |
| FileStructureInfo | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Сутність що представляє таблицю в базі даних |  |

Таблиця 3.25 – CRC-картка для класу LocoDataEntity

|  |  |
| --- | --- |
| LocoDataEntity | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Сутність що представляє таблицю в базі даних |  |

Таблиця 3.26 – CRC-картка для класу LocoEntity

|  |  |
| --- | --- |
| LocoEntity | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Сутність що представляє таблицю в базі даних | StorageManager, ContentPersister |

Таблиця 3.27 – CRC-картка для класу RecordStructureInfo

|  |  |
| --- | --- |
| RecordStructureInfo | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Сутність що представляє таблицю в базі даних |  |

Таблиця 3.28 – CRC-картка для класу SensorEntity

|  |  |
| --- | --- |
| SensorEntity | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Сутність що представляє таблицю в базі даних |  |

Таблиця 3.29 – CRC-картка для класу UserEntity

|  |  |
| --- | --- |
| UserEntity | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Сутність що представляє таблицю в базі даних | AuthenticationBean |

Таблиця 3.30 – CRC-картка для класу CustomException

|  |  |
| --- | --- |
| CustomException | |
| Базовий класс | Похідні класи |
| Exception | DataAccessException, ValidationException |
| Відповідальність | Зв’язки |
| Базовий клас для помилок |  |

Таблиця 3.31 – CRC-картка для класу DataAccessException

|  |  |
| --- | --- |
| DataAccessException | |
| Базовий класс | Похідні класи |
| CustomException | - |
| Відповідальність | Зв’язки |
| Описує помилку при доступі до даних |  |

Таблиця 3.35 – CRC-картка для класу ValidationException

|  |  |
| --- | --- |
| ValidationException | |
| Базовий класс | Похідні класи |
| CustomException | - |
| Відповідальність | Зв’язки |
| Описує помилку при перевірці даних |  |

За первинною класифікацією розробляється діаграма класів (Class Diagram)[x]. Діаграма класів — статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення. Діаграма класів, також, може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм.

При проектуванні діаграми класів програми (рис. 3.1) були використані наступні шаблони проектування [12]:

* MVC - шаблон поділяє роботу веб-додатку на три окремі функціональні ролі: модель даних (model), користувальницький інтерфейс (view) і керуючу логіку (controller). Таким чином, зміни, що вносяться в один з компонентів, надають мінімально можливий вплив на інші компоненти. Шаблон був застосований при проектуванні програмного інтерфейсу додатку, а саме – контролерів веб-запитів, що не залежать від механізму доставки запитів та їх формату (рис. 3.1);

mvc.emf

Рис. 3.1 – Діаграма класів реалізації MVC

* шаблон Interface – визначає список операцій що повинен забезпечити деякий об'єкт, без визначення реалізації. Шаблон дозволяє просто виконувати заміну компонентів системи, без потреби в модифікації алгоритмів в різних частинах системи. Також є актуальним при виконання модульних тестів за допомогою об'єктів заглушок. Даний шаблон використано для реалізації класів менеджерів: CollectionManager, DocumentManager, SectionManager та MetadataManager, абстрагувавши контролери від їх реалізації;
* шаблон Dependency Injection (впровадження залежності) – підхід до організації взаємодії компонентів, що дозволяє розділити процес використання компонентів, від процесу їх отримання. При такому підході, задача конфігурації компонентів покладається на зовнішнє керуюче середовище (у випадку проектованого проекту – Spring Framework IOC Container), що забезпечує передачу реалізацій компонентів від яких залежить система. Шаблон використовується в усіх шарах додатку;
* шаблон Видавник/Підписник (Publisher/Subscriber) – дозволяє абстрагувати відправлення повідомлення одержувачу, за допомогою черги повідомлень. Шаблон використано при реалізації внутрішньої шини подій репозиторію, в межах DocumentManager, та підсистеми відкладеного виконання команд (рис. 3.2). Використання шаблону дозволило розділити інтерфейс взаємодії компонентів що потребують створення відкладених задач, від компоненту призначеного для виконання цих задач;

pub_sub.emf

Рисунок 3.2 – Діаграма класів для реалізації схеми Видавник-Підписник

* шаблон Service Layer [12] – визначає для програми кордон і набір допустимих операцій з точки зору взаємодіючих з ним клієнтських. Він інкапсулює бізнес-логіку програми, керуючи транзакціями і керуючи відповідями в реалізації цих операцій. Даний шаблон використано для реалізації класів менеджерів: CollectionManager, DocumentManager, SectionManager та MetadataManager. Його використання довзолило відділити бізнес-логіку від механізму збередення даних від класів що відповідають за збереження даних (рис. 3.3). Таке рішення дозволило зменшити функціональну зв'язність між вузлами, та зробити додаток більш модульним, що позитивно вплило на функціональну зв’язність компонентів.

cld-1.emf

Рисунок 3.3 – Діаграма класів що додатку

### 3.3.1 Проектування генератору ідентифікаторів

При проектуванні системи, важлим аспектом було забезпечення можливості розподіленного функціонування екземплярів системи. Насамперед, необхідно організувати можливість використання не лише декількох екземплярів серверу додатків, а й декількох екземплярів серверу БД.

Розповсюдженою практикою є використання числових ідентифікаторів. При побудові розподіленої системи необхідно виділити проблему повтору ідентифікаторів при використанні простих методів їх генерації. Для вирішення цієї проблеми, можливо визначити унікальні діапазони ідентифікаторів для кожного серверу БД, або використовувати зовнішній лічильник.

Іншою практикою є використання UUID[13]. Перевагами UUID є, насамперед, можливість незалежної генерації унікальних ідентифікаторів, що майже не залежать від поточного стану системи. У випадку використання повністю випадкового UUID (UUID type 4), буде відсутня послідовний зв’язок між ідентифікаторами, що призведе до фрагментації сторінок БД та до зниження продуктивності. Також недоліком, є те, що у випадку використання бінарного представлення такого ідентифікатору (128 біт) відсутня можливість їх сортування штатними засобами роботи із числами, через специфічну структуру ідентифікатору. Для сортування необхідно виділити фіксовану та змінну частини, що розміщуються не послідовно у ідентифікаторі. У випадку використання строкового представлення UUID (у вигляді числа у 16-ричному кодуванні), ідентифікатор буде мати розмір 36 байт, та також не буде забезпечувати послідовність генерованих значень.

Тому було прийняте рішення що до розробки спрощеного алгоритму генерації ідентифікаторів.

Було виділено наступні критерії яким повинен задовольнити генератор:

* забезпечення унікальності ідентифікаторів на строк 100 років;
* можливість генерувати ідентифікатори на частоті не менш 100 на секунду;
* послідовність ідентифікаторів, що генеруються;
* можливість забезпечити захист від підбору наступних значень за попередніми, для посилених ідентифікаторів.

Виходячи із поставлених вимог, було прийняте рішення використовувати наступні значення для отримання ідентифікаторів:

* клас ідентифікатору (2 символи), що забезпечує можливість за ідентифікатором встановити його призначення, та забезпечити повну унікальність ідентифікаторів в межах програмного комплексу;
* ідентифікатор серверу в кластері (2 символи), що дозволяє декільком екземплярам незалежно генерувати ідентифікатори;
* поточна дата, кодована у вигляді 36-ричного числа. Значення кодується відносно 2000 року (4 символи);
* часовий зсув поточного часового поясу відносно UTC, до якого додано 12 годин, та записано у 36-ому кодуванні (1 символ). Часовий зсув дозволяє досягти монотонності часу, що втрачається через сезоні переводи часу;
* поточний час у UTC, у вигляді числа в 10-ичному кодуванні, що зберігає кількість секунд з початку дня (5 символів);
* значення локального циклічного лічильнику, у 36-ому кодуванні, що забезпечує 46655 унікальних значень на секунду(3 символи);
* опціональне, випадкове 3 байтове число, записане у 36-ому кодуванні, що ускладнює підбор наступного значення ідентифікатора (5 символів).

Використовуючи такі вхідні дані було досягнуто можливості генерувати ідентифікатори, що є послідовними майже на всьому протязі часової послідовності, з потужністю достатньою для роботи програмного комплексу, що проектується. Наприклад: DCA10E4BC383280RU782HK:DC – тип ідентифікатору, номер серверу в кластері(A1), 0E4B – дата(11 травня 2014 р.), C – часовий зсув (12-12 =0), 38328 – час (10:38:48), 0RU – лічильник(1002), 782HK – випадкова величина (12133784).

### 3.3.2 Проектування підсистеми розподілення прав

Для реалізації механізму розподілення прав було застосовано підхід, що має назву ACL (Access Control List, список контролю доступу) [14].

Для програмного комплексу, що проектується, ACL складається із наступних складових частин:

* дозволи (ролі), що встановлюють можливість виконати деяку дію над об'єктом, що захищається;
* об'єкти ACL – логічні елементи репозиторію, для яких можливо встановити права доступу;
* суб'єкти ACL (authority) – логічні типи користувачів системи, яким можна надавати доступ до елементів репозиторію. Для розроблюваної системи, такими суб'єктами є групи користувачів.

Спроектований механізм ACL включає можливість спадкування, що полягає в автоматичному встановленні прав доступу для всіх елементів, що за ієрархією репозиторію входять в керований об’єкт безпеки. Наприклад, встановлення допуску на колекцію, призведе до проеціювання допуску на усі документи колекції.

Списки контролю доступу є декларативними. Такий підхід дозволяє спростити безпосередню роботу із ACL, але ускладнює роботу з визначення реальних допусків на роботу із об'єктами. Для цього, за декларативним ACL будується ефективний ACL – агреговані списки контролю доступу для документів репозиторію. Ефективний ACL включає в себе інформацію про усі видані допуски для елементів репозиторію, що зберігаються в базі даних.

Принцип побудови ефективних ACL зображений на рис. 3.4.

Через можливість необхідності великої кількості обчислень, що повинна бути виконана в межах однієї транзакції (наприклад при видачі допуску на секцію що містить велику кількість документів), механізм обрахунку ефективного ACL було побудовано на базі внутрішнього планувальника відкладених задач. Такий підхід забезпечує можливість не блокувати таблиці БД на великий час, та розподілити обчислювальний процесс між екземплярами серверу, що працюють із одним екземпляром бази даних. Також така реалізація дозволяє реагувати на виникнення проблем при роботі із БД, та відновити виконання транзакції пізніше.

acl.emf

Рисунок 3.4 – Побудова ефективного ACE для документа

### 3.3.3 Проектування підсистеми перетворення змісту документів

Велика кількість розповсюджених користувацьких форматів є пропрієтарними, та можуть буди коректно використані лише за допомогою ПЗ що їх створило.

Для використання в внутрішніх алгоритмах, було прийняте рішення про необхідність уніфікованої конвертації ресурсів у формати що з одного боку були б відкритими, а зі іншого були придатні для використання кінцевим користувачем.

Можна виділити 3 найбільш розповсюджені типи документів, що використовуються в повсякденні. Нижче наведені ці типи, та відкриті формати в які були б придатні для обробки програмним забезпеченням, без використання спеціально програмного забезпечення:

* текстові документи – PDF (Portable Document Format);
* зображення – SVG (векторні) та PNG (растрові);
* відео – Vorbis Theora/WebM/Mpeg-2.

Конвертація документів буде виконуватися за допомогою наступного програмного забезпечення: текстові документи – Apache OpenOffice (LibreOffice), зображення – ImageMagic та відео – за допомогою декодера ffmpeg. Конвертація вкладень реалізована за допомогою підсистеми відкладеного виконання задач. Принцип роботи підсистеми конвертації наведено на рис. 3.5.

extrector.emf

Рисунок 3.5 – Схема взаємодії компонентів перетворення документів

### 3.3.4 Проектування підсистеми відкладеного виконання команд

Однією із цілей дипломного проектування є забезпечення розподіленої роботи програмного комплексу. Таким чином, необхідно організувати підтримку планування виконання команд на довільному сервері з кластеру.

Реалізація такої системи кращим чином вписується в модель масової обробки заявок. В системі, кожна задача класифікується як заявка деякого типу. Заявка ідентифікується за такими критеріями:

* ідентифікатор, що є унікальним кодом заявки;
* тип заявки, що встановлює тип обробника задачі;
* стан заявки, що встановлює поточний стан задачі;
* ідентифікатор документа, колекції чи секції з яким асоційована заявка;
* дата подачі заявки;
* дата взяття в обробку;
* дата закінчення обробки;
* дані, що необхідні обробнику заявки для виконання відкладеної команди.

Планування виконання задачі виконується шляхом додання відповідного запису до таблиці станів заявок. Дані необхідні для обробки заявки серіалізуються в формат JSON, та поміщаються у відповідне поля заявки.

Одиничне взяття в роботу заявки забезпечується транзакціональностью бази даних, що захищає від повторного переведення заявки із стану «Нова» до стану «В обробці».

Механізми обробки команд не підтримують транзакцію на протязі обробки команди, тому у випадку нештатної ситуації, наприклад аварійної зупинки серверу, заявка може залишитися не термінальному стані, зависнути. Для цього, для заявки кожного типу вводиться максимальний час обробки. У випадку перевищення встановленого часу обробки, Планувальник відновлення задач виконує переведення заявки до стану «Нова», що дозволяє планувальнику виконання задач взяти задачу в роботу.

Для забезпечення мінімального часу між створенням заявки, та її обробкою, та зменшення навантаження на сервер БД, комунікація між контролером відкладених задач та контролером виконання задач, виконується через систему доставки повідомлень за схемою «Постачальник-споживач».

Така схема дозволяє зменшити навантаження на сервер баз даних (зменшення інтервалу пошуку необроблених задач), мінімізувати затримку між створенням задачі та її обробкою, та прозоро зв’язати різні екземпляри додатку засобами черги повідомлень. Схема обробки заявок наведена на рис. 3.6.

task_mgr.emf

Рисунок 3.6 – Організація розподіленої відкладеної обробки задач

### 3.3.5 Проектування підсистеми доступу до вмісту

Однією з основних проблем реалізації доступу до вмісту був алгоритм авторизації запитів на отримання доступу до вмісту.

Архітектурно, сервіс використовує авторизацію шляхом аутентифікації користувача за сервісним токеном, та перевіркою доступів користувача. При необхідності створення ідентифікатору ресурсу для доступу, передача токену сесії у ідентифікаторі ресурсу є небажаною, через збільшення безпосередньо довжини ідентифікатору ресурсу та факту явної передачі аутентифікаційних даних.

Цю проблему можливо вирішити декількома шляхами:

* повна відсутність авторизації при доступі до контенту;
* реалізація підтримки аутентифікаційних cookie зі сторони серверу;
* реалізація механізму «полегшеної» авторизації зі сторони інтерфейсів, що забезпечують доступ до контенту.

Найпростішим методом реалізації доступу до контенту є авторизація доступу до URI контенту. Така методика використовується багатьма сервісами і полягає в тому, що механізми авторизації системи захищають від можливості отримання користувачем URI контенту. Такий підхід є актуальним при наявності дуже великих об’ємів даних що зберігаються, та у випадку коли отримання небажаного доступу до даних не є критичним. Тому використання такого підходу у рамках програмного комплексу є не можливим, через необхідність гарантій контрольованого доступу до ресурсів.

Іншим методом є аутентифікація користувача через cookie-браузера. Такий підхід спрощує механізми авторизації доступу, але його реалізація є проблемною з точки зору REST-архітектури веб сервісу. Також такий підхід не можливий при організації доступу із клієнтів що не базуються на веб-браузерах.

Через вищерозглянуті причини було прийняте рішення розробити спеціальний механізм доступу до змісту, що б авторизував доступ користувача до ресурсу, та для використання якого було б достатньо лише ідентифікатору ресурсу, за яким сервер міг би надати користувачу вміст ресурсу.

У межах такого механізму, доступ до контенту виконується в 2 етапи.

На першому етапі (рис. 3.7) виконується отримання так званого «білета» на доступ до ресурсу. На цьому етапі за запитом визначається користувач, документ до ресурсу якого виконується спроба доступу та IP-адреса з якої виконується запит. В момент запиту, сервер перевіряє можливість доступу користувача отримати доступ до ресурсу документа, і у випадку успішної перевірки доступу, сервер формує новий білет, що вказує на користувача, ресурс та адресу з якої користувач отримує доступ до серверу.

ticket1.emf

Рисунок 3.7 – Отримання білету на доступ до ресурсу

На другому етапі користувач отримує доступ до ресурсу за білетом (рис. 3.8). На цьому етапі виконується перевірка актуальності білету та адреси з якої виконується запит. У даній схемі контроль доступу виконується на рівні IP- адреси користувача, що дозволяє спростити доступ до ресурсу, та обмежити використання URI ресурсу лише IP-адресою з якої прийшов запит.

ticket2.emf

Рисунок 3.8 – Отримання змісту ресурсу за білетом

Таким чином було отримано механізм що спрощує доступ до ресурсів за їх URI виконуючи аутентифікацію за IP-адресою користувача.

Такий механізм призвів до іншої проблеми, необхідність зберігати інформації про білети на стороні серверу. Збереження такої інформації у реляційній БД може призвести до надлишкового навантаження на сервер БД, через те, що запити на доступ до ресурсу мають велику частоту, але їх зміст не є необхідним зберігати увесь час, через короткий час життя білетів.

Для збереження білетів на стороні серверу можливо використати пам’ять, але такий метод призводить до проблеми синхронізації білетів в межах кластеру. З іншого боку, можливо використати NoSQL рішення для збереження такої інформації, але це б призвело до необхідності додаткових ресурсів для серверу документ-орієнтованої бази даних, що є не ефективно.

Тому, було прийняте рішення зберігати дані в основному кеші додатку. У випадку одноекземплярного розгортання він буде розміщений у пам’яті додатку, а для багатоекземплярного – на спільному кеш-сервері.

## 3.4 Проектування динаміки системи

Для кожного варіанту використання можна побудувати діаграму кооперації (Collaboration Diagram). Даний тип діаграми дозволяє визначити які об‘єкти, як (за допомогою яких методів) і в якій послідовності взаємодіють для виконання якогось сценарію поведінки системи.

Для проектування послідовності дій була розроблена діаграма кооперації (рис. 3.9), що відображає послідовність взаємодії компонентів системи при видаленні колекції із системи. У взаємодії приймають участь наступні компоненти:

* контролер запитів на роботу з колекціями (ApiCollectionsController)4 менеджер колекцій (CollectionManager);
* менеджер документів (DocumentManager) для керування документами репозиторію;
* менеджер сховища (StorageManager), для роботи із вкладеними в документи додатками;
* репозиторій додатків (ContainerPersister), для збереження змісту вкладених додатків;
* репозиторій документів (DocumentRepository);
* репозиторій колекцій (CollectionRepository).

collab1.emf

Рисунок 3.9 – Діаграма кооперації при видалені колекції

Для реалізації підсистеми відкладеного виконання команд була розроблена діаграма кооперації (рис.10). В схемі виконання відкладених команд використовуються такі компоненти:

* інтерфейс планувальника задач SchedulerService;
* інтерфейс до бази даних Scheduler Service DAO;
* компонент для відправки даних на сервер повідомлень Scheduler Event Broker;
* контролер вивантаження запланованих задач із бази даних Task Fetch Scheduler;
* компонент, що виконує очікування нових повідомлень на сервері повідомлень Scheduler Event Consumer;
* контролер виконання задач Task Executor;
* компонент, що керує виконанням відкладених задач Task Processor Pool.

Така послідовність дій покриває потреби програмної підсистеми в реалізації обробки відкладених задач шляхом безпосередньої передачі повідомлення про створення нової задачі на системний брокер подій та створення запису в базі для журналювання стану системи.

У випадку виникнення помилок при передачі задачі на сервер доставки повідомлень, або у випадку програмного збою при обробці задачі за допомогою журналу задач, компонент Task Fetch Scheduler виконує ініціацію повторної обробки задачі. Така дія виконується за планувальником виклику процедур, період якої конфігурується зі сторони серверу, що дозволяє балансувати навантаження на систему.

Для контролю за виконанням задач, та забезпечення того, що б в один момент часу задача виконувалася лише одним виконавцем, компонент Task Executor синхронізує стан задачі із базою даних. Така операція є транзакційною, і сервер баз даних гарантує забезпечення цілісності стану системи. Компонент контролює оновлення станів задач за допомогою сервер баз даних, без кешування.

scheduler_seq.emf

Рисунок 3.10 – Діаграма кооперації підсистеми виконання відкладених задач

## 3.5 Проектування системи на фізичному рівні

На етапі фізичного проектування системи було створено діаграму компонентів (Component Diagram) (рис. 3.11) що структурує залежності між компонентами, та розділяє систему на структурні одиниці. Систему було розділено на презентаційний рівень (контролери API) та бізнес-логіку (сервісний рівень).

Було виділено наступні незалежні рівні:

* сервісний рівень, що реалізує основну бізнес-логіку компонентів з яких складається програмний комплекс, що забезпечують роботу основних алгоритмів програмної системи. На діаграмі 3.11 представлений групою «Сервісний рівень»;
* рівень об’єктів для доступу до даних, що призначений для абстракції компонентів бізнес-логіки системи від механізмів збереження даних. Компоненти цього рівня забезпечують збереження даних сутностей, що потребуються для роботи бізнес-логіки програмного комплексу, до реляційної бази даних або файлового сховища. Для компонентів що призначені для роботи із базою даних, на цьому рівні реалізується транзакціональність запитів. Представлений групою «Сервісний рівень» на діаграмі компонентів;
* рівень контролерів програмного комплексу, що забезпечують взаємодію між зовнішнім середовищем та компонентами бізнес-логіки. Контролери додатку також забезпечують трансформацію запитів-відповідей до програмного комплексу, та реалізують інтерфейси для забезпечення MVC;
* рівень аутентифікації що реалізує інтерфейс до серверу аутентифікації та авторизації. Контролери взаємодіють через загальний інтерфейс до підсистеми авторизації, що абстрагує логіку контролерів запитів від логіки роботи із сервером авторизації.

Комунікація між рівнями системи виконується за допомогою стандартизованих інтерфейсів.

component-1.emf

Рисунок 3.11 – Діаграма компонентів серверу

На основі отриманих результатів, було прийняте рішення про розділення серверу на 2 фізичних компоненти: сервер аутентифіцкації та сервер репозиторію. Таке рішення позитивно позначиться на зв’язності системи та дозволить розмежити споживання ресурсів. Фізичне розміщення програмного комплексу було відображено у вигляді діаграми розгортання (рис. 3.12).

deploymnt.emf

Рисунок 3.12 – Діаграма розгортання програмного комплексу

## 3.6 Вибір мови програмування

Виходячи з вимог поставлених до програмного комплексу було прийняте рішення використовувати мову Java. Це зумовлене тим, що проект є серверним додатком, а Java в першу чергу є мовою серверної розробки. Вибір Java також зумовлений тим, що це стандартом де-факто для розробки промислових додатків через офіційні гарантії зворотної сумісності, сталий синтаксис та високо продуктивність при наявності необхідних ресурсів. Також важливим фактором є те, що середовище Java існує для більшості операційних систем, що не обмежує користувача використанням продукту від одного поставника.

Для розробки було обрано JDK 7(Java Development Kit, комплект розробки Java).JDK7, на даний момент, є стабільною версією що підтримується розробником, корпорацією Oracle. Важливим також є підтримка зі сторони середовища останньої версії GC(Garbage Collector, збирач сміття – підсистема середовища, що відповідає за автоматичне звільнення пам’яті) що може працювати майже без зупинки додатку, та нової підсистеми вводу-виводу Java NIO (Java New Input/Output, новий ввід/вивід Java – набір бібліотек для неблокуючого вводу виводу), що дозволяє підвищити продуктивність роботи системи при роботі з великими файлами, що є актуальним для розроблюваного продукту.

Роботу серверу було вирішено базувати на технології Java Servlet API, що гарантує можливість виконання додатку в будь якому сумісному з JDK7 контейнері сервлетів.

Комплексну підтримку архітектурних рішень в додатку покладено на Spring Framework 3.3, що забезпечує наявність механізмів для зворотнього контролю залежностей (Dependency Injection) та декларативного керування транзакціями (Declarative Transactions). Spring Framework є стандартом для розробки серверних промислових додатків Java.

Для взаємодії із базою даних було прийняте рішення використовувати технології ORM (Object-relational mapping, Об'єктно-реляційна проекція), що дозволяє виконувати проеціювання сутностей із бази даних на класи Java. Як реалізацію ORM було обрано Hibernate 4, що добре інтегрується зі Spring та дозволяє зменшити кількість роботи по взаємодії з базою даних.

Для розробки додатку було обрано середовище Oracle Netbeans 7.4. Це зумовлено наявністю якісної підтримки мови Java та супутніх технологій, засобів відлагодження, тестування та профілювання. Середовище є безкоштовним для промислової розробки.

# 4 ВІДЛАГОДЖЕННЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

## 4.1 Вибір стратегії тестування

За направленістю тестування було поєднано два види тестування: низхідне та монолітне.

Низхідне тестування було використане в розробці класів бізнес-логіки додатку (включаючи DocumentManager, SectionManager, CollectionManager, тощо). При застосуванні цього методу, абстракції що реалізують роботу із базою даних було замінено на заглушки (Mock-objects [15]), що повертали передбачене значення. Метод активно використовується при розробці через тести (Test Driven Development) [16], що полягає в розробці тестів, за якими розроблюється логіка додатку. Метод забезпечує наявність покриття тестами більшої частини додатку, що підвищує якість програми та спрощує локалізацію помилок – виконати існуючий тест для локалізації відомої помилки, або створити новий для пошуку знайденої.

Тестування логіки додатку виконувалось за допомогою монолітного тестування. Усі розроблені тести представлені двома видами: модульні тести (Unit Test) та інтеграційні(Itegration Test) [17]. Модульні тести дозволили забезпечити коректну роботу окремих класів додатку та перевірити правильність роботи їх внутрішньої логіки. Модульні тести використовували лише реальні класи, замінюючи зовнішні залежності на Mock-об'єкти. Інтеграційні тести дозволили перевірити роботу системи в комплексі, використовуючи усі існуючі модулі, зовнішні сервіси та реальні бази даних. Також за допомогою інтеграційних тестів було протестовано зовнішній протокол серверу.

Для ефективного тестування були використані методи «білого ящика» та «чорного ящика».

Тестування за принципом білого ящика характеризується ступенем покриття тестами логіки (вихідний текст) програми. Вичерпне тестування за принципом білого ящика припускає виконання кожного шляху в програмі, але так як в програмі з циклами виконання кожного шляху звичайно нездійсненно, то тестування усіх шляхів не розглядається.

Для функцій зі складними умовами був використаний метод тестування «білим ящиком» – метод покриття умов та рішень. Даний метод дозволяє виявити непрохідні гілки умов, виявити оператори, які ніколи не будуть виконуватись. Метод покриття умов та рішень дозволяє виявити найбільшу кількість помилок серед методів «білого ящика». Даний метод полягає у записі числа тестів достатнього для того, щоб були покриті всі оператори, умови та рішення. Кожний оператор, умова та рішення повинні бути виконані хоча б один раз. Якщо після зіставлення тестів залишаються не покриті оператори, умови чи рішення, то потрібно доповнити свій набір тестів таким чином, щоб кожний оператор, умова чи рішення виконувались не менше одного разу.

Функції, в яких немає складних умов, тестувались методом покриття умов, так як цього достатньо для виявлення помилки.

Виконання тестів було забезпечене середовищем побудови додатки Apache Maven, що виконує керування виконанням тестів. Maven забезпечує виконання тестів при кожній побудові додатку та інформує користувача про наявність тестів що завершилися помилкою.

Метод тестування «чорним ящиком» припускає розробку тестів на основі аналізу специфікації програми. Тестування програми обмежується використанням невеликої підмножини всіх можливих вхідних даних. Правильно обраний тест цієї підмножини повинен володіти двома властивостями:

* зменшувати число тестів, які повинні бути розроблені для досягнення заздалегідь визначеної мети тестування;
* покривати значну частину інших можливих тестів, що в деякому ступені свідчать про наявність або відсутності помилок до і після застосування цієї обмеженої підмножини значень вхідних даних;
* покривати умови за яких виявлені помилки, що дозволяє однозначно підтвердити факт виправлення помилки.

Були використані наступні методи тестування «чорним ящиком»:

* метод припущення про помилку. Основна ідея полягає в переліку в деякому списку можливих помилок або ситуацій, в яких вони можуть з’явитися, а потім на основі цього списку виконується процес написання тестів. За допомогою даного методу можливо виявити помилки, завдяки тому, що в функцію передаються довільні дані.

## 4.2 Функції для тестування

SuccessfulResponse onCreateMetadata( String documentId, DocumentAttribute tagValue) – метод класу DocumentController призначений для додання метаданих до вказаного документу.

Вхід:

* documentId – ідентифікатор документу, для якого необхідно додати метадані;
* tagValue – об'єкт значення метаданих;
* таблиці БД: document та document\_metadata.

Вихід:

* оновлення списку метаданих документа (таблиця document\_metadata).

Текст методу:

public @ResponseBody

SuccessfulResponse onCreateMetadata(

@PathVariable("documentId") String documentId,

@RequestBody DocumentAttribute tagValue) {

documentManager.loadDocument(documentId);

if(tagValue.getAttributeId()==null||

tagValue.getAttributeId().isEmpty()) {

throw new RepositoryServiceException (

RepositoryServiceError.BadQueryParam,

"attributeId");

}

ArrayList<String> values = new ArrayList<String>();

values.addAll(tagValue.getValues());

documentManager.setMetdataItem(

documentId,tagValue.getAttributeId(),values);

eventSender.sendUpdateEvent(

new EvtRepositoryChanged(

EvtRepositoryChanged.NodeType.Document,

EvtRepositoryChanged.EventType.Update, documentId));

return new SuccessfulResponse();

}

Document loadExistingDocument(String documentId) – Метод класу DocumentManager, що завантажує документ за його ідентифікатором. У випадку відсутності документу ініціює виключення, що документ не знайдено.

Вхід:

* documentId – ідентифікатор документу;
* таблиця бази даних document.

Вихід:

* завантажений документ, або виключення, що документ не знайдено.

Текст методу:

Document loadExistingDocument(String documentId)

throws RepositoryException {

Document doc = documentRepository.findOne(documentId);

if (doc == null) {

throw new RepositoryException(

RepositoryErrors.DOC\_NOT\_FOUND);

}

return doc;

}

void setMetdataItem(String documentId,String metadataId, List<String> metadata) – метод класу DocumentManager, що встановлює вказані значення метаданих для документу.

Вхід:

* documentId – ідентифікатор документу для модифікації;
* metadataId – ідентифікатор метаданих що необхідно встановити;
* metadata – список значень метаданих;
* таблиці бази даних: document, document\_metadata.

Вихід:

* оновлені значення метаданих в таблиці document\_metadata.

Текст методу:

@Transactional

public void setMetdataItem(

String documentId, String metadataId,

List<String> metadata) throws RepositoryException {

metadataManager.ensureMetadataExists(metadataId);

Document doc = loadExistingDocument(documentId);

List<String> values = MetadataTool.extractValues(

doc, metadataId);

for(String value : values){

removeItemFromMetadataImpl(

doc.getDocumentMetadataList(), metadataId, value);

}

for(int i =0;i<metadata.size();i++){

DocumentMetadata dm = new DocumentMetadata(

i, metadataId, documentId);

dm.setMetadatavalue(metadata.get(i));

doc.getDocumentMetadataList().add(dm);

}

documentRepository.save(doc);

}

## 4.3 Розробка тестів методом «білого ящика»

Для тестування методу onCreateMetadata був обраний метод покриття умов та рішень. Даний метод полягає у записі числа тестів достатнього для того, щоб були покриті всі оператори, умови та рішення хоча б один раз. Необхідно обумовити, що методи loadDocument та setMetdataItem котрі викликаються методом що тестується, можуть ініціювати виключення, тому ці гілки потоку виконення також необхідно включити в список умов для покриття.

Список умов для тестування:

1. documentManager.loadDocument(documentId) throws Exception;
2. tagValue.getAttributeId()==null
3. tagValue.getAttributeId().isEmpty();
4. documentManager.setMetdataItem(documentId,tagValue.getAttributeId(),values);
5. return new SuccessfulResponse().

Тест 1

Вхід: documentId=’12345’, tagValue = {}, відсутність документу в таблиці document (див. табл. 4.1).

Таблиця 4.1 – Вміст БД для тесту 1

|  |  |  |  |
| --- | --- | --- | --- |
| document | | | |
| documentId | documentStatus | creationDate | modificationDate |
|  |  |  |  |

Вихід: RepositoryException { RepositoryErrors.DOC\_NOT\_FOUND }.

Тест 2

Вхід: documentId=’ DCA0000001’, tagValue={ attributeId=null }, стан БД наведено в табл. 4.2.

Таблиця 4.2 – Вміст БД для тесту 2

|  |  |  |  |
| --- | --- | --- | --- |
| document | | | |
| documentId | documentStatus | creationDate | modificationDate |
| DCA0000001 | A | 01-01-1900 | 01-01-1900 |

Вихід: RepositoryServiceException { RepositoryServiceError.BadQueryParam, ‘attributeId’}.

Тест 3

Вхід: documentId=’ DCA0000001’, tagValue={ attributeId=’’ }, зміст БД див. табл 4.3.

Таблиця 4.3 – Вміст БД для тесту 3

|  |  |  |  |
| --- | --- | --- | --- |
| document | | | |
| documentId | documentStatus | creationDate | modificationDate |
| DCA0000001 | A | 01-01-1900 | 01-01-1900 |

Вихід: RepositoryServiceException { RepositoryServiceError.BadQueryParam, ‘attributeId’}.

Тест 4

Вхід: documentId =’ DCA0000001’, tagValue={ attributeId=’dcmi.code’ }, зміст БД див. табл. 4.4.

Вихід: RepositoryException{ RepositoryErrors.TAG\_NOT\_EXISTS }.

Тест 5

Вхід: documentId =’ DCA0000001’, tagValue={ attributeId=’dcmi.title’,

values ={‘123’} }, зміст БД наведено в табл. 4.4.

Таблиця 4.4 – Вміст БД для тестів 4 та 5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| document | | | | | |
| documentId | documentStatus | | creationDate | | modificationDate |
| DCA0000001 | A | | 01-01-1900 | | 01-01-1900 |
| metadata\_definition | | | | | |
| metadataId | | multivaluedTag | | constraintType | |
| dcmi.title | | True | | Elib.string | |

Вихід: SuccessfulResponse.

За розробленими тестами була побудована таблиця покриття умов методу «DocumentController.onCreateMetadata» тестами (табл. 4.5). Також було проведено програмний аналіз покриття коду тестами (див. рис. 4.1). Результати програмного аналізу покриття коду збігаються з результатами отриманими аналітично. Для програмного аналізу коду використовувався інструмент Cobertura – засіб що обчислює відсоток покриття Java-коду тестами, що інтегрується із системою побудови проектів Maven, та аналізує результати виконання Unit-тестів що викликаються Maven при побудові виконуваних модулів прогограми.

Таблиця 4.5 – Покриття умов тестами для методу onCreateMetadata

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Тест | Умови | | | | | | | | | |
| 1) | | 2) | | 3) | | 4) | | 5) | |
| + | - | + | - | + | - | + | - | + | - |
| 1 | \* |  |  | \* |  | \* |  | \* |  | \* |
| 2 |  | \* | \* |  |  | \* |  | \* |  | \* |
| 3 |  | \* |  | \* | \* |  |  | \* |  | \* |
| 4 |  | \* |  | \* |  | \* | \* |  |  | \* |
| 5 |  | \* |  | \* |  | \* |  | \* | \* |  |

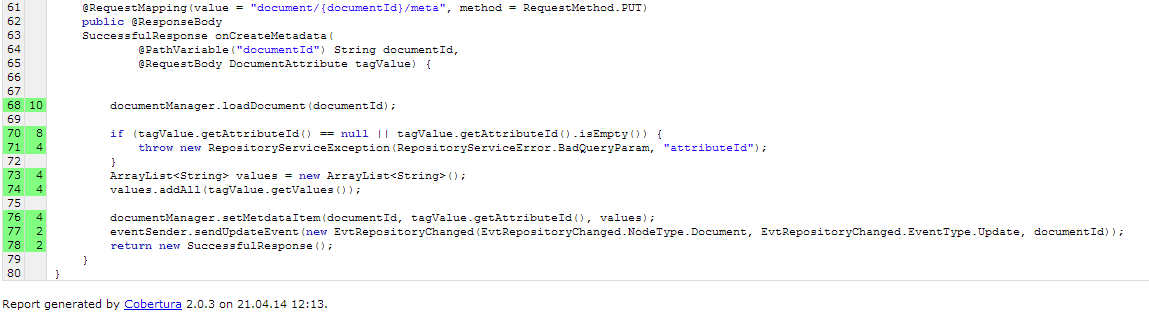


Рисунок 4.1 – Звіт про покриття методу тестами

## 4.4 Розробка тестів методом «чорного ящика»

Для тестування функції onCreateMetadata () був обраний метод припущення про помилку, що дозволяє перевірити роботу програми та знайти можливі помилки.

Для функції onCreateMetadata () були розроблені наступні тести:

Припустимо, що на вхід метода передається пустий ідентифікатор документу.

Тест 1

Вхід: documentId=null.

Вихід: очікується помилка «Документ не зайдено».

Припустимо, що на вхід передається ідентифікатор документу, але не передається значення атрибутів метаданих.

Тест 2

Вхід: documentId=’DC0001’, tagValue=null, вміст в ДБ (див. табл 4.6).

Таблиця 4.6 – Вміст БД для тесту 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| document | | | | | |
| documentId | documentStatus | | creationDate | | modificationDate |
| DC0001 | A | | 01-01-1900 | | 01-01-1900 |
| metadata\_definition | | | | | |
| metadataId | | multivaluedTag | | constraintType | |
| dcmi.title | | True | | Elib.string | |

Вихід: очікується помилка «Невірний запит. Некоректне значення параметру».

Припустимо, що на вхід передається коректний запит.

Тест 3

Вхід: documentId=’DC0001’,tagValue={ attributeId=’dcmi.title’. values = {‘123’}, стан БД(див. табл. 4.7).

Таблиця 4.7 – Вміст БД для тесту 3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| document | | | | | |
| documentId | documentStatus | | creationDate | | modificationDate |
| DC0001 | A | | 01-01-1900 | | 01-01-1900 |
| document\_metadata | | | | | |
| documentId | metadataId | | metadataOrder | | metadataValue |
| DC0001 | dcmi.title | | 0 | | Тест |
| metadata\_definition | | | | | |
| metadataId | | multivaluedTag | | constraintType | |
| dcmi.title | | True | | Elib.string | |

Вихід: оновлення значення поля метаданих «dcmi.title» документу «DC0001»(див. табл. 4.8).

Таблиця 4.8 – Вміст БД після виконання тесту 3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| document | | | | | |
| documentId | documentStatus | | creationDate | | modificationDate |
| DC0001 | A | | 01-01-1900 | | 01-01-1900 |
| document\_metadata | | | | | |
| documentId | metadataId | | metadataOrder | | metadataValue |
| ~~DC0001~~ | ~~dcmi.title~~ | | ~~0~~ | | ~~Тест~~ |
| DC0001 | dcmi.title | | 0 | | 123 |
| metadata\_definition | | | | | |
| metadataId | | multivaluedTag | | constraintType | |
| dcmi.title | | True | | Elib.string | |

Припустимо, що на вхід передається ідентифікатор існуючого документу, та існуючий атрибут метаданих, але його значення не відповідає припустимому формату, наприклад для формата дат.

Тест 4

Вхід: documentId=’DC0001’,tagValue={ attributeId=’dcmi.date’. values = {‘123,4,5’}, дані в БД (табл. 4.9).

Таблиця 4.9 – Вміст БД для тесту 4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| document | | | | | |
| documentId | documentStatus | | creationDate | | modificationDate |
| DC0001 | A | | 01-01-1900 | | 01-01-1900 |
| metadata\_definition | | | | | |
| metadataId | | multivaluedTag | | constraintType | |
| dcmi.title | | True | | Elib.string | |

Вихід: очікується помилка, що значення «123,4,5» є не припустимим для поля «dcmi.date».

На основі тестів за методом «чорного ящика» для методу onCreateMetadata() були побудовані інтеграційні тести. В результаті тестів, було виявлено помилки, що призводили до некоректної роботи серверних запитів. Після локалізації помилок, вони були успішно виправлені. Результатом аналізу помилок, стало прийняття рішення про необхідність автоматизованої валідації вхідних даних на відповідність граничним обмеженням.

Метод setMetdataItem() також було протестовано методом припущення про помилку. Для цього були розроблені тести.

Припустимо, що на вхід передається ідентифікатор існуючого документа, існуючого атрибуту метаданих та список значень є пустим.

Тест 1

Вхід: metadataId=’dcmi.title’,documentId=’DC0001’,values=[], вміст БД (табл. 4. 10).

Вихід: очікується видалення всіх атрибутів типу «dcmi.title» із документа «DC00001»(табл. 4.11).

Припустимо, що на вхід передаеться ідентифікатор існуючого документа, існуючого атрибуту метаданих та список значень має значення «null».

Тест 2

Вхід: metadataId=’dcmi.title’,documentId=’DC0001’,values=null, стан БД (див. табл. 4.10).

Вихід: очікується видалення всіх атрибутів типу «dcmi.title» із документа «DC00001»(див. табл. 4.11, видалення позначається як викреслені рядки).

Таблиця 4.10 – Вміст БД для тестів 1 та 2

|  |  |  |  |
| --- | --- | --- | --- |
| document | | | |
| documentId | documentStatus | creationDate | modificationDate |
| DC0001 | A | 01-01-1900 | 01-01-1900 |
| document\_metadata | | | |
| documentId | metadataId | metadataOrder | metadataValue |
| DC0001 | dcmi.title | 0 | Тест |

Таблиця 4.11 – Очікуваний стан БД післе тестів 1 та 2

|  |  |  |  |
| --- | --- | --- | --- |
| document | | | |
| documentId | documentStatus | creationDate | modificationDate |
| DC0001 | A | 01-01-1900 | 01-01-1900 |
| document\_metadata | | | |
| documentId | metadataId | metadataOrder | metadataValue |
| ~~DC0001~~ | ~~dcmi.title~~ | ~~0~~ | ~~Тест~~ |

Припустимо, що для документа вже задані значення атрибуту «dcmi.title». На вхід передаються нові значення.

Тест 3

Вхід: metadataId=’dcmi.title’,documentId=’DC0001’,values=[‘123’,’456’], дані в БД(табл. 4.12).

Таблиця 4.12 – Вміст БД для тесту 3

|  |  |  |  |
| --- | --- | --- | --- |
| document | | | |
| documentId | documentStatus | creationDate | modificationDate |
| DC0001 | A | 01-01-1900 | 01-01-1900 |
| document\_metadata | | | |
| documentId | metadataId | metadataOrder | metadataValue |
| DC0001 | dcmi.title | 0 | Тест |

Вихід: очікується заміна значеня поля «dcmi.title», документа «DC0001» на список значень переданих в запиті (табл. 4.13).

Таблиця 4.13 – Очікуваний вміст БД для тесту 3

|  |  |  |  |
| --- | --- | --- | --- |
| document | | | |
| documentId | documentStatus | creationDate | modificationDate |
| DC0001 | A | 01-01-1900 | 01-01-1900 |
| document\_metadata | | | |
| documentId | metadataId | metadataOrder | metadataValue |
| ~~DC0001~~ | ~~dcmi.title~~ | ~~0~~ | ~~Тест~~ |
| DC0001 | dcmi.title | 0 | 123 |
| DC0001 | dcmi.title | 1 | 456 |

В результаті тестів було виявлено, що метод setMetdataItem() не коректно реагує на вхідні дані, що входять в граничні умови. Після локалізації проблеми, вона була вирішена шляхом покращення перевірки вхідних параметрів методу, що призвело до успішного проходження розроблених тестів.

## 4.5 Відлагодження програми

Якщо тестування – діяльність, спрямована на виявлення помилок, то відлагодження спрямоване на встановлення точної природи відомої помилки, а потім – на виправлення цієї помилки. Ці два види діяльності зв'язані – результати тестування є вихідними даними для налагодження.

Відлагодження програми виконувалося з використанням методу індукції та методу просування від місця виникнення помилки до місця помилки. Таким чином, визначивши збійну операцію, можна локалізувати безпосереднє місце помилки.

Налагодження відбувалося шляхом завантаження Unit-тестів у режимі відладки, використовуючи наступні інструменти:

* Call Stack – список кадрів стеку викликів, за якими програма потрапила в поточний стан;
* Watches – інтерфейс що дозволяє виконувати моніторинг значень змінних в поточному контексті відладчика;
* Variables – інтерфейс, що відображає поточні значення змінних, наявних в поточному кадрі стеку;
* Threads – вікно, що відображає список ниток(thread), що активні в віртуальній машині Java. Дозволяє переключитися на стек викликів довільної нитки.

До місця виникнення ймовірної помилки ставиться точка зупину Breakpoint (рис. 4.2), на якій програма зупиниться і чекатиме покрокового виконання. Це дозволяє ініціювати виконання операцій, робота яких призводить до збою, та розпочати відлагодження з місця проблеми.

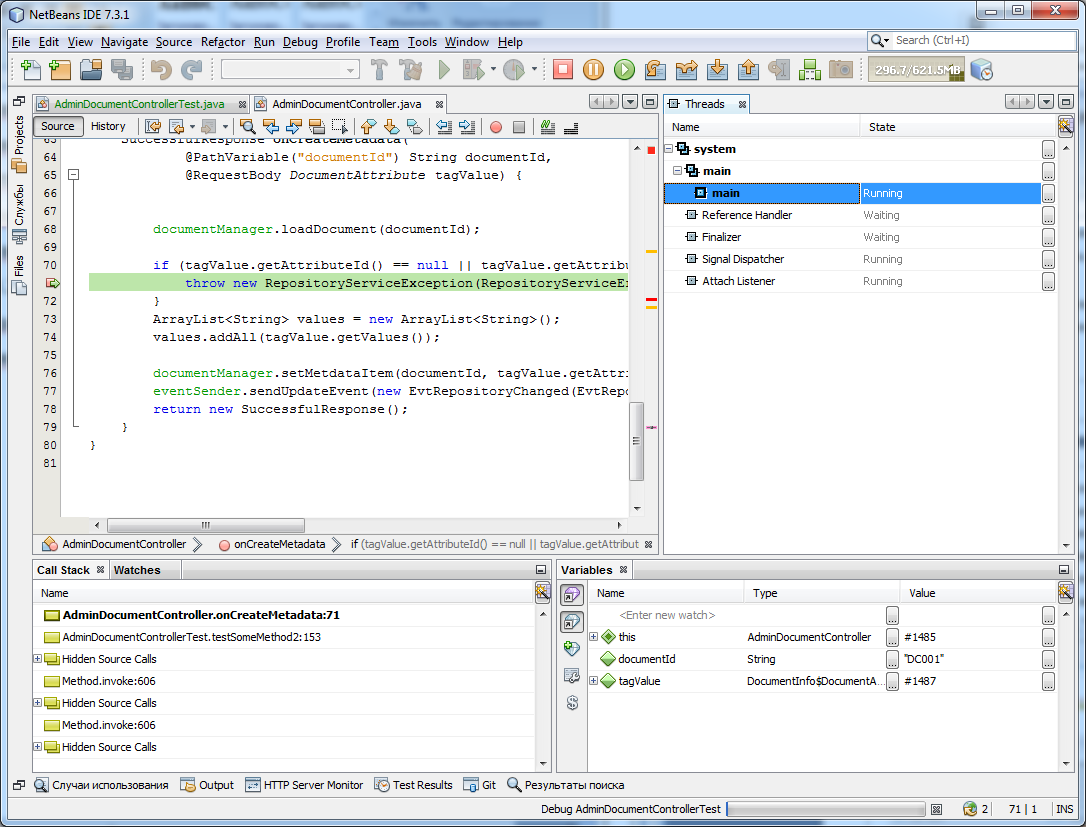


Рисунок 4.2 – Приклад використання точки зупину програми

# 5 ОХОРОНА ПРАЦІ

## 5.1 Аналіз шкідливих та небезпечних виробничих факторів

Шкідливі фактори – виробничі фактори, тривалий вплив яких на працюючого у визначених умовах людини приведе до захворювання, зниження працездатності. У залежності від рівня і тривалості впливу шкідливі фактори можуть класифікуватися і як небезпечні.

Основним робочим місцем спеціаліста відділу експлуатаційної підтримки є ПЕОМ з ВДТ. Згідно санітарно-гігієнічним вимогам умови праці співробітника відділу підтримки повинні відповідати I або II класу відповідно до Гігієнічної класифікації праці по показниках шкідливості і небезпеки факторів виробничого середовища, важкості і напруженості трудового процесу ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [18].

Робота спеціаліста з експлуатаційної підтримки пов'язана з наступними шкідливими факторами:

* недостатнє освітлення природним світлом;
* відблиски на екрані монітора;
* іонізуюче випромінювання;
* електромагнітне поле.

Недостатнє висвітлення приводить до швидкої стомленості очей, що у свою чергу, призводить до зниження продуктивності праці і росту кількості прийнятих помилкових рішень.

Відблиски на екрані монітора, що виникають при неправильному освітленні, приводить до погіршення зору, а у випадку тривалого впливу даного небезпечного фактору, може привести до повної втрати зору. З метою зниження рівня впливу на працівника даного шкідливого фактора, варто дотримувати вимоги НПАОП 0.00-1.31-99 «Правила охорони праці під час експлуатації електронно-обчислювальних машин» [19].

Основним джерелом іонізуючого випромінювання на робочому місці співробітника відділу підтримки є монітор – випромінювання α і β, що випускається люмінофором, згодом може призвести до променевої хвороби. Для зниження проникаючого випромінювання люмінофора рекомендується застосування моніторів із захисним покриттям екрана (напилювання свинцю), або (для виключення даного виду випромінювання) застосування рідкокристалічних панелей.

Небезпечні фактори – це виробничі фактори, вплив яких на працюючого у визначених умовах людини, приведе до травми, різкого погіршення здоров'я. До різкого погіршення здоров'я можна віднести отруєння, опромінення, удар електрострумом, тепловий удар та ін. ДСТУ 2293-99 «Охорона праці. Терміни та визначення основних понять» [20].

Робота спеціаліста експлуатаційної підтримки тісно пов'язана з використанням ПЕОМ. Виходячи з цього, можна виділити наступні небезпечні фактори:

* електронебезпека;
* пожежонебезпека.

При роботі з ПЕОМ найчастіше трапляються нещасні випадки, пов’язані з ураженням електричним струмом, які викликані дотиком до оголених місць струмоведучих частин устаткування, або частин, що знаходяться під напругою.

Вплив струму на людину залежить від його сили: струм до 0,6мА не відчувається людиною. Струм силою 6мА приводить до скорочення м'язів тієї частини, тіла, що піддалася його впливу. Цей струм називається «не відпускаючим». Значення струму, що перевищує 6мА, здатні викликати утрату свідомості і зупинку подиху, а при досягненні струмом порогу 100мА – смерть. При впливі на тіло людини струму в 3-4А виникає обвуглювання ділянок тіла.

Пожежі становлять особливу небезпеку для життя людини, і можуть призвести до великих матеріальних утрат. Під час пожежі людина може отримати опіки різного ступеня тяжкості, а також отруїтися чадним газом. Джерелами загоряння можуть виявитися електронні схеми ПЕОМ, що перегрілися.

## 5.2 Проектні заходи

Для якісної і зручної роботи співробітника відділу експлуатаційної підтримки необхідне проведення проектних заходів: відповідна облаштованість, належне дотримання ергономічних характеристик основних елементів робочого місця, санітарно-гігієнічних вимог і т.п.

Лінія електромережі для живлення ПЕОМ, периферійних пристроїв ПЕОМ і устаткування для обслуговування, ремонту і налагодження ПЕОМ виконується як окрема групова трьох провідна мережа, шляхом прокладки фазового, нульового робочого і нульового захисного провідників.

Підключення на розподільному щиті до одного контактного затиску нульового робочого і нульового захисного провідників заборонено.

Площа перетину нульового робочого і нульового захисного провідника в груповій трьох провідній мережі повинна бути не менш площі перетину фазового провідника.

Через те, що в робочому приміщенні відділу експлуатаційної підтримки, одночасно експлуатується більше п'яти персональних ПЕОМ, на видному доступному місці встановлюється аварійний вимикач, за допомогою якого можливо зробити знеструмлення приміщення (за винятком освітлення).

При роботі співробітника є недопустимими:

* експлуатація кабелів і проводів з ушкодженою чи утративши захисні властивості за час експлуатації ізоляцією;
* використання ушкоджених розеток, розгалужених і сполучних коробок, вимикачів і інших електроприладів, а також ламп, скло яких має сліди чи затьмарення опуклості;
* підвішування світильників безпосередньо на струмопровідних проводах, обгортання електроламп і світильників папером, тканиною й іншими займистими матеріалами, експлуатація їх із знятими ковпаками.

Виходячи з того, що приміщення в яких працюють спеціалісти експлуатаційної підтримки, відносяться до тих, у яких експлуатуються ВДТ (візуальний дисплейний термінал) і ПЕОМ (персональна електронно-обчислювальна машина), для них повинна бути визначена категорія по вибухонебезпечній і пожежній безпеці відповідно до НАПБ Б.03.002-2007 «Норми визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою» [21], і клас зони згідно ПЕУ (правила улаштування електроустановок). Відповідні позначення повинні бути нанесені на вхідні двері приміщення.

Будинки та ті їхні частини, повинні мати не нижче II ступеня вогнестійкості. Приміщення для обслуговування, ремонту і налагодження ПЕОМ повинні відноситися:

* по пожежонебезпеки до категорії В – пожежонебезпечні приміщення, де розташовуються тверді горючі речовини (ТГР), відповідно до НАПБ Б.03.002-2007 «Норми визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою» [21];
* по класу приміщення до категорії ІІа по ПЕУ.

Неприпустимим є розташування приміщень категорії А и Б (НАПБ Б.03.002-2007), а також виробництв із мокрими технологічними процесами поруч із приміщеннями, де розташовуються ПЕОМ, виконується їхнє обслуговування, налагодження і ремонт, а також над такими приміщеннями і під ними.

Робочі приміщення відділу підтримки повинні бути оснащені системою автоматичної пожежної сигналізації відповідно до вимог переліку однотипних за значенням об'єктів, що підлягають устаткуванню автоматичними установками пожежогасіння і пожежної сигналізації, затверджених ДБН В.2.5-56:2010 «Пожежна автоматика будинків і споруджень» [22] з димовими пожежними оповіщувачами та переносними вуглекислотними вогнегасниками з розрахунку 2 шт. на кожні 20 м2 площі приміщення з обліком гранично допустимої концентрації вогнегасної рідини відповідно до вимог Правил пожежної безпеки в Україні.

Стіни приміщень з ПЕОМ виготовляються з негорючих матеріалів. Підходи до засобів пожежогасіння повинні бути вільними.

За інструкцією "З охорони праці для спеціалістів з експлуатаційної підтримки" співробітник відділу планування інструктується перед початком роботи первинним інструктажем.

Первинний інструктаж завжди проводиться на робочому місці з безпосереднім показом робіт (стажування 1 місяць). Потім, через кожні 6 місяців проводиться повторний інструктаж.

Результати інструктажу заносяться в "Журнал реєстрації інструктажів з питань охорони праці". У журналі після проходження інструктажу повинен бути підпис людини, яка інструктувала та співробітника відділу прогнозування.

Площа, виділена для одного робочого спеціаліста підтримки, повинна складати не менш 6 м2 , а об’єм – не менш 20 м3.

Робочі місця, щодо вікон світлових прорізів повинні розташовуватися так, щоб природне світло падало з боку, переважно ліворуч.

При розташуванні робочих місць спеціалістів відділу підтримки необхідно дотримувати наступних вимог:

* робочі місця розташовуються на відстані не менш 1 м від стін із вікнами;
* відстань між бічними поверхнями ВДТ повинна бути не менше 1,2 м;
* відстань між тильною поверхнею ВДТ і екраном іншого ВДТ повинна бути не менш 2,5 м;
* прохід між рядами робочих місць повинний бути не менше 1 м.

Конструкція робочого місця співробітника відділу при роботі з ВДТ (при роботі сидячи) повинна забезпечувати підтримку оптимальної робочої пози з наступними ергономічними характеристиками: стопи ніг - чи на підлозі, чи на підставці для ніг; стегно - у горизонтальній площині; передпліччя - вертикально; лікті під кутом 70°-90° до вертикальної площини; зап'ястя - зігнуті під кутом не більш 20о щодо горизонтальної площини; нахил голови 15°-20° відносно вертикальної площини.

Висота робочої поверхні столу повинна бути в межах 680-800 мм, а ширина - забезпечувати виконання операцій у зоні досяжності моторного полючи.

Розміри столу, які рекомендуються: висота -725 мм, ширина ~ 600-1400 мм, глибина - 800-1000 мм.

Робочий стіл співробітника повинний мати простір для ніг висотою не менш 600 мм, шириною не менш 500 мм, глибиною на рівні колін не менш 450 мм, на рівні витягнутої ноги ~ не менш 650 мм.

Можливість обертання екрана ВДТ навколо горизонтальної і вертикальної осі.

Рівні шуму на робочому місці людини, яка працює із ВДТ та ЕОМ, визначені ДСН 3.3.6.037-99 «Санітарні норми виробничого шуму, ультразвуку та інфразвуку» [23].

Для забезпечення нормованих рівнів шуму у виробничих приміщеннях і на робочих місцях застосовуються шумопоглинаючі засоби, вибір яких обумовлюється спеціальними інженерно-акустичними розрахунками. При роботі з ЕОМ вказується спеціальний граничний спектр (ГС), що складає 55.

У якості шумопоглинаючих засобів повинні застосовуватися незпалимі чи трудно зпалимі спеціальні перфоровані плити, панелі, мінеральна вата з максимальним коефіцієнтом звукопоглинання в межах частот 31,5-8000 *Гц* чи інші матеріали аналогічного призначення, дозволені для обробки приміщень органами державного санітарно-епідеміологічного нагляду.

Параметри мікроклімату, іонного складу повітря, вміст шкідливих речовин на робочому місці, оснащеного ВДТ, повинні відповідати вимогам ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень» [24].

Для підтримки припустимих значень мікроклімату і концентрації позитивних і негативних іонів необхідно передбачити або установити прилади зволоження і (або) штучної іонізації, кондиціонування повітря.

При роботі з клавіатурою існують наступні вимоги:

* необхідно стійко розташувати клавіатуру на робочому столі, не допускаючи її коливання. Разом з цим може бути передбачена можливість поворотів і переміщень клавіатури. Положення клавіатури і кут її повороту повинні відповідати побажанням оператора (користувача). Якщо в конструкції клавіатури не передбачений простір для опори долонь, то їх необхідно розташувати на відстані не менш 10 см від краю столу. Під час роботи на клавіатурі необхідно сидіти прямо, не напружуючись;
* для зменшення несприйнятливого впливу на безпосереднього користувача пристроїв типу "миша" (змушена поза, необхідність постійного контролю за якістю дії) необхідно забезпечити вільну велику площу столу для переміщення "миші" і зручного упора ліктьового суглоба.

Для зниження напруженості роботи на ЕОМ необхідно рівномірно розподілити і чергувати характер робіт відповідно до їхньої складності.

Для зменшення негативного впливу на здоров'я працюючих виробничих факторів необхідно застосовувати регламентні перерви. У таблиці 1 приведений час регламентованих перерв користувача ПЕОМ у залежності від категорії і групи робіт.

Таблиця 5.1 – Час регламентованих перерв операторів ПЕОМ в залежності від категорії і групи робіт

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Категорія робіт | Група роботи | | | Час перерви при 8-годинній зміні, хв. |
| А, кількість знаків | Б, кількість знаків | В,година |
| Оптимальна -І | до 20 000 | до 15000 | до 2 | 20 |
| Припустима -ІІ | 21 000-40000 | до 30 000 | до 4 | 40 |
| Важка-ІІІ | понад 40 000  не більш 60 000 | понад 30 000  не більш 45000 | понад 4  не більш 6 | 60 |

Тривалість обідньої перерви визначається чинним законодавством про роботу.

При восьмигодинній робочій зміні регламентована перерва повинна бути такою:

* для I категорії робіт - через 2 години від початку зміни і через 2 години після обідньої перерви (кожен тривалістю 10 хв.);
* для II категорії робіт - через 2 години від початку зміни (тривалістю 15 хв.), через 1,5 і 2,5 годин після обідньої перерви (15 і 10 хв. відповідно чи тривалістю 5-10 хв*.* Через щогодини роботи, в залежності від характеру технологічного процесу);
* для Ш категорії робіт - через 2 години від початку зміни (тривалістю 15 хв.), через 1,5 і 2,5 годин після обідньої перерви (тривалістю 20 хв. Кожний тривалістю 5–15 хв. Щогодини роботи, у залежності від характеру технологічного процесу);
* під час нічної зміни, незалежно від групи і категорії робіт, тривалість регламентованих робіт збільшується на 60 хв.

У випадку виникнення у користувача зорового або дискомфорту інших неприємних суб'єктивних відчуттів, що настають, незважаючи на проходження санітарно-гігієнічних і ергономічних вимог, режимів роботи і відпочинку, варто застосувати індивідуальний підхід в обмеженні часу робіт і корекції тривалості перерв для відпочинку, або проводити заміну іншими видами робіт (не пов'язаних з використанням комп'ютера).

## 5.3 Безпека праці при виконанні робіт на ПЕОМ

### 5.3.1 Вимоги безпеки праці перед початком роботи на ПЕОМ

По інструкції «З охорони праці для спеціалістів з експлуатаційної підтримки» керівник інструктується перед початком роботи первинним інструктажем.

Первинний інструктаж завжди проводиться на робочому місці з безпосереднім показом робіт (стажування один місяць). Потім, через кожні шість місяців проводиться повторний інструктаж,

Результати інструктажу заносяться в «Журнал реєстрації інструктажів з питань охорони праці». У журналі після проходження інструктажу повинен бути підпис людини, яка інструктувала та керівника. Далі підготовка робочого місця повинна виконуватися відповідно до нижче приведених пунктів:

* увімкнути систему кондиціонування повітря в приміщенні;
* оглянути робоче місце і привести його в порядок: переконатися, що на ньому немає сторонніх предметів; все устаткування і блоки ПЕОМ з'єднані із системним блоком за допомогою сполучних шнурів;
* перевірити надійність установки апаратури на робочому столі. Монітор повинен стояти не на краю столу. Повернути монітор так, щоб було зручно дивитися на екран – під прямим кутом (а не збоку) – та ледь зверху вниз, при цьому екран повинен бути ледь нахилений – нижній його край ближче до співробітника відділу;
* перевірити загальний стан апаратури, або перевірити чи справні електропроводка, сполучні шнури, штепсельні вилки, розетки, перевірити заземлення захисного екрана;
* відрегулювати висвітлення робочого місця;
* відрегулювати і зафіксувати висоту крісла, зручний для співробітника нахил його спинки;
* включити апаратуру комп'ютера перемикачами на корпусі в послідовності: стабілізатор напруги, монітор, принтер (якщо необхідний друк), системний блок;
* відрегулювати яскравість світіння екрана до яскравості навколишніх його поверхонь у робочій зоні і не більше 3:1;
* при виявленні яких-небудь несправностей роботу не починати, повідомити про це керівника робіт.

### 5.3.2 Вимоги безпеки праці під час роботи на ПЕОМ

У період роботи за ПЕОМ необхідно передбачати перерви для відпочинку, які мають бути оптимальної тривалості (надто тривалі ведуть до порушення готовності до дії та розладу динамічного стереотипу). Тому, через кожні 40 - 45 хвилин необхідно робити перерву на 3 - хвилини, а через 2 години - на 15 хвилин. Середня сумарна тривалість роботи за ПЕОМ на день не повинна перевищувати 4 годин, а на тиждень - 20 годин.

При використанні захисного екрана або монітора із зниженим випромінюванням час роботи за ПЕОМ може бути збільшено.

Шкідливою є одна й та сама поза протягом тривалого часу. Тому в положенні сидячі треба час від часу змінювати фіксовані робочі пози, робити короткочасні перерви.

Під час роботи на ПЕОМ напружуються м'язи рук. Для підтримки їх тонусу під час перерви рекомендується проводити гімнастичні вправи.

З метою зниження негативного впливу монотонної діяльності доцільно чергувати операції введення тексту і цифрових даних, редагування тексту.

Періодично рекомендується виконувати комплекс вправ для очей:

* дивитися на мітку на віконному склі, що знаходиться на відстані 30 см від очей, потім перевести погляд вдалину (повторити кілька разів);
* рухи очима по колу до 10 разів за годинною стрілкою та навпаки - спочатку швидко, потім якомога повільніше (повторити вправу з заплющеними очима);
* самомасаж заплющених очей та шкіри навколо очей пальцями.

Забороняється залишати ввімкнені ПЕОМ і пристрої без нагляду.

Підключення і відключення роз'ємних кабелів пристроїв ПЕОМ робити тільки при відключенні їх з мережі.

Не можна користуватися біля ПЕОМ аерозолями (дезодорантами, тощо).

Не допускати попадання води та інших рідин в середину пристроїв комп'ютера.

При наявності електроструму на корпусі припинити роботу, вимкнути ПЕОМ від електромережі, сповістити керівника.

У випадку виникнення у співробітника зорового або дискомфорту інших неприємних суб'єктивних відчуттів, що настають, незважаючи на проходження санітарно-гігієнічних і ергономічних вимог, режимів роботи і відпочинку, варто застосувати індивідуальний підхід в обмеженні часу робіт і корекції тривалості перерв для відпочинку, або проводити заміну іншими видами робіт (не пов'язаних з використанням комп'ютера).

### 5.3.3 Вимоги безпеки праці після закінчення роботи на ПЕОМ

При завершенні роботи користувач повинен:

* закінчити і записати в пам'ять комп'ютера файл, що знаходився в роботі. Вийти з програмної оболонки і повернутися в середовище операційної системи;
* вимкнути системний блок, принтер, інші периферійні пристрої (якщо вони підключені до комп'ютера), вимкнути монітор. Вимкнути стабілізатор живлення, якщо комп'ютер підключений до мережі через нього. Штепсельні вилки витягнути з розетки. Накрити клавіатуру кришкою для запобігання потрапляння на неї пилу;
* прибрати робоче місце. Забрати усі необхідні документи (чи покласти їх у шухляду);
* доповісти керівнику про всі виявлені недоліки в роботі ПЕОМ.

### 5.3.4 Вимоги безпеки праці в аварійних ситуаціях

В разі ознак горіння (диму, запаху гару, тощо), припиненні подавання електроенергії або виявленні будь-яких несправностей необхідно негайно вимкнути ПЕОМ з електромережі. Якщо є потерпілі – надати їм першу медичну допомогу, викликати швидку допомогу за телефоном «103» або за телефоном найближчої медичної допомоги.

Якщо сталася пожежа, викликати пожежну частину за телефоном «101» та приступити до гасіння наявними засобами пожежогасіння. При виникненні аварійної ситуації виконувати всі вказівки керівника робіт по її усуненню.

# ВИСНОВКИ

В результаті роботи було виконано розробку програмного комплексу, що призначений для автоматизації обслуговування електронної бібліотеки та проаналізовано особливості розробки промислових додатків.

Для розробки продукту було використано новітні технології для побудови промислових Web-сервісів та спеціальне програмне забезпечення для поєднання вузлів додатку в єдиний комплекс.

Роботу програмного комплексу забезпечує реляційна БД, що складається із двох баз даних, призначених для збереження інформації про користувачів та структури каталогу бібліотеки.

Програмний комплекс реалізований у вигляді веб сервісу за архітектурою REST. Це дозволило спростити доступ до програмного комплексу та його обслуговування для адміністраторів комплексу. Використані архітектурні рішення дозволили використовувати програмний комплекс у багатоекземплярному розгортанні, що надає можливість балансувати навантаження між декількома фізичними серверами, та при підвищенні навантаження розгорнути додатковий екземпляр серверу на наявному обладнанні.

З іншого боку, для спрощення роботи із файловим сховищем було використано сховище за допомогою безпосередньо файлової системи. Це є вузьким місцем спроектованого програмного комплексу. Таким чином, для забезпечення безвідмовної роботи необхідні додаткові роботи з підтримки файлового сховища, через відсутність однозначного рішення з реалізації відмовостійкості файлової системи.

В межах роботи було використано проміжне програмне забезпечення, таке як сервер повідомлень Apache ActiveMQ, кеш-сервер Memcached та сховище даних Redis. Також активно використовується існуюче ПЗ для роботи із форматами даних документів, такі як Apache OpenOffice та ffmpeg.

В процесі розробки використовувалися новітні підходи до розробки програмного забезпечення. Проектування програми виконувалося за допомогою мови формального опису UML. Для проектування бази даних використовувалися моделі сутностей, за якими були побудовані сутності для програми. При написанні програмного коду використовувався підхід Test Driven Development, при якому спочатку формується специфікація у вигляді тестів, і лише за ними пишеться програмний код.

Розробка продукту була виконана для сучасної платформи Java 7, що забезпечило можливість роботи програмного продукту на будь якому сервері що підтримується Java. Також такий підхід зводить до мінімуму збитки від помилок в програмному продукті – у випадку помилки програма продовжить роботу, завершивши з помилкою лише збійний бізнес-процес.

Використані рішення виділяють продукт із ряду аналогів що доступні на ринку. Розроблена програма використовує сучасну сервіс-орієнтовану архітектуру, на відміну від аналогів що орієнтуються на локальне використання користувачем або повністю самостійну роботу.

Розроблений програмний комплекс є завершеним рішенням для електронної бібліотеки та готовий до експлуатації. Продукт може бути використаний у будь якій бібліотеці для організації електронного каталогу.

Для виправлення головного недоліку програмного комплексу можливий подальший розвиток у напрямку побудови сховища контенту поверх хмарних сховищ файлів. Таке рішення може забезпечити зменшення витрат на підтримку відмовостійкого файлового сховища, та зменшити збитки від виходу з ладу обладнання що використовується для збереження даних.

Також актуальним є питання реалізації клієнтів для доступу до репозиторію бібліотеки за допомогою мобільних пристроїв, використовуючи програмне забезпечення що використовує особливості їх операційних систем, для забезпечення максимальної продуктивності додатків.

# ЛІТЕРАТУРА

1. А. Б. Антопольский, Т. С. Маркарова, Е. А. Данилина Правовые и технологические проблемы создания и функционирования электронных библиотек — М.: ИНИЦ «Патент», 2008. — С. 207.
2. «DCMI Metadata Basics» [Електронний ресурс] — Режим доступу: <http://dublincore.org/metadata-basics/>.
3. Фаулер М. Архитектура корпоративных программных приложений / М. Фаулер, Д. Райс, М. Фоммел, Э. Хайет, Р. Ми, Р. Стаффорд – Вильямс, 2007. – 544 с.
4. «Date and Time Formats» [Електронний ресурс] — Режим доступу: <http://www.w3.org/TR/NOTE-datetime>.
5. «RFC 3066 — IETF» [Електронний ресурс] — Режим доступу: <https://www.ietf.org/rfc/rfc3066.txt>.
6. Фаулер М. UML. Основы. / М. Фаулер, К. Скотт; пер. с англ. А. Леоненков – СПб: Символ-Плюс, 2002. – 192 с.
7. «ER-модель данных» [Електронний ресурс] — Режим доступу: <http://ru.wikipedia.org/wiki/ER-%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85>.
8. «ООП» [Електронний ресурс] — Режим доступу: <http://ru.wikipedia.org/wiki/%D0%9E%D0%9E%D0%9F>.
9. «CRC-карта» [Електронний ресурс] — Режим доступу: <http://ru.wikipedia.org/wiki/CRC-%D0%BA%D0%B0%D1%80%D1%82%D0%B0>.
10. «Диаграммы UML» [Електронний ресурс] — Режим доступу:

<http://khpi-iip.mipk.kharkiv.edu/library/case/leon/gl8/gl8.html>.

1. Приемы объектно-ориентированного проектирования. Паттерны проектирования / [Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес]. – СПб.: Питер, 2001. – 368 с.
2. «Паттерн ServiceLayer» [Електронний ресурс] — Режим доступу: <http://design-pattern.ru/patterns/service-layer.html>.
3. «RFC 4122 - A Universally Unique IDentifier (UUID) URN Namespace» [Електронний ресурс] — Режим доступу: <http://tools.ietf.org/html/rfc4122>.
4. «ACL. Списки контроля доступа» [Електронний ресурс] — Режим доступу: <http://ru.wikipedia.org/wiki/ACL>.
5. «Mock-объект» [Електронний ресурс] — Режим доступу: <http://ru.wikipedia.org/wiki/Mock-%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82>.
6. Экстремальное программирование. Разработка через тестирование / [К. Бек]. – СПб.: Питер, 2003. – 224 с.
7. Искусство тестирования программ, 3-е издание / [Г. Майерс, Т. Баджетт, К. Сандлер] — М.: «Диалектика», 2012. — 272 с.
8. ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин».
9. НПАОП 0.00-1.31-99 «Правила охорони праці під час експлуатації електронно-обчислювальних машин».
10. ДСТУ 2293-99 «Охорона праці. Терміни та визначення основних понять»
11. НАПБ Б.03.002-2007 «Норми визначення категорій приміщень, будинків .та зовнішніх установок за вибухопожежною та пожежною небезпекою».
12. ДБН В.2.5-56:2010 «Пожежна автоматика будинків і споруджень».
13. ДСН 3.3.6.037-99 «Санітарні норми виробничого шуму, ультразвуку та інфразвуку».
14. ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень».