**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 программного 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](#_Toc390108400)

[1 ПРИЗНАЧЕННЯ, ПОСТАНОВКА ЗАДАЧІ ТА ОГЛЯД АНАЛОГІВ І ЛІТЕРАТУРИ 9](#_Toc390108401)

[1.1 Призначення та область застосування 9](#_Toc390108402)

[1.2 Постановка задачі 9](#_Toc390108403)

[1.3 Огляд літератури 10](#_Toc390108404)

[1.4 Аналіз аналогів 18](#_Toc390108405)

[1.4.1 Характеристика системи ruDi. 19](#_Toc390108406)

[1.4.2 Характеристика системи "Магістраль" ДЕ1м. 25](#_Toc390108407)

[1.5 Аналіз розглянутих аналогів 25](#_Toc390108408)

[2 ЗОВНІШНЄ ТА ЛОГІЧНЕ ПРОЕКТУВАННЯ 26](#_Toc390108409)

[2.1 Зовнішнє проектування 26](#_Toc390108410)

[2.1.1 Опис функціональних характеристик 26](#_Toc390108411)

[2.1.2 Формалізація задач 27](#_Toc390108412)

[2.2 Аналіз баз даних 29](#_Toc390108413)

[2.2.1 Аналіз системи керування базами даних 30](#_Toc390108414)

[2.2.2 Сутності та їх представлення в базі даних 30](#_Toc390108415)

[2.2.3 ER-модель бази даних 31](#_Toc390108416)

[2.2.4 Аналіз фізичного проекту бази даних 33](#_Toc390108417)

[2.2.4 Нормалізація бази даних 37](#_Toc390108418)

[3 ВНУТРІШНЄ ПРОЕКТУВАННЯ 39](#_Toc390108419)

[3.1 Вибір парадигми програмування 39](#_Toc390108420)

[3.2 Проектування програмних інтерфейсів 40](#_Toc390108421)

[3.2.1 Проектування RESTful протоколу 42](#_Toc390108422)

[3.2.2 Протокол веб-сервісу 44](#_Toc390108423)

[3.3 Проектування архітектури системи 44](#_Toc390108424)

[3.3.1 Проектування генератору ідентифікаторів 55](#_Toc390108425)

[3.3.2 Проектування підсистеми розподілення прав 57](#_Toc390108426)

[3.3.3 Проектування підсистеми перетворення змісту документів 59](#_Toc390108427)

[3.3.4 Проектування підсистеми відкладеного виконання команд 60](#_Toc390108428)

[3.3.5 Проектування підсистеми доступу до вмісту 62](#_Toc390108429)

[3.4 Проектування динаміки системи 65](#_Toc390108430)

[3.5 Проектування системи на фізичному рівні 68](#_Toc390108431)

[3.6 Вибір мови програмування 70](#_Toc390108432)

[4 ВІДЛАГОДЖЕННЯ ТА ТЕСТУВАННЯ ПРОГРАМИ 72](#_Toc390108433)

[4.1 Вибір стратегії тестування 72](#_Toc390108434)

[4.2 Функції для тестування 74](#_Toc390108435)

[4.3 Розробка тестів методом «білого ящика» 76](#_Toc390108436)

[4.4 Розробка тестів методом «чорного ящика» 79](#_Toc390108437)

[4.5 Відлагодження програми 84](#_Toc390108438)

[5 ОХОРОНА ПРАЦІ 86](#_Toc390108439)

[5.1. Аналіз шкідливих та небезпечних виробничих факторів 86](#_Toc390108440)

[5.2. Проектні заходи 87](#_Toc390108441)

[5.3. Безпека проці при виконанні робіт на ПЕОМ 91](#_Toc390108442)

[5.3.1. Вимоги безпеки праці перед початком роботи на ПЕОМ 91](#_Toc390108443)

[5.3.2. Вимоги безпеки праці під час роботи на ПЕОМ 92](#_Toc390108444)

[5.3.3. Вимоги безпеки праці після закінчення роботи на ПЕОМ 95](#_Toc390108445)

[5.3.4. Вимоги безпеки праці в аварійних ситуаціях 95](#_Toc390108446)

[ВИСНОВКИ 96](#_Toc390108447)

[ЛІТЕРАТУРА 98](#_Toc390108448)

# ВСТУП

На Українській залізниці планується поступовий перехід від планово попереджувальної системи ремонту рухомого складу, до системи ремонту рухомого складу з урахуванням його фактичного технічного стану. Такий перехід можливий тільки в тому разі, коли є найбільш достовірна інформація про стан кожної окремої одиниці рухомого складу. Таку інформацію можуть дати і тестові діагностичні системи, але, для економії часу та коштів, потрібно раціональніше впроваджувати функціональні засоби діагностування. До таких засобів належать бортові системи діагностування локомотивів.

На сьогоднішній день ремонт та технічне обслуговування локомотивів виконується згідно з фіксованим графіком, коли необхідність та вид чергового ремонту, як правило, визначаються пробігом локомотива. Такий підхід не враховує фактичний стан вузлів локомотивів і реальну необхідність в ремонті даного виду, що призводить до значних додаткових витрат. Крім того, відсутність інформації про обсяг і перелік ремонтних робіт для конкретних локомотивів ускладнює планування ремонту в депо та суттєво збільшує час його виконання.

Одним з варіантів вирішення цієї проблеми є удосконалення методів діагностування та прогнозування зміни технічного стану вузлів локомотивів, розробка методів визначення періодичності та обсягів виконання ремонтів на основі діагностичних даних та «історії» експлуатації локомотива.

Досвід розробки та впровадження систем управління локомотивним парком з використанням результатів діагностування за кордоном підтверджує ефективність такого підходу. Провідні світові компанії виробники локомотивів приділяють значну увагу впровадженню засобів діагностування та організації системи сервісного обслуговування локомотивів на основі результатів діагностування. [1-4].

Системи збору діагностичних даних встановлюються на багатьох локомотивах, під час проведення діагностування система формує запис показників всіх датчиків локомотива. Сформований запис зберігається в журналі показників датчиків локомотива, на кожному локомотиві існує свій автономний журнал (база даних) показників датчиків, який формується з багатьох записів.

Для аналізу діагностичної інформації в умовах депо Нижньодніпровськ Вузол та Красний Лиман використовується програмний комплекс Магістраль Центр розроблений НВО Квант-Радіоелектроніка. [5]. За допомогою цього комплексу інформація з бортового модуля пам’яті кожного електровоза переноситься до центральної діагностичної бази даних депо.

В рамках дипломного проекту буде розроблено програмний комплекс «Менеджер статистики локомотивів» призначений для перегляду статистики систем діагностування локомотивів.

«Менеджер статистики локомотивів» буде створено у вигляді веб-сервісу, для спрощення його обслуговування та модернізації з використанням інтерактивних діаграм, що дозволить універсальний віддалений доступ до даних кінцевому користувачу.

Такий шлях вирішення проблеми є актуальним та концептуально новим, через те, що проектоване рішення є орієнтованим на розширення функціональності за допомогою зовнішніх модулів.

Результатом проектування та розробки буде програмний комплекс аналіз роботи якого дозволить уточнювати перелік контрольованих параметрів, удосконалювати алгоритми контролю, періодичність опитування датчиків контролюючих параметри вузлів і агрегатів локомотивів, а також використовувати інформацію при розробці й проектуванні вузлів локомотивів для забезпечення їх придатності до контролю.

# 1 ПРИЗНАЧЕННЯ, ПОСТАНОВКА ЗАДАЧІ ТА ОГЛЯД АНАЛОГІВ І ЛІТЕРАТУРИ

## 1.1 Призначення та область застосування

Програмний комплекс, що розробляється, призначений для перегляду статистики систем діагностування локомотивів.

Функціональне призначення продукту полягає у наданні сервісу що зможе адмініструвати систему, виконувати пошук діагностичних даних локомотивів за період часу та відображати їх значення на діаграмі та таблицях. Експлуатаційне призначення – спрощення доступу до діагностичних даних локомотивів та надання можливості побудови порівняльних діаграм з характеристиками вузлів локомотивів.

Програма може бути розповсюджена та використана у локомотивному господарстві Укрзалізниці, де для збору діагностичних даних локомотивів використовується програмний комплекс Магістраль.

## 1.2 Постановка задачі

Необхідно розробити програмний комплекс, що надасть змогу видалено переглядати статистичну інформацію систем діагностування локомотивів.

Результатом розробки повинен стати сервіс, що надає програмні інтерфейси для перегляду статистичної інформації локомотивів. Кінцевому користувачу повинні бути надані можливості для побудови порівняльних діаграми з характеристиками вузлів локомотивів та можливість друку та збереження діаграм в графічному форматі. Порівняльна діаграма повинна відображати данні у вигляді графіка, забезпечувати масштабування та відображати дані за декількома локомотивами та декількома вузлами на одній площині. Програма повинна надати можливість завантаження файлу з статистичною інформацією про вузли локомотивів. Продукт повинен забезпечувати можливість клієнт-серверної взаємодії, без обмеження мовою програмування.

Продукт має відповідати наступним вимогам: гнучкі критерії для пошуку діагностичної інформації в базі даних, відмово стійкість серверу, забезпечення гнучкої системи контролю доступу до ресурсів.

## 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.

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, згідно [6], 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.

Новий рухомий склад українських залізниць обладнаний бортовими системами діагностування. Уперше на вітчизняних локомотивах такі системи були встановлені на електровозах серії ДЕ1. Система діагностування контролює параметри основних вузлів електровоза, таких як: тягові двигуни, допоміжні машини, пуско-гальмові резистори, акумуляторна батарея, перетворювач, букси. Також система контролює стан електричних апаратів силовому ланцюга й ланцюгів керування.

Система працює в чотирьох режимах, які дозволяють безупинно контролювати вузли, переглядати поточні значення параметрів і накопичену діагностичну інформацію. Вікно системи діагностування показане на рисунку 1.1, структура системи показана на рисунку 1.2.



Рисунок 1.1 - Вікно бортової системи діагностування електровоза ДЕ1:

кола — контрольовані вузли; цифри біля кіл — код вузла в системі діагностування

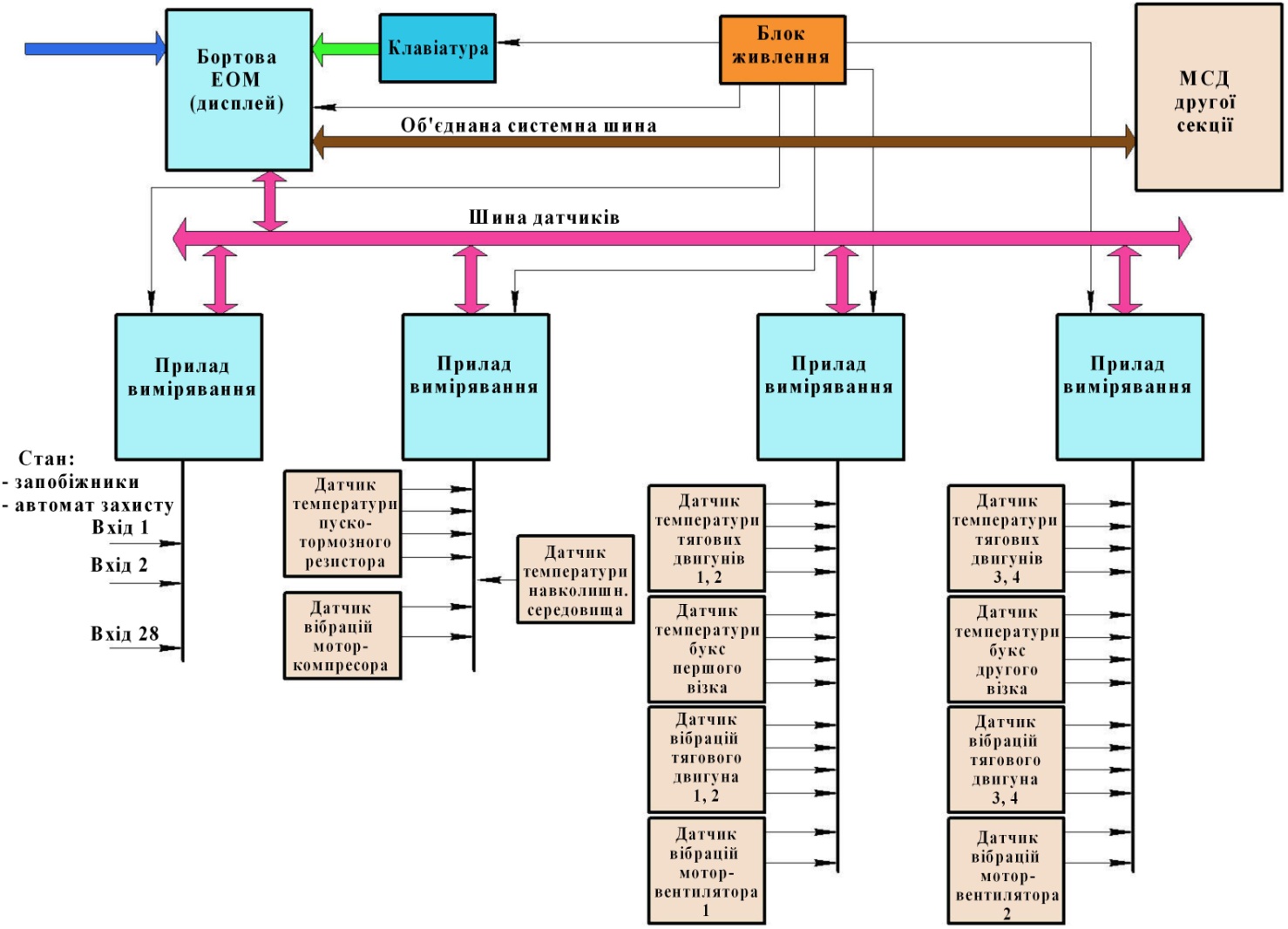


Рисунок 1.2 – Структурна схема МСД "МАГИСТРАЛЬ-ДЕ1М"

Технічна характеристика мікропроцесорної системи "МАГИСТРАЛЬ-ДЕ1М":

|  |  |  |
| --- | --- | --- |
| * Кількість контрольованих параметрів | * – | * 400 |
| * Частота опиту датчиків, Гц | * – | * (не більш), 1 |
| * Ємність модуля для реєстрації аварійних ситуацій, Мбайт | * – |  |
| * Напруга живлення, В | * – | * 50 |
| * Споживана потужність по ланцюгу 50 В, Вт | * – | * 126 |
| * Пусковий струм по ланцюгу 50 В, А | * – | * 15 |
| * Маса, кг | * – | * 150 |
| * Напрацювання на відмову, тис./ год | * – | * 120 |
| * Час підготовки системи до роботи, с | * – | * 60 |

Система забезпечує вимірювання аналогових і дискретних параметрів зокрема:

– температури БПТР (8 каналів);

– температури на вході і виході тягових двигунів (16 каналів);

– температури букс (16 каналів);

– температура навколишнього середовища (2 канали);

– віброприскорення на тягових двигунах в двох площинах (16 каналів);

**–** віброприскорення на моторвентиляторах і моторкомпрессорах;

– положення запобіжників і автоматів захисту в ланцюгах живлення.

Також прийом даних з пристрою управління тяговим електроприводом (УУТЕП):

– струми і напруга тягових двигунів;

– струми і напруга акумуляторних батарей;

– струми і напруга перетворювачів;

– струми вентиляторів і компресорів;

– напруга контактної мережі;

– стан контакторів, перемикачів, вимикачів, реле та іншого устаткування.

Виявлення відхилень в роботі устаткування по критерію перевищення допустимих і критичних значень для аналогових параметрів.

Візуальне і звукове сповіщення про досягнення параметрами критичних і гранично допустимих значень, а також про появу заборонених комбінацій дискретних сигналів).

Відображення поточного стану контрольованих параметрів на відео моніторі.

Реєстрацію інформації про відхилення від норми в роботі устаткування в незалежному модулі пам’яті.

Розшифровка даних, записаних в модулі пам'яті, і видача підсумкового зведення про кількість і характер зареєстрованих відхилень в роботі устаткування електровоза.

Прийом і відображення даних з швидкостеміра (швидкість, тиск, обмеження швидкості).

Розрахунок тягових і гальмівних зусиль і потужності споживаною тяговим і допоміжним устаткуванням і відображення результатів на відеомоніторі.

Програмні і апаратні засоби системи забезпечують функціонування систем діагностики двох секцій електровоза в єдиному комплексі, в якому обробляється, відображається на відеомоніторі і реєструється в модулі пам’яті вимірювальна інформація всього електровоза.

Система побудована за блоково-модульним принципом і складається з окремих конструктивно і функціонально закінчених елементів різних рівнів складності: приладів, блоків, вузлів і датчиків, що розміщуються в кузові електровоза і на візках. Всі компоненти системи з'єднуються між собою за допомогою джгутів монтажного комплекту.

Програмне забезпечення системи (ПО) розміщується в твердотільних пристроях (ЗУ) бортової ЕОМ (зберігаючих пристроях). Збір вимірювальної інформації на рівні системи організований за магістральним принципом. У кожній секції збір здійснюється по послідовній шині датчиків (ШД).

Як базові технічні засоби вимірювального поста використовуються уніфіковані, системно сумісні прилади, до складу яких входять один або два програмно керованих модуля збору інформації з сімейства IDEM-7000. Модулі збору інформації вирішують завдання комутації вимірювальних каналів, кодування вимірювальної інформації і передачі її в прилад (ЗУ) по послідовному каналу шини датчиків.

До складу системи кожної секції входять два блоки (основний і резервний) вторинного електроживлення.

Для накопичення і аналізу діагностичної інформації безпосередньо на електровозі в системі діагностування передбачений режим «Статистика». Додатково для розшифрування й накопичення інформації в умовах депо використовується програмне забезпечення для розшифрування й аналізу інформації.

Система діагностування електровозів серії ДС3 за своїми функціями аналогічна системі діагностування електровоза ДЕ1. Відмінною рисою цієї системи є більш зручний графічний інтерфейс при відображенні інформації. Системою також передбачене накопичення діагностичної інформації, розшифрування якої виконується представниками фірми Siemens.

Системи діагностування дизель-поїздів ДЕЛ02 і тепловозів ТЕП150 також виконують контроль дизеля і його систем, вібрації й електричних параметрів тягових електродвигунів, параметрів акумуляторної батареї, а також контроль роботи електричних апаратів, температури букс, тиску в гальмових циліндрах. Структура системи діагностування дизель поїзда СПРАВ-02 наведена на рисунку 1.3. Вікно системи діагностування показане на рисунку 1.4.

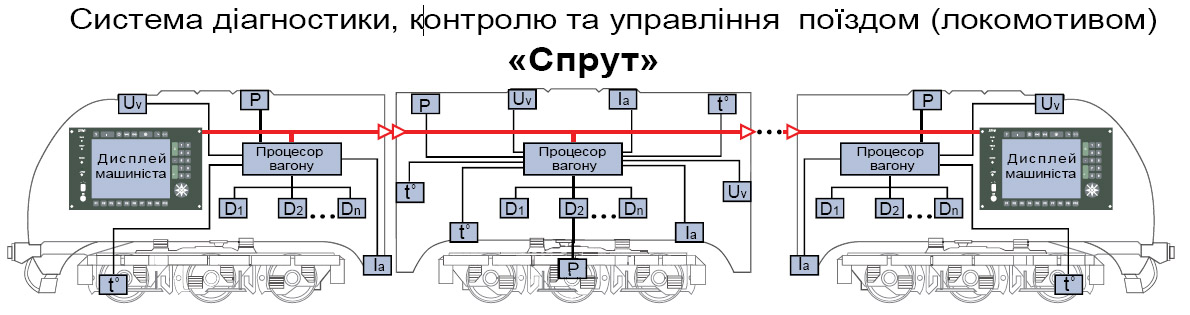


Рисунок 1.3. Структура системи діагностування дизель поїзда ДЕЛ02

Системи діагностування електровозів ДЕ1 і ДС3 виконують функції контролю й накопичення діагностичної інформації про основні вузли. На інших типах рухомого состава (електровози ВЛ11М6, ЕПЛ2Т, 2С5К и ін.) також використовуються бортові системи діагностування, структура яких аналогічна розглянутим системам. У цілому кількість діагностичних параметрів у бортових системах діагностування нових і існуючих локомотивів більшою мірою залежить від технічної можливості вимірювати той або інший параметр існуючими засобами діагностування, а також від кількості коштів, виділених розроблювачам на систему діагностування. Ефективність системи залежить не тільки від кількості діагностичних параметрів, але в значній мірі від організації роботи системи діагностування й використання отриманої інформації ремонтними службами.

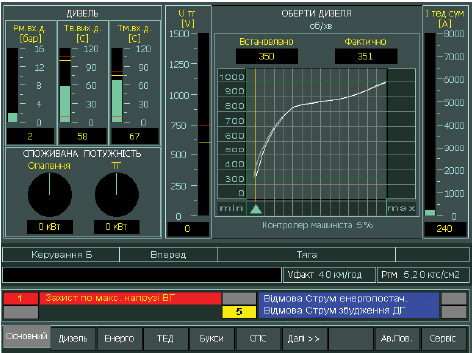


Рисунок 1.4 - Вікно системи діагностування дизель поїзда ДЕЛ-02

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

* підвищити безпеку руху (попереджати аварійні ситуації під час руху, контролювати дії локомотивної бригади);
* скоротити час пошуку несправності в електричних схемах (контролювати спрацьовування електричних апаратів);
* виявляти (а частіше — підтверджувати) факт настання відмови контрольованих вузлів.

Фактично системи діагностування виконують функцію моніторингу (безперервного спостереження) зміни технічного стану вузлів локомотивів.

З позиції технічної діагностики [7] основними завданнями будь-якої системи діагностування є:

* визначення працездатності об'єкта діагностування (справний/несправний);
* контроль функціонування (контроль того, щоб у процесі експлуатації не виникали відмови, що порушують працездатність);
* пошук несправності (вказівка місця виникнення несправності й способу її усунення);
* прогнозування зміни технічного стану об'єкта діагностування надалі;
* визначення попереднього стану об'єкта діагностування.

Існуючі системи діагностування тією чи іншою мірою виконують перші три завдання, але їх розв'язок не дозволяє чітко визначити обсяги ремонту й планувати його періодичність, що необхідно для переходу на ремонт рухомого состава з обліком його фактичного технічного стану. Це різко знижує ефективність використання систем діагностування й формує негативне відношення до самих систем діагностування з боку ремонтних служб.

Для підвищення ефективності систем діагностування необхідно налагодити зворотний зв'язок між діагностичними системами й ремонтними службами депо. Для цього слід розробити технологію запису, зберігання й обробки інформації від систем діагностування локомотива в депо. Із цією метою вхідна інформація повинна фіксуватися, накопичуватися, систематизуватися й використовуватися для розрахунків системи утримання кожного локомотива.

Тільки на підставі спільного аналізу діагностичної інформації й даних про відмови локомотивів у процесі експлуатації може бути розрахована раціональна система утримання та отриманий ефект від впровадження систем діагностування.

Крім того, подальший аналіз роботи систем діагностування дозволить уточнювати перелік контрольованих параметрів, удосконалювати алгоритми контролю, періодичність опитування датчиків контролюючих параметри вузлів і агрегатів локомотивів, а також використовувати інформацію при розробці й проектуванні вузлів локомотивів для забезпечення їх придатності до контролю.

## 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 спроектовано для збору діагностичних даних локомотивів у локомотивному господарстві де використовується програмний комплекс Магістраль. Іншою причиною є те, що існуючі рішення призначені для внутрішнього використання, та не надаються для придбання.

Саме тому, в цьому розділі будуть розглянуті також закордонні рішення бортових систем діагностування локомотивів.

### 1.4.1 Характеристика системи ruDi.

У Германії розроблена система ruDi для контролю і управління парком тягового рухомого складу. Вона забезпечує реєстрацію всіх експлуатаційних даних, технічну діагностику рухомого складу в оперативному режимі і визначає його місцезнаходження в межах європейської мережі. На основі даних, що поступають безперервно, виробляється оцінка міри завантаження, окремо для кожного типа рухомого складу і визначаються терміни його технічного обслуговування. Система дозволяє не лише визначати місцезнаходження рухомого складу в будь який час, але і сприяє оптимізації процесу управління парком рухомого складу і підвищенню якості організації його технічного обслуговування. У зв'язку з цим помітно підвищується надійність і експлуатаційна готовність рухомого складу.

Завдяки модульній компоновці система ruDi має можливість розширення незалежно від виробників рухомого складу, її можна встановлювати на рухомий склад як додаткове устаткування. Експлуатаційним компаніям система ruDi забезпечує оперативний контроль парку рухомого складу, поліпшення планерування поточного вмісту, точну документальну реєстрацію всіх виробничих робіт, а також підтримку при розрахунках експлуатаційних витрат протягом всього терміну життєвого циклу (LCC).

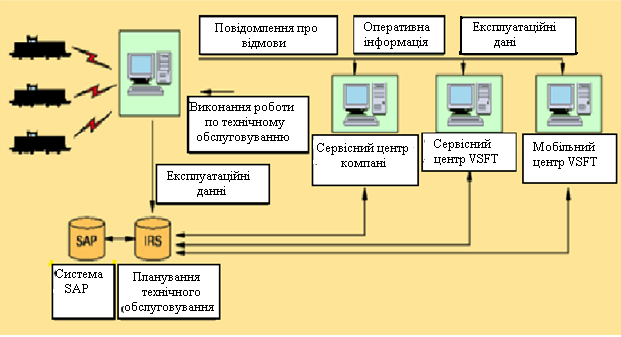


Рисунок 1.5 – Структура інтегрованої системи ruDi/VSMS

VSMS об'єднує в собі систему планування, організації і документальної реєстрації процесів технічного обслуговування тягового рухомого складу, а також системи реєстрації витрат і експлуатаційні показники. На всіх стадіях виготовлення, експлуатації і поточного обслуговування рухомого складу безперервно і в реальному масштабі часу реєструються експлуатаційні показники (години роботи, пробіг, витрата палива і т. д.). Крім того, проводиться аналіз відмов з класифікацією їх по серіях рухомого складу, типах вузлів і компаніях, що виробляють вузли та агрегати.

На основі отриманих даних ведеться автоматизоване планування заходів, щодо технічного обслуговування з визначенням потрібних ресурсів. Ці заходи документально реєструються системою, на основі чого формується «історія» поточного утримання рухомого складу і його вузлів з врахуванням інструкцій, що діють. В результаті доступності документації фахівці сервісних служб в будь-якій точці світу отримують інформацію в оперативному режимі.

З іншого боку, фахівцями сервісної служби виробляється безпосередня і точна реєстрація без використання паперових носіїв всіх проведених заходів щодо технічного обслуговування, а також пов'язаних з ними витрат .

Постійний контроль за витратами на обслуговуючий персонал і матеріальними ресурсами забезпечує їх облік протягом всього терміну служби (витрат LCC). Крім того, зібрані дані дозволяють проводити аналіз показників RAMS (надійність, експлуатаційна готовність, ремонтопридатність, безпека).

Постійна документальна реєстрація відповідних параметрів дозволяє систематично фіксувати всі виникаючі відхилення від нормального стану і аналізувати частоту і причини виникнення несправностей. Аналіз показників LCC і RAMS для різних цілей може проводитися з класифікацією за серією рухомого складу, типові вузла, а також по виробникові або компанії, що експлуатує рухомий склад. Таким чином, всі зібрані дані служать основою як для розрахунку показників LCC і RAMS, так і для розробки системи профілактичного і ремонтного обслуговування рухомого складу, що базується на чіткій інформації.

Важливою складовою частиною планування робіт по поточному утриманню тягового рухомого складу, управління і документування є реєстрація і обробка в реальному часі експлуатаційних характеристик (наприклад, частоти обертання дизелів, температури, тиску). Цю важливу задачу виконує система ruDi у складі VSMS, розробленою компанією Vossloh.

Система управління парком залізничного рухомого складу ruDi забезпечує постійну реєстрацію в реальному масштабі часу відповідних експлуатаційних показників і дозволяє проводити в оперативному режимі технічне діагностування і визначати місцезнаходження рухомого складу на будь – якій мережі.

Вона працює незалежно від встановлених на рухомому складі систем управління і тому не робить впливу на допуск рухомого складу до експлуатації. Проте за допомогою стандартних інтерфейсів, таких, наприклад, як інформаційна шина CAN, дані можуть бути отримані з систем управління і оброблені для здійснення функцій контролю.

За рахунок модульної структури система ruDi володіє високою гнучкістю вживання і може встановлюватися як на новому рухомому складі, так і тому, що експлуатується будь – яких компаніях – виробниках. Ще однією перевагою системи менеджменту рухомого складу ruDi є використання радіозв'язку для передачі даних (таких, наприклад, як сигнали від супутникових систем GSM або GPRS), завдяки чому система є мобільною.

Обробка даних може проводитися за бажанням або самим замовником, або сторонньою фірмою, що надає такі послуги. Якщо експлуатуюче підприємство користується послугами спеціалізованої фірми з обробки даних, то через певні проміжки часу воно отримує актуальну інформацію про свій рухомий склад або може само отримати таку інформацію через Інтернет.

Система ruDi має, наступні стандартні функції:

* дистанційна технічна діагностика;
* реєстрація експлуатаційних показників;
* визначення інтервалів технічного обслуговування;
* документальна реєстрація всіх виконаних робіт по технічному утриманню;
* перевірка технічного стану для сертифікації;
* аналіз витрат;
* визначення місцезнаходження рухомого складу;
* захист від розкрадань;
* визначення коефіцієнта використання рухомого складу.

Система ruDi складається з двох основних частин: мобільного устаткування рухомого складу і стаціонарної центральної інформаційної станції .

Основною складовою частиною устаткування рухомого складу є бортовий комп'ютер з радіо модемом (стандарт GSM/GPRS) і під'єднаним блоком супутникового позиціювання GPS для визначення місцезнаходження рухомого складу. При необхідності устаткування рухомого складу може доповнюватися індивідуальними датчиками.

Бортовий комп'ютер був спеціально розроблений для цих цілей. Завдяки дуже міцній конструкції він щонайкраще проявив себе в умовах залізничної експлуатації. У нього є вісім аналогових і вісім цифрових входів, а також достатній об'єм пам'яті для автономної експлуатації локомотиву без обміну інформацією з диспетчерським пунктом.

Концепція оцінки витрат життєвого циклу (LCC) набула широкого поширення у всіх галузях економіки, у тому числі і на залізничному транспорті. Головною перевагою цієї концепції для користувачів є поліпшення ухвалення рішень про інвестиції і можливість робити вплив на загальні витрати з використанням порівняльного аналізу витрат на придбання нової техніки і додаткових витрат, необхідних для ремонту старої. В даний час у зв'язку з постійним скороченням життєвих циклів продукції концепція LCC стає усе більш актуальною.

Основним завданням розрахунків LCC є завчасне планування, регулювання і контроль успішності життєвого циклу продукції. За допомогою методу LCC витрати і доходи можна аналізувати і прогнозувати вже на стадії розробки з метою регулювання подальших експлуатаційних витрат, ще до початку виробництва. Таким чином, розрахунок по методу LCC надає істотну допомогу при планування і ухваленні рішень, а також може використовуватися для обґрунтування або оцінки вже прийнятих рішень.

Метод LCC є інструментом системи менеджменту витрат, який служить для аналізу і регулювання аспектів економічної ефективності протягом всього життєвого циклу об'єкту від його розробки, реалізації і використання аж до утилізації. Крім того, можуть розроблятися різні моделі LCC з врахуванням особливостей підприємства і специфіки завдань.

На відміну від традиційних методів розрахунку, які відображають витрати і доходи, що виникають лише в період експлуатації об'єкту, в методі LCC враховуються, крім того, витрати передування і подальшого періодів, а також інші, пов'язані з об'єктом, такі, наприклад, як витрати в результаті простою. Таким чином, при складанні балансу враховуються не лише витрати, пов'язані з придбанням і виникаючі при використанні об'єкту. Розробка систем менеджменту витрат, таких, як LCC, не може замінити традиційних методів розрахунку витрат, а швидше служить для розширення їх можливостей.

Період, передуючий часу знаходження об'єкту на ринку, охоплює всі види діяльності, необхідні для його виготовлення. Сюди відносяться, наприклад, витрати на дослідження і розробку, а також витрати, пов'язані з впровадженням виробу (наприклад, презентація на виставці) і адаптацією до умов ринку.

Період, наступний за часом знаходження об'єкту на ринку, включає всі витрати, що виникають після виготовлення і реалізації об'єкту. До них відносяться, наприклад, витрати на гарантійне обслуговування, можливе повернення продукції, а також витрати на поточний вміст.

Так, витрати протягом життєвого циклу залізничного рухомого складу складаються з наступних витрат:

* інвестиційних;
* монтаж і введення в експлуатацію;
* споживання енергоносіїв;
* експлуатаційні та ремонтні витрати;
* компенсаційні в результаті простоїв і відмов;
* екологічні;
* на вивід з експлуатації і утилізацію.

Модель розрахунку витрат по методу LCC, призначена для виробника, детально враховує всі попередні витрати на такі роботи, як планування, дослідження і розробка, конструювання.

Система управління парком рухомого складу ruDi є основною складовою частиною сучасних моделей LCC, призначеною для планування і ухвалення рішень, а також для обґрунтування вже прийнятих рішень. Дані, отримані в системі ruDi, дозволяють аналізувати і регулювати витрати по поточному вмісту рухомого складу протягом всього життєвого циклу. Таким чином, система менеджменту рухомого складу ruDi є основою для розрахунку експлуатаційних витрат і витрат на поточне утримання.

### 1.4.2 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 ruDi розроблена в Германії. Проте, зважаючи на те, що у локомотивному господарстві Укрзалізниці використовується програмний комплекс Магістраль, закордонний аналог не може буди використаний для досягнення основних цілей задачі. Це зумовлено тим, що система ruDi не має засобів інтеграції з іншими подібними програмними комплексами, насам перед с комплексом Магістраль. В свою чегру Магістраль не забезпечує потрібний функціонал для доступу до діагностичних даних локомотивів. Тому розроблювальний програмний продукт є унікальним і впершу чергу призначений на обробку даних від програмного комплексу Магістраль.

# 2 ЗОВНІШНЄ ТА ЛОГІЧНЕ ПРОЕКТУВАННЯ

## 2.1 Зовнішнє проектування

### 2.1.1 Опис функціональних характеристик

Дана робота призначена для надання сервісу керування та роботи зі сховищем документів електронної бібліотеки. Виходячи з постановки задачі, приведена характеристика функціональних вимог до програми, що розробляється.

Програмний продукт повинен надавати такі можливості:

* побудова електронного каталогу документів;
* встановлення формалізованих атрибутів для документів;
* прикріплення додатків до документів, та надання доступу до них.

Вимоги до структури каталогу документів:

* каталог складається із розділів та колекцій, що мають ім’я та опис (не є обов’язковим);
* розділ може включати підрозділи та колекції;
* колекція може включати в себе тільки документи.

Документ в репозиторії складається із атрибутів, та вкладених додатків. В ролі додатку можуть виступати будь які бінарні дані, наприклад файли.

Вимоги до атрибутів документів (далі  модель метаданих):

* можливі атрибути документів є формалізованими та включають: код, назву, тип значення та опис (не є обов’язковим);
* кожен елемент моделі метаданих не є обов’язковим;
* деякі атрибути можуть включати декілька значень;
* необхідно надання можливості редагування моделі метаданих: додання, редагування та видалення атрибутів;
* за основу моделі метаданих необхідно взяти Дублінське ядро;
* необхідна підтримка таких типів значень атрибутів:
  + звичайний текст;
  + дати та час, згідно рекомендаціям Date and Time Formats [4];
  + формалізовані скорочення країн [5].

### 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):

* прогляд розділу;
* прогляд колекції;
* прогляд документу;
* скачування додатків;
* пошук документу за атрибутами.

usecase-1.emf

Рисунок 2.1 – Діаграма варіантів використання для користувача системи

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

* керування користувачами;
  + реєстрація;
  + видалення;
* редагування каталогу;
  + редагування розділів;
  + редагування колекцій;
* редагування документів;
  + створення/видалення документів;
  + редагування метаданих;
  + редагування додатків;
* редагування моделі метаданих;
  + додання/видалення типів метаданих;
  + редагування атрибутів метаданих.

usecase-2.emf

Рисунок 2.2 – Діаграма варіантів використання для адміністратора систему

## 2.2 Аналіз баз даних

Програмний комплекс, що розробляється, буде використовувати дві бази даних: база «LibSecurity», що призначена для збереження інформації про користувачів, та їх сесії, та «LibRepository» – для збереження інформації про структуру каталогу, документи та їх вміст. Таке рішення дозоляє розділити сутності, зменшити зв'язок між ними, та забезпечити незалежне розміщення моделей даних на різних екземплярах бази даних, що позитивно вплине на можливості масштабування системи що проектується.

### 2.2.1 Аналіз системи керування базами даних

У якості серверу баз даних обрано PostgreSql – реляційну СУБД промислового рівня з відкритим вихідним кодом. Це зумовлено підтримкою великої кількості платформ на яких може працювати сервер, його технічними характеристиками та ціною (продукт є безкоштовним навіть для промислового використання).

Сервер баз даних надає вбудовані можливості для реалізації повнотекстового пошуку за текстовими полями БД, з використанням стеммінгу ( процес знаходження основи слова для заданого вихідного слова). Також система надає можливості з сегментування таблиць за довільним критерієм, що дозволяє виконувати фізичне розміщення даних в декількох таблицях, та отримувати доступ до них через відображення. Оптимізатор запитів серверу підтримує потужні стратегії кешування та перебудови запитів, що забезпечує підвищену продуктивність при використанні засобів об’єктно-реляційного проеціювання для доступу до моделі даних.

PostgreSql підтримує потокове та виразове репліціювання. Це дозволяє організувати дзеркалювання змісту бази даних, і у випадку відмови основного екземпляру сховища виконати переключення на резервний екземпляр, з актуальним змістом.

### 2.2.2 Сутності та їх представлення в базі даних

Для організації механізми аутентифікації були розроблені наступні сутності:

* користувач(SystemUser)– аутентифікаційні дані користувачів та інформація про них: ПІБ та e-mail;
* сесія(Session) – аутентифікована користувацька сесія, включає ідентифікатор, та дату останньої активності;
* група(UserGroup) – користувацька група: назва, опис;
* членство в групі(GroupMembership) – входження користувача до групи, регламентує зв'язок між логіком користувача та групою.

Для збереження інформації про каталог репозиторії були визначені наступні сутності:

* визначення метаданих(MetadataDefinition) – зберігає тип та ідентифікатор;
* інформація про метадані(MetadataDefinitionInfo) – локалізована назва елементу метаданих;
* документ(Document) – документ в бібліотеці, що включає ідентифікатор та статус;
* вкладення документу(DocumentAttachment) – інформація про додані до документу файли, що включає: ідентифікатор, логін автора, тип контенту, розміри файлу;
* метадані документу(DocumentMetadata) – метадані пов'язані із документом, що включає ідентифікатор документа, тип метаданих, порядковий номер атрибуту та значення;
* колекція(DocumentCollection) – інформація про колекції бібліотеки: ідентифікатор, назва, опис та стан;
* секція(DocumentSection) – інформація про розділи бібліотеки: назва, опис, стан та ідентифікатор;
* роль(AclRole) – визначення ролі в системі: ідентифікатор, назва та опис;
* допуск(AclGrant) – інформація про видачу ролі для деякої групи, що включає: тип допуску, роль, ідентифікатор групи та логін ініціатора;
* запланована задача(ControllerTask) – журнал запланованих фонових задач, що включає: тип задачі, стан, дати взяття в роботу та завершення та сериалізованої інформації про задачу.

### 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 наведені ER-моделі баз «LibRepository» (рис. 2.3) та «LibSecurity»(2.4).

er-2.emf

Рисунок 2.3 – ER модель бази LibRepository

er-1.emf

Рисунок 2.4 – ER-модель бази LibSecurity

### 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 «LibSecurity» (таблиці 2.1-2.4) з переліком полів та їх типів.

Таблиця 2.1 – Характеристика таблиці system\_user

|  |  |
| --- | --- |
| system\_user | |
| Назва поля | Тип |
| userLogin | Varchar(20) |
| username | Varchar(255) |
| userLastName | Varchar(255) |
| userMiddleName | Varchar(255) |
| userEmail | Varchar(255) |
| passVerifier | Varchar(40) |
| passSalt | Varchar(40) |
| creationDate | Timestamp |
| creatorLogin | Varchar(20) |

Таблиця 2.2 – Характеристика таблиці session

|  |  |
| --- | --- |
| session | |
| Назва поля | Тип |
| Sid | Char(22) |
| Login | Varchar(20) |
| lastTouch | Timestamp |
| Active | Boolean |
| sessRemoteIp | Varchar(39) |

Таблиця 2.3 – Характеристика таблиці user\_group

|  |  |
| --- | --- |
| user\_group | |
| Назва поля | Тип |
| groupId | Varchar(64) |
| groupName | Varchar(255) |
| groupDescription | Text |
| groupCreationDate | Timestamp |
| groupType | Char(1) |

Таблиця 2.4 – Характеристика таблиці group\_membership

|  |  |
| --- | --- |
| group\_membership | |
| Назва поля | Тип |
| GroupId | Varchar(64) |
| groupUserLogin | Varchar(20) |

За сутностями бази «LibRepository» також були спроектовані фізичні таблиці для бази даних (таблиці 2.5-2.14), що наведені нижче із переліком полів та їх типів.

Таблиця 2.5 – Характеристика таблиці document\_collection

|  |  |
| --- | --- |
| document\_collection | |
| Назва поля | Тип |
| collectionId | Char(17) |
| sectioned | Char(17) |
| collectionName | Varchar(255) |
| collectionDescription | Text |
| modificationDate | Timestamp |
| collectionStatus | Char(1) |

Таблиця 2.6 – Характеристика таблиці document

|  |  |
| --- | --- |
| document | |
| Назва поля | Тип |
| documentId | Char(17) |
| documentStatus | Char(1) |
| creationDate | Timestamp |
| modificationDate | Timestamp |

Таблиця 2.7 – Характеристика таблиці document\_metadata

|  |  |
| --- | --- |
| document\_metadata | |
| Назва поля | Тип |
| documentId | Char(17) |
| metadataId | Varchar(100) |
| metadataOrder | Int4 |
| metadataValue | Text |

Таблиця 2.8 – Характеристика таблиці document\_section

|  |  |
| --- | --- |
| document\_section | |
| Назва поля | Тип |
| sectionId | Char(17) |
| sectionName | Varchar(255) |
| sectionDescription | Text |
| modificationDate | Timestamp |
| sectionStatus | Char(1) |

Таблиця 2.9 – Характеристика таблиці acl\_grant

|  |  |
| --- | --- |
| acl\_grant | |
| Назва поля | Тип |
| roleId | Varchar(20) |
| grantedObject | Varchar(20) |
| userGroupId | Varchar(64) |
| grantDirection | Char(1) |
| grantedLogin | Varchar(20) |
| grantedDate | Timestamp |

Таблиця 2.10 – Характеристика таблиці acl\_role

|  |  |
| --- | --- |
| acl\_role | |
| Назва поля | Тип |
| roleId | Varchar(20) |
| roleName | Varchar(100) |
| roleDesc | Text |

Таблиця 2.11 – Характеристика таблиці document\_attachment

|  |  |
| --- | --- |
| document\_attachment | |
| Назва поля | Тип |
| documentId | Char(17) |
| atcRef | Char(17) |
| roleId | Varchar(20) |
| creatorLogin | Varchar(20) |
| creationDate | Timestamp |
| isPrimary | Boolean |
| mimeType | Varchar(255) |
| projectedReference | Char(17) |
| attachmentSize | Int8 |
| originalName | Varchar(255) |
| fileHash | Char(32) |

Таблиця 2.12 – Характеристика таблиці metadata\_definition

|  |  |
| --- | --- |
| metadata\_ definition | |
| Назва поля | Тип |
| metadataId | Varchar(100) |
| multivaluedTag | Boolean |
| constraintType | Varchar(30) |

Таблиця 2.13 – Характеристика таблиці metadata\_definition\_info

|  |  |
| --- | --- |
| metadata\_definition\_info | |
| Назва поля | Тип |
| metadataId | Varchar(100) |
| metadataLang | Char(4) |
| metadataName | Varchar(255) |
| metadataDescription | Text |

Таблиця 2.14 – Характеристика таблиці controller\_task

|  |  |
| --- | --- |
| controller\_task | |
| Назва поля | Тип |
| tasked | Varchar(17) |
| taskType | Char(1) |
| taskState | Char(1) |
| taskBlob | Text |
| repObjectId | Varchar(17) |
| created | Timestamp |
| started | Timestamp |
| finished | Timestamp |

### 2.2.4 Нормалізація бази даних

Перша нормальна форма (1НФ) вимагає, щоб кожне поле таблиці БД було неподільним і не містило повторюваних груп.

Для забезпечення 1НФ для таблиці system\_user поле userFIO було розділене на 3 окремі поля (див. рис. 2.5), а в document\_metadata було додано компонент ключа metadataOrder.

nf1.emf

Рисунок 2.5 – Зміни в моделі для забезпечення НФ 1

Друга нормальна форма (2НФ) вимагає, щоб всі поля таблиці залежали від первинного ключа, тобто щоб первинний ключ однозначно визначав запис та не був надлишковим. Ті поля, які залежать лише від частини первинного ключа мають бути виділенні в окрему таблицю.

Для того, щоб база даних відповідала (2НФ) інформацію про локалізацію назв метаданих було винесено до таблиці metadata\_definition\_info(рис. 2.6).

nf2.emf

Рисунок 2.6 – Зміни в моделі для забезпечення НФ 2

Третя нормальна форма (3НФ) вимагає, щоб в таблиці не було транзитивних залежностей між не ключовими полями, тобто щоб значення будь-якого поля яке не входить в первинний ключ, не залежало від іншого поля, яке також не входить в первинний ключ.

Для цього в таблиці document\_attachment було створено композитний первинний ключ (documented,atcId), що відображує залежність вкладення до документу від самого документу (рис. 2.7).

nf3.emf

Рисунок 2.7 – Зміни в моделі для забезпечення НФ 3

# 3 ВНУТРІШНЄ ПРОЕКТУВАННЯ

## 3.1 Вибір парадигми програмування

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 структуризації каталогу документів бібліотеки що ставиться в рамках дипломного проекту відноситься до задач що добре вкладається в концепцію об’єктно-орієнтованого програмування.

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

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;
* своя реалізація через 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. Також це спрощує процедуру маршрутизації засобами балансувальників запитів, через відсутність необхідності аналізувати зміст запиту, орієнтуючись лише на URL. 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, тому веб-сервіс повинен підтримувати обидва, що надасть можливість програмісту використовувати найбільш зручний для нього протокол.

### 3.2.1 Проектування RESTful протоколу

Для архітектури REST є характерним те, що запит ідентифікується за допомогою URI ресурсу, та веб-методу. Призначення веб-методів що встановлюється REST наведені в таблиці 3.1.

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» |

Такий підхід до організації API має назву CRUD(create read update delete, cтворення читання оновлення видалення)-репозиторій.

Протокол повинен підтримувати вхідні запити як у форматі XML, так і в JSON. Для ідентифікації вхідного формату запиту буде використовуватися HTTP-заголовок «Content-Type», що є стандартним заголовком HTTP:

* для XML: application/xml;
* для JSON: application/json.

За замовчуванням, для визначення формату відповіді буде використовуватися значення заголовку «Content-Type», тобто сервер відформатує відповідь в тому форматі в якому отримав запит.

Для випадків коли необхідно отримати відповідь в форматі відмінному від формату запиту, або при відсутності встановити заголовок «Content-Type» протокол підтримує встановлення параметру запиту «format», що підтримує 2 значення:

* /resource/1?format=xml – для отримання відповіді у XML;
* /resource/1?format=json – для отримання відповіді в форматі JSON.

Важливою частиною протоколу обробка помилок. Формат відповіді про помилку має бути стандартизованим, та не залежити від запиту. Також необхідно мати на увазі питання локалізації. Раціональним є надати можливість локалізації клієнтському додатку. Виходячи з цього можна виділити мінімально необхідні дані в відповіді для ідентифікації помилки що виникла:

* код помилки – унікальний номер помилки;
* текст помилки – пояснення, що означає помилка;
* параметри за якими було створено пояснення, для надання можливості додатку відтворити локалізовану версію повідомлення.

Через різну семантику протоколів, формат відповіді для XML та JSON будуть відрізнятися. Так, XML дозволяє ідентифікувати тип відповіді за допомогою назви кореневого тегу, а JSON – лише шляхом встановлення додаткових змінних. Тому для JSON – відповідь може включати об'єкт «data», що зберігає відповідь у випадку успіху, та об’єкт «error» у випадку помилки. Для XML – кореневий тег однозначно ідентифікує об'єкт відповіді.

Приклади відповідей що інформують про помилку для XML та JSON наведені в таблиці 3.2.

Таблиця 3.2 – Формати відповідей, що інформують про помилку

|  |  |
| --- | --- |
| Формат відповіді | Приклад |
| XML | <?xml version="1.0" encoding= "UTF-8" standalone="yes"?>  <error status="A006" description="Session not exists">  <param>1</param>  </error> |
| JSON | {  "error":{  "status":"A006",  "description":"Session not exists",  "params" :[ "1"]  }  } |

### 3.2.2 Протокол веб-сервісу

При проектуванні серверу аутентифікації було виділено користувацькі (табл. 3.4) API, та розроблена таблиця кодів можливих помилок( табл. 3.3).

Таблиця 3.3 – Коди помилок серверу аутентифікації

|  |  |  |
| --- | --- | --- |
| Код | Опис | Аргументи |
| A001 | При обробцi запиту виникла внутрiшня непередбачена помилка | вiдсутнi |
| A002 | Невiрний пароль або логiн | вiдсутнi |
| A003 | Користувача заблоковано | вiдсутнi |
| A004 | Сессiя не iснує | 1. iдентифiкатор сессiї |
| A005 | Сессiя є не активною | 1. iдентифiкатор сессiї |
| A006 | Група не iснує | 1. iдентифiкатор групи |
| A007 | Група вже iснує | 1. iдентифiкатор групи |
| A008 | Некоректний запит до серверу | 1. iм’я параметру |
| A009 | Відмова серверу БД | вiдсутнi |
| A010 | Некоректний запит до БД. | вiдсутнi |
| A011 | В доступi вiдмовлено | вiдсутнi |
| A012 | Користувач з таким логiном вже iснує | 1. логiн користувача |
| A013 | Користувач з таким логiном не iснує | 1. логiн користувача |

Таблиця 3.4 – Користувацький API серверу аутентифікації

|  |  |  |  |
| --- | --- | --- | --- |
| Запит | HTTP-метод | Тіло запиту | Призначення |
| /createSession | POST | «аутентифікаційні дані» | Створює користувацьку сессію |
| /sessionInfo/<id> | GET | - | Повертає об'єкт «сессія» із ідентифікатором <id> |
| /logout/<id> | POST | - | Закриває сессію із ідентифікатором <id> |
| /ping/<id> | GET | - | Оновлює стан активності сессії з ідентифікатором <id> |

## 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. 3.5 – 3.37).

Таблиця 3.5 – CRC-картка для класу CollectionRepository

|  |  |
| --- | --- |
| CollectionRepository | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Реалізує інтерфейс до бази колекцій |  |

Таблиця 3.6 – CRC-картка для класу DocumentRepository

|  |  |
| --- | --- |
| DocumentRepository | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Реалізує інтерфейс до бази документів |  |

Таблиця 3.7 – CRC-картка для класу MetadataRepository

|  |  |
| --- | --- |
| MetadataRepository | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Реалізує інтерфейс до бази метаданих |  |

Таблиця 3.8 – CRC-картка для класу SectionRepository

|  |  |
| --- | --- |
| SectionRepository | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Реалізує інтерфейс до бази розділів |  |

Таблиця 3.9 – CRC-картка для класу ReferenceGenerator

|  |  |
| --- | --- |
| ReferenceGenerator | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Генерація розподілених ідентифікаторів |  |

Таблиця 3.10 – CRC-картка для класу CollectionsManager

|  |  |
| --- | --- |
| CollectionsManager | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Логіка операцій над колекціями | ReferenceGenerator, CollectionRepository, SectionRepository |

Таблиця 3.11 – CRC-картка для класу ConstraintManager

|  |  |
| --- | --- |
| ConstraintManager | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Реалізує алгоритми валідації значень метаданих |  |

Таблиця 3.12 – CRC-картка для класу MetadataManager

|  |  |
| --- | --- |
| MetadataManager | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Реалізує логіку роботи з метаданими | ConstraintManager, MetadataRepository |

Таблиця 3.13 – CRC-картка для класу DocumentManager

|  |  |
| --- | --- |
| DocumentManager | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Реалізує логіку роботи з документами | ConstraintManager, DocumentRepository, MetadataManager, ReferenceGenerator |

Таблиця 3.14 – CRC-картка для класу AclControlBean

|  |  |
| --- | --- |
| AclControlBean | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Реалізує керування таблицями контролю доступу | EntityManager |

Таблиця 3.15 – CRC-картка для класу AclReaderBean

|  |  |
| --- | --- |
| AclReaderBean | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Реалізує операції з побудови списків контролю доступу | EntityManager |

Таблиця 3.16 – CRC-картка для класу DocumentAcl

|  |  |
| --- | --- |
| DocumentAcl | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Реалізує операції з обробкою списків доступу для документів | AclControlBean, AclReaderBean, DocumentManager |

Таблиця 3.17 – CRC-картка для класу RepositorySearch

|  |  |
| --- | --- |
| RepositorySearch | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Реалізує операції з пошуку документів в репозиторії | EntityManager, MetadataManager |

Таблиця 3.18 – CRC-картка для класу SchedulerService

|  |  |
| --- | --- |
| SchedulerService | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Реалізує роботу з планування відкладених задач | ReferenceGenerator, EntityManager |

Таблиця 3.19 – CRC-картка для класу TicketService

|  |  |
| --- | --- |
| TicketService | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Реалізує роботу з формування білетів на скачування документів | StorageManager, DocumentAcl, ReferenceGenerator, TicketPersister |

Таблиця 3.20 – CRC-картка для класу TicketPersister

|  |  |
| --- | --- |
| TicketPersister | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Реалізує роботу з збереження білетів на завантаження документів | CacheManager |

Таблиця 3.21 – CRC-картка для класу StorageManager

|  |  |
| --- | --- |
| StorageManager | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Реалізує логіку роботи з додатками до документів | EntityManager, ReferenceGenerator, ContainerPersister |

Таблиця 3.22 – CRC-картка для класу ContainerPersister

|  |  |
| --- | --- |
| ContainerPersister | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Реалізує логіку збереження файлів додатків на сервері |  |

Таблиця 3.23 – CRC-картка для класу ApiCollectionsController

|  |  |
| --- | --- |
| ApiCollectionsController | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Реалізує логіку обробки запитів до колекцій | CollectionManager, SectionManager |

Таблиця 3.24 – CRC-картка для класу ApiDocumentController

|  |  |
| --- | --- |
| ApiDocumentController | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Реалізує логіку обробки запитів до документів | DocumentManager |

Таблиця 3.25 – CRC-картка для класу ApiMetadataController

|  |  |
| --- | --- |
| ApiMetadataController | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Реалізує логіку обробки запитів до метаданих | MetadataManager |

Таблиця 3.26 – CRC-картка для класу AttachmentsController

|  |  |
| --- | --- |
| AttachmentsController | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Реалізує логіку обробки запитів до додатків до документів | StorageManager |

Таблиця 3.27 – CRC-картка для класу GrantsController

|  |  |
| --- | --- |
| GrantsController | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Реалізує логіку обробки запитів до списків контролю доступу | AclReaderBean, AclControlBean |

Таблиця 3.28 – CRC-картка для класу Document

|  |  |
| --- | --- |
| Document | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Реалізує збереження даних що характеризують документ в бібліотеці | DocumentRepository, Documentanager |

Таблиця 3.29 – CRC-картка для класу Attachment

|  |  |
| --- | --- |
| Attachment | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Описує дані що характеризують додаток, що вкладений в документ Document | StorageManager, ContentPersister |

Таблиця 3.30 – CRC-картка для класу Role

|  |  |
| --- | --- |
| Role | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Описує дані що характеризують роль в електронній бібліотеці | AclReaderBean, AclControlBean |

Таблиця 3.31 – CRC-картка для класу Permission

|  |  |
| --- | --- |
| Permission | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Описує дані що характеризують допуск на деяку дію | AclReaderBean, AclControlBean |

Таблиця 3.32 – CRC-картка для класу User

|  |  |
| --- | --- |
| User | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Описує ідентифікаційні дані користувача системи | AuthenticationBean |

Таблиця 3.33 – CRC-картка для класу Ticket

|  |  |
| --- | --- |
| Ticket | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Описує дані що сберігає сервер про білет на доступ до ресурсу | TicketService, TicketPersister |

Таблиця 3.34 – CRC-картка для класу MetadataTag

|  |  |
| --- | --- |
| MetadataTag | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Описує значення метадані, та їх атрибути що асоціюються із документом | MetadataManager, MetadataRepository |

Таблиця 3.35 – CRC-картка для класу DocumentContent

|  |  |
| --- | --- |
| DocumentContent | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Описує вміст документу, що було виділено із його вкладень | DocumentManager, StorageManager |

Таблиця 3.36 – CRC-картка для класу ControllerTask

|  |  |
| --- | --- |
| ControllerTask | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Описує вміст відкладеної задачі що зберігається в журналі задач | SchedulerService |

Таблиця 3.37 – CRC-картка для класу Collection

|  |  |
| --- | --- |
| Collection | |
| Базовий класс | Похідні класи |
| - | - |
| Відповідальність | Зв’язки |
| Описує дані для роботи із колекціями в пежах програми | CollectionManager |

За первинною класифікацією розробляється діаграма класів (Class Diagram) [10]. Діаграма класів відображає класи і зв’язки між ними.

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

* MVC - цей 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 (GrantsController, AttachmentsController, ApiMetadataController), 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 |  | \* |  | \* |  | \* |  | \* | \* |  |

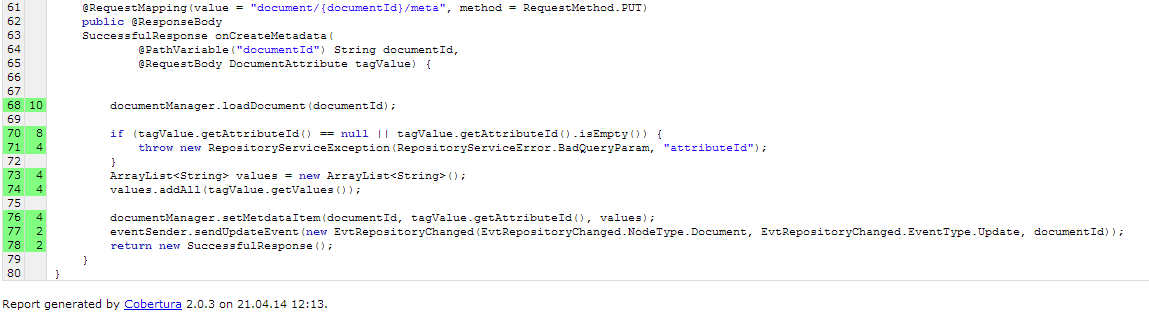


Рисунок 4.1 – Звіт про покриття методу тестами

## 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.

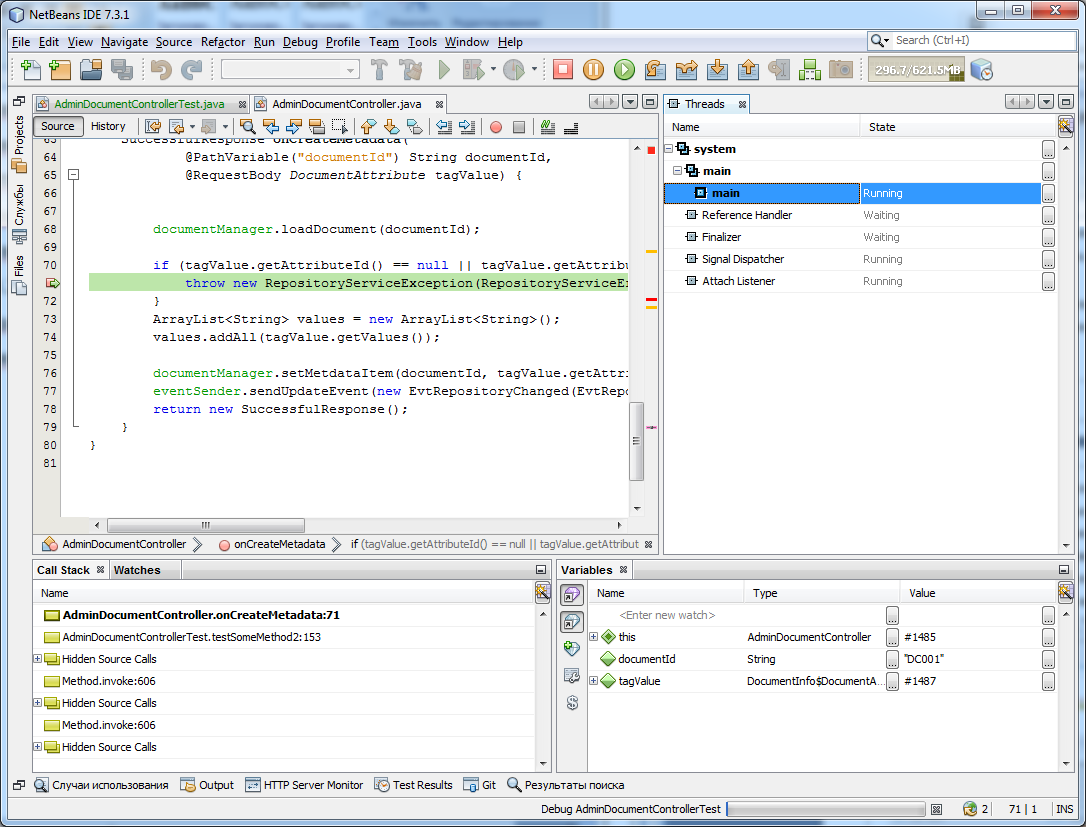


Рисунок 4.2 – Приклад використання точки зупину програми

# 5 ОХОРОНА ПРАЦІ

## 5.1. Аналіз шкідливих та небезпечних виробничих факторів

Відповідно до ДСТУ 2293-99 «Охорона праці. Терміни та визначення основних понять» 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.

Умови праці співробітника відділу, який працює з ПЕОМ, повинні відповідати I або II класу відповідно до Гігієнічної класифікації праці показників шкідливості і небезпеки факторів виробничого середовища, ваги і напруженості трудового процесу відповідно до ДСанПІН 3.3.2-007-1998 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин».

Робота на ПЕОМ пов'язана з наступними шкідливими факторами:

випромінення монітору;

відблиски на екрані монітора;

електромагнітне поле.

Випромінення монітору призводить до швидкої стомленості очей, що у свою чергу, призводить до зниження продуктивності праці і може спричинити хронічні захворювання очей.

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 дотримуватись вимог нормативних документів.

Електромагнітне поле призводить до підвищеної втомлюваності людини, може викликати головний біль. Тривалий вплив електромагнітного поля призводить до погіршення стану здоров’я людини, та може викликати хронічні захворювання.

Відповідно до ДСТУ 2293-99 «Охорона праці. Терміни та визначення основних понять» 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.

До небезпечних факторів відносяться:

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 0,6мА приводить до скорочення м'язів тієї частини, тіла, що піддалася його впливу. Цей струм називається «не відпускає». Значення струму, що перевищує 0,6мА, 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 з ПЕОМ, необхідне проведення проектних заходів: відповідна облаштованість, належне дотримання ергономічних характеристик основних елементів робочого місця, санітарно-гігієнічних вимог і т.п.

Рівні електромагнітного випромінювання і магнітних полів на робочому місці повинні відповідати вимогам нормативних документів України.

За способами захисту людини від ураження електричним струмом відео-термінали, ПЕОМ, периферійні пристосування ПЕОМ і оснащення для обслуговування, ремонту і налагодження ПЕОМ повинні відповідати I класу захисту згідно нормативних документів з охорони праці чи повинні бути заземлені відповідно до НПАОП 40.1-1.21-98 «Правила безпечної експлуатації електроустановок споживачів».

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.

У приміщенні, де одночасно експлуатується чи обслуговується більше п'яти персональних ПЕОМ, на видному доступному місці встановлюється аварійний вимикач, за допомогою якого можливо зробити знеструмлення приміщення (за винятком освітлення).

При роботі неприпустимо:

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.

Для всіх споруджень та приміщень, у яких експлуатуються ВДТ і ПЕОМ, повинна бути визначена категорія по вибухонебезпечній та пожежній безпеці і клас зони згідно ПУЕ. Відповідні позначення повинні бути нанесені на вхідні двері приміщення.

Будинки та ті їхні частини, у яких розміщуються ПЕОМ, повинні мати не нижче II ступеня вогнестійкості відповідно до НАПБ Б.03.002-2007 «Норми визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою». Приміщення для обслуговування, ремонту і налагодження ПЕОМ повинні відноситися:

по пожежонебезпеки до категорії В – пожежонебезпечні приміщення, де розташовуються тверді горючі речовини (ТГР);

по класу приміщення до категорії ІІа по ПУЕ.

Неприпустимим є розташування приміщень категорії А и Б, 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.

Приміщення з ПЕОМ 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 нормативними документами України, 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, вказаних у документі НАПБ А.01.001-2004 «Правила пожежної безпеки в Україні».

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

По інструкції "По охороні праці для операторів" 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 місця з ВДТ чи ПЕОМ, повинна складати не менш 6 м2 , а об’єм – не менш 20 м3.

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

При розташуванні робочих місць із ВДТ необхідно дотримувати наступних вимог:

робочі місця з ВДТ 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 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 для ВДТ повинна бути в межах 680-800 мм, а ширина - забезпечувати виконання операцій у зоні досяжності моторного поля.

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

Робочий стіл для ВДТ 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.

Клавіатуру слід розміщувати на поверхні столу або на спеціальній, регульований за висотою, робочій поверхні окремо від столу на відстані 100 - 300 мм від краю, ближчого до працівника. Кут нахилу клавіатури має бути в межах 5 - 15 град.

Розміщення принтера або іншого пристрою введення-виведення інформації на робочому місці має забезпечувати добру видимість екрану відео термінала, зручність ручного керування пристроєм введення-виведення інформації в зоні досяжності моторного поля: по висоті 900 - 1300 мм, по глибині 400 - 500 мм.

При потребі високої концентрації уваги під час виконання робіт з високим рівнем напруженості суміжні робочі місця з відео терміналами та персональними ЕОМ необхідно відділяти одне від одного перегородками висотою 1,5 - 2 м.

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».

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

## 5.3. Безпека проці при виконанні робіт на ПЕОМ

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

Підготовка робочого місця повинна виконується відповідно до нижче приведених пунктів:

увімкнути систему кондиціювання 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

Користувачі ЕОМ повинні слідкувати за тим, щоб відео термінали, ЕОМ, периферійні пристрої ЕОМ та устаткування для обслуговування, ремонту та налагодження ЕОМ були справними і випробуваними відповідно до чинних нормативних документів.

Щоденно перед початком роботи необхідно проводити очищення екрану відео терміналу від пилу та інших забруднень.

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.

Є неприпустимими такі дії:

- виконання обслуговування, ремонту та налагодження ЕОМ безпосередньо на робочому місці користувача ЕОМ;

- зберігання біля відео термінала та ЕОМ паперу, дискет, інших носіїв інформації, запасних блоків, деталей тощо, якщо вони не використовуються для поточної роботи;

- відключення захисних пристроїв, самочинне проведення змін у конструкції та складі ЕОМ, устаткування або їх технічне налагодження;

- робота з відео терміналами, в яких під час роботи з'являються нехарактерні сигнали, нестабільне зображення на екрані тощо;

- праця на матричному принтері зі знятою (трохи піднятою) верхньою кришкою.

Вимоги безпеки під час обслуговування, ремонту та налагодження ЕОМ

Монтаж, обслуговування, ремонт та налагодження ЕОМ, заміна деталей, пристроїв, блоків повинні здійснюватись тільки при повному відключенні живлення.

Забороняється з'єднувати та роз'єднувати кабелі при підключеній напрузі.

У тих випадках, коли монтаж, обслуговування, ремонт та налагодження ЕОМ або її пристроїв, блоків при відключеному живленні неможливі, виконання цих робіт допускається за умови додержання таких вимог:

- устаткування, допоміжна апаратура та прилади повинні бути заземлені;

- роботи виконуються не менше ніж двома працівниками;

- працівники повинні виконувати роботу інструментом з ізольованими ручками, стоячи на діелектричному килимку, або бути в діелектричних калошах.

Засоби захисту та інструмент необхідно щоразу перед застосуванням оглянути і при виявленні несправностей негайно замінювати.

Користування несправними захисними засобами та інструментом є неприпустимим.

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. Боднарь, Б.Е. Повышение эксплуатационной надежности подвижного состава: Развитие систем диагностирования [Текст] / Б.Е. Боднарь , А.Б. Очкасов // Локомотив-информ. – 2011.- №1-2. - С.56-58
2. Осяев, А.Т. О системе обслуживания локомотивов за рубежом [Текст] / А.Т. Осяев, В.А. Никифоров.// Вестник ВНИИЖТа. – 2012,- №5.- С.56 – 62.
3. Горский, А.В. Стратегия интелектуального ремонта локомотивов [Текст] / А.В. Горский, А.А. Воробьев, А.В. Скребков // Локомотив. – 2012. - №7.- С.33-35.
4. Боднарь, Б.Е. Использование диагностической информации при разработке системы управления техническим состоянием локомотивов [Текст] / Б.Е. Боднарь, Е.Б Боднарь, А.Б.Очкасов // Локомотив-информ. – 2011.- №3-4. - С.10-13.
5. ГКИУ. 468262.006 РЭ. Система диагностики «МАГИСТРАЛЬ-ДЭ1М». Руководство по эксплуатации.- Днеп.: УЭлНИИ. 1996.
6. Горский А.В., Воробьев А.А., Куанышев Б.М. Ремонт - только по результатам диагностики / Локомотив.- 1998.- № 12.- С. 37-39.
7. Бадьян И.О. Аппаратура микропроцессорной системы управления и диагностики электровоза // Системы транспортной автоматики.- 2004.- № 4.- С. 48-52.
8. «RFC 3066 — IETF» [Електронний ресурс] — Режим доступу: <https://www.ietf.org/rfc/rfc3066.txt>.
9. Фаулер М. UML. Основы. / М. Фаулер, К. Скотт; пер. с англ. А. Леоненков – СПб: Символ-Плюс, 2002. – 192 с.
10. «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>.
11. «ООП» [Електронний ресурс] — Режим доступу: <http://ru.wikipedia.org/wiki/%D0%9E%D0%9E%D0%9F>.
12. «CRC-карта» [Електронний ресурс] — Режим доступу: <http://ru.wikipedia.org/wiki/CRC-%D0%BA%D0%B0%D1%80%D1%82%D0%B0>.
13. «Диаграммы UML» [Електронний ресурс] — Режим доступу:  
    <http://khpi-iip.mipk.kharkiv.edu/library/case/leon/gl8/gl8.html>.
14. Приемы объектно-ориентированного проектирования. Паттерны проектирования / [Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес]. – СПб.: Питер, 2001. – 368 с.
15. «Паттерн ServiceLayer» [Електронний ресурс] — Режим доступу: <http://design-pattern.ru/patterns/service-layer.html>.
16. «RFC 4122 - A Universally Unique IDentifier (UUID) URN Namespace» [Електронний ресурс] — Режим доступу: <http://tools.ietf.org/html/rfc4122>.
17. «ACL. Списки контроля доступа» [Електронний ресурс] — Режим доступу: <http://ru.wikipedia.org/wiki/ACL>.
18. «Mock-объект» [Електронний ресурс] — Режим доступу: <http://ru.wikipedia.org/wiki/Mock-%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82>.
19. Экстремальное программирование. Разработка через тестирование / [К. Бек]. – СПб.: Питер, 2003. – 224 с.
20. Искусство тестирования программ, 3-е издание / [Г. Майерс, Т. Баджетт, К. Сандлер] — М.: «Диалектика», 2012. — 272 с.
21. ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин».
22. НПАОП 0.00-1.31-99 «Правила охорони праці під час експлуатації електронно-обчислювальних машин».
23. ДСТУ 2293-99 «Охорона праці. Терміни та визначення основних понять»
24. НАПБ Б.03.002-2007 «Норми визначення категорій приміщень, будинків .та зовнішніх установок за вибухопожежною та пожежною небезпекою».
25. ДБН В.2.5-56:2010 «Пожежна автоматика будинків і споруджень».
26. ДСН 3.3.6.037-99 «Санітарні норми виробничого шуму, ультразвуку та інфразвуку».
27. ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень».