ЗМІСТ

СПИСОК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

СЕД	– Системи електронного документообігу
СКБД	– Система керування базами данних
API	–Application Programming Interface
IDE	– Integrated Development Environment
Java SE SDK	– Java Standard Edition Software Development Kit
JSON	– JavaScript Object Notation

ВСТУП

В сучасних організаціях системи електронного документообігу стають обов’язковим елементом інфраструктури. З їх допомогою підвищується ефективність діяльності компаній.

Електронний документообіг – це спосіб організації роботи з документами, при якому основна маса документів використовується в електронному вигляді і зберігається централізовано. Система електронного документообігу – програмне забезпечення, розраховане на багато користувачів, яке передбачає організацію роботу з електронними документами, а також взаємодію між співробітниками. Вже сьогодні існує багато безкоштовних і платних рішень, якими може скористатись організація. Однак всі вони мають ті чи інші недоліки. По-перше, всі вони потребують інсталяції і кропіткого налаштування. В деяких випадках їх потрібно підлаштувати під потреби підприємства. По-друге, такі системи потребують постійної підтримки фахівцями, а це додатковий персонал.

Звітом має бути строга форма в документі Excel. Метою створення системи є спрощення роботи зі звітами. Була потрібна система, яка могла б інтегруватися у вже існуючі обчислювальні й інформаційні потужності підприємства, працювати на застосовуваних у компанії технологіях, надавати зручний і зрозумілий графічний інтерфейс, який дозволив би користувачу без підготовки відразу почати працювати з нею.

На основі перерахованих вище вимог і недоліках, було прийнято рішення про необхідність створення власної внутрішньокорпоративної системи.

Метою даної комплексної магістерської роботи є розробка та реалізація WEB-орієнтованої системи електронного документообігу підприємства з територіально-розподіленою структурою.

Метою другої частини комплексної магістерської роботи є розробка серверної частини системи електронного документообігу підприємства.

Для досягнення поставленої мети в роботі необхідно вирішити наступні завдання:

- дослідження існуючих аналогів СЕД;
- аналіз технічного завдання;
- функціональний опис системи електронного документообігу;
- логічне проектування системи електронного документообігу;

- вибір та обґрунтування технічних вимог та інструментальних засобів для фізичного проектування системи електронного документообігу;
- розробка серверної частини системи електронного документообігу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження предмета, цілей і особливостей системи електронного документообігу

Зараз організації, що переходять на електронний документообіг, в першу чергу думають про ефективність. Підвищення ефективності можливо двома способами – через збільшення результату і зменшення витрат. Сучасні системи електронного документообігу використовують обидва ці способи. Так, зниження витрат сприяють:

- Скорочення витрат на паперові документи;
- Скорочення непродуктивних витрат робочого часу співтробітників. За оцінками західних консалтингових компаній, частка витрат часу на виконання рутинних, непродуктивних операцій над документами може становити до 20-30% усього робочого часу (а на практиці – до 60-70%). Знизити такі витрати – одна з найважливіших цілей впровадження СЕД.

На результативність діяльності організації при впровадженні СЕД впливають:

- Прискорення інформаційних потоків;
- Змінення корпоративної культури.

Впроваджуючи систему електронного документообігу, організації найчастіше планують вирішити такі завдання:

- Підвищення ефективності управління шляхом автоматизації контролю виконання, більшої прозорості діяльності підрозділів і окремих співробітників;
- Автоматизація бізнес-процесів з їх одночасною оптимізацією;
- Забезпечення підтримки накопичення, управління і організації доступу до корпоративної інформації і знань;
- Протоколювання діяльності організації в цілому, її окремих підрозділів, робочих груп, співробітників з використанням цієї інформації для підтримки прийняття рішень;
- Скорочення обороту паперових документів;
- Спрощення і здешевлення зберігання документів, що використовуються в поточній діяльності, за рахунок створення оперативного електронного архіву.

1.2 Аналіз існуючих систем

Для визначення вимог, функцій розроблюваної системи електронного документообігу, було проведено необхідний аналіз існуючих і функціонуючих в глобальній мережі інтернет аналогічних СЕД[1].

На сьогоднішній день існує безліч систем документообігу. Розглянемо деякі з них.

1) Docsvision

Система Docsvision (рис. 1.1) – програмний продукт, призначений для створення автоматизованих корпоративних рішень з управління документами і бізнес-процесами. Включає предметно-орієнтовану платформу з відкритими інтерфейсами прикладного програмування для розробки замовлених додатків і готові типові додатки з можливостями параметричної настройки.

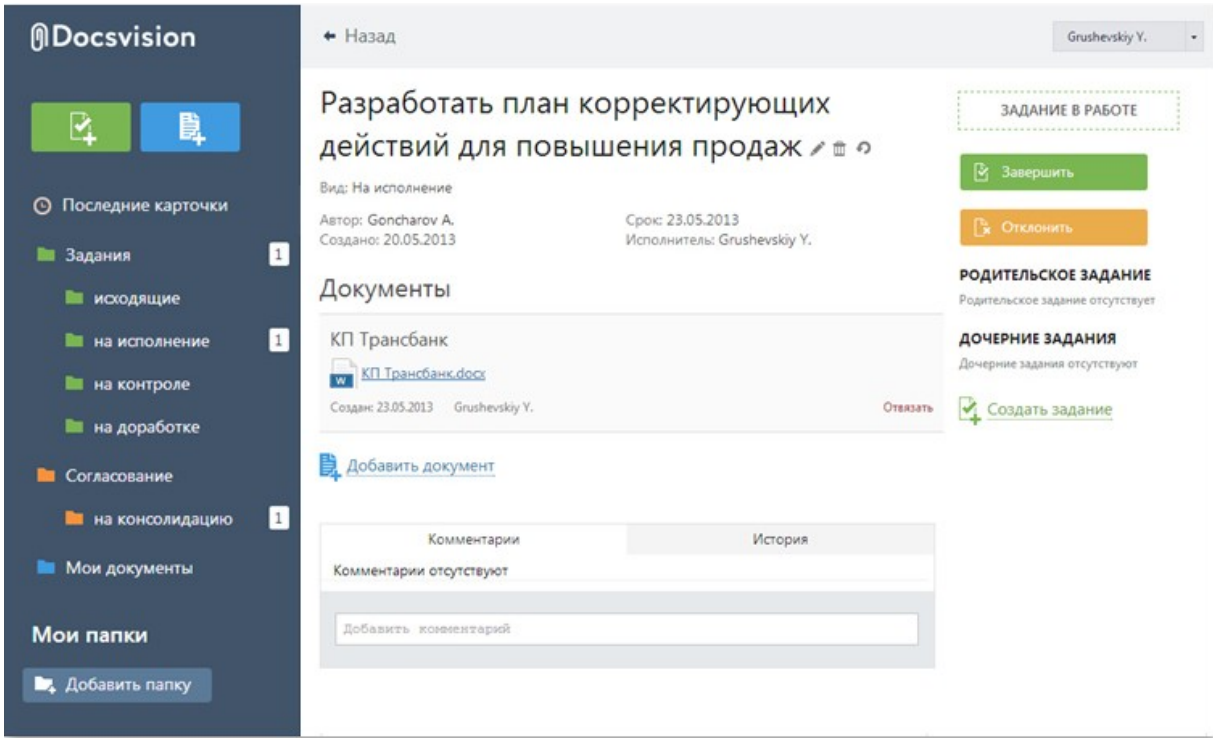


Рисунок 1.1 – Головна сторінка Docsvision

Платформа Docsvision 5 є базисом для електронного документообігу – створення системи управління документами і бізнес-процесами і включає в себе основні технології та базові об'єкти, призначені для зберігання інформації, доступу до неї, а також забезпечують користувачеві зручний інтерфейс для роботи. Платформа складається з клієнтської і серверної частин.

Конструктори, модулі, шлюзи до інших систем і готові програми Docsvision дозволяють гнучко налаштувати систему під рішення конкретних бізнес-завдань замовника.

2) Naumen DMS

Система управління документообігом Naumen DMS володіє широким функціоналом в таких областях, як управління документами і бізнес-процесами. Екран оброблених документів наведено на рис.1.2.

Функції зберігання структурованих і неструктурованих даних в електронному вигляді є основними в Naumen DMS. Підсистема прав доступу розмежовує доступ до даних, які містяться в окремих об'єктах системи (документи, папки, довідники, журнали).

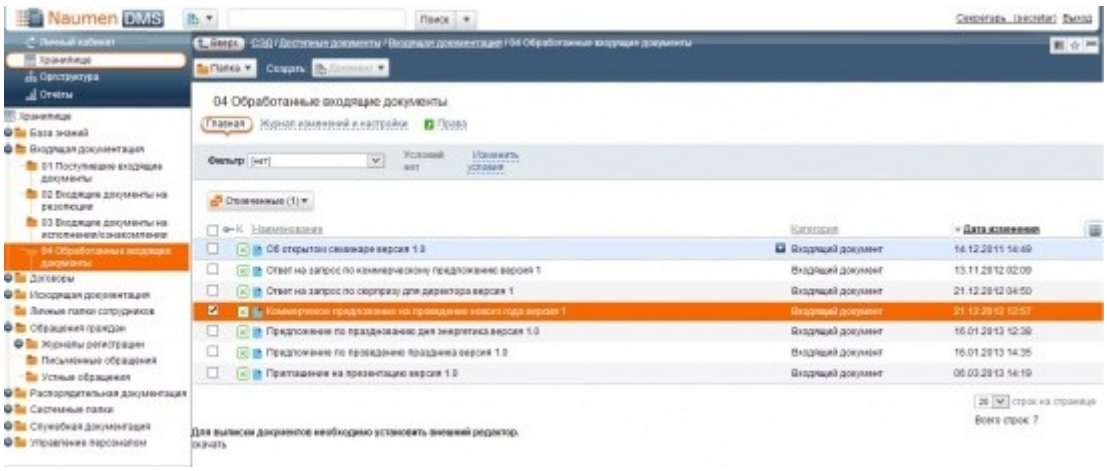


Рисунок 1.2 – Екран оброблених документів Naumen

Система має ефективну і гнучкою підсистемою пошуку. Пошук може проводитися за назвою і змістом документа, в тому числі за змістом прикладеного файлу, а також за різними критеріями пошуку. Кожен користувач може зберігати пошукові запити для подальшого використання.

Naumen DMS дозволяє налаштовувати процеси обробки документів двома способами. Спрощена розробка процесів виконується за допомогою настройки життєвих циклів документів і на базі типових дій з документами (реєстрація, переміщення, створення нової версії, запуск типового подпроцесса і ін.) Для спрощеної налаштування може бути використаний власний дизайнер життєвих циклів Naumen LCM, що дозволяє здійснювати настройку в графічному вигляді.

Для більш складних процесів, в тому числі і для інтеграційних міжсистемних процесів, передбачено окремо додаток-редактор і сервер

виконання процесів, що використовує стандартизовану нотацію і модель бізнес-процесів – BPMN (англ. Business Process Model and Notation). Разом з Naumen DMS поставляється сервер виконання процесів Activiti BPM, що володіє також і графічним дизайнером процесів. Дана технологія дозволяє розробляти системи класу BPM і EAI.

3) DocSpace

DocSpace – це програмна платформа, повністю інтегрована в Microsoft SharePoint, призначена для створення рішень в області автоматизації управлінського документообігу і діловодства, а також вирішення інших завдань з управління неструктурованих контентом і організації колективної роботи. Електронний архів для пошуку документів наведено на рис.1.3.

Програмна платформа DocSpace – це гнучкий інструмент для швидкого створення рішень в області електронного документообігу, мінімізації витрат впровадження і подальшого використання, зручною адаптації під індивідуальні потреби Замовника.

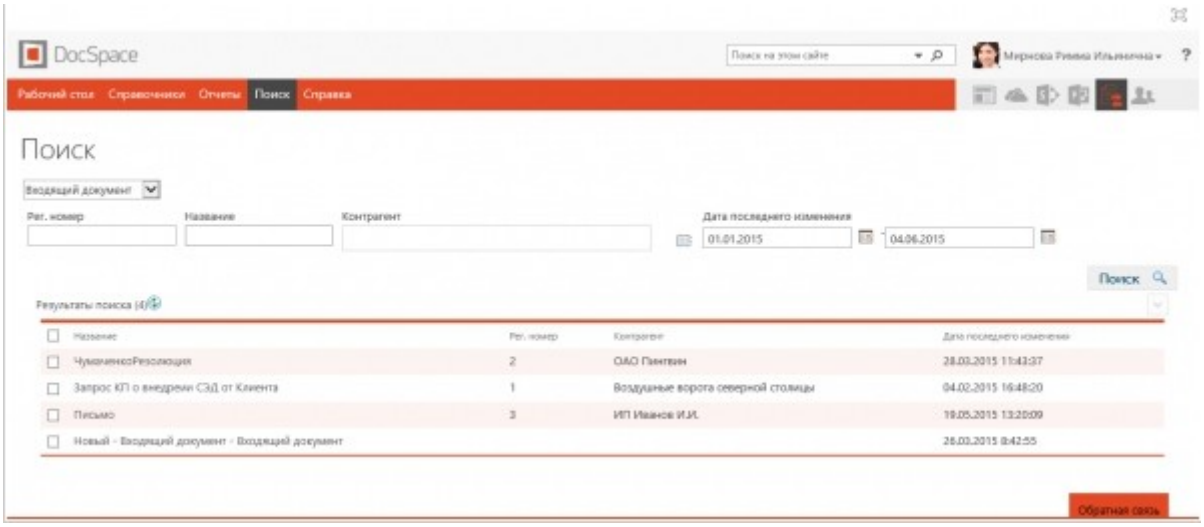


Рисунок 1.3 – Електронний архів DocSpace

Коробкова СЕД / ECM-система DocSpace – це готові конфігурацій для автоматизації типових процесів електронного документообігу на основі платформи DocSpace.

Завдання системи – забезпечити якісний менеджмент по відношенню до документації організації як повноцінному ресурсу управління.

Для співробітників СЕД / ECM-система DocSpace є зручним веборієнтированное простір для щоденної роботи, управління документами і завданнями.

4) Directum

DIRECTUM – система управління корпоративним контентом (Enterprise Content Management), на базі можливостей якої будується повноцінна система електронного документообігу та інфраструктура ефективної взаємодії співробітників підприємства від рівня топ-менеджменту до кінцевих виконавців. В системі DIRECTUM передбачений варіант глобального пошуку (рис. 1.4), при якому результатом може стати будь-який об'єкт системи (документи, папки, завдання, завдання). Цей варіант зручний в тому випадку, коли користувач не знає, який об'єкт є шуканим, і йому необхідно скористатися всіма джерелами інформації для отримання необхідних даних.

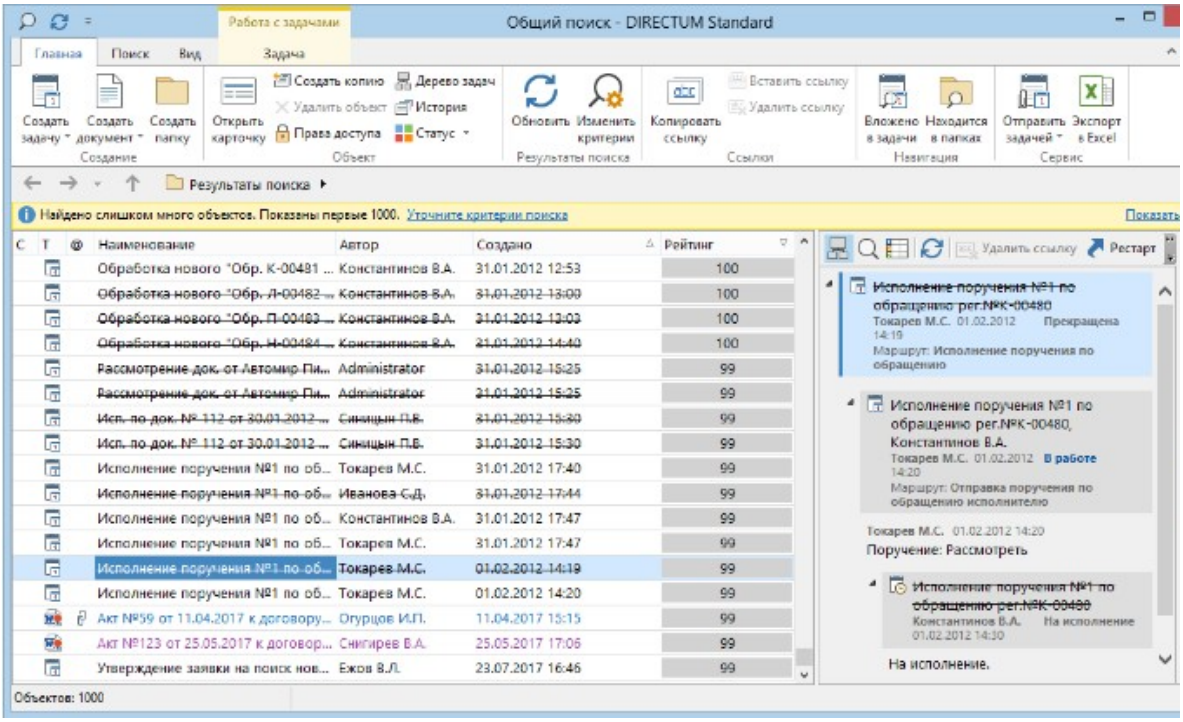


Рисунок 1.4 – Екран загального пошуку

DIRECTUM повністю відповідає концепції ECM згідно моделі, розробленої асоціацією з питань управління інформацією і зображеннями (AIIM), і включає в себе функції:

- Введення і перетворення документів;
- Управління спільною роботою;
- Довготривалого зберігання документів;
- Забезпечення їх цілісності;
- Доставки інформації.

Архітектура системи включає потужну ECM-платформу, яка забезпечує високу масштабованість і гнучкість рішень, територіально розподілену роботу, інтеграцію з корпоративним IT-оточенням, інтерфейси доступу до системи.

Готові модулі та технічні рішення закривають прикладні завдання з управління та погодженням документів, роботі з нарадами, договорами, ведення діловодства і т.д.

Специфічні завдання бізнесу реалізуються в широкому наборі готових бізнес-рішень. Кожне з них включає технічне рішення, послуги бізнес-консалтингу та навчання, методики впровадження і прорахунок кінцевого ефекту.

5) ELMA ECM

ELMA – система управління компанією, що дозволяє побудувати ефективну взаємодію сотрудніковкомпанії і контролювати їх діяльність з метою підвищення якості роботи всієї компанії. Інтерфейс системи наведено на рис. 1.5.

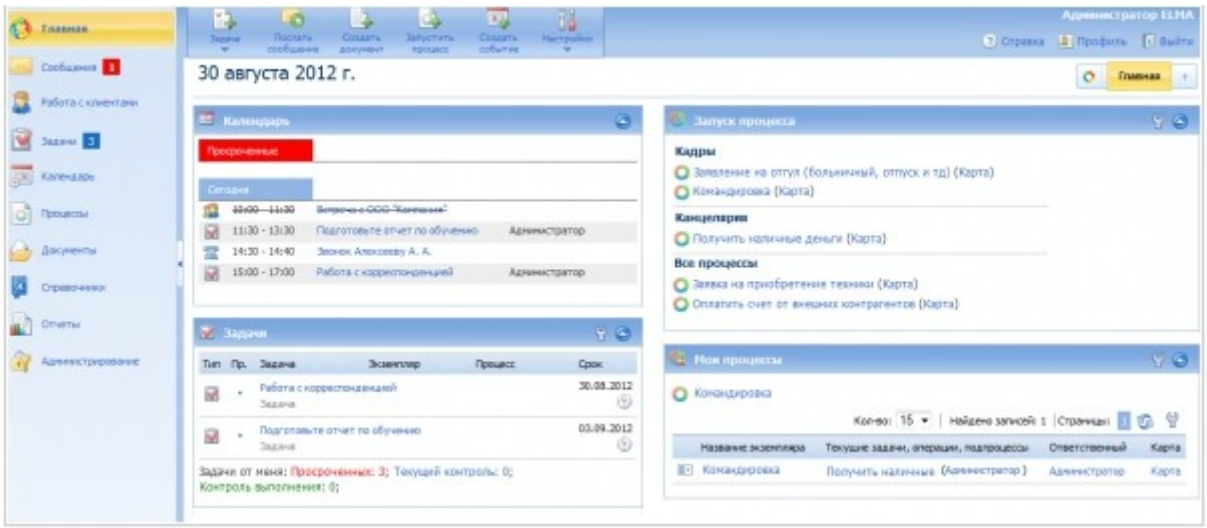


Рисунок 1.5 – Інтерфейс системи ELMA

ELMA складається з набору додатків для управління компанією, які можуть бути придбані і функціонувати як окремі додатки, так і разом в єдиному інформаційному просторі. Електронний документообіг автоматизує типові процеси діловодства і ведення електронного документообігу. Використання механізму WorkFlow дозволяє швидко налаштувати систему відповідно до потреб компанії.

Управління бізнес-процесами. Додаток реалізує концепцію BPM (Business Process Management), що дозволяє будувати гнучкі адаптивні інформаційні системи; рішення, здатні оперативно змінюватися разом зі зміною бізнес-процесів компанії. Для моделювання бізнес-процесів додаток використовує міжнародний стандарт BPMN.

Управління показниками. Додаток дозволяє реалізувати в компанії принципи цільового управління і винагороди за результатами праці. Впровадження в «легкій формі» дозволить сформувати набір цільових показників ефективності, і налагодити оперативний контроль за досягненням планових значень показників в розрізі цілей і стратегії розвитку компанії.

1.3 Постанова завдання

У всіх перерахованих вище систем є кілька недоліків.

По-перше, вони вимагають розгортки і кропіткого налаштування. В окремих випадках, необхідна доробка фахівцями.

По-друге, вони пропонують зайвий функціонал, який в даному випадку не потрібний і надлишковий.

По-третє, деякі з них потребують чималих потужностей системи.

По-четверте, практично всі подібні системи платні і покупка такої системи може стати серйозним ударом по бюджету підприємства.

Зважаючи на ці недоліки і специфічні вимоги, було прийнято рішення розробити свою систему.

Наша програма має будуватися на основі структури MVC. Model-view-controller (Модель-представлення-контролер) – це не шаблон проекту, це конструкційний шаблон, який описує спосіб побудови структури нашого застосування, сфери відповідальності та взаємодія кожної з частин в даній структурі[2].

Вперше вона була описана в 1979 році, звичайно ж, для іншого оточення. Тоді не існувало концепції веб додатки. Tim Berners Lee (Тім Бернерс Лі) посіяв насіння World Wide Web (WWW) на початку дев'яностих і назавжди змінив світ. Шаблон, який ми використовуємо сьогодні, є адаптацією оригінального шаблону до веб розробці.

Шалена популярність даної структури в веб додатках склалася завдяки її включенню в два середовища розробки, які стали дуже популярними: Struts і Ruby on Rails. Ці дві середовища розробки намітили шляхи розвитку для сотень робочих середовищ, створених пізніше. Ідея, яка лежить в основі

конструкційного шаблону MVC, дуже проста: потрібно чітко розділяти відповідальність за різне функціонування в наших програмах (рис. 1.6)/

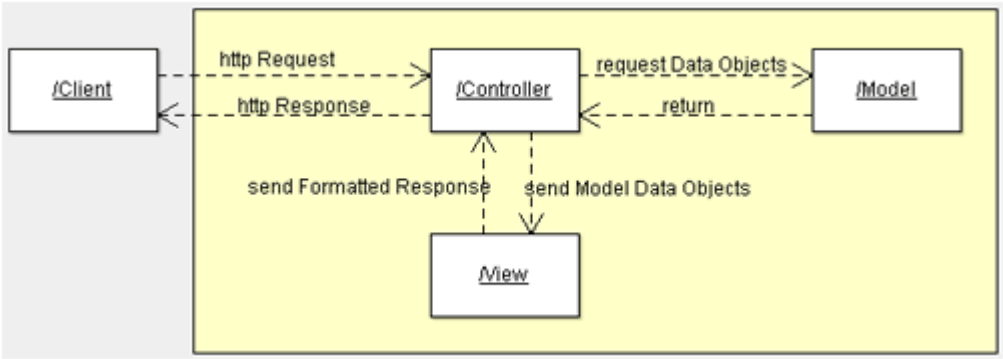


Рисунок. 1.6 – Загальна схема структури MVC

Додаток розділяється на три основних компоненти, кожен з яких відповідає за різні завдання. Давайте детальніше розберемо компоненти на прикладі:

Контролер (Controller)

Контролер керує запитами користувача (одержувані у вигляді запитів HTTP GET або POST, коли користувач натискає на елементи інтерфейсу для виконання різних дій). Його основна функція – викликати і координувати дію необхідних ресурсів і об'єктів, потрібних для виконання дій, що задаються користувачем. Зазвичай контролер викликає відповідну модель для задачі і вибирає відповідний вид.

Модель (Model)

Модель – це дані і правила, які використовуються для роботи з даними, які представляють концепцію управління додатком. У будь-якому додатку вся структура моделюється як дані, які обробляються певним чином. Що таке користувач для додатка - повідомлення або книга? Тільки дані, які повинні бути оброблені відповідно до правил (дата не може вказувати в майбутнє, e-mail повинен бути в певному форматі, ім'я не може бути довшим X символів, і так далі).

Вид (View)

Вид забезпечує різні способи представлення даних, які отримані з моделі. Він може бути шаблоном, який заповнюється даними. Може бути кілька різних видів, і контролер вибирає, який підходить якнайкраще для поточної ситуації.

Веб додаток зазвичай складається з набору контролерів, моделей і видів. Контролер може бути влаштований як основний, який отримує всі запити і викликає інші контролери для виконання дій в залежності від ситуації.

2 ВИБІР І ОБГРУНТУВАННЯ ПРОГРАМНИХ ЗАСОБІВ ТА АРХІТЕКТУРИ СИСТЕМИ

2.1 Загальна архітектура системи

В будь-якій мережі, яка побудована на сучасних мережевих технологіях, присутні елементи клієнт-серверного взаємодії, найчастіше на основі дворівневої архітектури (рис. 2.1). Дворівневою (two-tier, 2-tier) вона називається через необхідність розподілу трьох базових компонентів між двома вузлами (клієнтом і сервером)[3].

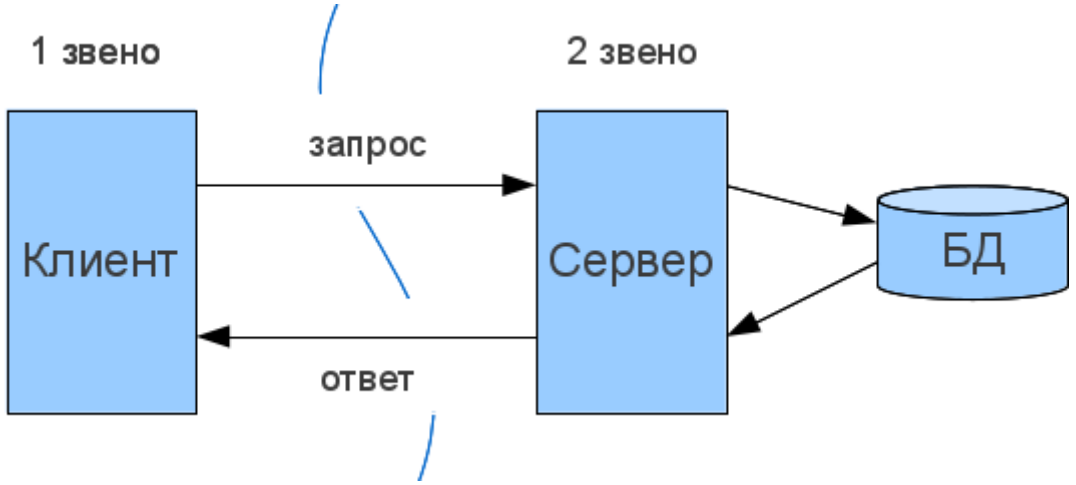


Рисунок 2.1 – Дворівнева «клієнт-серверна» архітектура

Дворівнева архітектура використовується в клієнт-серверних системах, де сервер відповідає на клієнтські запити безпосередньо і в повному обсязі, при цьому використовуючи тільки власні ресурси. Тобто сервер не викликає сторонні мережеві програми, але не звертається до сторонніх ресурсів для виконання будь-якої частини запиту.

Розташування компонентів на стороні клієнта або сервера визначає наступні основні моделі їх взаємодії в рамках двухзвенной архітектури:

- сервер терміналів – розподілене представлення даних;
- файл-сервер – доступ до віддаленої базі даних і файлових ресурсів;
- сервер БД – віддалене уявлення даних;
- сервер додатків – віддалений додаток.

Історично першою з'явилася модель розподіленого представлення даних (модель сервер терміналів). Вона реалізовувалася на універсальній ЕОМ (мейнфрейми), що виступала в ролі сервера, з підключеними до неї

алфавітно-цифровими терміналами. Користувачі виконували введення даних з клавіатури терміналу, які потім передавалися на мейнфрейм і там виконувалася їх обробка, включаючи формування «картинки» з результатами. Ця «картинка» і поверталася користувачеві на екран терміналу.

З появою персональних комп'ютерів і локальних мереж, була реалізована модель файлового сервера, який представляв доступ файлових ресурсів, в т.ч. і до віддаленої бази даних. В цьому випадку виділений вузол мережі є файловим сервером, на якому розміщені файли бази даних. На клієнтах виконуються додатки, в яких поєднані компонент уявлення та прикладної компонент (СУБД і прикладна программма), що використовують підключену віддалену базу як локальний файл. Протоколи обміну при цьому представляють набір низькорівневих викликів операцій файлової системи.

Така модель показала свою неефективність з огляду на те, що при активній роботі з таблицями БД виникає велике навантаження на мережу. Частковим вирішенням цієї проблеми є підтримка тиражування (реплікації) таблиць і запитів. В цьому випадку, наприклад при зміні даних, оновлюється не вся таблиця, а тільки модифікована її частина.

З появою спеціалізованих СУБД з'явилася можливість реалізації іншої моделі доступу до віддаленої базі даних – моделі сервера баз даних[5]. В цьому випадку ядро СУБД функціонує на сервері, прикладна програма на клієнті, а протокол обміну забезпечується за допомогою мови SQL. Такий підхід в порівнянні з файловим сервером веде до зменшення завантаження мережі й уніфікації інтерфейсу "клієнт-сервер». Однак, мережевий трафік залишається досить високим, крім того, як і раніше неможливо задовільний адміністрування додатків, оскільки в одній програмі поєднуються різні функції.

З розробкою і впровадженням на рівні серверів баз даних механізму збережених процедур з'явилася концепція активного сервера БД. У цьому випадку частина функцій прикладного компонента реалізовані у вигляді збережених процедур, що виконуються на стороні сервера. Решта прикладна логіка виконується на стороні клієнта. Протокол взаємодії – відповідний діалект мови SQL[6].

Переваги такого підходу очевидні:

- можливо централізоване адміністрування прикладних функцій;
- зниження вартості володіння системою (TOC, total cost of ownership) за рахунок оренди сервера, а не його покупки;

- значне зниження мережевого трафіку (тому що передаються не SQL-запити, а виклики збережених процедур).

Основний недолік – обмеженість коштів розробки збережених процедур у порівнянні з мовами високого рівня.

Реалізація прикладного компонента на стороні сервера надає наступну модель – сервер додатків. Перенесення функцій прикладного компонента на сервер знижує вимоги до конфігурації клієнтів і спрощує адміністрування, але представляє підвищені вимоги до продуктивності, безпеки і надійності сервера[7].

На сьогоднішній день намічається тенденція повернення до того, з чого починалася клієнт-серверна архітектура – до централізації обчислень на основі моделі термінал-сервера. У сучасній реінкарнації термінали відрізняються від своїх алфавітно-цифрових предків тим, що маючи мінімум програмних і апаратних засобів, представляють мультимедійні можливості. Роботу терміналів забезпечує високопродуктивний сервер, куди винесено все, аж до віртуальних драйверів пристроїв, включаючи драйвери відеопідсистеми.

2.2 Вибір системи керування базами даних

У якості бази даних була обрана Oracle Database. Oracle Database або Oracle RDBMS – це об'єктно-реляційна система, яка підтримує деякі технології, які реалізують об'єктно-орієнтований підхід, тобто забезпечують управління створення та використання баз даних[8].

Oracle Database поставляється в чотирьох різних редакціях, орієнтованих на різні сценарії розробки і розгортання додатків. Крім того, корпорація Oracle пропонує кілька додаткових програмних продуктів, які розширюють можливості Oracle Database 11g для роботи з конкретними прикладними пакетами[9].

Нижче перераховані існуючі редакції СКБД Oracle Database 11g: Oracle Database 11g Standard Edition One характеризується безпрецедентною простотою експлуатації, міццю і вигідним співвідношенням ціни і продуктивності для додатків масштабу робочих груп, окремих підрозділів або програм, які потребують середовищі інтернет / інтранет. Працюючи в різних середовищах, починаючи від односерверних конфігурацій для малого бізнесу і закінчуючи розподіленими середовищами великих філій, Oracle Database 11g Standard Edition One володіє всіма функціональними

можливостями для забезпечення роботи критичних для бізнесу додатків. Редакція Standard Edition One ліцензується тільки для серверів, які мають не більше двох процесорів.

Вибір обумовлений наявністю у підприємства ліцензії даної бази і її активним використанням вже на поточний момент.

2.3 Вибір мови програмування

В якості мови розробки програми, її серверної частини-контролерів, сервісів, інтерфейсів доступу до даних у базі даних, пропонується Java, платформа J2EE.

Одна з головних переваг мови Java – її незалежність від платформи, на якій виконуються програми. Таким чином, один і той же код можна запускати під управлінням операційних систем Windows, Linux, FreeBSD, Solaris, Apple Mac та ін. Це стає дуже важливим, коли програми завантажуються за допомогою глобальної мережі інтернет і використовуються на різних платформах.

Крім того, Java – повністю об'єктно-орієнтована мова. Всі сутності в мові Java є об'єктами, за винятком небагатьох основних типів (primitive types), наприклад чисел[10].

Важливо і те, що розробляти на Java програми, які не містять помилок, значно легше, ніж, наприклад, на C++.

Вся справа в тому, що розробниками мови Java з компанії Sun був проведений фундаментальний аналіз програм мовою C + +. Аналізувалися "вузькі місця" вихідного коду, які й призводять до появи помилок, що дуже важко виявити. Тому було прийнято рішення проектувати мову Java з урахуванням можливості створювати програми, в яких була б передбачена та мінімізована вірогідність появи найбільш поширених помилок.

Якщо розглядати ці аспекти і брати до уваги поставлені завдання, то можна стверджувати, що на даний момент немає більш зручного і надійного засобу для реалізації проекту[11].

Для спрощення розробки та підвищення якості проекту, ми будемо використовувати кілька додаткових технологій. Перша з них-Spring Framework або коротко Spring.

Spring

Spring – універсальний фреймворк з відкритим вихідним кодом для Java-платформи. Незважаючи на те, що Spring Framework не забезпечує яку-

небудь конкретну модель програмування, він став широко поширеним в Java-співтоваристві головним чином як альтернатива і заміна моделі Enterprise JavaBeans[12].

Spring Framework надає велику свободу Java-розробникам в проектуванні, крім того, він надає добре документовані і легкі у використанні засоби вирішення проблем, що виникають при створенні додатків корпоративного масштабу.

Між тим, особливості ядра Spring Framework можуть бути застосовні в будь-якому Java-додатку, і існує безліч розширень і удосконалень для побудови веб-додатків на Java Enterprise платформі. З цих причин Spring отримав велику популярність і визнається розробниками як стратегічно важливий фреймворк.

Spring Framework може бути розглянутий як колекція менших фреймворків або фреймворків у фреймворці (рис. 2.2).

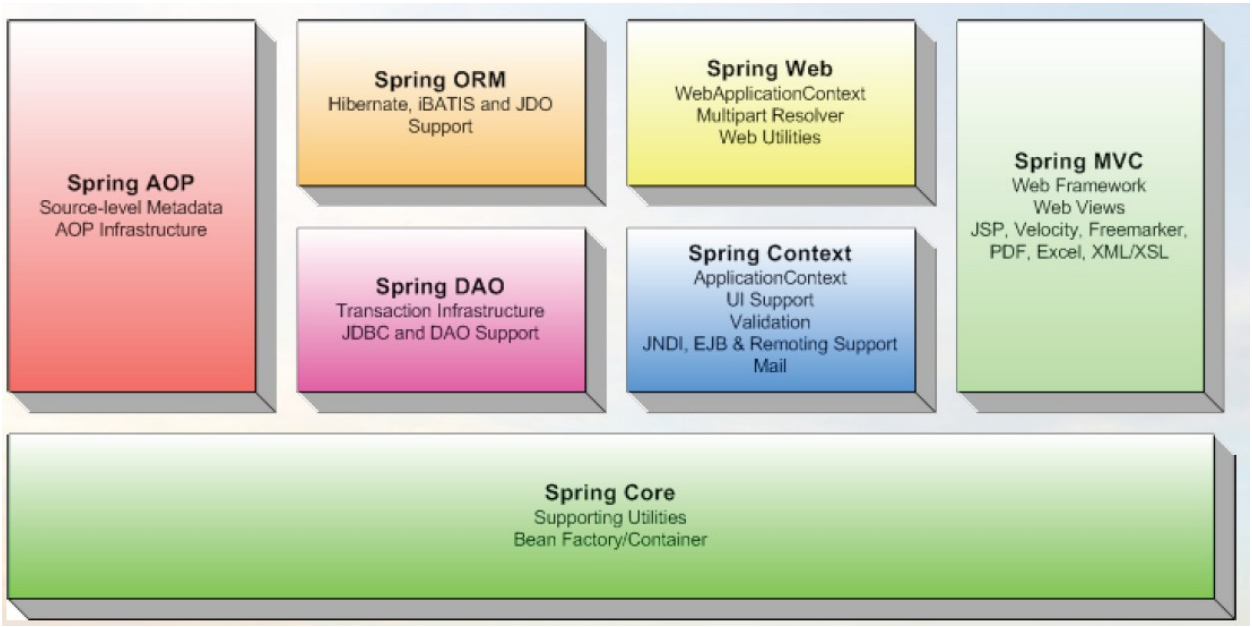


Рисунок 2.2 – Базові модулі фрейморвку Spring

Більшість цих фреймворків може працювати незалежно один від одного, проте, вони забезпечують більшу функціональність при спільному їх використанні. Ці фреймворки діляться на структурні елементи типових комплексних программ. Ми будемо використовувати наступні:

- Inversion of Control контейнер: конфігурування компонентів додатків і управління життєвим циклом Java об'єктів;

- Фреймворк доступу до даних: працює з системами керування базами даних на Java платформі використовуючи JDBC і Object-relational mapping, забезпечуючи вирішення завдань, які повторюються у великому числі Java-based environments;
- Фреймворк Model-view-controller: каркас, заснований на HTTP і сервлетах що надає безліч можливостей для розширення та налаштування (customization);
- Фреймворк аутентифікації і авторизації: конфігуруємий інструментарій процесів аутентифікації і авторизації, підтримує багато популярних і ставших індустріальними стандартами протоколів, інструментів, практик через дочірній проект Spring Security (раніше відомий як Acegi);
- Тестування: каркас, що підтримує класи для написання модульних та інтеграційних тестів.

Inversion of Control контейнер

Центральною частиною Spring Framework є Inversion of Control контейнер, який надає засоби конфігурування та управління об'єктами Java за допомогою рефлексії. Контейнер відповідає за управління життєвим циклом об'єкта: створення об'єктів, виклик методів ініціалізації та конфігурування об'єктів шляхом зв'язування їх між собою.

Об'єкти створювані контейнером також називаються «керовані об'єкти» або Beans. Зазвичай конфігурування контейнера здійснюється шляхом завантаження XML файлів, що містять визначення Bean'ів та надають інформацію необхідну для створення bean'ів. Об'єкти можуть бути отримані або за допомогою пошуку залежності, або впровадженням залежності. Пошук залежності – шаблон проектування, коли викликаючий об'єкт запитує у об'єкта-контейнера екземпляр об'єкта з певним ім'ям або певного типу. Впровадження залежності – шаблон проектування, коли контейнер передає примірники об'єктів за їх іменем іншим об'єктам або за допомогою конструктора, або властивості, або фабричного методу[13].

Фреймворк доступу до даних

Spring надає свій шар доступу до баз даних і підтримує всі популярні бази даних.

Для всіх цих фреймворків, Spring надає такі особливості:

- Управління ресурсами – автоматичне отримання і звільнення ресурсів бази даних;

- Обробка виключень – переклад винятків при доступі до даних у виключення Spring;
- Транзакційність – прозорі транзакції в операціях з даними;
- Розпакування ресурсів – отримання об'єктів бази даних з пулу з'єднань;
- Абстракція для обробки BLOB та CLOB.

Фреймворк Model-view-controller

Spring MVC – фреймворк орієнтований на запити і надає деякі можливості для розробника:

- Ясний та прозорий поділ між шарами в MVC і запитами;
- Стратегія інтерфейсів – кожен інтерфейс робить тільки свою частину роботи;
- Інтерфейс завжди може бути замінений альтернативною реалізацією;
- Інтерфейси тісно пов’язані з Servlet API;
- Високий рівень абстракції для веб-додатку.

Фреймворк аутентифікації і авторизації- Spring Security

Spring Security – це Java / Java EE фреймворк, що надає механізми побудови систем аутентифікації та авторизації, а також інші можливості забезпечення безпеки для промислових програм, створених за допомогою Spring Framework.

JPA

Для спрощення роботи з базою даних і даними з неї, було прийнято рішення використовувати JPA. Java Persistence API – API, що входить з версії Java 5 до складу платформ Java SE і Java EE, надає можливість зберігати в зручному вигляді Java-об'єкти в базі даних.

Підтримка збереження даних, що надається JPA, покриває області:

- безпосередньо API, заданий в пакеті javax.persistence;
- платформо-незалежна об'єктно-орієнтована мова запитів Java Persistence Query Language;
- метаінформація, що описує зв'язки між об'єктами;
- генерація DDL для сутностей.

Ми будемо використовувати чистий JPA, тому що це-реалізація стандарту, яка дозволить нам не бути залежними від конкретної бази даних (будь то Oracle DB, MySQL, Postgre SQL) і вільно переходити з однієї бази на іншу в разі потреби.

LiquiBase

LiquiBase – незалежна від бази даних бібліотека, з відкритим вихідним кодом для управління версіями та змінами схем баз даних. Вона допоможе організувати додаткові зміни в базі даних з різними наборами змін і застосувати їх для бази даних.

LiquiBase не єдиний засіб міграції база даних. Є багато рішень, такі як Artisan і Laravel. Але LiquiBase вирішує багато проблем, які не роблять інструменти міграції бази даних, такі як підтримка кількома розробниками різних систем управління базами даних. Крім того, серйозним недоліком в більшість інструментів є те, що вони не знають що змінити. Замість того щоб зосередитися на зміні, вони раз у раз порівнюють два знімка схеми бази даних для створення сценарію міграції. Так, наприклад, перейменування стовпців розглядається як операція drop + add, яка може привести до втрати даних. LiquiBase це зміна розуміє.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ

Проект повинен бути розбитий на наступні підпроекти:

- підпроект domain (підпроект з класами-об'єктами, що описують сутності в базі даних і взаємозв'язку між ними)
- підпроект dao (data-access-object – підпроект, який реалізує роботу з базою – доступ до таблиць, збереження, отримання, зміна даних)
- підпроект service (підпроект різних сервісів)
- підпроект view (підпроект, що зберігає в собі контролери сторінок, сторінки, а також javascript, css код та зображення для сторінок)
- підпроект db-scripts (підпроект з liquibase скриптами для автоматичної генерації енобхідної інформації в базі даних для початку робот із проектом)

Підпроекти та їх зв’язки зображені на рис. 3.1.

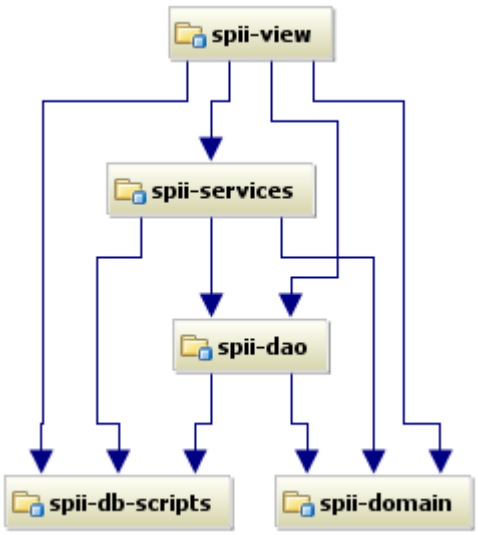


Рисунок. 3.1 – Підпроекти та їх зв’язки

3.1 Підпроект domain

Даний підпроект містить класи, які описують сутності з бази даних (діаграма таблиць та їх взаємозв'язків представлена на рис. 4.2.), а так само деякі інші об'єкти, примірники яких не зберігаються в базі, а створюються лише під час виконання специфічних операцій, наприклад інформація про завантаження і за статусом завантаження при завантаженні звітів в базу.

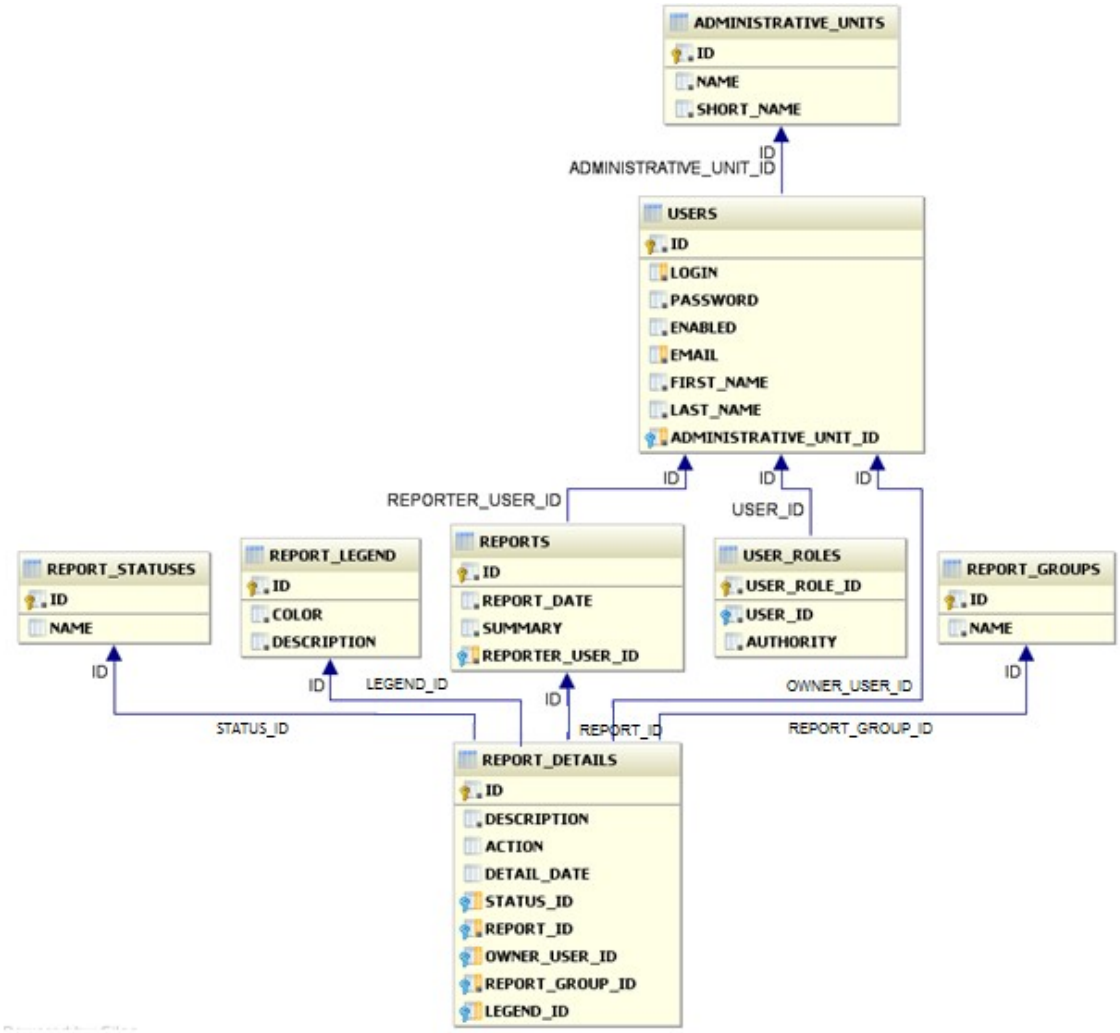


Рисунок 3.2 – Таблиці в базі та взаємозв’язки між ними

Підпроект містить один інтерфейс-DomainObject, що повинен успадковувати інтерфейс java.io.Serializable. Інтерфейс не повинен містити жодних методів. Всі класи, що описують сутності з бази повинні імплементувати даний інтерфейс.

У підпроекту реалізовані наступні класи:

- адміністративний юніт (AdministrativeUnit)
- звіт (Report)
- деталі звіту (ReportDetail)
- групи звітів(ReportGroup)
- легенда звіту(ReportLegend)
- статус звіту(ReportStatus)
- користувач (User)

- ролі користувачів (UserRole)
- статус (Status)
- завантаження (Upload)

Всі класи, їх взаємозв'язки в підпроекті представлені на рис. 3.3.

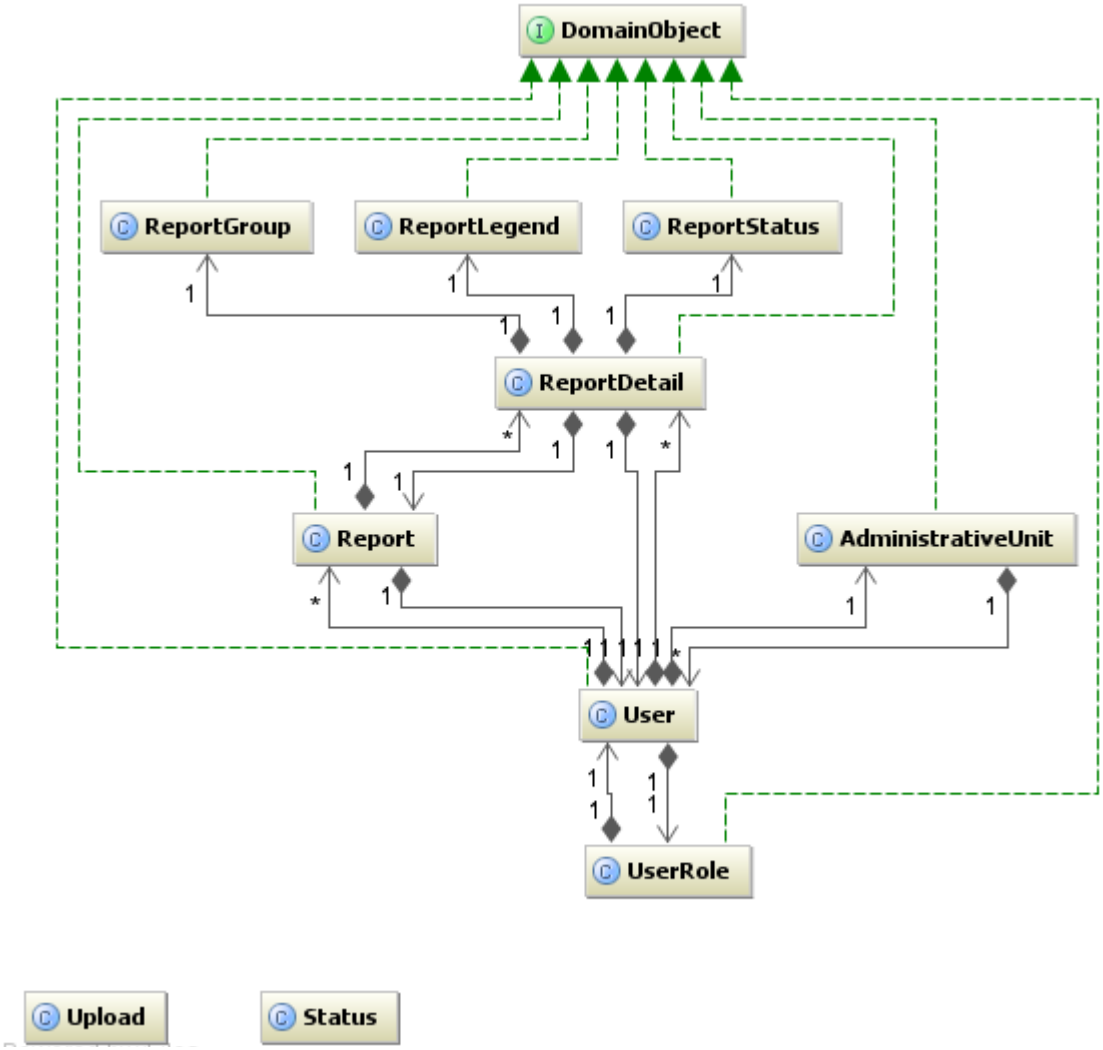


Рисунок 3.3 – Класи і взаємозв'язки класів у підпроекту domain

Класи, що описують сутності в базі являють собою сутності (Entity). Розглянемо створення цих класів на прикладі класу Report.

3.1.1 Класс Report

Лістинг коду классу Report представлено в Додатку А.

Класс має наступні анотації:

- @Entity – ознака того, що даний клас являє собою сутність, яку треба зберігати в базі даних
- @Table – вказує на таблицю, яку необхідно використовувати для зберігання в базі
- В класі є наступні поля:
- private Long id – ідентифікатор запису
- private Date date – дата створення звіту
- private String summary – інформація звіту
- private User reporter – поле-посилання на творця звіту
- private List<ReportDetail> reportDetails – список деталей звіту

3.1.1.1 Поля класу Report та їх опис

Поле id

Опис поля:

```
@Id
@GeneratedValue(strategy = GenerationType.SEQUENCE)
@Column(name = "ID", unique = true, nullable = false)
private Long id;
```

Анотації поля:

- @Id – говорить про те, що дане поле є ідентифікатором. Анотація @ Id є обов'язковою, тобто як мінімум одне поле класу повинно бути нею позначено.
- @GeneratedValue – анотація дає вказівку, що дана величина буде генеруватися, причому вказівку strategy = GenerationType.SEQUENCE говорить про те, що значення буде створено за допомогою послідовності в базі.
- @Column – анотація для вказівки імені стовпця, з яким пов'язано поле класу. Параметр "unique = true" вказує, що поле повинно мати унікальне значення, а параметр "nullable = false" не дозволить зберегти дані, якщо дане поле не заповнено.

Поле date

Опис поля:

```
@Temporal(TemporalType.DATE)
@Column(name="REPORT_DATE",nullable = false)
private Date date;
```

Анотації поля:

- @Temporal(TemporalType.DATE) – говорить про те, що поле буде використовуватися для зберігання дати
- @Column – анотація для вказівки імені стовпця, з яким пов'язано поле класу. Параметр "nullable = false" не дозволить зберегти дані, якщо дане поле не заповнено.

Поле summary

Опис поля:

```
@Column(name = "SUMMARY",nullable = false)
private String summary;
```

Анотації поля:

- @Column – анотація для вказівки імені стовпця, з яким пов'язано поле класу. Параметр "nullable = false" не дозволить зберегти дані, якщо дане поле не заповнено.

Поле reporter

Опис поля:

```
@ManyToOne
@JoinColumn(name = "reporter_user_id", nullable = false)
private User reporter;
```

Анотації поля:

- @ManyToOne – анотація вказує, що поле вказує на інший клас, який пов'язаний з поточним класом зв'язком багато-до-одного.
- @JoinColumn – дана анотація, по суті, схожа на @ Column. Відмінністю є те, що вона вказує, що колонка до того ж є сполучною. Параметр "nullable = false" не дозволить зберегти дані, якщо дане поле не заповнено.

Поле reportDetails

Опис поля:

```
@OneToMany(mappedBy = "report", cascade = CascadeType.ALL, fetch = FetchType.EAGER)
private List<ReportDetail> reportDetails;
```

Анотації поля:

- @OneToMany – запис говорить про те, що поле служить для зв'язку один-до-багатьох. Параметр "mappedBy =" report "" вказує на поле в класі ReportDetail, яке відповідає за посилання на клас Report. Параметр "cascade = CascadeType.ALL" – вказує правило каскаду. У даному випадку це означає, що всі дії, які відбуватимуться з сутністю, що знаходиться в екземплярі класу Report повинні бути

зроблені і з усіма сутностями деталей звіту, які прив'язані до даної конкретної сутності звіту. Параметр "fetch = FetchType.EAGER" означає, що при завантаженні звіту, система відправить запит не тільки на завантаження самої сутності Report, але і на завантаження всієї деталей звіту, які прив'язані до даної суті.

3.1.1.2. Конструктори класу Report

В класі Report представлено 2 конструктори:

- стандартний порожній конструктор
- конструктор з аргументами

У зв'язку з тим, що ми хочемо використовувати конструктор з аргументами, для коректної роботи ми зобов'язані привести порожній конструктор в описі сутності. Він має вигляд:

```
public Report(){}
```

Конструктор з параметрами приймає в себе дату створення звіту, користувача-творця звіту, опис (summary) звіту, а так само список деталей звіту.

```
public Report(Date date, User reporter, String summary,
 List<ReportDetail>  reportDetails) {
        this.date = date;
        this.reporter = reporter;
        this.summary = summary;
        this.reportDetails = reportDetails;
    }
```

3.1.1.3. Інші методи класу Report

Так як одним з базових принципів ООП є інкапсуляція полів і методів класу, всі поля класу Report мають модифікатор доступу "private". Це означає, що для роботи з полями нам необхідно створити так звані сеттери і геттери.

Клас Report з усіма полями, конструкторами і методами представлений на рис. 3.4.

Всі сучасні середовища розробки підтримують генерацію цих методів.

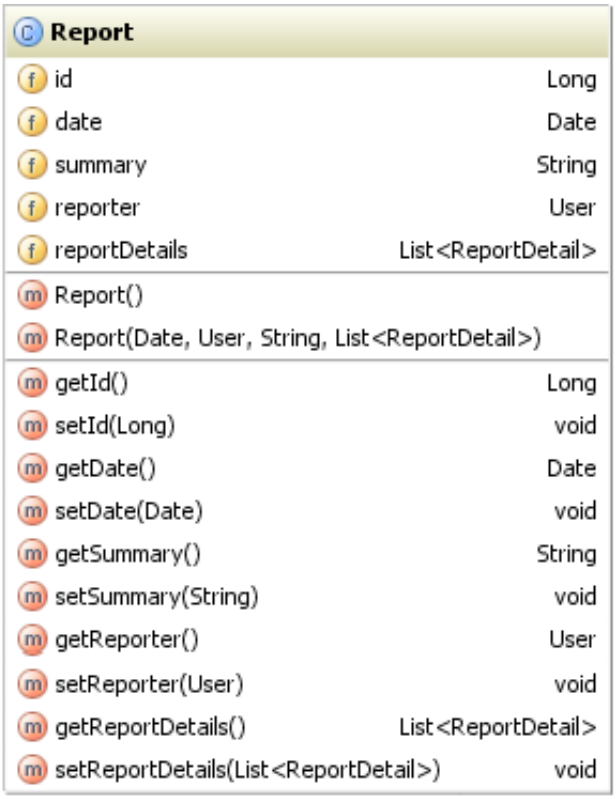


Рисунок 3.4 – Поля, конструктори та методи класу Report

Ми так само скористаємося автоматичною генерацією коду. Отримуємо наступний набір методів:

```
public Long getId() {
    return this.id;}
public void setId(Long id) {
    this.id = id;}
public Date getDate() {
    return this.date;}
public void setDate(Date date) {
    this.date = date;}
public String getSummary() {
    return this.summary;}
public void setSummary(String summary) {
    this.summary = summary;}
public User getReporter() {
    return this.reporter;}
public void setReporter(User reporter) {
    this.reporter = reporter;}
public List<ReportDetail> getReportDetails() {
    return this.reportDetails;}
public void setReportDetails(List<ReportDetail> reportDetails) {
    this.reportDetails = reportDetails;}
```

Не будемо детально розглядати інші класи. Нижче наведені діаграми, що ілюструють поля, методи, конструктори класів, а також взаємозв'язки між класами (рис. 3.5, рис. 3.6, рис. 3.7).

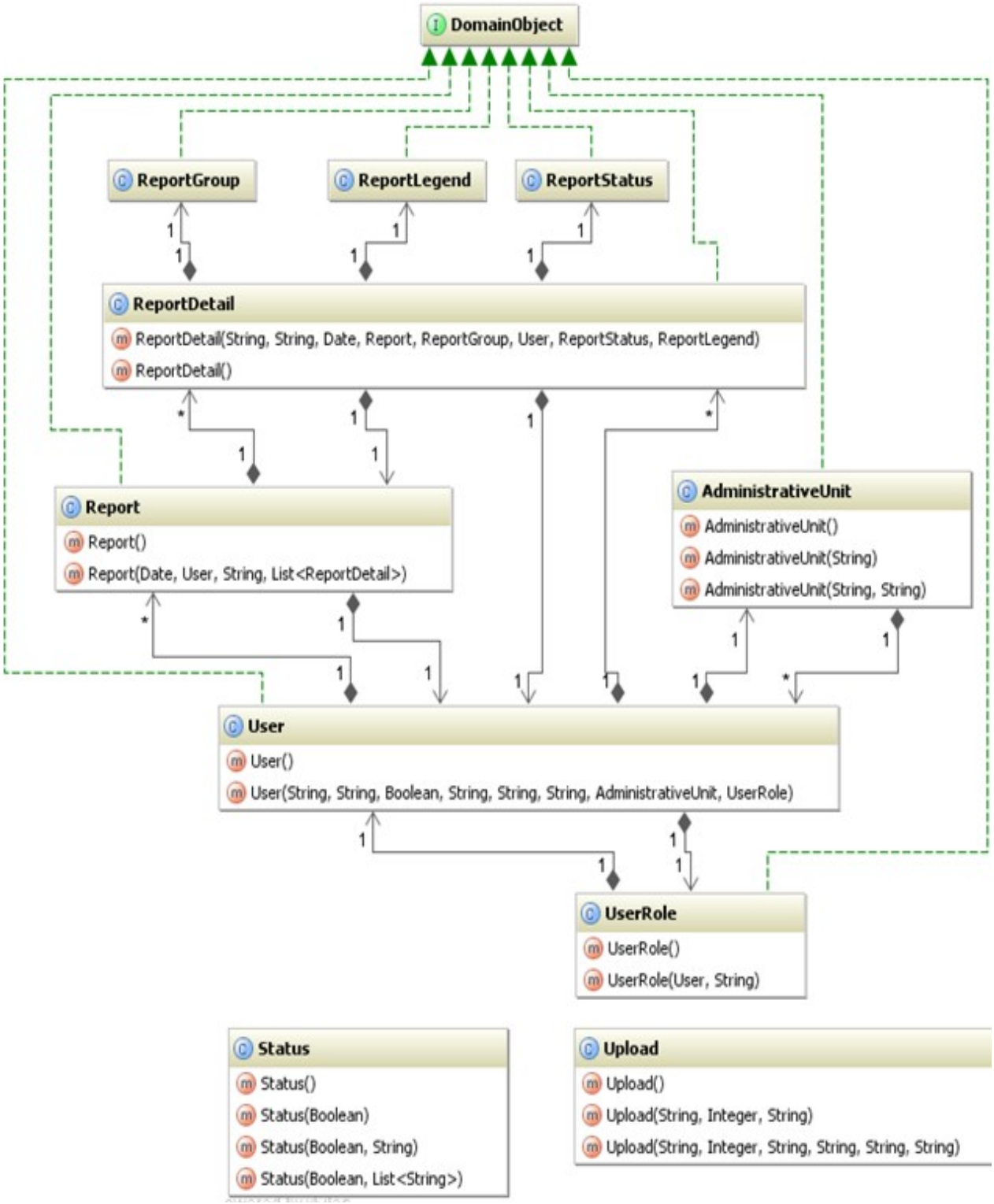


Рисунок 3.5 – Конструктори класів та їх взаємозв’язок в підпроекті domain

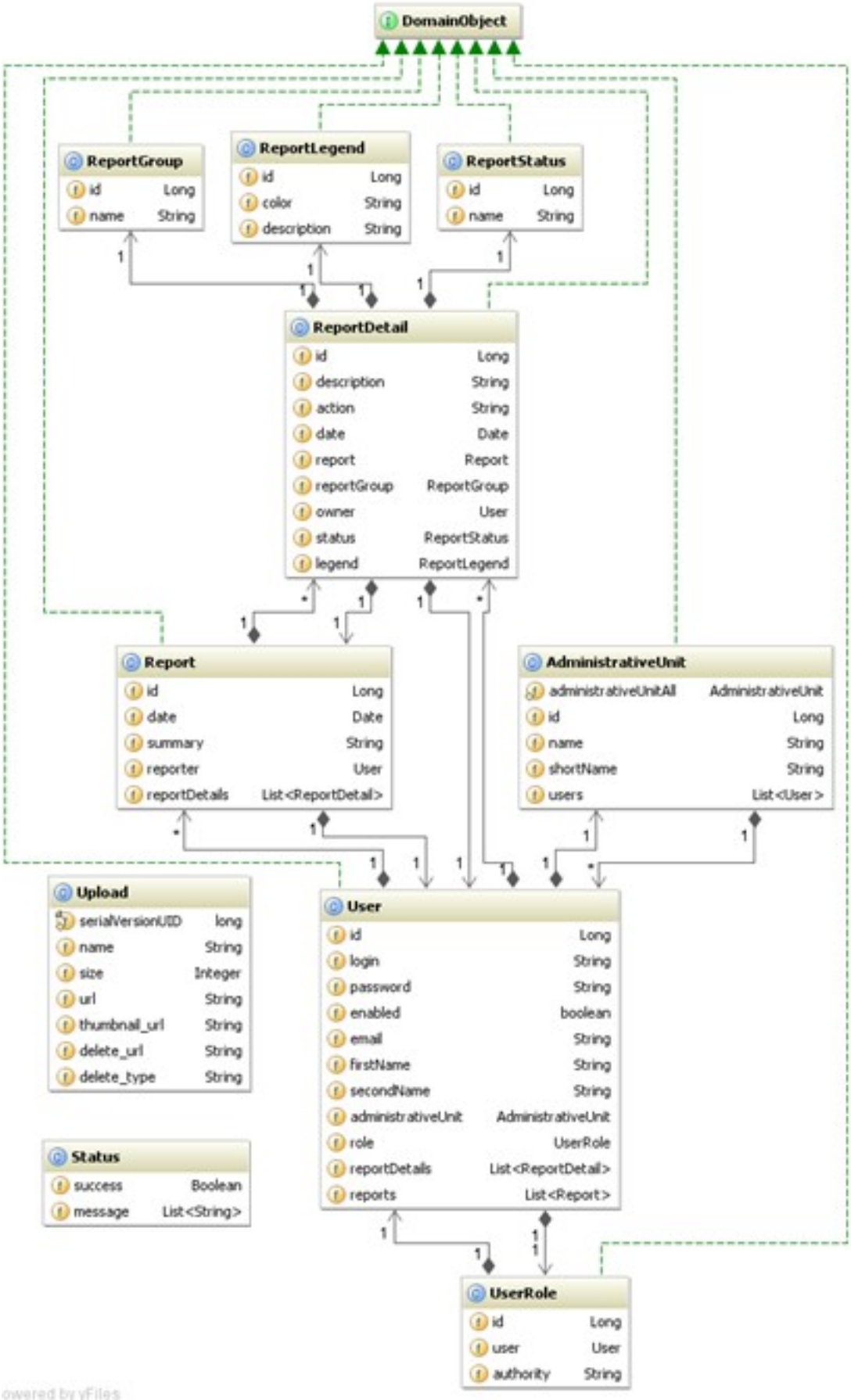


Рисунок. 3.6 – Поля класів та їх взаємозв’язок в підпроекті domain

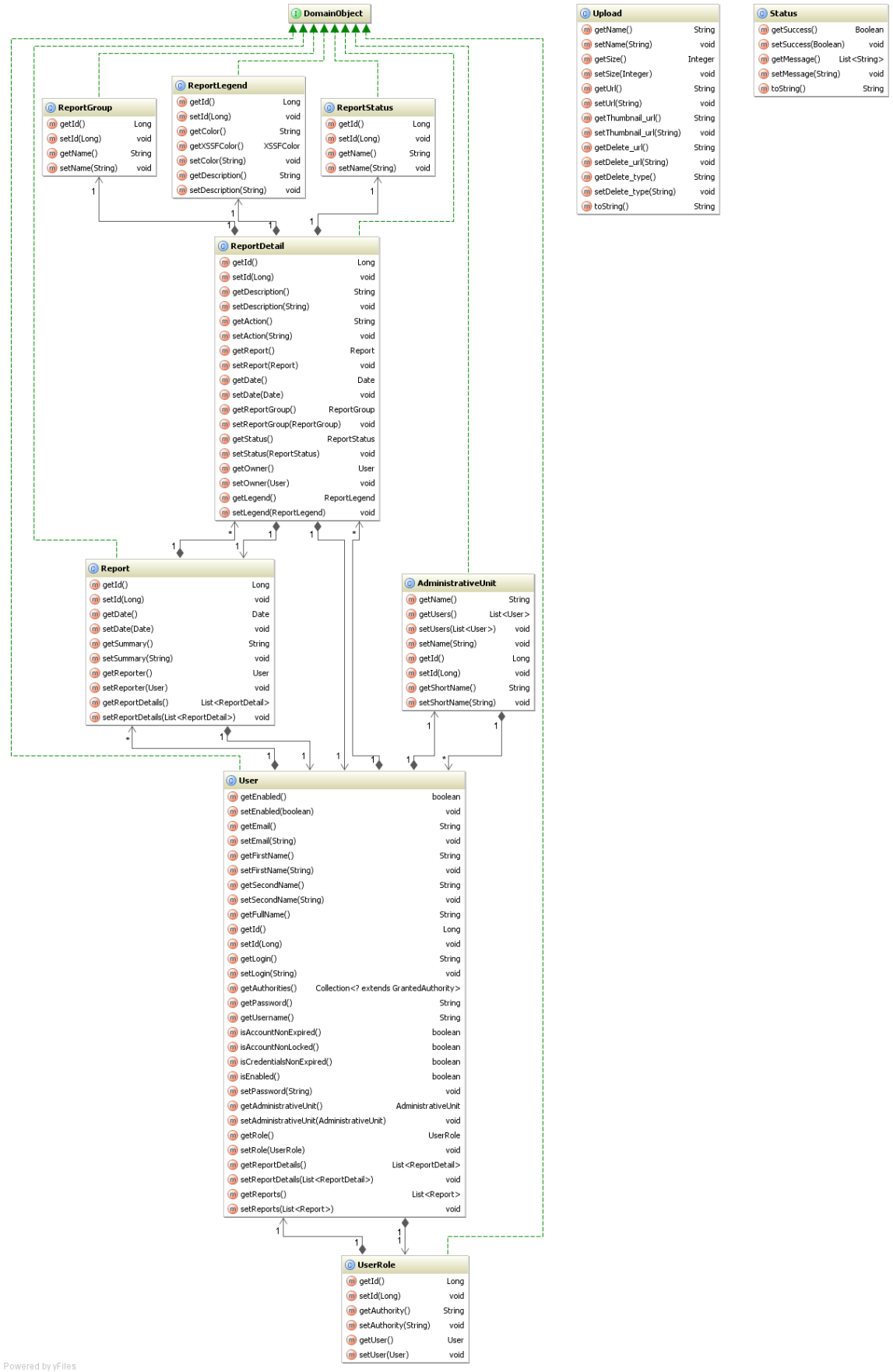


Рис. 3.7. Методи класів та їх взаємозв’язок в підпроекті domain

3.2 Підпроект dao

Даний підпроект реалізує роботу з базою даних, відповідає за доступ до таблиць, збереження, отримання, зміну даних в них. Всі dao-класи повинні мати подібний базовий набір методів:

- отримати запис з бази по id
- отримати всі записи з таблиці
- зберегти запис
- видалити запис
- оновити дані записи
- виконати запит

Тому був реалізований базовий клас, який надає всі ці методи. Інші класи успадковують цей клас. Поверх класів реалізований шар інтерфейсів через які ми і будемо звертатися до класів (безпосередньої реалізації інтерфейсів).

Мають бути реалізовані наступні класи:

- GenericDaoJpa
- AdministrativeUnitDaoJpa
- ReportDaoJpa
- ReportDetailsDaoJpa
- ReportGroupDaoJpa
- ReportLegendDaoJpa
- ReportStatusDaoJpa
- UserDaoJpa
- UserRoleDaoJpa

Набір інтерфейсів:

- GenericDao
- AdministrativeUnitDao
- ReportDao
- ReportDetailsDao
- ReportGroupDao
- ReportLegendDao
- ReportStatusDao
- UserDao
- UserRoleDao

В Додатку Б приведена діаграма класів і інтерфейсів, а також їх зв'язок.

Розглянемо реалізацію класів і інтерфейсів на прикладі GenericDao, AdministrativeUnitDao.

3.2.1 Інтерфейс GenericDao

Лістинг інтерфейсу GenericDao:

```
package com. customname.spii.dao;

import com. customname.spii.entity.DomainObject;
import java.util.List;

public interface GenericDao<T extends DomainObject>  {
    T getById(Long id);
    List<T> getAll();
    void save(T object);
    T update(T object);
    void delete(T object);


    List<T> executeQuery(String query);
}
```

У даному інтерфейсі описаний набір методів, які ми повинні реалізувати в класі GenericDaoJpa. Методи:

- T getById(Long id) – метод отримання запису в таблиці по id
- List<T> getAll() – метод отримання всіх даних з таблиці
- void save(T object) – метод збереження запису
- T update(T object) – метод оновлення запису
- void delete(T object) – метод видалення запису
- List<T> executeQuery(String query) – метод виконання заздалегідь-сформованого запиту

3.2.2 Інтерфейс AdministrativeUnitDao

Лістинг інтерфейсу AdministrativeUnitDao:

```
package com. customname.spii.dao;
import com.customname.spii.entity.AdministrativeUnit;
import java.util.List;
public interface AdministrativeUnitDao extends
GenericDao<AdministrativeUnit>{
    public List<AdministrativeUnit>
  getAdministrativeUnitsByName(String longName, String shortName);
}
```

Даний інтерфейс успадковує всі методи інтерфейсу GenericDao, а також реалізує один свій.

Метод public List<AdministrativeUnit> getAdministrativeUnitsByName(String longName, String shortName) дозволяє отримати запис про адміністративний юніт по його імені.

3.2.3 Клас GenericDaoJpa

Лістинг класу GenericDaoJpa представлено в Додатку В.

Клас і його методи працюють з generic, який успадковує DomainObject, який був створений в підпроекту domain. Це дозволяє нам використовувати розроблені нами стандартні методи з усіма класами з підпроекту domain. Таким чином ми отримуємо універсальний набір базових dao-методів.

3.2.3.1 Анотації класу GenericDaoJpa

Над класом розташовані анотації:

- @SuppressWarnings("unchecked") – використовується для «придушення» попереджень, які створюються компілятором
- @Transactional – говорить про те, що методи являють собою транзакції, тобто логічну одиницю роботи з даними. Транзакція може бути виконана або цілком і успішно, дотримуючись цілісності даних і незалежно від інших транзакцій, що йдуть параллельно, або не виконана взагалі і тоді вона не повинна мати ніякого ефекту.

3.2.3.2 Поля класу GenericDaoJpa

Клас має два поля:

- private Class<T> type – поле типу класу, що зберігає клас об'єкта з яким ми будемо працювати надалі
- protected EntityManager entityManager – менеджер сутностей. Стандартний менеджер, наданий JPA. Має свій набір методів, який ми будемо використовувати при реалізації методів у класі.

3.2.3.3 Методи класу GenericDaoJpa

В класі GenericDaoJpa представлені базові методи для роботи з таблицями в базі.

Метод getById

Лістинг методу:

```
        public T getById(Long id) {
        return (T) entityManager.find(type, Long.parseLong("" +
id));
        }
```

Метод getById дозволяє через стандартний менеджер сутностей отримати запис з таблиці за ідентифікатором.

Метод getAll

Лістинг методу:

```
public List<T> getAll() {
          return entityManager.createQuery(
                              "select obj from " + type.getName() +
        " obj").getResultList();
    }
```

У методі ми через entityManager створюємо запит, використовуючи createQuery, який вибере з таблиці всі записи. Методом getResultList ми виконаємо запит і отримаємо результати, які будуть повернуті у вигляді списку.

Метод save

Лістинг методу:

```
public void save(T object) {
entityManager.persist(object);
}
```

Метод зберігає новий запис у базу, використовуючи метод менеджера сутностей persist.

Метод update

Лістинг методу:

```
   public T update(T object) {
return entityManager.merge(object);
   }
```

Як параметр в метод передається об'єкт-екземпляр класу-опису запису з бази. Метод зберігає зміни до запису в базу, використовуючи метод менеджера сутностей merge.

Метод delete

Лістинг методу:

```
   public void delete(T object) {
entityManager.remove(entityManager.merge(object));
   }
```

Як параметр в метод передається об'єкт-екземпляр класу-опису запису з бази. Метод видаляє запис з бази, використовуючи метод менеджера сутностей remove.

Метод executeQuery

Лістинг методу:

```
public List<T> executeQuery(String query) {
return entityManager.createQuery(query).getResultList();
}
```

Як параметр в метод передається рядок-запит. Метод виконує запит, використовуючи метод менеджера сутностей createQuery і виконує його, використовуючи getResultList.

Всі поля, методи і конструктори класу представлені на рис. 3.8.

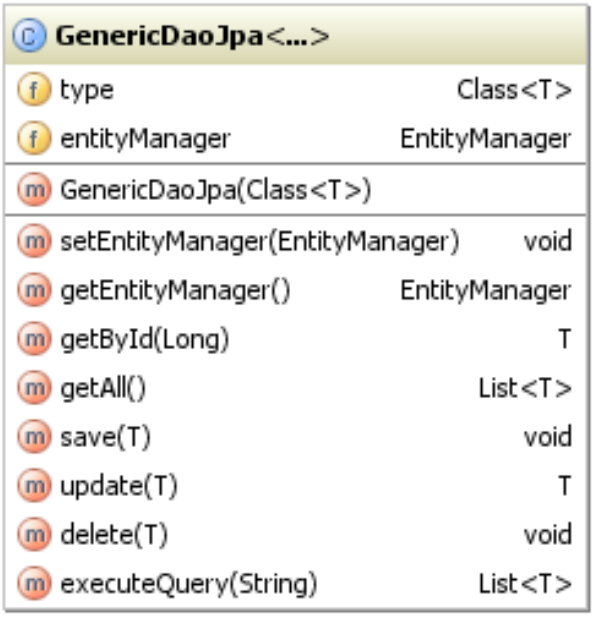


Рисунок. 3.8 – Поля, методи і конструктори класу GenericDaoJpa

3.2.4 Клас AdministrativeUnitDaoJpa

Лістинг класу AdministrativeUnitDaoJpa:

```
package com.customname.spii.dao.impl;

import com.customname.spii.dao.AdministrativeUnitDao;
import com. customname.spii.entity.AdministrativeUnit;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository("administrativeUnitDao")
public class AdministrativeUnitDaoJpa extends
GenericDaoJpa<AdministrativeUnit> implements AdministrativeUnitDao{

    public AdministrativeUnitDaoJpa(){
        super(AdministrativeUnit.class);}
```

```
    public List<AdministrativeUnit> getAdministrativeUnitsByName(String
longName, String shortName) {
        String queryString = "SELECT obj FROM AdministrativeUnit obj WHERE
NAME='" + longName +"' OR short_name = '" + shortName + "'";
        List<AdministrativeUnit> admUnit = this.executeQuery(queryString);
        if (admUnit != null && admUnit.size() > 0) {
            return admUnit;
        }
        return null;
    };
}
```

Методи, конструктори класу і його зв'язок з батьками зображено на рис. 3.9.

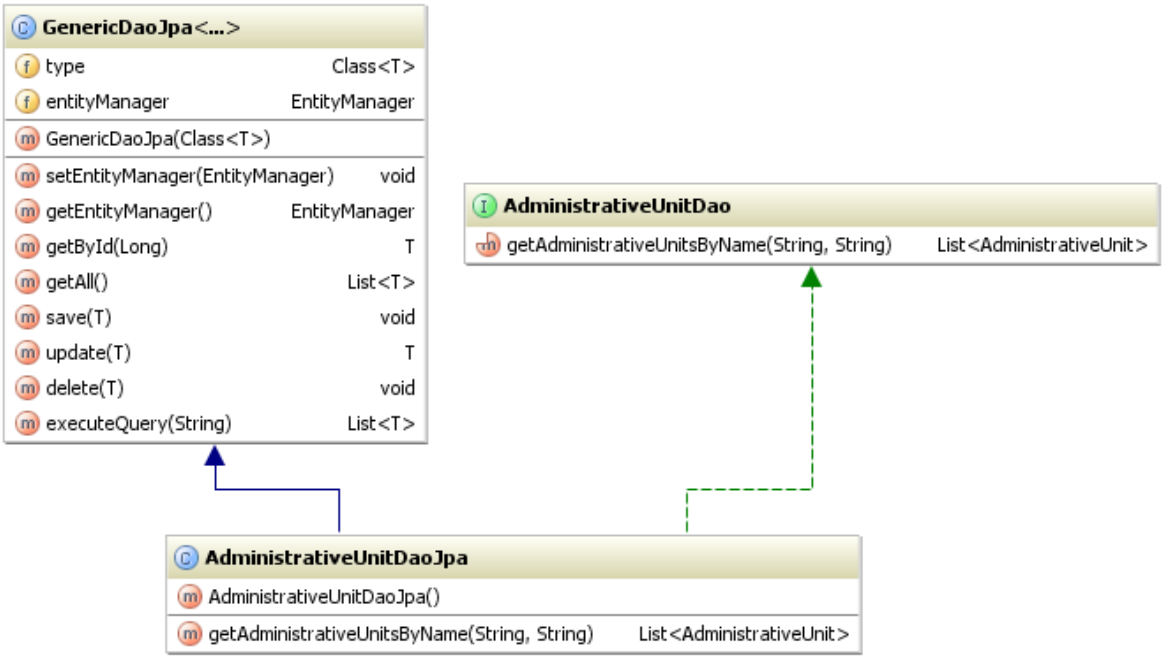


Рисунок 3.9 – Методи, конструктори класу AdministrativeUnitDaoJpa та його зв’язок з батьками

Клас успадковує GenericDaoJpa і реалізує інтерфейс AdministrativeUnitDao. В інтерфейсі визначено метод getAdministrativeUnitsByName. Метод дозволяє отримати запис про адміністративний юніт по повному або короткому імені. У разі, якщо жодного запису не знайдено, то повернеться null. Якщо є збіги, то повернеться список записів. Як правило, метод не повинен повертати список в якому більше одного запису. Однак, якщо вказати повне ім'я одного бранча та скорочене іншого, то метод поверне список з двох елементів (невеликий хак).

3.2.5. Інші класи підпроекту dao

Не будемо детально розглядати решту класів підпроекту dao. Нижче наведені діаграми класів з їх методами, конструкторами і зв'язками(рис 3.10, рис. 3.11, рис. 3.12, рис. 3.13, рис. 3.14, рис. 3.15).

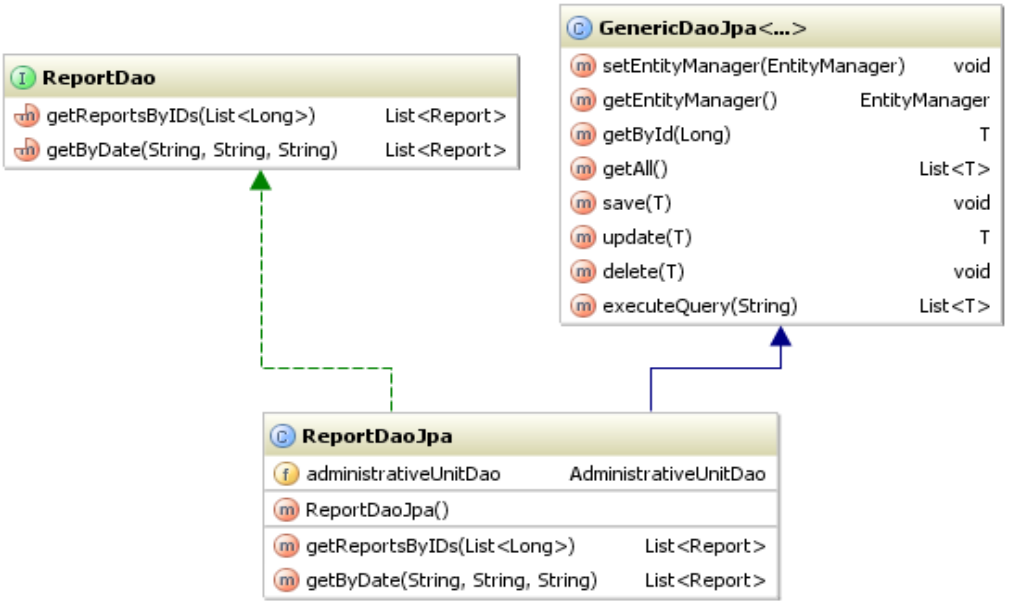


Рисунок 3.10 – Методи, конструктори класу ReportDaoJpa і його зв'язок з батьками

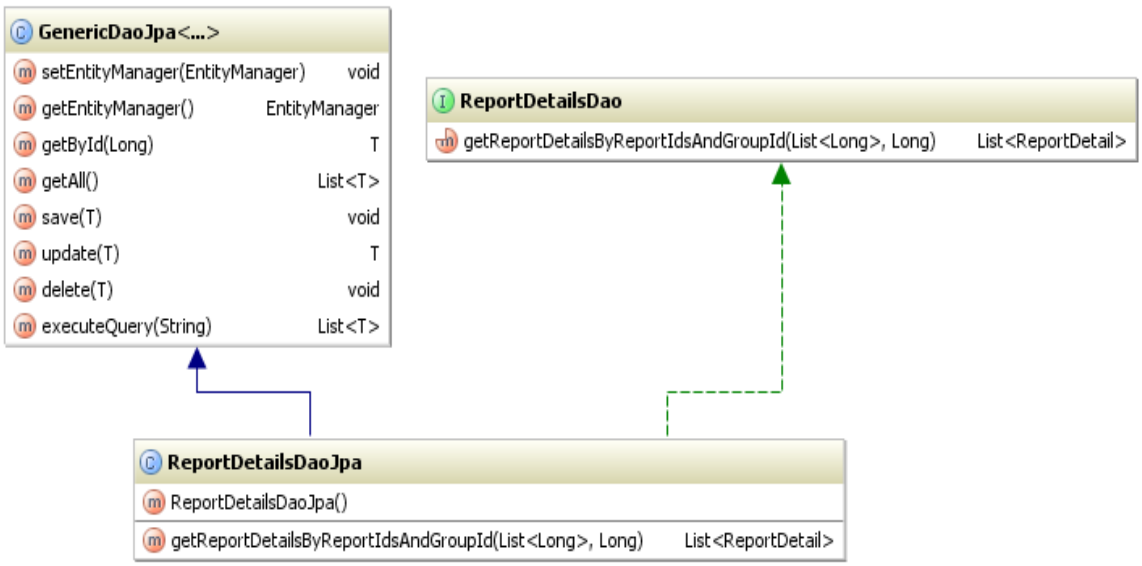


Рисунок 3.11 – Методи, конструктори класу ReportDetailsDaoJpa і його зв'язок з батьками

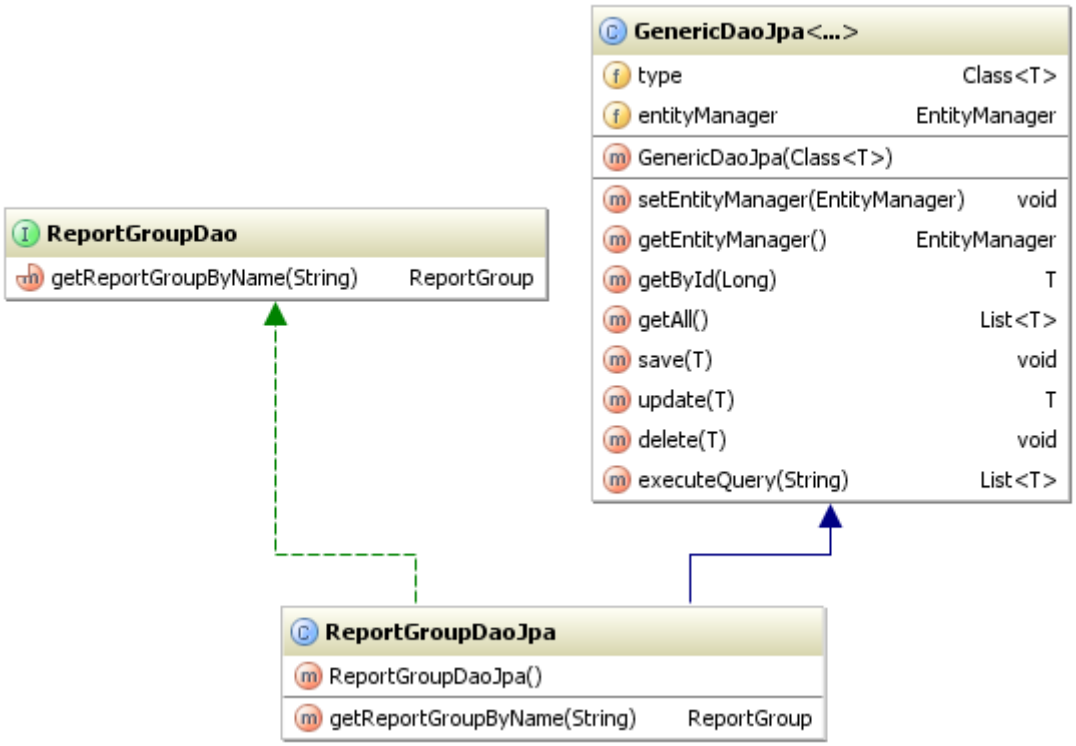


Рисунок 3.12 – Методи, конструктори класу ReportGroupDaoJpa і його зв'язок з батьками

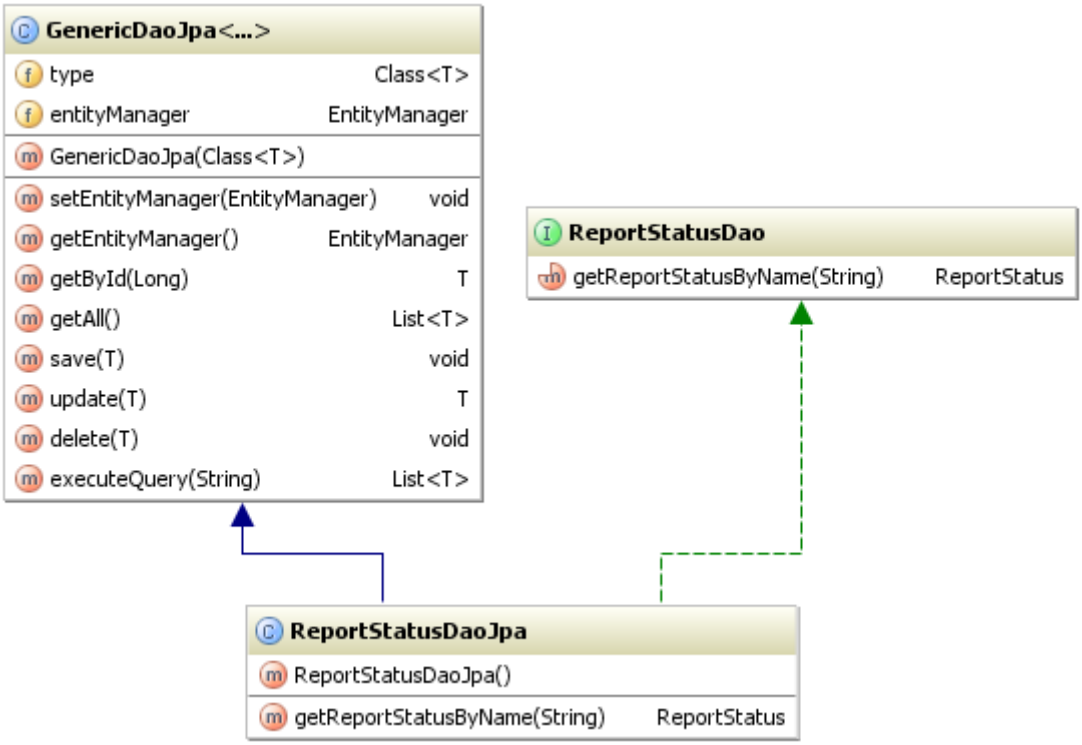


Рисунок 3.13 – Методи, конструктори класу ReportStatusDaoJpa і його зв'язок з батьками

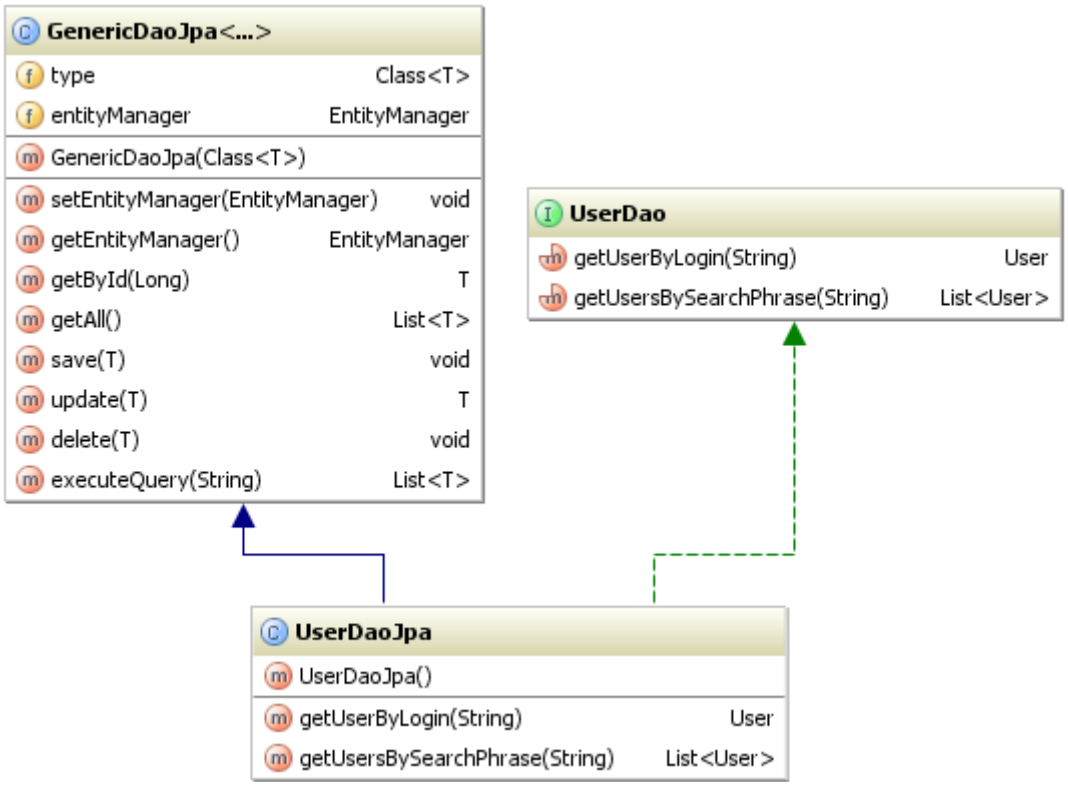


Рисунок 3.14 – Методи, конструктори класу UserDaoJpa і його зв'язок з батьками

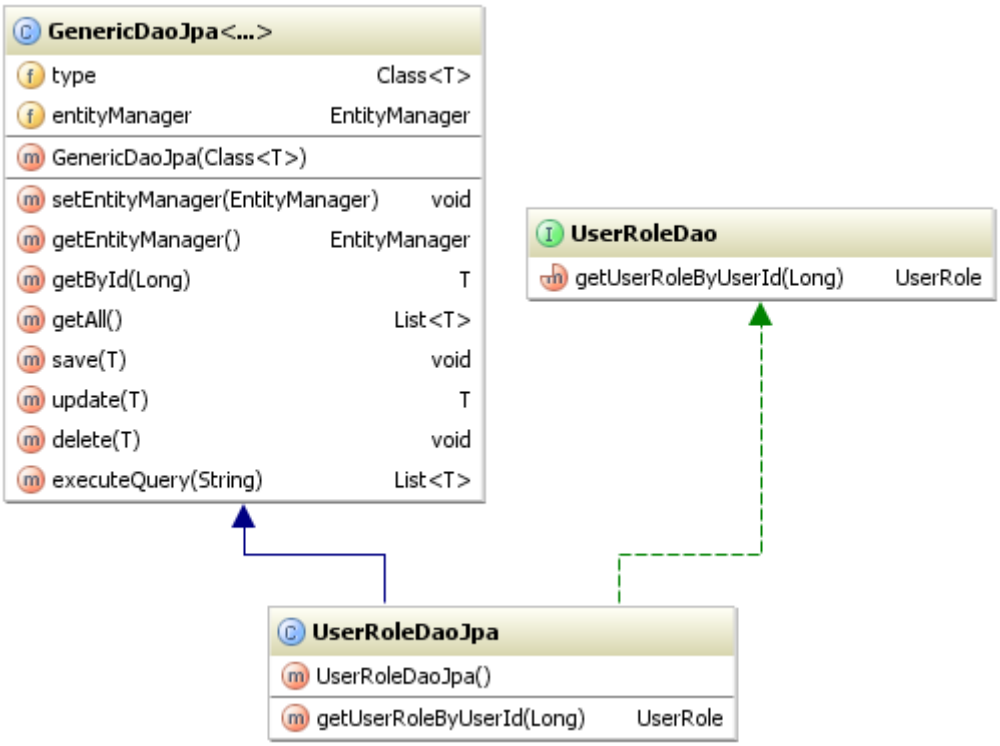


Рисунок 3.15 – Методи, конструктори класу UserRoleDaoJpa і його зв'язок з батьками

3.3 Підпроект service

Підпроект service є прошароком між контролерами сторінок і dao нашого проекту. Сервіси, що надаються в підпроекті, виконують більшу частину роботи з даними-їх обробку, перетворення, перевірку і так далі.

Розглянемо реалізацію сервісів і утиліт на прикладі сервісу ReportService, який використовує сервіс ReportPageDataProcessor і необхідного для нього класу-утиліти StringToJsonConverter. Доступ до сервісів надається через набір інтерфейсів. Повний список набору інтерфейсів разом з наданими методами представлений на рис. 3.16.

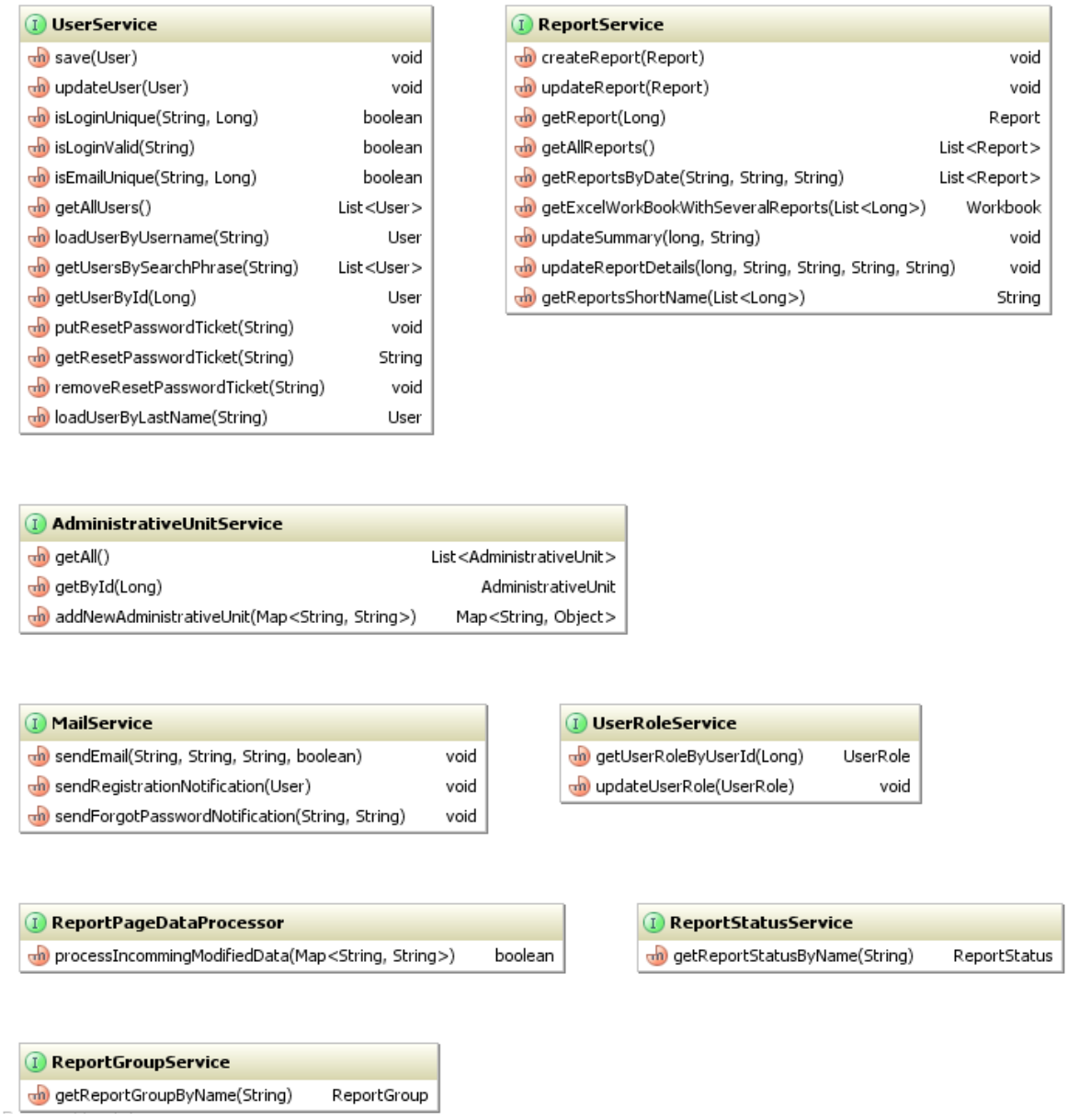


Рисунок 3.16 Інтерфейси і їхні методи підпроекту service

До складу підпроекту сервісів входять безпосередньо самі сервіси та набір класів-утиліт.

Класи-утиліти необхідні, щоб розвантажити сервіси від надлишкового для кожного окремого сервісу функціоналу та надання зручного доступу до необхідних для всіх сервісів функцій (наприклад, конвертації рядка в JSON-об'єкт). Класи-утиліти винесені в окремий пакет Utils. Їх реалізація, так само як і реалізація сервісів, розділена на інтерфейси та безпосередню реалізацію класів-утиліт.

3.3.1 Інтерфейс StringToJsonConverter

Лістинг інтерфейсу:

```
package com.customname.spii.services.Util;
import org.json.simple.JSONObject;
import java.util.List;
public interface StringToJsonConverter {
    public List<JSONObject> convertStringToJson(String data);
}
```

Варто відзначити, що для реалізації інтерфейсу і класу була використана бібліотека JSON.simple. JSON.simple – це проста Java бібліотека для роботи з JSON, читання і запису JSON даних, яка повністю відповідає JSON специфікації (RFC4627).

Інтерфейс надає всього один метод convertStringToJson. Метод приймає в себе серіалізований JSON-об'єкт і повертає список JSON-об'єктів з полів яких вже можна діставати інформацію.

3.3.2 Клас StringToJsonConverterImpl

3.3.2.1 Анотації класу

Анотація класу @Component необхідна, для того, щоб Spring зміг зробити ін'єкцію. Вона дозволяю автоматично зареєструвати компонент у контексті Spring для подальшої ін'єкції.

3.3.2.2 Методи класу

У класі, як і в інтерфейсі, представлений всього один метод – convertStringToJson. У методі створюється ArrayList об'єктів JSONObject. У разі, якщо в отриманому рядку є всього один набір даних (тобто немає

підрядка "}, {"), то ми відразу намагаємося конвертувати його в JSONObject і повертаємо результат з методу. В іншому випадку, ми видаляємо першу і останню фігурні дужки з рядка ("{", "}") і розбиваємо наш рядок з даними на окремі підрядка-об'єкти, які зберігають дані про один запис. Потім у циклі ми конвертуємо дані з кожного підрядка в JSONObject і повертаємо список цих об'єктів.

3.3.3 Інтерфейс ReportPageDataProcessor

Лістинг інтерфейсу:

```
package com. customname.spii.services;
import java.util.Map;
public interface ReportPageDataProcessor {
    public boolean processIncommingModifiedData(Map<String,
String> modifiedData);
}
```

Інтерфейс надає всього один метод – processIncommingModifiedData.

Даний метод приймає об'єкт типу Map, в якому містяться дані про змінений summary і details звіту. Ключ – це summary або details, вміст по ключу - це рядок – серіалізований JSON-об'єкт із зміненими даними. Метод повертає true, у разі якщо в об'єкті modifiedData є якісь дані для зміни summary або details звіту. В іншому випадку метод повертає false.

3.3.4 Сервіс ReportPageDataProcessorImpl

3.3.4.1 Анотації класу

Анотація класу @Component необхідна, для того, щоб Spring зміг зробити ін'єкцію. Вона дозволяю автоматично зареєструвати компонент у контексті Spring для подальшої ін'єкції.

3.3.4.2 Поля класу

У класі представлено два поля:

- ReportService reportService – посилання на сервіс звітів
- StringToJsonConverter stringToJson – посилання на конвертер серіалізованого рядка в JSONObject

Обидва поля позначені анотацією @ Autowired. Властивості класу з анотацією @ Autowired заповнюються відповідними значеннями відразу

після створення bean'a і перед тим, як будь-який з методів класу буде викликаний.

3.3.4.3 Методи классу

Клас ReportPageDataProcessorImpl надає такі методи:

- public boolean processIncommingModifiedData(Map<String, String> modifiedData) – єдиний метод, доступний через інтерфейс «назовні». Є первинним методом обробки даних
- public boolean processSummaryData(Map<String, String> modifiedData) – метод для обробки та збереження зміненого Summary звіту
- public boolean processReportDetails(Map<String, String> modifiedData) – метод для обробки та збереження деталей звіту
- public String dataPreprocessing(Map<String, String> modifiedData, String key) – метод обробки значення, одержуваного за ключем

Метод dataPreprocessing

Дані на початку обробки представлені на рис. 3.17.

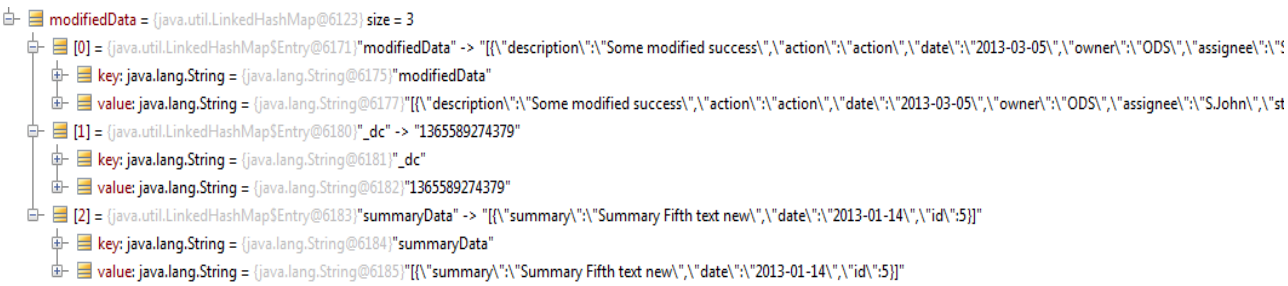


Рисунок 3.17 – Об'єкт, щозберігає дані зі змінами для звітів

Видно, що рядки з яких ми будемо отримувати дані, обгорнуті в квадратні дужки. Для обробки методом класу StringToJsonConverter ці дужки не потрібні. Метод dataPreprocessing призначений для отримання даних по ключу з об'єкта типу Map і видалення цих дужок. Після цього, дані повертаються для подальшої обробки.

Метод processSummaryData

На рис. 3.18. наведено алгоритм роботи методу. Метод призначений для обробки і збереження змін до Summary звіту.

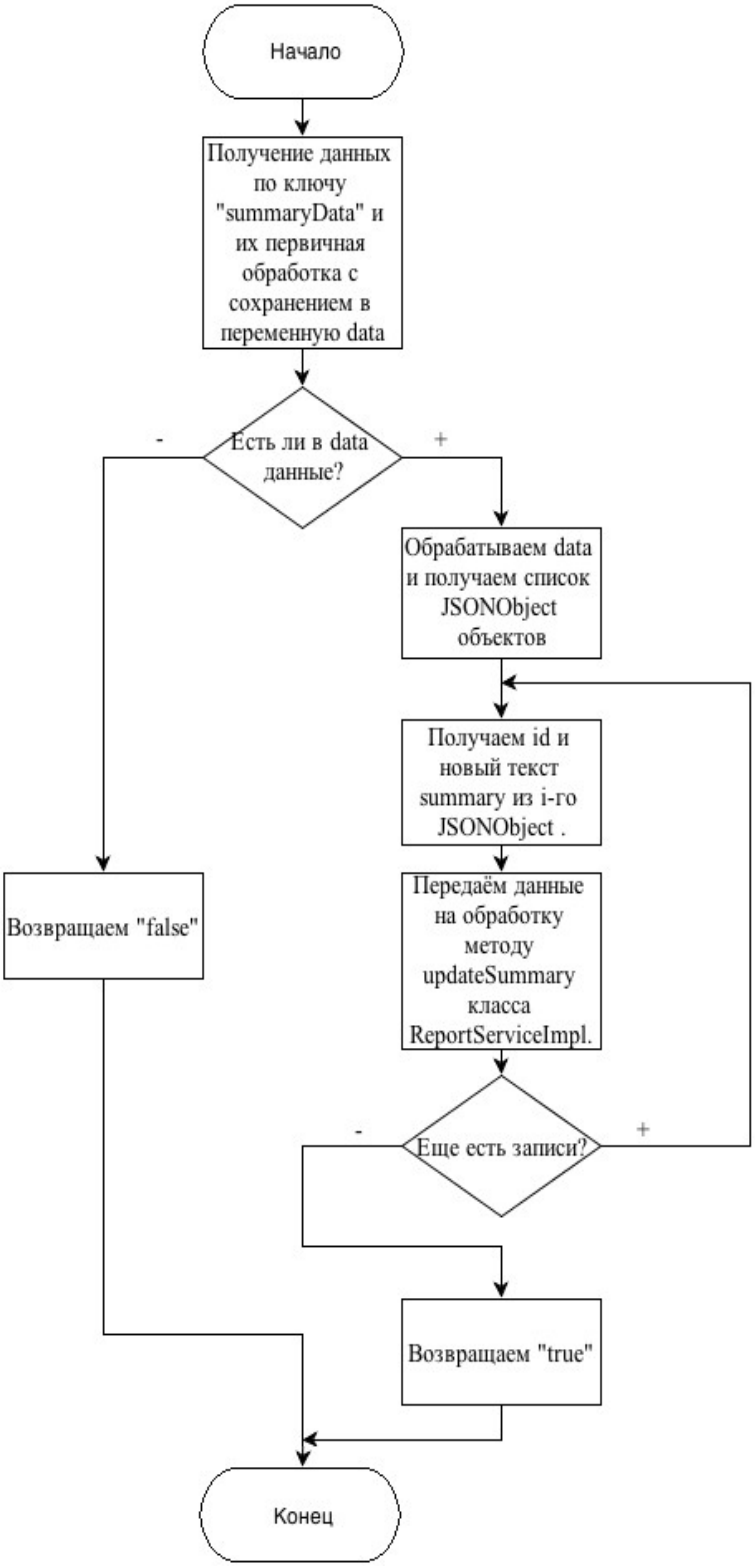


Рисунок 3.18 – Алгоритм роботи методу processSummaryData

Метод processReportDetails

Метод призначений для збереження деталей звіту. Метод відрізняється від методу processSummaryData наступним:

- метод отримує з вхідних даних поля:
    1) id
    2) description
    3) action
    4) status
    5) owner
- з reportService для подальшої обробки даних викликається метод updateReportDetails.

В іншому метод і алгоритм його роботи аналогічні методу processSummaryData.

Метод processIncommingModifiedData

Метод необхідний для виклику методів processSummaryData, processReportDetails і повертає логічне «АБО» результату їх роботи.

3.3.5 Інтерфейс ReportService

Лістинг інтерфейсу ReportService представлено у Додатку Г.
Інтерфейс надає наступний набір методів:

- createReport – створити новий звіт
- updateReport – оновити звіт
- getReport – отримати звіт за id
- getAllReports – отримати усі звіти
- getReportsByDate – отримати звіти за проміжок часу
- getExcelWorkBookWithSeveralReports – отримати WorkBook зі звітом
- updateSummary – оновити Summary звіта\ звітів
- getReportsShortName – отримати коротке ім’я бранчу, котрому налажать звіти

3.3.6 Сервіс ReportService

Лістинг сервісу ReportService представлено у Додатку Д.

3.3.6.1 Анотації класу

Над класом розташована анотація @ Service ("reportService"). Вона є спадкоємцем анотації @ Component і необхідна, для того, щоб Spring зміг

зробити ін'єкцію. Аттрибут всередині дужок-це атрибут value. Він задає ім'я біна, за допомогою якого ми надалі зможемо його використовувати.

3.3.6.2 Поля класу

У класу ReportServiceImpl наступні поля:

- ReportDao reportDao
- UserDao userDao
- ReportStatusService reportStatusService
- ReportGroupService reportGroupService
- UserService userService
- ReportDetailsDao reportDetailsDao
- ReportStatusDao reportStatusDao
- ReportGroupDao reportGroupDao

Всі поля позначені анотацією @Autowired. Поля класу з анотацією @Autowired заповнюються відповідними значеннями відразу після створення bean'a і перед тим, як будь-який з методів класу буде викликаний.

3.4 Підпроект spii-db-scripts

Підпроект spii-db-scripts містить налаштування підключення до бази і liquibase скрипти для створення в базі всієї структури таблиць, необхідних для роботи програми, а так само заповнення таблиць даними (користувачі, тестові дані)[14].

Дані для таблиць зберігаються в окремих csv-файлах. CSV (від англійського Comma-Separated Values – значення, розділені комами) – текстовий формат, призначений для представлення табличних даних. Кожен рядок файлу – це один рядок таблиці. Значення окремих колонок розділяються розділовим символом-комою.

Кожна зміна в базі міститься в тегу changeSet. Набір тегів в ньому визначає набір дій, які виконуються в кожному конкретному changeSet. Це може бути створення, видалення таблиці, перейменування таблиці або її полів, додавання або видалення з неї даних.

3.4.1 Створення таблиці за допомогою liquibase скрипта

Лістинг скрипта:

```
    <changeSet author="dermenzhy" id="8">
            <createTable tableName="reports">
                <column name="id" type="NUMBER(10)">
                    <constraints primaryKey="true" nullable="false"/>
                </column>
                <column name="reportDate" type="DATE">
                    <constraints nullable="false"/>
                </column>
                <column name="summary" type="VARCHAR2(4000)">
                    <constraints nullable="false"/>
                </column>
                <column name="reporter" type="NUMBER(10)">
                                        <constraints  nullable="false"
foreignKeyName="FK$REPORT$REPORTER" references="users(id)"/>
                </column>
            </createTable>
    </changeSet>
```

Даний changeSet створює таблицю "reports". У таблиці представлені наступні колонки:

- Id (тип: NUMBER(10)) – ідентифікатор запису в базі
- reportDate (тип: DATE) – дата створення звіту
- summary (тип: VARCHAR2(4000)) – короткий виклад звіту
- reporter (тип: NUMBER(10)) – ідентифікатор користувача, який створив звіт

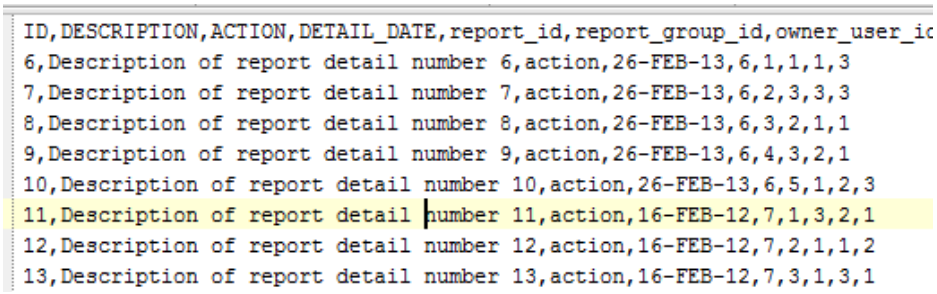
Колонка "reporter" є зовнішнім ключем, який дозволить знайти творця звіту в базі.

3.4.2 CSV-файл зі звітами

Однією з вимог введення системи в експлуатацію було створення набору тестових даних- п'ятдесяти звітів в базі з деталями по кожному звіту. Для генерації цього числа звітів було створено спеціальний додаток. Результатом роботи програми, є текст, наведений на рис. 3.19. та рис. 3.20.

```
ID,REPORT_DATE,SUMMARY,REPORTER_USER_ID
6,26-FEB-13,New summary number 6,3
7,16-FEB-12,New summary number 7,3
8,17-MAY-12,New summary number 8,1
9,7-DEC-13,New summary number 9,4
10,16-APR-12,New summary number 10,3
11,17-JUN-12,New summary number 11,3
12,25-DEC-12,New summary number 12,1
13,4-DEC-12,New summary number 13,4
14,16-MAR-12,New summary number 14,2
15,13-SEP-12,New summary number 15,3
16,5-SEP-13,New summary number 16,1
```

Рисунок 3.19 – Частина тексту csv-файлу з данними звіту



```
ID,DESCRIPTION,ACTION,DETAIL_DATE,report_id,report_group_id,owner_user_ic
6,Description of report detail number 6,action,26-FEB-13,6,1,1,1,3
7,Description of report detail number 7,action,26-FEB-13,6,2,3,3,3
8,Description of report detail number 8,action,26-FEB-13,6,3,2,1,1
9,Description of report detail number 9,action,26-FEB-13,6,4,3,2,1
10,Description of report detail number 10,action,26-FEB-13,6,5,1,2,3
11,Description of report detail number 11,action,16-FEB-12,7,1,3,2,1
12,Description of report detail number 12,action,16-FEB-12,7,2,1,1,2
13,Description of report detail number 13,action,16-FEB-12,7,3,1,3,1
```

Рисунок 3.20 – Частина тексту csv-файлу з деталями звіту

Отримані дані були збережені у файлах reportsNewOne.csv і reportDetailsNewOne.csv і використані для заповнення даних у таблицях. Це було зроблено за допомогою наступних changeSet:

```
      <changeSet author="dermenzhy" id="1">
              <comment>Add new reports</comment>
              <loadData  tableName="REPORTS"
file="liquibase/changelogs/csv/reports/reportsNewOne.csv"/>
          </changeSet>
          <changeSet author="dermenzhy" id="2">
              <comment>Add new report details</comment>
              <loadData  tableName="REPORT_DETAILS"
file="liquibase/changelogs/csv/reports/reportDetailsNewOne.csv"/>
          </changeSet>
```

3.5 Підпроект spii-view

Підпроект містить в собі представлення (view) нашого MVC-додатку. У ньому знаходяться контролери сторінок (з допоміжними класами) і безпосередньо самі сторінки з javascript і css-файлами.

3.5.1 Контролери сторінок

Контролери відповідають за повернення необхідної інформації при роботі користувача з нашою програмою – сторінок, при зверненні до URL-адрес та даних, при отриманні відповідних запитів. Всі класи з підпроекту spii-view представлені на діаграмі на рис. 3.21.

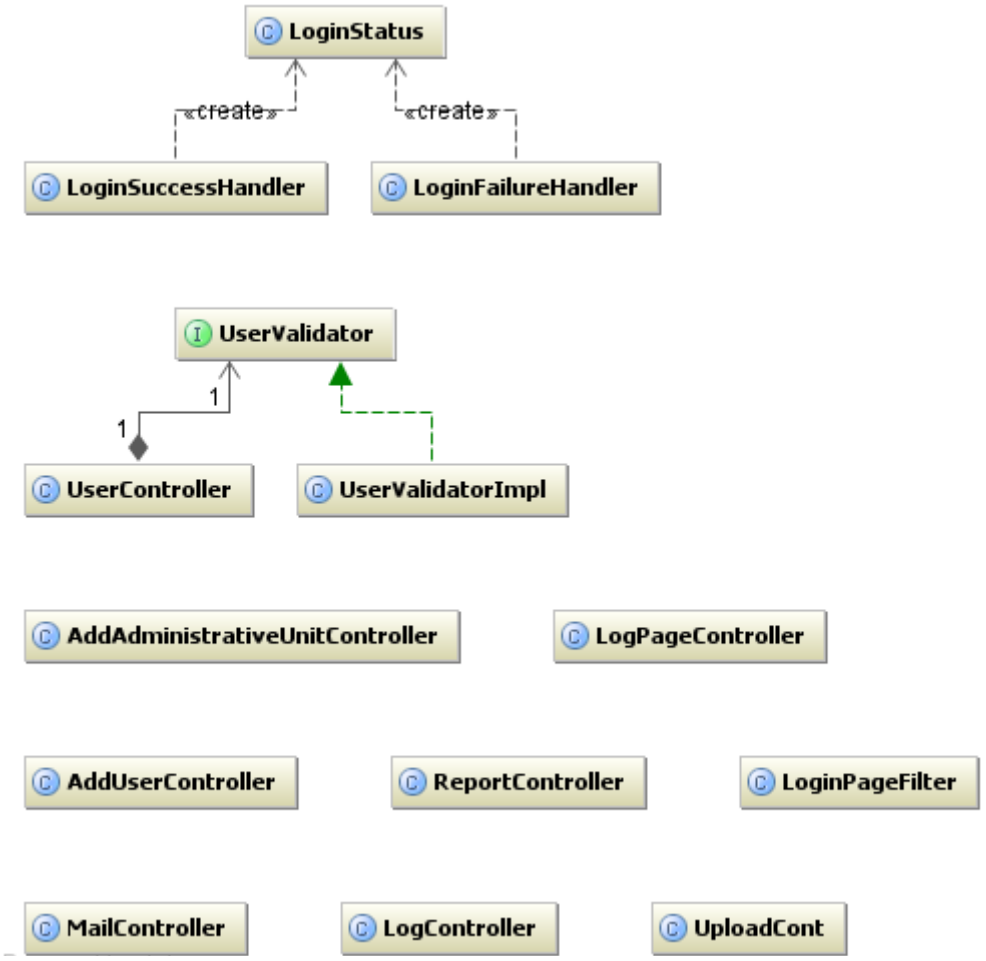


Рисунок 3.21 – Класи підпроекту spii-view

3.5.2 Клас ReportController

3.5.2.1 Анотації класу

Над класом розташована анотація @ Controller. Вона є спадкоємцем анотації @ Component і необхідна, для того, щоб Spring зміг зробити ін'єкцію (повідомляє Spring про те, що даний клас є bean і його необхідно довантажити при старті програми).

3.5.2.2 Поля класу

Клас ReportController містить наступні поля:

- ReportService reportService – сервіс звітів
- UserService userService – сервіс користувачів
- private ReportPageDataProcessor reportPageDataProcessor – сервіс – обробник даних

Всі поля позначені анотацією @Autowired. Поля класу з анотацією @Autowired заповнюються відповідними значеннями відразу після створення bean'a і перед тим, як будь-який з методів класу буде викликаний.

3.5.2.3 Методи класу

У класі ReportController є чотири методи, але ми розглянемо тільки два з них-це метод збереження змінених даних звіту і метод збереження звіту:

- saveModifiedData
- create

Метод saveModifiedData

Метод призначений для збереження змінених даних звіту. Метод приймає в себе об'єкт з якого по строковому ключу можна отримати рядок даних. Повертає метод аналогічний об'єкт. Алгоритм роботи методи представлений на рис. 3.22.

Анотації методу

У методі присутні три анотації. Анотація @RequestMapping визначає тип запиту та його URI. Типу запиту визначається параметром "method". Рядок "method = RequestMethod.GET" означає, що буде виконаний гет-запит. URI вказаний в параметрі value – "value =" / saveModifiedData " ".

Якщо необхідно, щоб результат роботи методу в контролері був виведений безпосередньо в тіло відповіді на запит, а не послужив адресою переходу і не був поміщений як параметр в модель, потрібно вказати безпосередньо перед методом анотацію @ ResponseBody. Цією інструкцією ми віддає відповідь безпосередньо браузеру, минаючи шар представлень. Тобто, те, що віддаємо в методі, те і отримає браузер.

Анотація @RequestParam транслює значення параметра запиту в змінну. Нам немає необхідності піклуватися про тип – Spring призведе змінну до зазначеного типу самостійно.

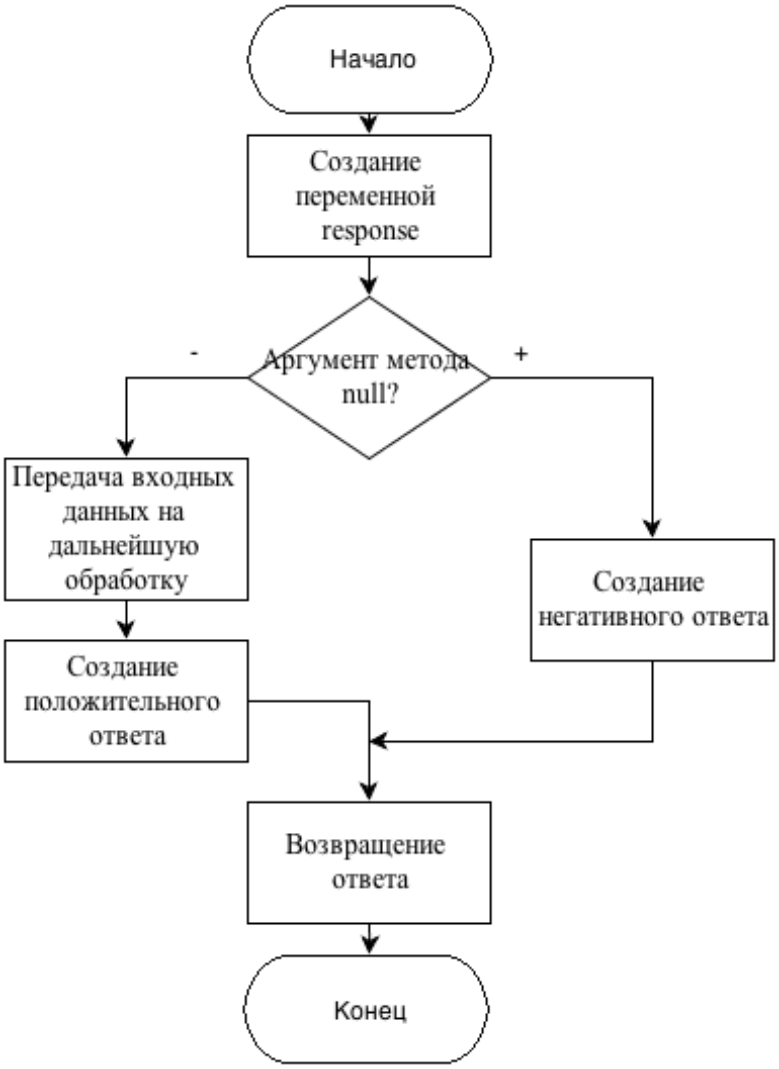


Рисунок 3.22 – Алгоритм роботи методу saveModifiedData

Метод create

Метод призначений для збереження нового звіту в базу. Метод приймає аргумент типу Report. Метод вертає об'єкт типу "Map". Він містить у собі результат операції-"true", у разі успішного завершення операції, "false" у випадку невдачі (так само в цьому випадку в об'єкті буде збережено повідомлення про помилку).

Анотації методу

У методі присутні три анотації. Анотація @ RequestMapping визначає тип запиту та його URI. Типу запиту визначається параметром "method". Рядок "method = RequestMethod.GET" означає, що буде виконаний гет-запит. URI вказаний в параметрі value – "value =" / createReport " ".

Якщо необхідно, щоб результат роботи методу в контролері був виведений безпосередньо в тіло відповіді на запит, а не послужив адресою

переходу і не був поміщений як параметр в модель, потрібно вказати безпосередньо перед методом анотацію @ ResponseBody. Цією інструкцією ми віддає відповідь безпосередньо браузеру, минаючи шар представлень. Тобто, те, що віддаємо в методі, те і отримає браузер.

HTTP-запит крім заголовку і параметрів має також основну частину – тіло запиту. Його вміст також може бути розпізнано як аргумент в методі контролера. Для того, щоб це відбулося, необхідно вказати @ RequestBody в оголошенні цього аргументу.

3.5.3. Контролер сторінки пошуку користувачів

Розглянемо створення контролерів сторінок на прикладі контролера сторінки пошуку користувачів. Контролер сторінки повертає єдиний аргумент - ім'я *. Jsp файлу, которийотвечает за цю сторінку.

Лістинг контролера:

```
        /**
         * @return {String}
         */
                @RequestMapping(value = "/userSearchPage", method = 
RequestMethod.GET)
         public String userSearchPage() {
             return "userSearchPage";
        }
```

3.5.3.1 Анотації контролера сторінки пошуку користувачів

Анотація @ RequestMapping визначає тип запиту та його URI. Тип запиту визначається параметром "method". Рядок "method = RequestMethod.GET" означає, що буде виконаний get-запит. URI вказаний в параметрі value - "value =" / userSearchPage ".

ВИСНОВКИ

Електронний документообіг – це спосіб організації роботи з документами, при якому основна маса документів використовується в електронному вигляді і зберігається централізовано. Система електронного документообігу – програмне забезпечення, розраховане на багато користувачів, яке передбачає організацію роботу з електронними документами, а також взаємодію між співробітниками.

Темою магістерської роботи є розробка системи для обороту звітності для підприємств з географічно-розподіленою структурою.

Метою роботи було спрощення процедури роботи з звітами для топ-менеджерів компанії.

Для досягнення зазначеної мети перед роботою було поставлено ряд завдань:

- Створити єдиний репозиторій звітів для контролю діяльності офісу;
- Здійснити експорт/імпорт звітів в в/з форми Excel за для зняття ризику помилок (людського фактору);
- Об’єднати усі види тижневих звітів філій підприємств.

У ході роботи були досліджені існуючі рішення, було проведено їх аналіз, виявлені переваги і недоліки. На цій підставі було прийнято рішення про створення нової системи.

На основі завдань, які повинна вирішувати система, був описаний функціонал, який необхідно було реалізувати, складені макети сторінок.

Був проведений аналіз технічних засобів, якими можна реалізувати описані функції системи, і прийнято рішення створювати систему на мові Java з використанням таких технологій, як:

- Spring
- JPA
- Liquibase

У якості бази даних було вирішено використовувати СУБД компанії Oracle. Для зборки проекту використовувався maven[15].

Результатом проекту стала форма звіту зі зручним інтерфейсом, що дозволяє створювати, редагувати і переглядати існуючі звіти. Програма підтримує роботу зі звітами в MS Excel і надає багаторолевий доступ до функціональності проекту, вирішуючи поставлені цілі і завдання.

ПЕРЕЛІК ПОСИЛАНЬ

1. Пошукова система гугл [Електроний ресурс] – Режим доступу: https://www.google.com.ua/

2. Головний екран Wikipedia [Електроний ресурс] − Режим досупу: http://ru.wikipedia.org/wiki/Main_Page

3. Збірник питань-відповідей на IT-тематику [Електроний ресурс] – Режим доступу: http://stackoverflow.com/

4. Документація по базах даних компанії Oracle. [Електронний ресурс] – Режим доступу: http://docs.oracle.com/javaee/6/tutorial/doc/

5. Документація по базах даних компанії Oracle. [Електронний ресурс] – Режим доступу: http://www.oracle.com/pls/xe102/homepage

1. Шилдт Герберт. Java 8. Полное руководство. – М .: ООО "И.Д. Вильяме", 2015 - 1376 с.

2. Відеоуроки з створення серверу на Java [Електроний ресурс] – Режим доступу: https://www.youtube.com/watch?v=4Almffj2Gms

3. Відеоуроки роботи з JSON в Java [Електроний ресурс] – Режим доступу: https://www.youtube.com/watch?v=C79JS3NFKnY

4. Робота з JSON [Електроний ресурс] – Режим доступу: http://goloburdin.blogspot.com/2011/03/json-java.html

5. Форум на IT-тематику [Електроний ресурс] – Режим доступу: https://toster.ru/q/57674

6. Збірник навчальних матеріалів. [Електронний ресурс] – Режим доступу: http://www.mkyong.com/

7. Докуметація по Spring. [Електронний ресурс] – Режим доступу: http://www.springsource.org/

8. Збірник статей. [Електронний ресурс] – Режим доступу: http://habrahabr.ru/

9. Документація по liquibase. [Електронний ресурс] – Режим доступу: http://www.liquibase.org/documentation/index.html

6. Документація по maven. [Електронний ресурс] – Режим доступу: http://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

Д О Д А Т К И

ДОДАТОК А ЛІСТИНГ КЛАСУ Report

```
package com.customname.spii.entity;
import javax.persistence.*; import java.util.Date; import java.util.List;
@Entity @Table(name = "REPORTS") public class Report implements DomainObject{
@Id
@GeneratedValue(strategy = GenerationType.SEQUENCE)
@Column(name = "ID", unique = true, nullable = false)
private Long id;
@Temporal(TemporalType.DATE)      @Column(name="REPORT_DATE",nullable = false)
private Date date;
@Column(name = "SUMMARY",nullable = false)
private String summary;
@ManyToOne
@JoinColumn(name = "reporter_user_id", nullable = false)       private User
reporter;
@OneToMany(mappedBy   =   "report",cascade   =   CascadeType.ALL,   fetch   =
FetchType.EAGER
rivate List<ReportDetail> reportDetails;
public Report(){}
public  Report(Date  date,  User  reporter,  String  summary,  List<ReportDetail>
reportDetails) {
this.date = date;
this.reporter = reporter;
this.summary = summary;
this.reportDetails = reportDetails; }
public Long getId() {
return this.id;     }
public void setId(Long id) {
this.id = id;     }
public Date getDate() {
return this.date;     }
public void setDate(Date date) {
this.date = date;     }
public String getSummary() {
return this.summary;     }
public void setSummary(String summary) {
this.summary = summary;     }
public User getReporter() {
return this.reporter;     }
public void setReporter(User reporter) {
```

```
this.reporter = reporter;     }
public List<ReportDetail> getReportDetails() {
return this.reportDetails;     }
public void setReportDetails(List<ReportDetail> reportDetails){
this.reportDetails = reportDetails; } }
```

ДОДАТОК Б ДІАГРАМА КЛАСІВ, ІНТЕРФЕЙСІВ І ЇХ ЗВ'ЯЗОК

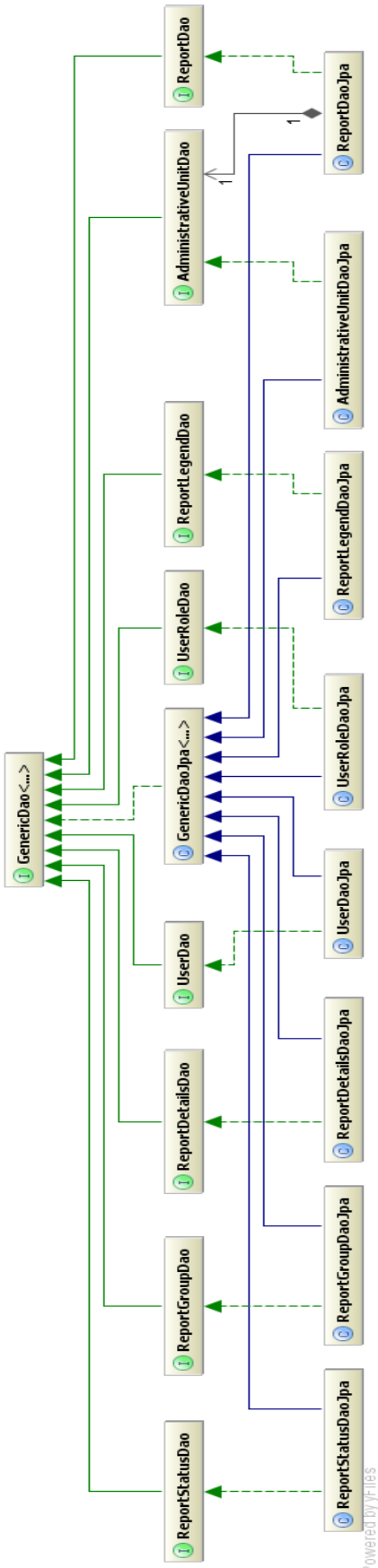


Рисунок Б.1 – Діаграма класів і інтерфейсів

ДОДАТОК В ЛІСТИНГ КЛАСУ GenericDaoJpa

```
package com.exigen.spii.dao.impl;

import com.exigen.spii.dao.GenericDao;
import com.exigen.spii.entity.DomainObject;
import org.springframework.transaction.annotation.Transactional;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import java.util.List;
@SuppressWarnings("unchecked")
@Transactional
public class GenericDaoJpa<T extends DomainObject> implements GenericDao<T> {
    private Class<T> type;
    protected EntityManager entityManager;
    public GenericDaoJpa(Class<T> type) {
        super();
        this.type = type;
    }
    @PersistenceContext
    public void setEntityManager(EntityManager entityManager) {
        this.entityManager = entityManager;}
    public EntityManager getEntityManager() {
        return entityManager;}
    public T getById(Long id) {
        return (T) entityManager.find(type, Long.parseLong("" + id));}
    public List<T> getAll() {
        return entityManager.createQuery(
                                "select  obj  from  "  +  type.getName()  +  "
obj").getResultList();}
    public void save(T object) {
        entityManager.persist(object);}
    public T update(T object) {
        return entityManager.merge(object);}
    public void delete(T object) {
        entityManager.remove(entityManager.merge(object));}
    @Override
    public List<T> executeQuery(String query) {
        return entityManager.createQuery(query).getResultList();}}
```

ДОДАТОК Г ЛІСТИНГ ІНТЕРФЕЙСУ ReportService

```
package com.customname.spii.services;
import com.customname.spii.entity.Report;
import org.apache.poi.ss.usermodel.Workbook;
import java.util.List;

public interface ReportService {
void createReport(Report newReport);
void updateReport(Report report);
Report getReport(Long reportId);
List<Report> getAllReports();
List<Report> getReportsByDate(String beginDate, String endDate, String
branch);
public Workbook getExcelWorkBookWithSeveralReports(List<Long>reportIdList);
public void updateSummary(long id, String summary);
public void updateReportDetails(long id, String description, String action,
String status, String owner);
public String getReportsShortName(List<Long> reportList);
}
```

ДОДАТОК Д ЛІСТИНГ КЛАСУ ReportService

```
@Service("reportService")
public class ReportServiceImpl implements ReportService {
@Autowired
ReportDao reportDao;
@Autowired
UserDao userDao;
@Autowired
ReportStatusService reportStatusService;
@Autowired
ReportGroupService reportGroupService;
@Autowired
UserService userService;
@Autowired
ReportDetailsDao reportDetailsDao;
@Autowired
ReportStatusDao reportStatusDao;
@Autowired
private ReportGroupDao reportGroupDao;
@Transactional
public void createReport(Report newReport) {
Authentication auth =SecurityContextHolder.getContext().getAuthentication();
newReport.setReporter(userService.loadUserByUsername(auth.getName()));
for(ReportDetail rd:newReport.getReportDetails()){
rd.setReport(newReport);
rd.setOwner(userDao.getById(rd.getOwner().getId()));
rd.setReportGroup(reportGroupService.getReportGroupByName(rd.getReportGroup()
.getName()));
rd.setStatus(reportStatusService.getReportStatusByName(rd.getStatus().getName
()));}
reportDao.save(newReport);}
@Override
public void updateReport(Report report) {
reportDao.update(report);}
@Override
public Report getReport(Long reportId) {
return reportDao.getById(reportId);}
@Override
public List<Report> getAllReports() {
return reportDao.getAll();}
@Override
public List<Report> getReportsByDate(String beginDate, String endDate,String
branch) {
return reportDao.getByDate(beginDate, endDate, branch);}
public void updateSummary(long id, String summary) {
Report report = reportDao.getById(id);
UserRole userRole
=(UserRole)SecurityContextHolder.getContext().getAuthentication().getAuthorit
ies().iterator().next();
if(userRole.getAuthority().equals("ROLE_EXECUTOR")) {
Long currentUserId = userRole.getUser().getId();
Long reporterId = report.getReporter().getId();
if(!currentUserId.equals(reporterId)) {
throw new AccessDeniedException("Unauthorized operation for this user");}}
report.setSummary(summary);
reportDao.update(report);
}
public void updateReportDetails(long id, String description, String action,
String status, String owner) {
ReportDetail reportDetail = reportDetailsDao.getById(id);
```

```
UserRole userRole =
(UserRole)SecurityContextHolder.getContext().getAuthentication().getAuthoriti
es(
).iterator().next();
if(userRole.getAuthority().equals("ROLE_EXECUTOR")){
Long currentUserId = userRole.getUser().getId();
Long reporterId = reportDetail.getReport().getReporter().getId();
if(!currentUserId.equals(reporterId)) {
throw new AccessDeniedException("Unauthorized operation for
this user");}}
reportDetail.setDescription(description);
reportDetail.setAction(action);
if (!reportDetail.getStatus().getName().equals(status)) {
ReportStatus reportStatus =
reportStatusDao.getReportStatusByName(status);
if (reportStatus != null) {
reportDetail.setStatus(reportStatus);}}
if (!reportDetail.getOwner().getLogin().equals(owner)) {
User user = userDao.getUserByLogin(owner);
if (user != null) {
reportDetail.setOwner(user);}}
reportDetailsDao.update(reportDetail);}
@Override
public String getReportsShortName(List<Long> reportList){
List<String> branchShortNames = new ArrayList<String>();
for(Long reportId : reportList){
String reportReporterBranchShortName =
this.getReport(reportId).getReporter().getAdministrativeUnit().getShortName()
;
if(!
branchShortNames.contains(reportReporterBranchShortName))branchShortNames.add
(reportReporterBranchShortName);}
return branchShortNames.size() == 1 ? branchShortNames.get(0) :"ALL";}
```