МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра інженерії програмного забезпечення

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Олексій Горський_____
"____"_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
МАГІСТРА

Тема: "Програмний засіб для міграції і синхронізації даних між алм/срм системами"

Виконавець: Міщенко Андрій Миколайович

Керівник: д.т.н професор Степанов Михайло Миколайович

Нормоконтролер: к.т.н доцент Радішевський Микола Федорович

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра інженерії програмного забезпечення
Освітній ступінь магістр
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення»
Форма навчання заочна

ЗАТВЕРДЖУЮ
Завідувач кафедри
Олексій Горський
"___" _____ 2023 р

ЗАВДАННЯ
на виконання кваліфікаційної роботи студента
Міщенка Андрія Миколайовича

1. Тема кваліфікаційної роботи: «Програмний засіб для міграції і синхронізації даних між алм/срм системами»
затверджена наказом ректора від 04.10.2023 р. № 2034/ст.

2. Термін виконання проекту: з 02.10.2022 р. по 31.12.2023 р.

3. Вихідні дані до роботи: програмний продукт розробити за допомогою інтегрованого середовища розробки IntelliJ IDEA.

4. Зміст пояснювальної записки:
1. Аналіз існуючих програмних продуктів для міграції даних та актуальність їх розробки.
2. Актуалізація розробки застосунків для міграції даних.
3. Вимоги до програмного застосунку.
4. Структура програмного продукту.
5. Прототип програмного забезпечення.

5. Перелік обов'язкових слайдів презентації:
1. Актуальність проблеми
2. Об'єкт та предмет дослідження
3. Основні етапи розробки
4. Архітектура додатку.

5. Функціональності додатку.
6. Тестування та результати
7. Переваги та перспективи

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1.	Розробка плану роботи, назви розділів ПЗ та затвердження їх керівником	2.10-8.10	done
2.	Проведення дослідження структури	9.10-15.10	done
3.	Написання пояснювальної записки	16.10-22.10	done
4.	Написання першого розділу	23.10-29.10	done
5.	Представлення першого розділу на кафедру	30.10-31.10	done
6.	Написання другого розділу	1.11-8.11	done
7.	Представлення другого розділу на кафедру	9.11-10.11	done
8.	Написання третього розділу	13.11-20.11	done
9.	Представлення третього розділу на кафедру	21.11-22.11	done
10.	Оформлення графічного матеріалу для презентації	23.11-28.11	done
11.	Завершення написання ПЗ. Проходження нормо контролю. Друк ПЗ Отримання відгуку керівника. Підготовка презентації та доповіді на передзахист.	29.11-10.12	done
12	Передзахист кваліфікаційної роботи. Отримання рецензії	11.12-17.12	done
13	Підготовка документів до захисту та здача їх секретарю ДЕК	18.12-24.12	done
14	Захист кваліфікаційної роботи	25.12-31.12	done

Дата видачі завдання 02.10.2023 р.

Керівник дипломної роботи: _____ д.т.н. професор Михайло СТЕПАНОВ

Завдання прийняв до виконання: _____ Андрій МІЩЕНКО

РЕФЕРАТ

Пояснювальна записка до дипломного проєкту «Програмний засіб для міграції і синхронізації даних між алм/срм системами»: __ с., __ рис., _ табл., __ інформаційних джерел.

КЛЮЧОВІ СЛОВА: МІГРАЦІЯ ДАНИХ, СИНХРОНІЗАЦІЯ, ALM-СИСТЕМИ, CRM-СИСТЕМИ, ЕФЕКТИВНІСТЬ, БЕЗПЕКА ДАНИХ, АВТОМАТИЗАЦІЯ, ІНТЕГРАЦІЯ.

Об’єкт розробки – різноманітні інструменти, алгоритми та функціонал, призначені для перенесення, трансформації і синхронізації даних між різними джерелами та призначеннями, здебільшого вони охоплюють розробку як самостійних програм, так і комплексних рішень для міграції даних.

Мета роботи – дослідження, розробка та впровадження програмого продукта для міграції даних з метою оптимізації та полегшення цього процесу. Важливою частиною є створення ефективного та безпечного засоба для переміщення, трансформації та синхронізації інформації між різними інформаційними системами. Також, метою є розгляд аспектів автоматизації, забезпечення високої продуктивності та забезпечення відповідності стандартам безпеки даних під час міграції.

ABSTRACT

Explanatory note for the diploma project 'Software Tool for Data Migration and Synchronization between ALM/CRM Systems': __ pages, __ figures, __ tables, __ information sources.

KEYWORDS: DATA MIGRATION, SYNCHRONIZATION, ALM SYSTEMS, CRM SYSTEMS, EFFICIENCY, DATA SECURITY, AUTOMATION, INTEGRATION.

The object of development is various tools, algorithms, and functionality designed for transferring, transforming, and synchronizing data between different sources and destinations. Mostly, they encompass the development of both standalone programs and comprehensive solutions for data migration.

The aim of the work is to research, develop, and implement a software product for data migration with the goal of optimizing and facilitating this process. An essential part is the creation of an efficient and secure means for moving, transforming, and synchronizing information between different information systems. Additionally, the goal includes addressing aspects of automation, ensuring high productivity, and compliance with data security standards during migration

ЗМІСТ

ВСТУП 8

РОЗДІЛ 1 АНАЛІЗ ТА ОГЛЯД СУЧАСНИХ АЛМ І СРМ СИСТЕМ 9

1.1. Що таке ALM/CRM системи? Їхня роль у бізнес процесах. 16

1.2. Існуючі продукти на ринку ALM/CRM застосунків 16

1.3. Роль даних у ALM та CRM Системах 28

Висновки 29

РОЗДІЛ 2. ІСНУЮЧІ ПРОГРАМНІ ПРОДУКТИ ДЛЯ МІГРАЦІЇ ДАНИХ ТА АКТУАЛЬНІСТЬ ЇХ РОЗРОБКИ 9

2.1. Актуальність розробки програмного застосунка для міграції даних 16

2.2. Існуючі продукти на ринку програмних застосунків 16

2.3. Визначення проблематики дипломної роботи 28

Висновки 47

РОЗДІЛ 3. РОЗРОБКА ВИМОГ ДО ПРОГРАМНОГО ДОДАТКУ МІГРАЦІЇ ДАНИХ 48

3.1. Вимоги до програмного додатку з міграції даних 49

3.2. Вимоги до операційної системи 50

3.3. Обґрунтування вибору інструментальних засобів 51

3.4. Апаратна частина інформаційної системи 52

3.5. Система інтегрованого середовища розробки об’єктно-орієнтованого програмування 52

3.6. Огляд допоміжних сервісів для взаємодії і інтеграції з основним додатком 56
Висновки 58

РОЗДІЛ 4. РОЗРОБКА ТА НАЛАГОДЖЕННЯ ПРОГРАМНОГО ПРОДУКТУ 59
4.1. Схема роботи програмного додатку 60
4.2. Проектування функціональних можливостей додатку 60
4.3. Проектування та реалізація програмного додатку 62
4.4. Детальний опис та демонстрація роботи програми 66
4.5. Програмна реалізація 71
Висновки 74

ВИСНОВКИ 75
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 76
ДОДАТОК А. Текст програми 77

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

УПР – управління повітряним рухом

ПВП – правила візуального польоту

ППП – правила польотів по приладах

КДП - контрольно-диспетчерські пункти

РЛУ – радіолокаційне управління

ЗПС – злітно-посадкова смуга

АТС – авіаційна транспортна система

МПЛ – місцеві повітряні лінії

ПС – повітряне судно

ПАНО – правила аеронавігаційного обслуговування

САПР – система автоматизованого проектування

ОС – операційна система

ООП – об’єктно-орієнтоване програмування

ВДТ – візуальні дисплейні термінали

РОЗДІЛ 1
АНАЛІЗ ТА ОГЛЯД СУЧАСНИХ АЛМ І СРМ СИСТЕМ

Зростання та еволюція інформаційних технологій супроводжується постійним розвитком систем управління, які спрямовані на поліпшення ефективності та організації процесів у різних галузях бізнесу. У цьому розділі ми зосередимося на аналізі та огляді сучасних систем управління життєвим циклом програмного забезпечення (ALM) та систем управління відносинами з клієнтами (CRM), розглядаючи їх функціональність, переваги та тенденції в їх розвитку.

Сучасне бізнес-середовище вимагає від підприємств високої адаптивності та оперативності. ALM та CRM системи стали ключовими інструментами для досягнення цих цілей, забезпечуючи злагоджену роботу команд, ефективне управління розробкою та ефективну взаємодію з клієнтами. У цьому контексті важливо ретельно розглянути їхні можливості та вплив на стратегічні та операційні аспекти бізнесу.

1.1. Що таке ALM/CRM системи? Їхня роль у бізнес процесах

У світі швидко розвиваючихся технологій ALM (Application Lifecycle Management) та CRM (Customer Relationship Management) системи стали ключовими інструментами для підтримки та оптимізації бізнес-процесів у сучасних компаніях.

ALM системи відіграють важливу роль у керуванні життєвим циклом розробки програмного забезпечення. Вони об'єднують в собі різні етапи від

концепції до випуску продукту, забезпечуючи ефективну координацію завдань та ресурсів. Планування, розробка, тестування, випуск та моніторинг — ALM системи дозволяють компаніям прискорити процес розробки та знизити ризики, пов'язані з випуском програмного продукту.

Сучасні бізнеси також активно використовують CRM системи для підтримки взаємодії з клієнтами та оптимізації бізнес-процесів в сфері обслуговування. Збір та аналіз інформації про клієнтів, автоматизація процесів продажів та підвищення якості обслуговування — це лише кілька функцій, які роблять CRM системи невід'ємною частиною сучасного бізнесу.

Обидві системи допомагають підвищити ефективність та конкурентоспроможність компаній. ALM забезпечує структурований підхід до розробки, тоді як CRM робить акцент на вдосконаленні взаємовідносин з клієнтами. Інтеграція цих систем дозволяє підприємствам максимально використовувати переваги автоматизації управління як життєвим циклом продукту, так і відносинами з клієнтами.

У підсумку, ALM та CRM системи є краєвидними технологіями, що допомагають компаніям адаптуватися до змін у сучасному бізнес-середовищі. Вони стали необхідними інструментами для досягнення успіху, забезпечуючи високий рівень ефективності та задоволення потреб як клієнтів, так і розробників.

1.2. Що таке ALM/CRM системи? Їхня роль у бізнес процесах

Системи управління життєвим циклом додатків (Application Lifecycle Management - ALM) стали важливим елементом сучасного програмного розробництва, забезпечуючи комплексний підхід до управління всім циклом життя програмного продукту. У цьому огляді розглянемо три передові ALM-системи: Jira, Microsoft Azure DevOps та Micro Focus Octane.

- Jira

Jira вирізняється своєю гнучкістю та ефективністю управління робочими завданнями. Розроблений компанією Atlassian, цей інструмент пропонує широкий спектр можливостей, включаючи трекінг завдань, контроль версій та автоматизацію тестування. Jira також інтегрується з різними іншими інструментами розробки, що полегшує спільну роботу команд.

- Microsoft Azure DevOps

Azure DevOps від Microsoft є повністю інтегрованою ALM-платформою, яка об'єднує у собі засоби для планування проектів, управління версіями, автоматизації тестування та постачання програмного забезпечення. Завдяки високому рівню інтеграції з іншими продуктами Microsoft, ця система дозволяє розробникам працювати ефективно та швидко.

- Micro Focus Octane

Octane від Micro Focus є сучасною ALM-платформою, яка враховує особливості роботи в гібридних та мікросервісних середовищах. Вона пропонує інструменти для керування завданнями, відстеження вимог, тестування та аналізу продуктивності. Octane також підтримує автоматизацію та інтеграцію з різноманітними іншими інструментами розробки.

Висновок

За останні роки сучасні ALM-системи, такі як Jira, Microsoft Azure DevOps та Micro Focus Octane, стали необхідними інструментами для ефективного управління розробкою програмного забезпечення. Їхня гнучкість, інтеграційна спроможність та розширюваність роблять їх популярними серед розробників та менеджерів проектів, допомагаючи їм досягати високої якості та швидкості в роботі.

У сучасному світі взаємодія з клієнтами є критичним елементом успіху будь-якого бізнесу. Customer Relationship Management (CRM) системи

виконують ключову роль у цьому процесі, допомагаючи компаніям ефективно управляти відносинами з клієнтами. У цьому огляді ми розглянемо три передові CRM-системи: Salesforce, HubSpot CRM та Zoho CRM.

- Salesforce

Salesforce визнаний як лідер серед CRM-платформ та пропонує повний спектр інструментів для управління продажами, маркетингом та обслуговуванням клієнтів. З його допомогою підприємства можуть створювати та вдосконалювати відносини з клієнтами, використовуючи інтуїтивно зрозумілі інструменти та аналітику.

- HubSpot CRM

HubSpot CRM пропонує великий набір інструментів для автоматизації маркетингу, управління продажами та обслуговування клієнтів. Особливістю є безкоштовна версія з базовим функціоналом, що робить його доступним для малих бізнесів. HubSpot CRM також славиться інтеграцією з іншими продуктами HubSpot.

- Zoho CRM

Zoho CRM відомий своєю гнучкістю та можливістю налаштування, що робить його популярним серед різноманітних компаній. Забезпечуючи управління контактами, продажами та маркетингом, Zoho CRM дозволяє підприємствам ефективно взаємодіяти з клієнтами та збільшувати продуктивність.

Висновок

Salesforce, HubSpot CRM та Zoho CRM представляють сучасне покоління CRM-систем, спрямоване на максимізацію ефективності та покращення відносин з клієнтами. Вони відзначаються своєю

функціональністю, інтуїтивно зрозумілим інтерфейсом та можливістю інтеграції з іншими інструментами, роблячи їх важливими інструментами для сучасних бізнесів, які прагнуть досягти виняткової якості обслуговування клієнтів.

1.3. Роль даних у ALM та CRM Системах

Дані у системах ALM визначають всі аспекти розробки програмного забезпечення. Вони починають свій шлях від етапу збору вимог, де важливо визначити очікування та потреби користувачів. На етапі розробки дані визначають стан робочих завдань, використання ресурсів та управління версіями коду. Збором та аналізом даних у системі ALM підприємство може прискорити час розробки, зменшити витрати та забезпечити високу якість продукту.

Щодо CRM систем - дані використовуються для створення повноцінного портрету клієнта. Вони включають інформацію про покупки, попередні взаємодії, предпочтення та інші аспекти взаємодії. Збір та аналіз даних дозволяють підприємствам не тільки персоналізувати комунікації, але й прогнозувати майбутні потреби клієнтів, створюючи стратегії маркетингу та обслуговування, спрямовані на зміцнення відносин.

Інтеграція даних між ALM та CRM системами стає стратегічним елементом для компаній, які прагнуть максимізувати віддачу від своїх інформаційних ресурсів. Здатність обмінюватися даними між цими системами сприяє більш ефективній взаємодії розробників, менеджерів проектів та відділів обслуговування клієнтів. Це дозволяє не тільки прискорити розробку продукту, але й забезпечити постійний обіг інформації для удосконалення стратегій обслуговування клієнтів.

Висновок

Роль даних у системах ALM та CRM визначається їхнім внеском у процес розробки та взаємодії з клієнтами. Ці дані не лише оптимізують внутрішні процеси підприємства, а й створюють умови для постійного удосконалення взаємодії з клієнтами, що є критичним для успіху в конкурентному бізнес-середовищі.

Висновок

Аналіз та огляд сучасних систем управління життєвим циклом проекту (АЛМ) та систем управління відносинами з клієнтами (CPM) виявив різноманіття та важливість таких інструментів у сфері інформаційних технологій та управління проектами. Отже, в цьому розділі визначено ключові особливості та тренди на ринку таких систем, їх переваги та недоліки.

Однією з головних тенденцій сучасних АЛМ систем є інтеграція різноманітних інструментів у єдиний робочий процес, що забезпечує зручність та ефективність в управлінні проектами. Модульність та розширюваність систем дозволяють адаптувати їх до потреб конкретного проекту чи команди.

Огляд сучасних CPM систем підкреслив важливість взаємодії з клієнтами та автоматизацію процесів ведення відносин. Аналіз різних систем дозволив виокремити ключові функції, такі як керування контактами, ефективний моніторинг звернень та аналітика для прийняття обґрунтованих рішень.

У результаті цього аналізу були визначені проблемні аспекти та потреби, які спонукали до розробки програмного додатку для міграції та

синхронізації даних між різними АЛМ/CPM системами. Відповідно до цього, подальший розділ дипломної роботи присвячено розробці та налагодженню такого програмного продукту, який не лише враховує сучасні тенденції, але й вирішує конкретні завдання користувачів у галузі управління проектами та відносинами з клієнтами.

РОЗДІЛ 2
АНАЛІЗ ІСНУЮЧИХ ПРОГРАМНИХ ПРОДУКТІВ ДЛЯ МІГРАЦІЇ ДАНИХ ТА АКТУАЛЬНІСТЬ ЇХ РОЗРОБКИ

2.1. Актуальність розробки програмного застосунка для міграції даних

Перш ніж приступити до вивчення проблематики та аналізу ринку подібних додатків, варто декілька слів сказати про цільові платформи, між якими і проводять міграцію даних, треба наголосити, що обидва типи систем (АЛМ і CPM) грають важливу роль у сучасному бізнесі, допомагаючи підтримувати ефективність і конкурентоспроможність компаній, але вони спрямовані на різні аспекти діяльності. АЛМ важливий для розробки програмного забезпечення, тоді як CPM зосереджений на відносинах з клієнтами і продажах.

Розробка програмних засобів для міграції і синхронізації даних між системами управління життєвим циклом програмного забезпечення (АЛМ) та системами управління відносинами з клієнтами (CPM) залишається актуальною і важливою задачею з численними практичними застосуваннями. Ось деякі ключові аргументи, які підтримують актуальність цієї розробки:

Розповсюдженість систем АЛМ і CPM: Багато компаній та організацій використовують системи АЛМ для управління розробкою програмного

забезпечення і CPM для управління відносинами з клієнтами. Ці системи зберігають великі обсяги даних, і їх інтеграція та синхронізація стають ключовими завданнями.

Необхідність обміну даними: Організації потребують обмінюватися даними між АЛМ і CPM системами для забезпечення ефективної роботи. Це може включати в себе передачу вимог до програм, спроби звітування, відстеження помилок, інформацію про релізи тощо.

Забезпечення консистентності даних: Важливо, щоб дані в обох системах були узгоджені і консистентні. Це сприяє уникненню конфліктів, непорозумінь і покращує прийняття рішень.

Підвищення продуктивності і якості: Забезпечення ефективної міграції і синхронізації даних допомагає підвищити продуктивність команд розробки та підвищити якість програмного забезпечення.

Постійний розвиток і зміна систем: Як системи АЛМ і CPM постійно розвиваються і оновлюються, з'являються нові функціональні можливості і змінюються структури даних. Тому програмні засоби для міграції і синхронізації даних повинні постійно адаптуватися до нових вимог.

Конкурентні переваги: Розробка власного програмного засобу для міграції і синхронізації даних може надати компанії конкурентну перевагу, спрямовуючи її на власну інфраструктуру та бізнес-процеси.

Зростання обсягів даних: Обсяги даних постійно зростають, і потреба в їхній обробці і передачі між системами лише посилюється.

Таким чином, розробка програмних засобів для міграції і синхронізації даних між АЛМ і CPM системами залишається актуальною і важливою задачею для багатьох організацій, і це є раціональним обгрунтуванням для подальших досліджень і розробки в цій області.

2.2. Існуючі продукти на ринку програмних застосунків для міграції

На ринку існують різні програмні засоби для міграції і синхронізації даних між системами управління життєвим циклом програмного забезпечення (АЛМ) та системами управління відносинами з клієнтами (CPM). Ці продукти можуть мати різний функціонал, цінову політику і можливості. Ось кілька прикладів таких продуктів:

1. Microsoft Power Automate (раніше відомий як Microsoft Flow):
   - *Опис*: Microsoft Power Automate - це сервіс автоматизації робочих процесів, який дозволяє створювати потоки роботи для автоматичної міграції та синхронізації даних між різними додатками, включаючи АЛМ та CPM системи.
   - *Функціональність*: Power Automate надає широкий спектр дій та інтеграцій для автоматизації робочих процесів і обміну даними.

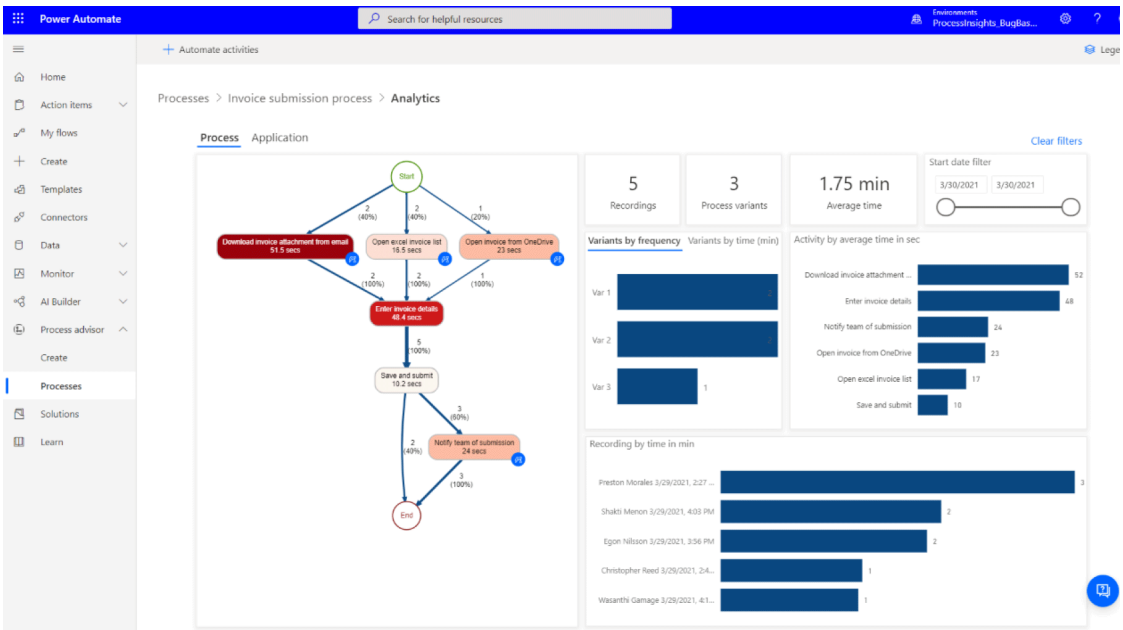

Рис. 2.1. Інтерфейс застосунку Microsoft Power Automate

2. Zapier:

- *Опис*: Zapier - це інтернет-сервіс, який дозволяє створювати зв'язки між різними додатками та автоматизувати завдання, включаючи міграцію та синхронізацію даних між різними системами.
- *Функціональність*: За допомогою Zapier можна створювати засоби інтеграції (запи) для різних додатків і визначати правила автоматичної обробки подій та даних.

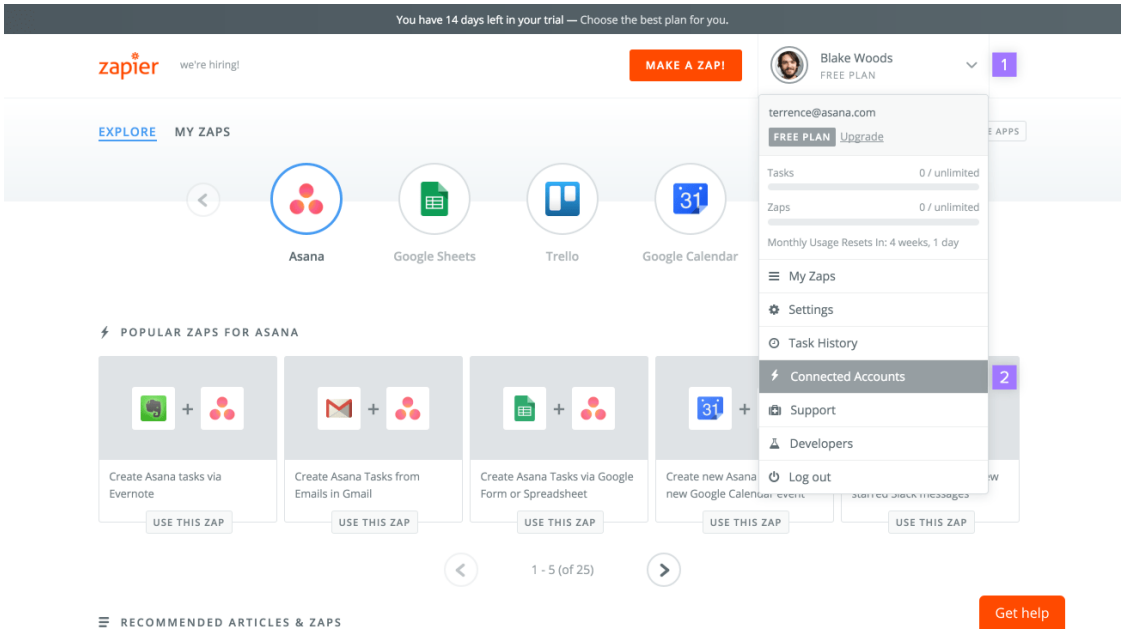

Рис. 2.2. Інтерфейс застосунку Zapier

3. Integrate.io (раніше Talend):

- *Опис*: Integrate.io - це інтеграційний платформа, яка дозволяє створювати і керувати потоками даних між різними джерелами даних, включаючи АЛМ і CPM системи.

- *Функціональність*: Ця платформа надає інструменти для витягування, перетворення, завантаження даних і автоматизованої синхронізації.

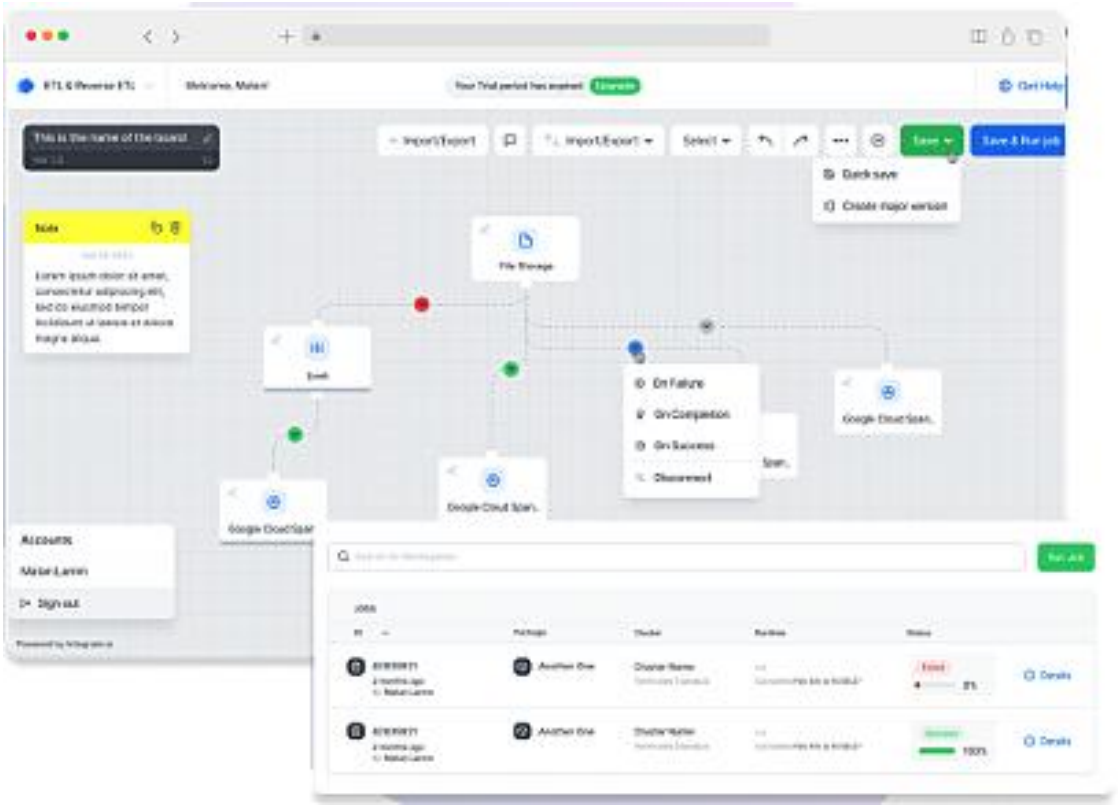

Рис. 2.3. Інтерфейс застосунку Integrate.io

4. Apache Nifi:

- *Опис*: Apache Nifi - це відкрите програмне забезпечення для обробки та забезпечення потоків даних. Воно може бути використано для міграції і синхронізації даних між різними джерелами і системами.
- *Функціональність*: Apache Nifi надає різні процесори для обробки даних та можливості інтеграції з різними джерелами та призначеннями.

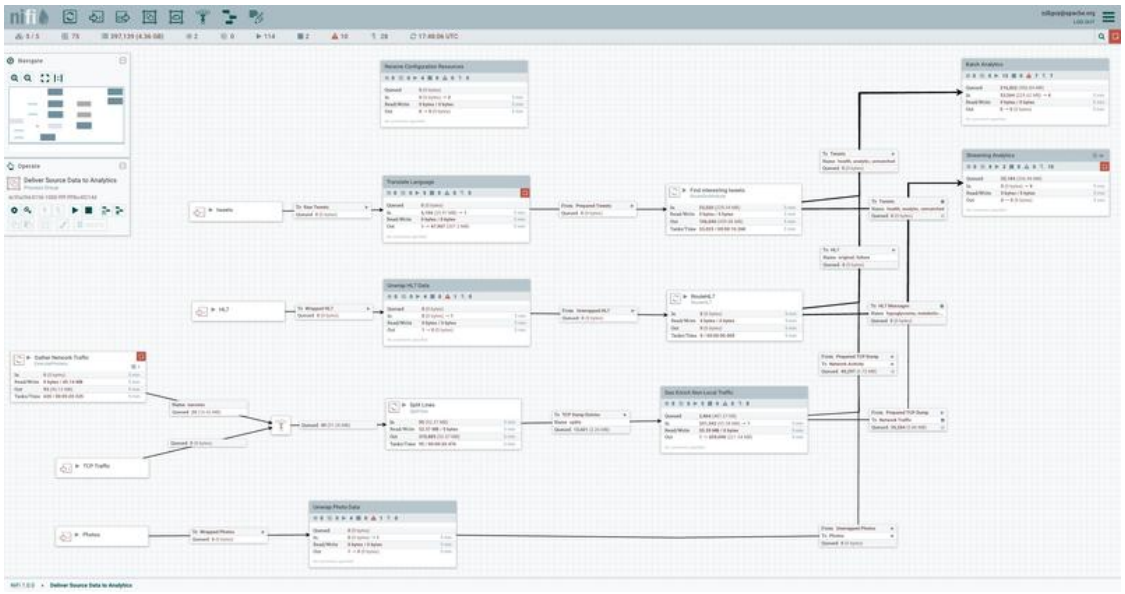

Рис. 2.4. Інтерфейс застосунку Apache Nifi

5. Skyvia:

- *Опис*: Skyvia - це хмарний сервіс для інтеграції та синхронізації даних між різними джерелами, включаючи бази даних, CRM, інші хмарні служби і системи.
- *Функціональність*: Skyvia надає інструменти для конфігурації правил синхронізації, маппінгу даних та планування завдань обміну даними.

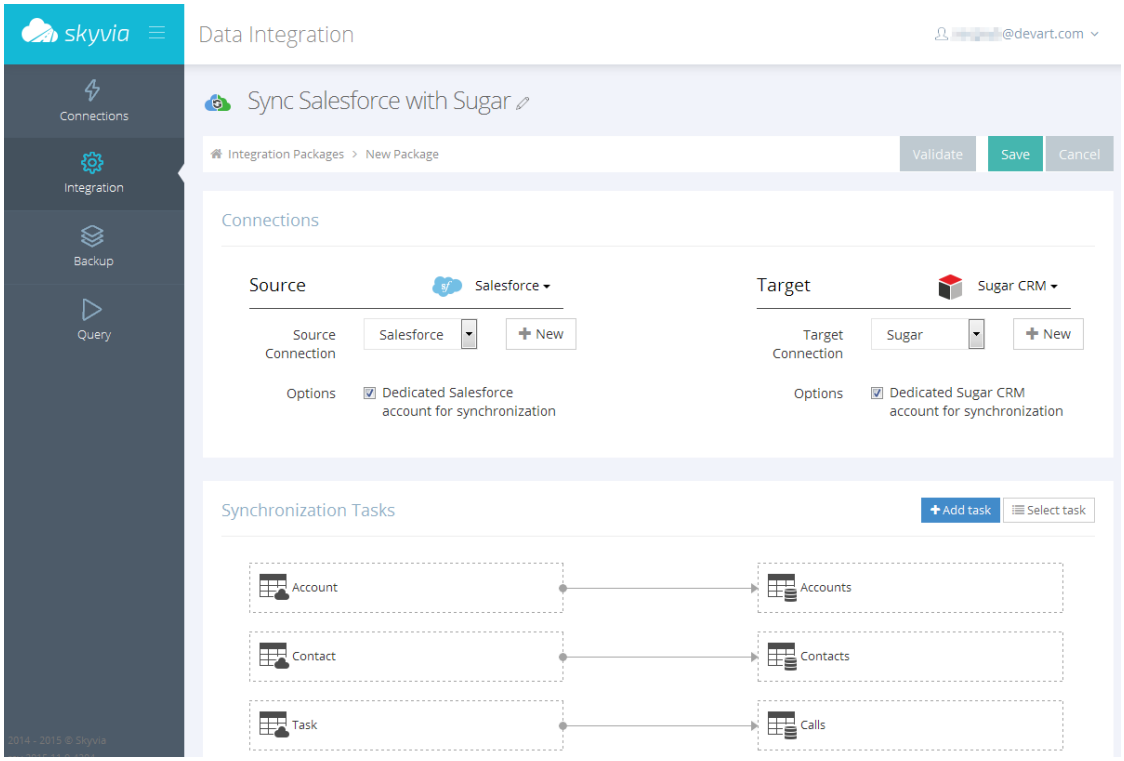

Рис. 2.5. Інтерфейс застосунку Apache Nifi

2.3. Визначення проблематики дипломної роботи

У процесі вивчення проблематики вибору програмних засобів для міграції і синхронізації даних між АЛМ та/або СРМ системами були ідентифіковані головні проблеми, які варто враховувати при виборі відповідного рішення. Основними проблемами є:

- **Швидкість передачі даних**: Зі зростанням обсягу даних у системах АЛМ і СРМ виникає необхідність у програмних засобах, які забезпечують швидку передачу і обробку даних. Повільна передача може призвести до затримок у робочих процесах і негативно вплинути на продуктивність.

- **Надійність і цілісність даних**: Важливо гарантувати, що дані не втрачаються та залишаються цілими під час міграції і синхронізації. Помилки або конфлікти можуть призвести до серйозних проблем для бізнес-процесів.
- **Сумісність і інтеграція**: Вибраний програмний засіб повинен бути сумісним з існуючими системами АЛМ і СРМ, а також мати можливості інтеграції з ними без значущих труднощів.
- **Забезпечення безпеки даних**: Збереження конфіденційності і безпеки даних під час їх передачі та обробки є важливою проблемою, особливо у випадку обробки чутливої інформації.

З метою успішної інтеграції та забезпечення надійної міграції та синхронізації даних між АЛМ та/або СРМ системами, необхідно враховувати вищезазначені проблеми і обирати програмні засоби, які відповідають конкретним потребам та вимогам проекту.

Мета дипломної роботи – подумати над концепцією простого і надійного додатку для міграції даних, створити MVP (Minimum Viable Product - Мінімально життєздатний продукт) цього ПЗ, яке б могло зацікавити потенційних замовників, які планують перейти з однєї платформи на аналогічну іншу.

Для досягнення мети необхідно використати такі технології, за допомогою яких будуть виконуватись задачі.

- використання інтегрованого середовища розробки Intellij Idea (>2023.2.2) та використання мови програмування Java;
- використання багатоплатформового інструменту для розробки застосунків та ігор;
- розробити структуру та ієрархію інтерфейсу;

- розробити ергономічний користувацький інтерфейс.

Висновок

У процесі вивчення актуальності розробки програмного застосунка для міграції даних та аналізу існуючих продуктів на ринку програмних застосунків, було отримано важливу інформацію, яка відображає стан та перспективи цієї галузі. На підставі проведених досліджень можна зробити наступні висновки:

Актуальність розробки програмного застосунка для міграції даних підтверджується рядом факторів. Спричинені зростанням обсягів даних, змінами в бізнес-процесах та вимогами законодавства, потребою у забезпеченні безпеки та ефективності міграційних операцій. Це свідчить про необхідність постійного вдосконалення програмних засобів для цієї області.

Аналіз існуючих продуктів показав, що на ринку існують різноманітні рішення для міграції даних. Однак, існує певний простір для розробки нових продуктів, які б задовольняли специфічні потреби користувачів, були болісною точкою для існуючих рішень.

Дослідження показало, що існуючі програмні продукти для міграції даних мають свої переваги та обмеження. Це вказує на можливість розробки нових інноваційних рішень, які б вирішували проблеми та покращували ефективність процесу міграції.

Для подальшого розвитку галузі рекомендується проводити подальший моніторинг та дослідження ринку, враховувати зміни в технологічному середовищі та вимоги користувачів. Розробка нових програмних продуктів для міграції даних може стати важливим напрямком для інновацій та покращення бізнес-процесів.

Отже, аналіз показав, що розробка програмного застосунка для міграції даних є актуальною і має потенціал для подальшого розвитку. Ринок продуктів для міграції даних відкриває можливості для створення нових інноваційних рішень, які задовільнять потреби користувачів та покращать процеси обробки даних.

РОЗДІЛ 3
РОЗРОБКА ВИМОГ ДО ПРОГРАМНОГО ДОДАТКУ МІГРАЦІЇ

3.1. Вимоги до програмного додатку з міграції даних

Аналіз наукових джерел, та джерел з інженерії знань показав, що найбільш розповсюдженими програмами для створення проектів, пов’язаних із розробкою засобів роботи з серверними системами є мови Java, а саме IDE Intellij Idea, JavaScript для графічної частини та ін.

Вибір мови Java у середовищі Intellij Idea зумовлено тим, що інструменти для програмування на Java абсолютно безкоштовні, їх багато (як і інформації про них самих). Так Intellij Idea - безкоштовно-поширюване середовище розробки, яке використовує широкий функціонал та має гідний рівень якості.

Java володіє великою множиною стандартних бібліотек і класів та детальною документацією, що полегшує написання найпростіших, але широко використовуваних методів. Саме такий підхід дозволяє писати код, слідуючи найкращим практикам програмування. Це досягається завдяки таким фреймворками як Spring, Sencha, Gradle.

Ще однією із особливостей Java є повна кроссплатформенність. Правильно спроектований додаток потенційно може однаково працювати як на Windows, так і на MacOS, Linux і багатьох інших операційних системах.

І найважливіше. Комплект для розробників Java за замовчуванням постачається з попередньо встановленим фреймворком Gradle, який ми й будемо використовувати для управління життєвим циклом розробки нашим конвертером.

Одна з головних особливостей цього фреймворка - декларативний опис проекту. Це означає, що розробнику не потрібно приділяти увагу кожному аспекту збірки - всі необхідні параметри налаштовані за замовчуванням. Зміни потрібно вносити лише в тому обсязі, в якому програміст хоче відхилитися від стандартних налаштувань.

Ще одна перевага проекту - гнучке управління залежностями. Gradle вміє довантажувати в свій локальний репозиторій сторонні бібліотеки, вибирати необхідну версію пакету, обробляти транзитивні залежності.

Визначимося із вимогами до створюваного нами програмного додатку, а саме з функціональними та не функціональними вимогами.

Табл. 3.1.

Специфікація функціональних вимог

Код вимоги	**Опис вимог**
1.	**Введення даних**
F1.1	Додаток повинен надавати можливість введення параметрів для підключення до джерела та цільової системи.
F1.2	Користувач повинен мати можливість вказати конкретні об'єкти (такі як задачі, баги, вимоги тощо) для міграції.
2.	**Маппінг даних**
F2.1	Додаток повинен надавати можливість визначити відповідність полів між джерелом та цільовою системою.
F2.2	Має бути можливість створювати та зберігати мапу полів для подальшого використання
3.	**Перевірка та валідація даних**
F3.1	Додаток повинен забезпечувати механізми перевірки та валідації даних перед міграцією
F3.2	Має бути система повідомлень про помилки та попередження для користувача
4.	**Перенос даних**

F4.1	Система повинна надавати можливість розпочати процес міграції з урахуванням введених параметрів та мапування
F4.2	Повинна бути можливість відслідковування стану міграції та виведення звітів
5.	**Завершення процесу міграції**
F5.1	Після завершення міграції повинна бути забезпечена можливість перевірки та підтвердження успішності операції
F5.2	Користувач повинен отримати повідомлення про завершення процесу та отримати результати міграції
6.	**Захист даних**
F6.1	Додаток повинен забезпечувати захист конфіденційності та цілісності даних під час міграції.
7.	**Автоматизація**
F7.1	Забезпечення можливості автоматизації процесу міграції для зменшення вручнії роботи та підвищення ефективності.

Табл. 3.2.

Специфікація нефункціональних вимог

Код вимоги	Опис вимог
NF1	Мова програмування JAVA
NF2	Можливість роботи під ОС Windows.
NF3	Забезпечити систему логуванням для відстеження подій і

	можливість пошуку помилок.

3.2. Вимоги до операційної системи

Для розробки програм на мові Java використовується спеціальне програмне забезпечення. Найновіші версії системного програмного забезпечення, необхідного для підтримки, можна завантажити з сайту компанії Azul (https://www.azul.com/): JRE, JDK. Перший додаток JRE – це програма для запуску і виконання програм (середовище виконання Java) Java Runtime Environment (JRE).

Для створення програм також має бути встановленим комплект розробки програмного забезпечення - JDK (Java Development Kit). Він містить компілятор, стандартні бібліотеки тощо.

Середовище Intellij Idea розроблене для широкого кола операційних систем, таких як Linux, Microsoft Windows і Mac OS. Для її запуску потрібно JVM (Java Virtual Machine) - віртуальна Java-машина.

Табл. 3.3.

Системні вимоги до Java Intellij Idea

Вимога	**Мінімальне значення**	**Рекомендоване значення**
Версія Java	11	11 і вище
Оперативна пам’ять	4 Гб	8 Гб та більше
Вільний простір на ЖД	8 Гб	10 Гб та більше
Процесор	1,5 ГГц	2,5 ГГц та більше

У табл. 3.3 представлені мінімальні і рекомендовані системні вимоги для роботи Intellij Idea.

3.3. Обґрунтування вибору інструментальних засобів

Вимоги до операційної системи (ОС) для додатку з міграції даних між ALM системами, написаного на мові Java та використовуючого серверну структуру, можуть бути визначені з урахуванням технічних характеристик та залежностей програмного забезпечення. Ось деякі загальні вимоги:

Операційна система сервера:

Додаток має підтримуватися та бути сумісним із популярними серверними операційними системами, такими як Windows Server, Linux (наприклад, Ubuntu, CentOS), або іншими, в залежності від вимог користувача та інфраструктури підприємства.

Версія Java Runtime Environment (JRE):

Додаток, написаний на мові Java, вимагає встановлення та коректної конфігурації Java Runtime Environment (JRE) або Java Development Kit (JDK). Важливо визначити сумісні версії JRE або JDK, які підтримуються додатком.

Ресурси сервера:

Визначення мінімальних та рекомендованих системних ресурсів (процесор, обсяг оперативної пам'яті, дисковий простір), необхідних для ефективної роботи додатку.

База даних:

Визначення підтримуваних систем управління базами даних (наприклад, MySQL, Oracle, Microsoft SQL Server) та їх версій для забезпечення взаємодії з даними під час міграції.

Мережеві вимоги:

Визначення мережевих протоколів та вимог до мережевої інфраструктури для забезпечення ефективного обміну даними між додатком та ALM системами.

Серверні технології:

Опис використовуваних серверних технологій, таких як веб-сервери (наприклад, Apache, Nginx) або контейнери додатків (наприклад, Tomcat, Jetty).

Безпека:

Визначення заходів забезпечення безпеки, таких як доступ через SSL/TLS, можливості аутентифікації та авторизації, шифрування даних та інші заходи безпеки.

В контексті розробки додатку для міграції даних між системами управління життєвим циклом додатків (ALM), визначення вимог грає критичну роль у забезпеченні успішності та ефективності проекту. В даному аналізі були висвітлені як функціональні, так і нефункціональні вимоги, спрямовані на забезпечення зручного та безперебійного процесу міграції даних.

Функціональні вимоги визначають точний функціонал, який має надавати додаток, такі як маппінг даних, валідація, перенос даних та інші, що становлять основу для успішної міграції.

Нефункціональні вимоги, у свою чергу, визначають параметри продуктивності, надійності, безпеки та інші аспекти, які визначають загальну якість додатку. Інтеграція з різними операційними системами, підтримка різних версій баз даних, висока стійкість до відмов та ефективне використання ресурсів сервера - це лише кілька з нефункціональних вимог, які забезпечують оптимальне функціонування додатку.

За умови виваженого використання вимог та інструментів, розробка додатку для міграції даних в середовищі ALM може бути виконана ефективно та забезпечити користувачам зручність у роботі та надійність процесу міграції.

3.4. Апаратна частина інформаційної системи

Таким чином апаратна частина інформаційної системи для додатку міграції даних буде виглядати наступним чином:

1. **Сервер**: Використаємо потужне серверне обладнання з достатньою кількістю оперативної пам'яті та швидким процесором для оптимальної роботи додатку. Рекомендується система з підтримкою многозадачності та великою пропускною здатністю.
2. **Операційна система**: Слід обирати оптимізовану для серверів операційну систему, таку як Linux або Windows Server, для забезпечення стабільності та надійності.
3. **База даних** (Derby):
   - Серверна конфігурація: Підготуємо апаратну конфігурацію сервера для ефективної роботи з базою даних Derby. До цього слід враховувати параметри пам'яті та обсяги дискового простору для оптимізації продуктивності.
   - Забезпечення резервного копіювання: Також у планах - розробка системи регулярного резервного копіювання для забезпечення надійності та безпеки даних.
4. **Клієнтська частина** (Ext.js):
   - Комп'ютери користувачів: Використовуймо комп'ютери з сучасними веб-браузерами для оптимального використання інтерфейсу на базі Ext.js. Перед цим переконайтеся, що вони мають достатньо продуктивні характеристики для роботи з великими обсягами даних.
   - Мережева інфраструктура: Варто забезпечите стабільне мережеве підключення, оскільки великі обсяги даних передаються між клієнтом та сервером.
5. Моніторинг та аналіз продуктивності:

- Інструменти моніторингу: Встановимо інструменти для моніторингу продуктивності сервера та бази даних, також для збору логів за допомогою Log4j, що дозволить вчасно виявляти проблеми та вирішувати їх.
- Журналювання подій: Використаємо систему журналювання, включаючи Log4j, для детального відстеження подій та виявлення можливих помилок.

За допомогою Log4j, ми забезпечимо ефективне логування подій, що дозволить нам здійснювати детальний моніторинг роботи додатку та вчасно вживати заходів для збереження його стабільності та надійності.

Висновок

Даний розділ виявився ключовим у формуванні стратегії, функціональності та архітектури програмного засобу для міграції даних між різними системами управління проектами. У процесі розробки вимог було враховано низку аспектів, що визначають ефективність, гнучкість та надійність додатку.

Визначення основних функціональних вимог було виконано з урахуванням потреб користувачів і можливостей систем управління проектами (ALM/CPM). Передбачено можливість вибору конкретних елементів для міграції, надання конфігураційних опцій та налаштувань для кожної підтримуваної системи, а також можливість синхронізації між різними ресурсами.

Підсумкова архітектура програмного продукту, що виникла з цих вимог, реалізує клієнт-серверну архітектуру з використанням Java для основної логіки та Ext.js для графічного інтерфейсу. Враховано аспекти

безпеки та забезпечено можливість конфігурації зв'язків з базами даних для зберігання конфігурацій та інших даних.

Цей розділ визначає надійність та готовність програмного продукту до реалізації. Описані функціональності, визначені вимоги та стратегії реалізації, формують основу для подальших етапів розробки та тестування, забезпечуючи стійкість та ефективність у реальних умовах використання. Усе це сприяє створенню високопродуктивного та витонченого інструменту для міграції та синхронізації даних між різними системами управління проектами.

РОЗДІЛ 4
РОЗРОБКА ТА НАЛАГОДЖЕННЯ ПРОГРАМНОГО ПРОДУКТУ

4.1. Схема роботи програмного додатку

Міграція даних — це процес переміщення даних із вихідних ресурсів до місць призначення, і їх сценарії можуть відрізнятися. Може бути багато причин, чому організація починає проект міграції даних, від оновлення додатків і впровадження нових систем підприємства до повної реорганізації в результаті злиття компанії. Аналізуючи досвід міграції даних у великій кількості інформаційних систем, можна підсумувати типовий процес міграції даних, який включає: аналіз формату даних і цільової структури вихідних ресурсів, формулювання планів міграції та перетворення даних, визначення зв'язків між елементами (ієрархія). об'єктів) ;Визначити порядок передачі даних на основі ієрархії залежностей. Механізм міграції має гарантувати, що всі записи у вихідному ресурсі переглядаються та переносяться, забезпечує цілісність даних і враховує всі залежності під час міграції записів. Запис можна перенести, лише якщо він незалежний або якщо записи, від яких він залежить, перенесено до цільового ресурсу. Тому основним правилом, що визначає порядок перегляду та передачі, є залежності між записами. До речі, вони визначаються зовнішніми ключами. Механізм міграції даних реалізовано у вигляді процедури, яка спочатку використовує виклики команд API, а потім рекурсивно викликає команди, пов'язані з типом, із ядра програми. Тому послідовність ходів між ними повинна починатися з елемента, що містить незалежний запис. Тоді основним порядком, визначеним для колекції таблиць у схемі бази даних, є зв'язок залежності між таблицями. Іноді ви можете ігнорувати цей зв'язок і просто вимкнути всі схеми перед міграцією та перебудувати їх після завершення всіх операцій з даними. Наприклад, виняток робиться, коли нова версія бази даних уже запущена. Виконайте сценарій, який змінює об'єкти в новій версії бази даних. Передача даних безпосередньо з необхідними перетвореннями - миттєво. Запуск сценаріїв для відновлення вимкнених індексів, додавання

перетворень тощо. Після завершення процесу міграції даних. Найпростіший варіант - створити проміжну програму. Він повинен підключитися до вихідної та цільової баз даних і виконати необхідні перетворення.

Далі – детально розглянемо загальну архітектуру програмного засобу для міграції і синхронізації даних між системами управління життєвим циклом програмного забезпечення (АЛМ) та системами управління вимогами (СРМ). Детально аналізується структура клієнт-серверної архітектури, основні компоненти та їх взаємодія.

4.1.1. Клієнт-Серверна Архітектура

Однією з ключових особливостей розробленого програмного засобу є його клієнт-серверна архітектура. Клієнт-серверна архітектура - це підход до створення програм, при якому функції програми розподілені між двома основними компонентами: клієнтом і сервером. У випадку додатка для міграції даних між системами управління життєвим циклом програмного забезпечення, така архітектура забезпечує чітку розділеність відповідальностей і покращує роботу додатку.

- **Серверна частина (Java Core):** Відповідає за обробку команд для міграції, виклик API-запитів до ресурсів АЛМ систем, та управління коннекторами. Забезпечує обробку важливих функцій, таких як мапування даних між системами, валідація та обробка помилок.
- **Клієнтська частина (Ext.js UI):** Графічний інтерфейс, що надає зручний інтерфейс користувачу для введення параметрів міграції, визначення прав доступу, та взаємодії з користувачем. Відображає стан міграцій та надає можливість користувачу вносити зміни та керувати процесом.

Доречність Клієнт-Серверної Архітектури**:**

- **Розділення Завдань:** Клієнт та сервер відповідають за різні аспекти роботи програми, що полегшує розробку, тестування та супровід коду.
- **Масштабованість:** Клієнт-серверна архітектура дозволяє легко масштабувати систему, додавати нові функції без великих змін, а також підтримувати багатокористувацькість.
- **Централізоване Управління:** Серверна частина може зберігати конфігураційні файли, виконувати моніторинг та забезпечувати централізоване управління міграціями.

Плюси та Мінуси**:**

Плюси:

- **Ефективність та Швидкість:** Клієнт-серверна архітектура дозволяє ефективно використовувати ресурси сервера та забезпечує швидкий обмін даними між клієнтом та сервером.
- **Легкість Супроводу:** Розділення функцій між клієнтом та сервером спрощує супровід коду та відладку.
- **Безпека:** Сервер може контролювати доступ до даних, забезпечуючи високий рівень безпеки.

Мінуси:

- **Залежність від Мережі:** Робота клієнтської частини може бути обмеженою або неможливою без доступу до сервера, що робить систему залежною від мережевого з'єднання.
- **Складність Розробки:** Потребує вивчення та розуміння роботи як клієнтської, так і серверної частин програми, що може бути складним для невеликих команд розробників.
- **Оновлення:** Оновлення системи може вимагати оновлення як клієнтської, так і серверної частин, що може бути складним у великих організаціях.

Клієнт-серверна архітектура є доречною для додатку міграції даних між АЛМ системами через свою здатність ефективно розділяти завдання, полегшувати супровід та забезпечувати масштабованість.

4.1.2. Ядро компонентів Java

Java Core в даному додатку є мотором, який відповідає за обробку основних команд для міграції даних між системами управління життєвим циклом програмного забезпечення (АЛМ). Він інтегрується з коннекторами для кожної підтримуваної АЛМ/СРМ системи, виконує мапування даних, ініціює виклики API-запитів та координує весь процес міграції.

Ключові Компоненти Java Core:

- **Менеджер Коннекторів:** Координує взаємодію з різними АЛМ/СРМ системами, забезпечуючи їх підтримку. Включає в себе інтерфейси для додавання нових коннекторів.
- **Модуль Мапування Даних:** Відповідає за визначення відповідностей між структурами даних у різних АЛМ системах, а також за конвертацію та трансформацію даних.
- **Обробник Команд:** Аналізує та виконує команди, отримані від графічного інтерфейсу, включаючи ініціювання процесу міграції, виклик API-запитів та контроль над ходом операцій.
- **Модуль Валідації та Обробки Помилок:** Забезпечує перевірку введених даних, правильність конфігурації та виведення зрозумілих повідомлень про помилки.

При використанні Java Core компонентів у додатку міграції даних між АЛМ системами виявляються раціональність та переваги наступних аспектів:

Java дозволяє досягти високого рівня портативності коду на різних платформах. Це робить впровадження та підтримку додатку більш простим

та ефективним, оскільки код може легко адаптуватися до різних операційних систем та середовищ.

Можливість паралельної обробки багатьох міграцій та запитів в Java Core забезпечує високу продуктивність та ефективність. Це дозволяє оптимізувати використання ресурсів та прискорює виконання завдань, особливо в умовах великого обсягу даних та інтенсивного використання системи.

Система спеціально розроблена з урахуванням можливості легкої додавання нових функцій та підтримки нових систем управління життєвим циклом програмного забезпечення (АЛМ). Модульна архітектура сприяє швидкій і безпечній інтеграції нового функціоналу та підтримки різних АЛМ систем без суттєвих змін в основному коді. Це робить систему гнучкою та готовою до швидкого розвитку відповідно до зростаючих потреб користувачів.

Плюси:

- **Інтеграція та Розширюваність:** Легка інтеграція з новими АЛМ системами та можливість додавання нових функцій через модульність.
- **Ефективність:** Java має високий рівень продуктивності та може ефективно обробляти великий обсяг даних.
- **Стандартизація:** Використання Java сприяє стандартизації коду та полегшує тестування та супровід.

Мінуси:

- **Можливий Overhead:** Мова Java може викликати невеликий overhead у порівнянні з деякими іншими мовами програмування.
- **Комплексність Розробки:** Розробка під Java може вимагати більше часу та зусиль у порівнянні з іншими мовами, особливо для невеликих проектів.

Java Core компоненти є ключовими для ефективної роботи додатку міграції даних між АЛМ системами. Їхні функції забезпечують необхідність для роботи з різними системами, мапування та валідацію даних, а також керування самим процесом міграції. Урахування раціональних властивостей та обмежень дозволяє створити потужний та гнучкий інструмент для міграції даних між АЛМ системами.

4.1.3. Графічний Інтерфейс (Ext.js)

Графічний інтерфейс (GUI) в розробці додатку для міграції даних між системами управління життєвим циклом програмного забезпечення (АЛМ) грає ключову роль у взаємодії з користувачем та забезпеченні зручності управління та контролю міграційним процесом.

Доречність Використання Ext.js:

- **Модерність та Естетика:** Ext.js, як розширення JavaScript, дозволяє створювати сучасний та естетичний інтерфейс, що сприяє зручному та приємному користувацькому досвіду.
- **Багатофункціональність:** Фреймворк Ext.js надає багатий набір компонентів та інструментів для швидкої і простої реалізації різноманітних елементів інтерфейсу, таких як таблиці, форми, графіки, дерева та інші.
- **Кросплатформенність:** Його кросплатформенність дозволяє розробникам створювати інтерфейс, який спрацьовує на різних пристроях та браузерах, що є важливим аспектом у сучасному розробницькому середовищі.

Плюси та Мінуси Використання Ext.js в Графічному Інтерфейсі:

Плюси:

- **Швидка Розробка:** Ext.js спрощує процес створення інтерфейсу, пропонуючи готові компоненти та інструменти, що робить розробку ефективною та швидкою.
- **Багатофункціональність:** Забезпечення різноманітних компонентів та можливостей, які полегшують реалізацію різних елементів інтерфейсу та взаємодію з користувачем.
- **Документація та Спільнота:** Ext.js має велику активну спільноту розробників та добре документований API, що полегшує розвиток та підтримку.

Мінуси:

- **Вагомий Розмір:** Розмір бібліотеки Ext.js може бути великим, що викликає певний overhead при завантаженні сторінки та може вплинути на швидкодію.
- **Вартість та Ліцензійна Політика:** Хоча Ext.js є безкоштовним для деяких використань, вона має комерційну ліцензію для більш розширених застосувань, що може бути фактором для певних проектів.
- **Залежність від JavaScript:** Якщо користувач вимикає JavaScript у своєму браузері, чи відключається через помилку, це може призвести до втрати функціональності інтерфейсу.

Висновок:

Використання Ext.js у графічному інтерфейсі додатку для міграції даних між АЛМ системами є раціональним вибором через його зручність, ефективність та можливості швидкої розробки. Однак, важливо враховувати його розмір, вартість та залежність від JavaScript при оцінці його придатності для конкретного проекту.

4.1.4. База Даних (Derby)

Для зберігання налаштувань, стану міграцій та інших важливих даних використовується вбудована база даних Apache Derby.

База даних є центральним елементом будь-якого додатку, оскільки саме тут зберігаються та організовуються всі дані, необхідні для міграції. У контексті проекту з міграції даних від АЛМ чи CPM системи, база даних виконує роль сховища для конфігураційних даних, інформації про міграції, а також результатів операцій.

Використання бази даних дозволяє зберігати дані у структурованому форматі, що робить їх легко доступними та оброблюваними. Це також дозволяє ефективно керувати версіями та історією міграцій, що є ключовими аспектами в управлінні життєвим циклом даних.

Derby - це вбудована, легка та потужна база даних, написана на мові Java. У контексті розробки додатку для міграції даних між системами управління життєвим циклом програмного забезпечення (АЛМ) чи системами управління відносинами з клієнтами (CPM), Derby може бути використана для зберігання конфігураційних даних, журналу подій, або результатів міграції.

Доречність Використання Derby:

- Вбудованість: Derby підтримує вбудовану архітектуру, що означає, що база даних може бути вбудована безпосередньо в додаток. Це спрощує розгортання та використання, зокрема для малих та середніх проектів.
- Легкість Використання: Derby надає простий SQL-інтерфейс та зручність взаємодії з мовою програмування Java, що сприяє легкості використання та розробці.

- Потужність та Ефективність: Незважаючи на свою легкість, Derby може обробляти значні обсяги даних та виконувати складні запити, що робить його потужним інструментом для зберігання та обробки інформації.

Плюси та Мінуси Використання Derby:

Плюси:

- Легкість Вбудовування: Здатність вбудовувати базу даних безпосередньо в додаток спрощує розгортання та управління залежностями.
- Легкість Розробки: Простий SQL-інтерфейс та підтримка Java роблять Derby привабливим для розробників, особливо для тих, хто вже використовує мову Java у своїх проектах.
- Вбудована Безкоштовність: Derby є безкоштовним програмним забезпеченням, що робить його доступним для використання без додаткових витрат.

Мінуси:

- Масштабування: Для дуже великих обсягів даних та високих навантажень може виникнути обмеження щодо масштабування в порівнянні з деякими іншими базами даних.
- Спрощена Функціональність: У порівнянні з деякими ентерпрайз-рішеннями, Derby може мати обмежену функціональність та інструменти для роботи з даними.
- Обмежена Спільнота та Ресурси: Незважаючи на активний спільноту, кількість ресурсів та інструментів для Derby може бути меншою порівняно з деякими більш популярними базами даних.

Використання Derby в додатку для міграції даних може виглядати наступним чином:

- Створення Таблиць: Визначення таблиць для зберігання конфігураційних даних, інформації про міграції та іншої сутності, пов'язаної з додатком.
- Використання SQL-Запитів: Використання SQL-запитів для вставки, вибірки та оновлення даних в базі даних.
- Обробка Помилок та Запитів: Включення механізму обробки помилок та управління транзакціями для забезпечення надійності та цілісності даних.

Висновок:

Використання Derby у додатку для міграції даних між АЛМ або СРМ системами є доцільним з урахуванням його легкості вбудовування, потужності та доступності

4.1.5. Взаємодія Компонентів

Програмний засіб для міграції і синхронізації даних між АЛМ/СРМ системами, що має клієнт-серверну архітектуру, складається з декількох основних компонентів, які взаємодіють між собою для забезпечення функціональності системи.

1) Графічна частина: Цей компонент, написаний на Ext.js, відповідає за візуалізацію даних і взаємодію з користувачем. Він дозволяє користувачам вводити параметри і валідувати доступ до кожної АЛМ/СРМ системи, що підтримується, а також налаштовувати з'єднання.

Рис. 4.1. Схема взаємодії користувача з графічною частиною

2) Core частина: Це серце системи, написане на Java. Вона обробляє основні команди для міграції даних. Вона взаємодіє з коннекторами для кожної АЛМ/СРМ системи, що підтримується, викликаючи API запити до ресурсів.

Рис. 4.2. Схема взаємодії користувача з графічною частиною і ядром програми

3) Коннектори: Ці компоненти, також написані на Java, відповідають за взаємодію з конкретними АЛМ/СРМ системами. Вони використовують API цих систем для доступу до даних і виконують запити, що надходять від core частини.

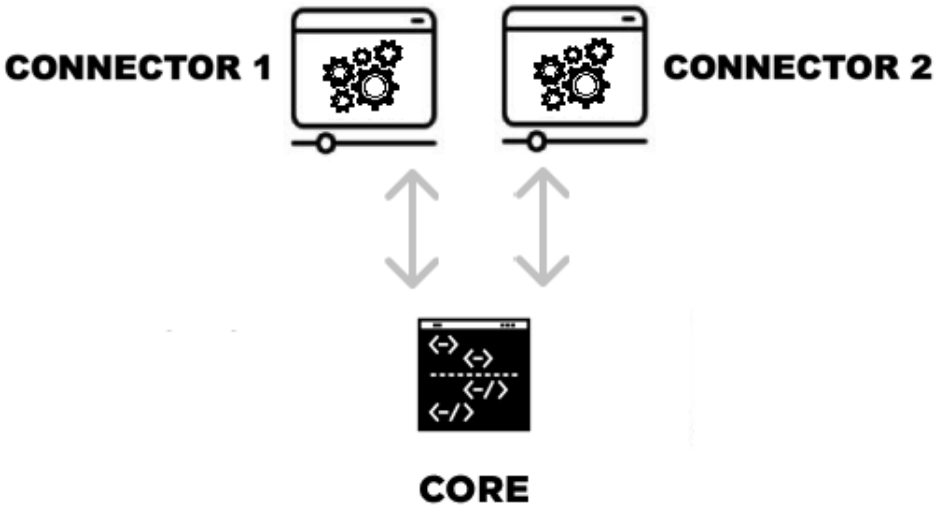

Рис. 4.3. Схема взаємодії ядра програми з коннекторами

4) База даних: Система використовує базу даних Derby для зберігання інформації про конфігурації, стан міграції і інші дані. Core частина взаємодіє з базою даних для зберігання і отримання даних.

Рис. 4.4. Схема взаємодії ядра із базою даних

4.1.6. Загальна Схема Роботи

Процес роботи додатку розпочинається з ініціалізації та завантаження ядра додатку. Графічний інтерфейс взаємодіє з ядром через визначений API, передаючи параметри для міграції та отримуючи результати операцій.

1. Користувач взаємодіє з графічним інтерфейсом, введенням параметрів міграції та налаштувань.

2. Графічний інтерфейс передає введені дані до ядра додатку.
3. Ядро обробляє команди міграції, викликаючи відповідні API запити до ресурсів АЛМ/СРМ систем.
4. Комунікація з коннекторами забезпечує передачу даних між додатком та системами.
5. Результати операцій обробляються ядром та повертаються до графічного інтерфейсу для відображення користувачеві.
6. Додаток зберігає конфігурації та інші дані в базі даних.

Така взаємодія між компонентами додатку забезпечує ефективну та надійну роботу системи міграції та синхронізації даних між різними системами управління життєвим циклом програмного забезпечення.

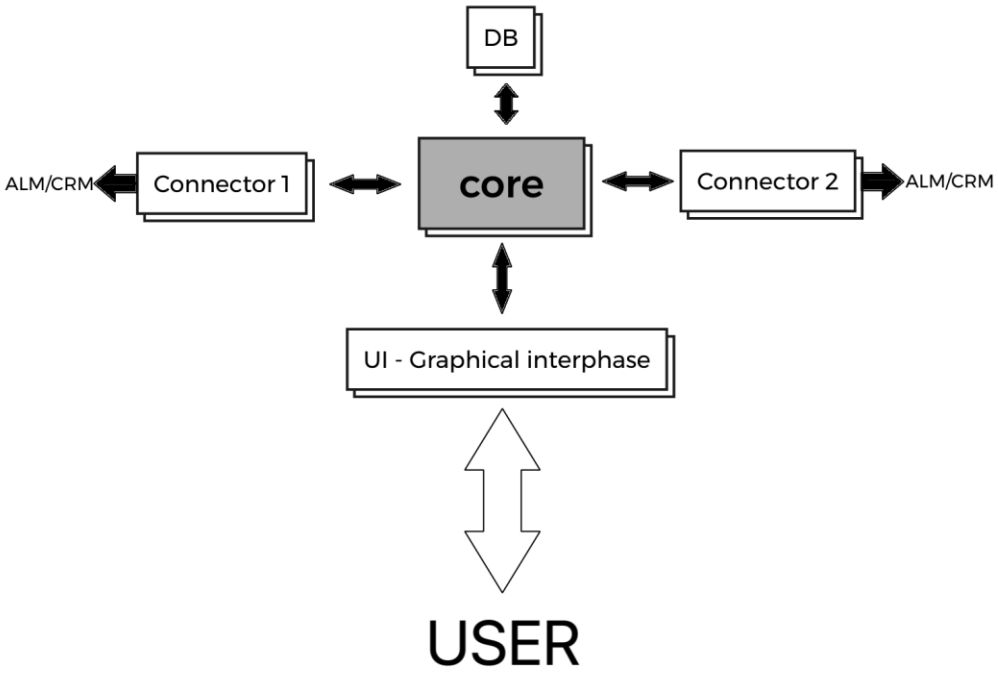

Рис. 4.5. Схема взаємодії всіх ключових елементів додатку

4.1.7. Проектування функціональних можливостей додатку

Ядро програми

Так як в основі архітектури проекту лежить шаблон проектування MVC в подальшій побудові проекту використовувався архітектурний стиль REST. REST — це стиль архітектури програмного забезпечення для розподілених систем. Зв'язок між сервером та клієнтом здійснюється повідомленнями по HTTP протоколу. Основини форматами передачі даних являється JSON та XML. Архітектура проекту була розділена на 4 шари, а саме:

1. Шар зв'язку з реляційною базою даних
2. Шар взаємодії між ресурсами
3. Шар бізнес логіки
4. Шар представлення інтерфейсу користувача

На рисунку 4.6 відображається зв’язок між чотирма шарами, архітектура була побудована так щоб користувач не зміг напряму вплинути на дані в реляційній базі даних

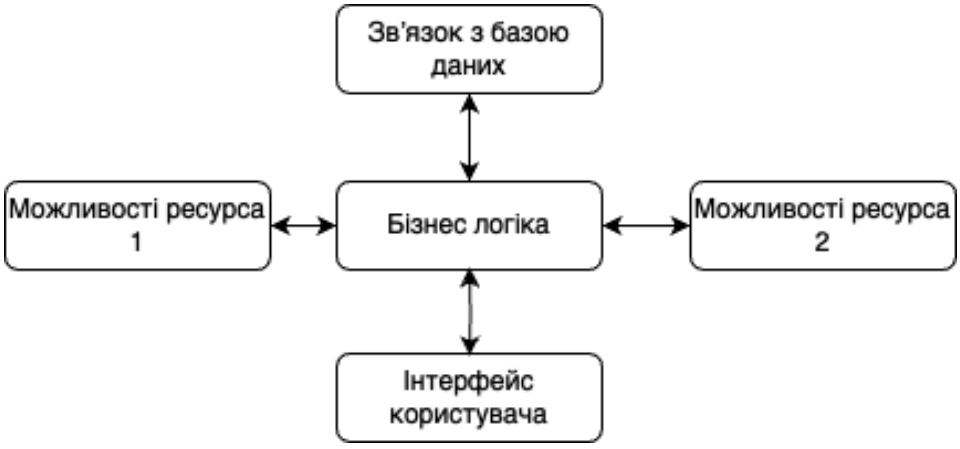

Рис. 4.6. Схема бізнес-логіки взаємодії компонентів

Під час роботи користувача з програмою основними об’єктами які оброблюються це формати даних XML або JSON, БД користувача та SQL-запит. В кінці ми отримаємо оновлений ресурс 2 в якому будуть перенесені дані з ресурсу 1. Рисунок 4.7.

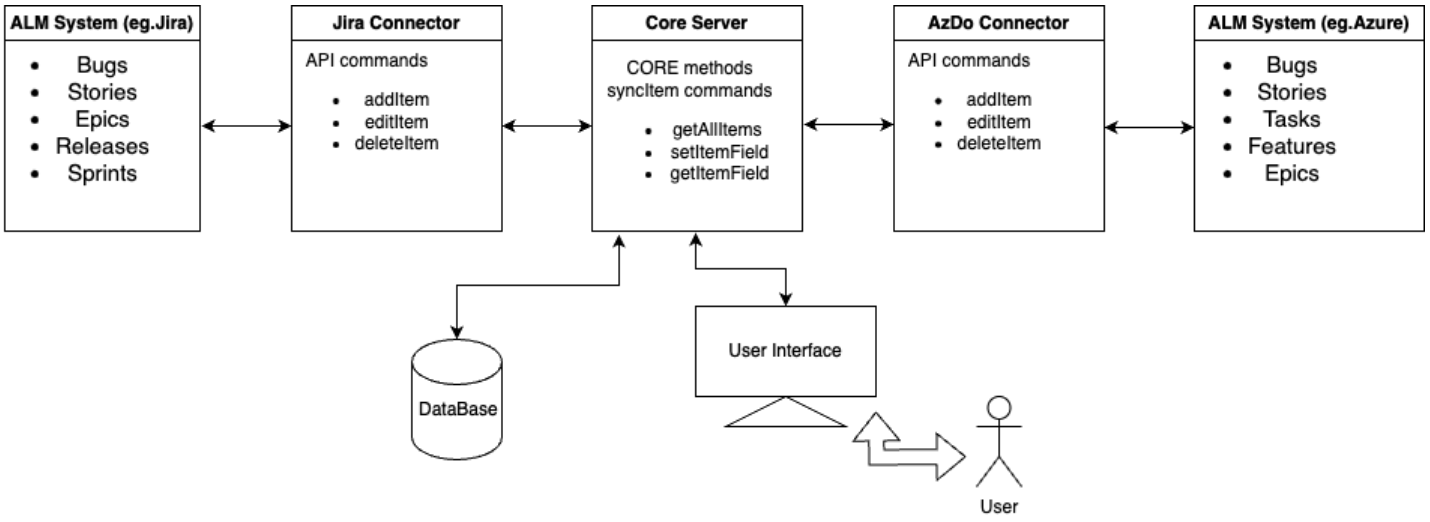

Рис. 4.7. Схема взаємодії основних модулів

4.2. Проектування функціональних можливостей додатку

4.2.1. Бізнес-логіка

Бізнес логіка в проекті представлена за допомогою чотирьох сервісів:

- RestClient
- UI
- RestService
- ConnectorUtil

Розгляньмо, кожен модуль окремо

RestClient – є ключовим модулем у core частині додатку, написаній на мові програмування Java. Цей модуль відповідає за взаємодію з системами управління життєвим циклом програмного забезпечення (ALM/CPM) шляхом використання HTTP-запитів до їхніх API. Розглянемо основні характеристики та функціональність модулю RestClient.

Основні Компоненти та Функції:

1. Клас RestClient:

Мета: Забезпечити зручний інтерфейс для виконання HTTP-запитів до API систем управління життєвим циклом програмного забезпечення.

Основні Методи:

- `sendGetRequest(String url)`: Відправка HTTP GET-запиту до вказаної URL-адреси.
- `sendPostRequest(String url, String payload)`: Відправка HTTP POST-запиту з вказаним тілом запиту (payload) до вказаної URL-адреси.
- `sendPutRequest(String url, String payload)`: Відправка HTTP PUT-запиту з вказаним тілом запиту (payload) до вказаної URL-адреси.
- `sendDeleteRequest(String url)`: Відправка HTTP DELETE-запиту до вказаної URL-адреси.

2. Обробка Відповідей:

Мета: Забезпечити обробку відповідей від сервера системи управління життєвим циклом програмного забезпечення.

Основні Функції:

- Розпакування та обробка JSON-відповідей.
- Обробка HTTP-статус кодів для визначення успішності або помилки запиту.

3. Конфігурація З'єднання:

Мета: Забезпечити можливість налаштування параметрів з'єднання, таких як таймаути, автентифікація тощо.

Основні Параметри:

- URL-адреса базового API.
- Таймаути для встановлення з'єднання та отримання відповіді.
- Дані автентифікації (якщо необхідно).

Переваги та Сценарії Використання:

a) Зручний Інтерфейс: RestClient надає зручний та легкий інтерфейс для використання в основній частині додатку. Розробники можуть швидко та просто взаємодіяти з API систем управління життєвим циклом програмного забезпечення.
b) Підтримка Різних Методів: RestClient підтримує різні HTTP-методи (GET, POST, PUT, DELETE), що робить його універсальним для різних типів запитів.
c) Обробка Помилок та Винятків: Модуль реалізує механізми обробки помилок та винятків, що дозволяє відстежувати та обробляти проблеми під час виконання запитів.
d) Можливість Конфігурації: RestClient дозволяє розробникам налаштовувати параметри з'єднання, щоб вони відповідали конкретним вимогам проекту.

Модуль RestClient є невід'ємною частиною ядра додатку, забезпечуючи ефективну взаємодію з системами управління життєвим циклом програмного забезпечення та покращуючи загальну функціональність системи.

UI – Модуль UI у core частині додатку відповідає за користувацький інтерфейс, який взаємодіє з користувачем. Даний модуль написаний з використанням Ext.js, що є потужною JavaScript-бібліотекою для створення веб-інтерфейсів. Розглянемо ключові характеристики та функціональність UI-модулю.

Основні Компоненти та Функції:

1. Головний Вікно (Main Window):

Опис: Головне вікно додатку, яке містить основний робочий простір та інші елементи інтерфейсу.

Функціонал:

- Забезпечує доступ до всіх ключових функцій додатку.
- Можливість відкривати різні розділи та вкладки.

2. Панель Інструментів (Toolbar):

Опис: Розміщена вгорі головного вікна, містить іконки та кнопки для швидкого доступу до функцій.

Функціонал:

- Забезпечує швидкий доступ до основних операцій, таких як міграція, синхронізація, налаштування тощо.

3. Панель Навігації (Navigation Panel):

Опис: Знаходиться з боку головного вікна та містить навігаційні елементи.

Функціонал:

- Надає структурований доступ до різних розділів та функціоналу додатку.

- 4. Робочий Простір (Workspace):

Опис: Це основна область для взаємодії з даними та виконання операцій.

Функціонал:

- Забезпечує відображення важливої інформації та результатів операцій.

5. Форми та Діалоги:

Опис: Використовуються для введення даних, налаштувань та відображення повідомлень.

Функціонал:

- Забезпечують взаємодію з користувачем для введення необхідної інформації чи виведення повідомлень.

6. Компоненти Данних (Data Components):

Опис: Відображають та редагують дані, наприклад, таблиці, дерева, графіки.

Функціонал:

- Дозволяють користувачам переглядати, редагувати та взаємодіяти з даними.

7. Валідація та Звіти:

Опис: Забезпечують валідацію введених даних та генерацію звітів про операції.

- Функціонал:
- Повідомлення про помилки та підтвердження успішних операцій.

Переваги та Сценарії Використання:

1. Зручний та Інтуїтивний Інтерфейс: UI-модуль розроблений з урахуванням принципів інтуїтивного дизайну, що робить користування додатком зручним та зрозумілим.
2. Многозадачність та Потужні Функції: Забезпечує можливість одночасно виконувати різні функції, такі як міграція, синхронізація, налаштування тощо.
3. Гнучка Налаштування та Розширення: Модуль дозволяє розробникам легко додавати нові функції та компоненти для вдосконалення функціональності.
4. Інтерактивність та Відгуки: Забезпечує інтерактивні елементи та відгуки користувачеві під час виконання операцій.
5. Адаптивний та Сучасний Дизайн: Модуль використовує адаптивний дизайн, що дозволяє йому коректно виглядати та працювати на різних пристроях та розширеннях екранів.

UI-модуль виконує важливу роль у взаємодії користувача з додатком, надаючи зручний та ефективний інтерфейс для виконання ключових операцій та взаємодії з даними.

RestService – є ключовим модулем у core частині додатку, написаний на мові програмування Java. Цей модуль відповідає за взаємодію зі сторонніми системами управління життєвим циклом програмного забезпечення (АЛМ/СРМ) шляхом використання HTTP-запитів до їхніх API. Розглянемо основні характеристики та функціональність модулю RestService.

Основні Компоненти та Функції:

1. Клас RestService:

Мета: Забезпечити інтерфейс для взаємодії з зовнішніми системами через HTTP-запити.

Основні Методи:

- `executeGetRequest(String url)`: Виконання HTTP GET-запиту до вказаної URL-адреси.
- `executePostRequest(String url, String payload)`: Виконання HTTP POST-запиту з вказаним тілом запиту (payload) до вказаної URL-адреси.
- `executePutRequest(String url, String payload)`: Виконання HTTP PUT-запиту з вказаним тілом запиту (payload) до вказаної URL-адреси.
- `executeDeleteRequest(String url)`: Виконання HTTP DELETE-запиту до вказаної URL-адреси.

2. Обробка Відповідей:

Мета: Забезпечити обробку відповідей від сервера системи управління життєвим циклом програмного забезпечення.

Основні Функції:

- Розпакування та обробка JSON-відповідей.
- Обробка HTTP-статус кодів для визначення успішності або помилки запиту.

3. Обробка Помилок та Винятків:

Мета: Забезпечити механізми обробки помилок та винятків, що можуть виникнути під час взаємодії з API.

Основні Параметри:

- Обробка винятків, пов'язаних з неправильними запитами, недоступністю сервера тощо.

4. Інтеграція з RestTemplate:

Мета: Використання бібліотеки RestTemplate для спрощення виконання HTTP-запитів та отримання відповідей.

- Основні Переваги:
- Уніфікований та зручний спосіб взаємодії з API.

5. Конфігурація З'єднання:

- Мета: Забезпечити можливість налаштування параметрів з'єднання, таких як таймаути, автентифікація тощо.

- Основні Параметри:
- URL-адреса базового API.
- Таймаути для встановлення з'єднання та отримання відповіді.
- Дані автентифікації (якщо необхідно).

Переваги та Сценарії Використання:

1. Зручний Інтерфейс для Взаємодії з API:- RestService надає простий та зручний інтерфейс для виконання HTTP-запитів та обробки відповідей API.

2. Обробка Помилок та Винятків:Модуль реалізує ефективні механізми обробки помилок, що дозволяє виявляти та вирішувати проблеми взаємодії з API.
3. Гнучка Налаштування: RestService дозволяє налаштовувати параметри з'єднання для відповідності конкретним вимогам додатку чи API.
4. Інтеграція з RestTemplate: Використання RestTemplate спрощує взаємодію з API та полегшує процес розробки.
5. Можливість Розширення: Модуль дозволяє розробникам розширювати та адаптувати його для взаємодії з різними API та сервісами.

Модуль RestService виконує важливу функцію взаємодії з API сторонніх систем, забезпечуючи ефективну інтеграцію та обробку даних у core частині додатку.

CONNECTORUTIL є важливим модулем у core частині додатку, написаний на мові програмування Java. Цей модуль відповідає за роботу з коннекторами для кожної системи управління життєвим циклом програмного забезпечення (АЛМ/СРМ). ConnectorUtil надає додатковий інтерфейс для взаємодії з різними АЛМ/СРМ системами, що дозволяє ефективно керувати міграцією та синхронізацією даних. Розглянемо основні характеристики та функціональність модулю ConnectorUtil.

Основні Компоненти та Функції:

1. Клас ConnectorUtil:

Мета: Забезпечити інтерфейс для взаємодії з різними коннекторами систем управління життєвим циклом програмного забезпечення.

Основні Методи:

- `connectToALMSystem(String almSystemURL, String username, String password)`: З'єднання з конкретною АЛМ/СРМ системою з використанням вказаних облікових даних.
- `migrateData(Connector connector, Data data)`: Виконання міграції даних з допомогою конкретного коннектора.
- `syncData(Connector connector, Data data)`: Виконання синхронізації даних з допомогою конкретного коннектора.
- `disconnectFromALMSystem(Connector connector)`: Роз'єднання від конкретної АЛМ/СРМ системи.

2. Робота з Коннекторами:

Мета: Забезпечити інтерфейс для роботи з конкретними коннекторами для кожної підтримуваної системи управління життєвим циклом програмного забезпечення.

Основні Функції:

- Завантаження та ініціалізація конкретного коннектора в залежності від системи.
- Передача даних конкретному коннектору для виконання міграції та синхронізації.

3. Додаткові Функції:

Мета: Надання додаткових функцій для зручності роботи з коннекторами та системами управління життєвим циклом програмного забезпечення.

Приклади Функцій:

- Отримання списку підтримуваних АЛМ/СРМ систем.
- Перевірка доступності конкретної АЛМ/СРМ системи перед підключенням.

Переваги та Сценарії Використання:

1. Універсальна Взаємодія з Різними Системами: ConnectorUtil дозволяє взаємодіяти з різними АЛМ/СРМ системами за допомогою єдиного інтерфейсу, що полегшує роботу з різними платформами.
2. Міграція та Синхронізація Даних: Надає функції для виконання міграції та синхронізації даних, що є основною метою додатку.
3. Гнучкість та Розширюваність: Модуль дозволяє додавати нові коннектори для підтримки додаткових АЛМ/СРМ систем.
4. Безпека та Облікові Записи: Забезпечує безпеку облікових записів, що використовуються для підключення до систем управління життєвим циклом програмного забезпечення.
5. Інтеграція з Core Частиною Додатку: Легко інтегрується з іншими модулями core частини додатку, забезпечуючи єдиний механізм взаємодії з коннекторами.

Модуль ConnectorUtil відіграє ключову роль у забезпеченні взаємодії з різними АЛМ/СРМ системами, що дозволяє додатку ефективно керувати міграцією та синхронізацією даних.

4.2.2. Структура БД та спосіб використання

Для зберігання конфігурацій, стану міграцій та інших даних, додаток використовує вбудовану базу даних Apache Derby. Це забезпечує стійкість та ефективність зберігання інформації про стан міграцій та конфігурації системи. Ось основні елементи схеми і таблиці

- LOGINUSER - Для забезпечення аутентифікації користувачів та контролю доступу до системи в базі даних Apache Derby була створена таблиця LOGINUSER. Ця таблиця містить інформацію про користувачів, які

мають доступ до програмного засобу для міграції і синхронізації даних між системами АЛМ/СРМ. Розглянемо структуру цієї таблиці.

Поля Таблиці:

- **USER_ID**: Унікальний ідентифікатор користувача, що автоматично генерується базою даних.
- **USERNAME**: Логін користувача, який використовується для входу в систему.
- **PASSWORD**: Захешований пароль користувача, що забезпечує безпеку при зберіганні.
- **ROLE**: Роль користувача в системі (наприклад, адміністратор, оператор тощо).

Опис Полів Таблиці

- **USER_ID**: Унікальний ідентифікатор генерується автоматично при додаванні нового користувача. Використовується як основний ключ таблиці.
- **USERNAME**: Логін користувача, який вводиться при вході в систему. Повинен бути унікальним.
- **PASSWORD**: Захешований пароль, який зберігається в базі даних. Забезпечує безпеку паролю користувача.
- **ROLE**: Роль користувача в системі. Визначає рівень доступу та права користувача в системі.

Ця таблиця використовується для аутентифікації користувачів при вході в систему. Поля USERNAME та PASSWORD порівнюються з введеними даними під час авторизації. ROLE визначає рівень доступу та можливості користувача в системі.

Ця таблиця є ключовою для забезпечення безпеки та ідентифікації користувачів, які мають доступ до функціональностей програмного засобу для міграції і синхронізації даних між системами АЛМ/СРМ.

- SYNCSET - в базі даних відповідає за зберігання інформації про синхронізаційні набори даних у системі. Ці набори визначають, які конкретні дані слід синхронізувати між різними системами управління життєвим циклом програмного забезпечення (АЛМ/СРМ). Розглянемо структуру цієї таблиці.

Поля Таблиці:

- **SET_ID**: Унікальний ідентифікатор синхронізаційного набору, що автоматично генерується базою даних.
- **SET_NAME**: Назва синхронізаційного набору для легкого ідентифікації та адміністрування.
- **SOURCE_SYSTEM**: Назва джерела (системи, з якої відбувається синхронізація).
- **TARGET_SYSTEM**: Назва цільової системи (системи, на яку відбувається синхронізація).
- **SYNCHRONIZATION_TYPE**: Тип синхронізації (наприклад, двостороння або одностороння).
- **LAST_SYNC_TIMESTAMP**: Дата та час останньої синхронізації для відстеження актуальності даних.

Опис Полів Таблиці:

- **SET_ID**: Унікальний ідентифікатор генерується автоматично при додаванні нового синхронізаційного набору. Використовується як основний ключ таблиці.

- **SET_NAME**: Назва синхронізаційного набору для легкого визначення його призначення.
- **SOURCE_SYSTEM**: Назва системи, з якої відбувається синхронізація даних.
- **TARGET_SYSTEM**: Назва системи, на яку відбувається синхронізація даних.
- **SYNCHRONIZATION_TYPE**: Визначає тип синхронізації, такий як двостороння або одностороння.
- **LAST_SYNC_TIMESTAMP**: Зберігає дату та час останньої синхронізації для відстеження актуальності даних.

Застосування Таблиці SYNCSET:

Ця таблиця дозволяє адміністраторам визначати різні синхронізаційні набори, контролювати напрямки синхронізації між різними системами, та відстежувати час останньої синхронізації. Вона є ключовим елементом для ефективного управління синхронізацією даних в додатку для міграції і синхронізації між АЛМ/CPM системами.

- SYNCHRONIZATION - призначена для зберігання інформації про кожну окрему синхронізаційну операцію між різними ресурсами.

Поля Таблиці:

- **SYNC_ID**: Унікальний ідентифікатор синхронізаційної операції, автоматично генерується базою даних.
- **SET_ID**: Зовнішній ключ, який посилається на ідентифікатор синхронізаційного набору в таблиці **SYNCSET.**
- **SOURCE_RESOURCE**: Назва ресурсу, з якого проводиться синхронізація.

- **TARGET_RESOURCE**: Назва ресурсу, на який проводиться синхронізація.
- **SYNC_START_TIMESTAMP**: Дата та час початку синхронізації.
- **SYNC_END_TIMESTAMP**: Дата та час завершення синхронізації (може бути NULL, якщо синхронізація ще не завершена).
- **STATUS**: Статус синхронізації (наприклад, "Успішно", "Помилка", "В процесі").
- **ERROR_MESSAGE**: Повідомлення про помилку (якщо є) під час синхронізації.

Опис Полів Таблиці:

- **SYNC_ID**: Унікальний ідентифікатор генерується автоматично при кожній новій синхронізації.
- **SET_ID**: Зовнішній ключ, який посилається на ідентифікатор синхронізаційного набору в таблиці **SYNCSET.**
- **SOURCE_RESOURCE**: Вказує на ресурс, з якого беруться дані для синхронізації.
- **TARGET_RESOURCE**: Вказує на ресурс, на який передаються дані під час синхронізації.
- **SYNC_START_TIMESTAMP**: Зберігає дату та час початку синхронізаційної операції.
- **SYNC_END_TIMESTAMP**: Зберігає дату та час завершення синхронізаційної операції.
- **STATUS**: Визначає статус синхронізації, чи була вона успішною, чи виникла помилка.
- **ERROR_MESSAGE**: Якщо статус синхронізації є "Помилка", то це поле може містити повідомлення про помилку.

Застосування Таблиці SYNCHRONIZATION:

Ця таблиця використовується для відстеження історії синхронізаційних операцій між різними ресурсами. Дані, збережені в цій таблиці, дозволяють адміністраторам та користувачам відстежувати прогрес синхронізації, а також виявляти та вирішувати проблеми, які можуть виникнути під час цього процесу.

- PROJECTMAP – дана таблиця відповідає за зберігання інформації про маппінг воркспейсів, які слугують контейнерами для елементів ресурсів у системі управління життєвим циклом програмного забезпечення (ALM/CPM). Ця таблиця дозволяє визначити структуру та взаємозв'язки між різними воркспейсами та їхніми елементами. Розглянемо структуру цієї таблиці.

Поля Таблиці:

- **WORKSPACE_ID**: Унікальний ідентифікатор воркспейсу, який автоматично генерується базою даних.
- **PARENT_WORKSPACE_ID:** Ідентифікатор батьківського воркспейсу, до якого належить даний воркспейс (може бути NULL для кореневого воркспейсу).
- **WORKSPACE_NAME**: Назва воркспейсу, яка ідентифікує його в системі.
- **RESOURCE_TYPE**: Тип ресурсу, який зберігається в даному воркспейсі (наприклад, "Документ", "Код", "Тест-кейс" і т. д.).
- **RESOURCE_ID**: Унікальний ідентифікатор конкретного ресурсу в межах даного воркспейсу.

Опис Полів Таблиці:

- **WORKSPACE_ID**: Унікальний ідентифікатор, який призначається кожному воркспейсу для однозначної ідентифікації.

- **PARENT_WORKSPACE_ID**: Вказує на ідентифікатор батьківського воркспейсу, до якого належить даний воркспейс. Для кореневого воркспейсу значення може бути NULL.
- **WORKSPACE_NAME**: Назва воркспейсу, яка використовується для зручності ідентифікації.
- **RESOURCE_TYPE**: Визначає тип ресурсу, який зберігається в даному воркспейсі.
- **RESOURCE_ID**: Унікальний ідентифікатор конкретного ресурсу в межах даного воркспейсу.

Застосування Таблиці PROJECTMAP:

Ця таблиця використовується для створення ієрархії воркспейсів та їхніх елементів у системі управління життєвим циклом програмного забезпечення. Зберігання інформації про батьківські та дочірні воркспейси, а також типи та ідентифікатори ресурсів у кожному воркспейсі, дозволяє зручно навігувати та взаємодіяти з різними елементами системи.

- CALCULATEDVALUE - призначена для зберігання інформації про значення, які підлягають обчисленню та зміні в процесі синхронізації між системами управління життєвим циклом програмного забезпечення (АЛМ/СРМ). Ця таблиця дозволяє визначати певні налаштування, які використовуються для автоматичного обчислення значень в ході синхронізації. Розглянемо структуру цієї таблиці.

Поля Таблиці:

- **CALCULATION_ID**: Унікальний ідентифікатор для кожного обчисленого значення, що автоматично генерується базою даних.
- **SET_ID**: Зовнішній ключ, який посилається на ідентифікатор синхронізаційного набору в таблиці **SYNCSET**.

- **RESOURCE_TYPE**: Тип ресурсу, до якого відноситься обчислене значення (наприклад, "Документ", "Код", "Тест-кейс" і т. д.).
- **RESOURCE_ID**: Унікальний ідентифікатор конкретного ресурсу, до якого відноситься обчислене значення.
- **PROPERTY_NAME**: Назва властивості (поля), до якого вноситься зміна в результаті обчислення.
- **CALCULATION_FORMULA**: Формула обчислення значення властивості для даного ресурсу.
- **LAST_CALCULATION_TIMESTAMP**: Дата та час останнього обчислення значення.

Опис Полів Таблиці:

- **CALCULATION_ID**: Унікальний ідентифікатор, який призначається кожному обчисленому значенню для однозначної ідентифікації.
- **SET_ID**: Зовнішній ключ, який посилається на ідентифікатор синхронізаційного набору в таблиці **SYNCSET.**
- **RESOURCE_TYPE**: Вказує на тип ресурсу, для якого визначається обчислене значення.
- **RESOURCE_ID**: Унікальний ідентифікатор конкретного ресурсу, до якого відноситься обчислене значення.
- **PROPERTY_NAME**: Назва властивості (поля) ресурсу, до якого вноситься зміна в результаті обчислення.
- **CALCULATION_FORMULA**: Формула обчислення, яка визначає, яким чином змінюється властивість.
- **LAST_CALCULATION_TIMESTAMP**: Зберігає дату та час останнього обчислення значення для відстеження актуальності.

Застосування Таблиці CALCULATEDVALUE:

Ця таблиця використовується для налаштування та зберігання обчислених значень, які можуть змінюватися в процесі синхронізації між різними ресурсами систем управління життєвим циклом програмного забезпечення. Такий підхід дозволяє автоматизувати та керувати певними аспектами даних під час синхронізації.

DATASOURCE - відповідає за зберігання інформації про джерела даних, що використовуються для аутентифікації з ресурсами, вибору користувача, дефолтного проекту та інших налаштувань в системі управління життєвим циклом програмного забезпечення (ALM/CPM). Ця таблиця дозволяє зберігати параметри для підключення до різних джерел даних та налаштувань для кожного користувача. Розглянемо структуру цієї таблиці.

Поля Таблиці:

- **DATA_SOURCE_ID**: Унікальний ідентифікатор джерела даних, автоматично генерується базою даних.
- **USER_ID**: Зовнішній ключ, який посилається на ідентифікатор користувача в таблиці **LOGINUSER.**
- **SOURCE_NAME**: Назва джерела даних для ідентифікації.
- **CONNECTION_STRING**: Рядок підключення до джерела даних.
- **USERNAME**: Логін користувача для аутентифікації на джерелі даних.
- **PASSWORD**: Пароль користувача для аутентифікації на джерелі даних (захешований для безпеки).
- **DEFAULT_PROJECT**: Назва дефолтного проекту для користувача на даному джерелі даних.
- **OTHER_SETTINGS**: Додаткові налаштування, які можуть бути збережені у форматі JSON.

- **LAST_AUTH_TIMESTAMP**: Дата та час останньої успішної аутентифікації користувача.

Опис Полів Таблиці:

- **DATA_SOURCE_ID**: Унікальний ідентифікатор для кожного джерела даних для однозначної ідентифікації.
- **USER_ID**: Зовнішній ключ, який посилається на ідентифікатор користувача в таблиці **LOGINUSER.**
- **SOURCE_NAME**: Назва джерела даних для зручності ідентифікації.
- **CONNECTION_STRING**: Рядок підключення, який визначає, як з'єднатися з джерелом даних.
- **USERNAME**: Логін користувача для аутентифікації на джерелі даних.
- **PASSWORD**: Захешований пароль користувача для безпеки.
- **DEFAULT_PROJECT**: Назва дефолтного проекту, який вибирається за замовчуванням при підключенні.
- **OTHER_SETTINGS**: Додаткові налаштування у форматі JSON, які можуть змінюватися в залежності від потреб системи.
- **LAST_AUTH_TIMESTAMP**: Дата та час останньої успішної аутентифікації користувача для відстеження актуальності.

Застосування Таблиці DATASOURCE: Ця таблиця використовується для зберігання параметрів підключення до джерел даних, аутентифікації користувачів

- SYNCRITERIA - відповідає за зберігання інформації про критерії синхронізації, такі як передумови та фільтри, для вибору елементів із ресурсів у системі управління життєвим циклом програмного забезпечення

(АЛМ/CPM). Ця таблиця дозволяє визначати умови, за якими обираються елементи для синхронізації між різними ресурсами. Розглянемо структуру цієї таблиці.

Поля Таблиці:

- **CRITERIA_ID**: Унікальний ідентифікатор критерію синхронізації, автоматично генерується базою даних.
- **SET_ID**: Зовнішній ключ, який посилається на ідентифікатор синхронізаційного набору в таблиці **SYNCSET.**
- **RESOURCE_TYPE**: Тип ресурсу, для якого визначається критерій синхронізації (наприклад, "Документ", "Код", "Тест-кейс" і т. д.).
- **CONDITION_NAME**: Назва критерію або умови для ідентифікації.
- **CONDITION_EXPRESSION**: Вираз, що визначає умови для вибору елементів для синхронізації.

Опис Полів Таблиці:

- **CRITERIA_ID**: Унікальний ідентифікатор для кожного критерію синхронізації для однозначної ідентифікації.
- **SET_ID**: Зовнішній ключ, який посилається на ідентифікатор синхронізаційного набору в таблиці **SYNCSET.**
- **RESOURCE_TYPE**: Тип ресурсу, для якого визначається критерій синхронізації.
- **CONDITION_NAME**: Назва критерію або умови для ідентифікації.
- **CONDITION_EXPRESSION**: Вираз, що визначає умови для вибору елементів для синхронізації.

Застосування Таблиці SYNCRITERIA:

Ця таблиця використовується для визначення умов і фільтрів, за якими вибираються елементи під час синхронізації між різними ресурсами. Умови

можуть включати, наприклад, обмеження за статусом елементів, датою останньої модифікації, або інші параметри, які визначають, які елементи підлягають синхронізації. Це дозволяє налаштовувати поведінку синхронізації відповідно до потреб користувача та вимог системи управління життєвим циклом програмного забезпечення.

ITERATION – призначена для зберігання інформації про окремий ітераційний цикл синхронізації між різними ресурсами в системі управління життєвим циклом програмного забезпечення (АЛМ/СРМ). Ця таблиця дозволяє відстежувати та керувати окремими ітераціями процесу синхронізації. Розглянемо структуру цієї таблиці.

Поля Таблиці:

- **ITERATION_ID**: Унікальний ідентифікатор для кожної ітерації синхронізації, автоматично генерується базою даних.
- **SET_ID**: Зовнішній ключ, який посилається на ідентифікатор синхронізаційного набору в таблиці **SYNCSET.**
- **START_TIMESTAMP**: Дата та час початку ітерації синхронізації.
- **END_TIMESTAMP**: Дата та час завершення ітерації синхронізації (може бути NULL, якщо ітерація ще не завершена).
- **STATUS**: Статус ітерації синхронізації (наприклад, "Успішно", "Помилка", "В процесі").
- **ERROR_MESSAGE**: Повідомлення про помилку (якщо є) під час ітерації синхронізації.
- **SYNCHRONIZED_ITEMS_COUNT**: Кількість синхронізованих елементів під час ітерації.
- **ITERATION_PARAMETERS**: Додаткові параметри ітерації у форматі JSON.

Опис Полів Таблиці:

- **ITERATION_ID**: Унікальний ідентифікатор, який призначається кожній ітерації для однозначної ідентифікації.
- **SET_ID**: Зовнішній ключ, який посилається на ідентифікатор синхронізаційного набору в таблиці **SYNCSET.**
- **START_TIMESTAMP**: Зберігає дату та час початку ітерації синхронізації.
- **END_TIMESTAMP**: Зберігає дату та час завершення ітерації синхронізації (може бути NULL, якщо ітерація ще не завершена).
- **STATUS**: Визначає статус ітерації синхронізації, чи була вона успішною, чи виникла помилка.
- **ERROR_MESSAGE**: Якщо статус ітерації є "Помилка", то це поле може містити повідомлення про помилку.
- **SYNCHRONIZED_ITEMS_COUNT**: Кількість синхронізованих елементів під час даної ітерації.
- **ITERATION_PARAMETERS**: Додаткові параметри ітерації, які можуть змінюватися в залежності від потреб системи.

Застосування Таблиці ITERATION:

Ця таблиця використовується для відстеження та керування окремими ітераціями процесу синхронізації. Інформація в цій таблиці дозволяє системі взаємодіяти з окремими етапами синхронізації, визначати їх статус, кількість синхронізованих елементів та інші параметри, необхідні для ефективного управління синхронізацією.

DATAPOOLACTION - призначена для зберігання інформації про набір всіх елементів, їхніх полів та значень, які попадають в конкретну ітерацію синхронізації між різними ресурсами в системі управління життєвим циклом

програмного забезпечення (АЛМ/СРМ). Ця таблиця важлива для відстеження змін та оновлень даних під час синхронізації. Розглянемо структуру цієї таблиці.

Поля Таблиці:

- **ACTION_ID**: Унікальний ідентифікатор для кожної дії в наборі даних, автоматично генерується базою даних.
- **ITERATION_ID**: Зовнішній ключ, який посилається на ідентифікатор ітерації в таблиці **ITERATION.**
- **RESOURCE_TYPE**: Тип ресурсу, до якого відноситься дія (наприклад, "Документ", "Код", "Тест-кейс" і т. д.).
- **RESOURCE_ID**: Унікальний ідентифікатор конкретного ресурсу, до якого відноситься дія.
- **FIELD_NAME**: Назва поля (філда) ресурсу, до якого вноситься зміна.
- **OLD_VALUE**: Старе значення поля до синхронізації.
- **NEW_VALUE**: Нове значення поля після синхронізації.

Опис Полів Таблиці:

- **ACTION_ID**: Унікальний ідентифікатор, який призначається кожній дії для однозначної ідентифікації.
- **ITERATION_ID**: Зовнішній ключ, який посилається на ідентифікатор ітерації в таблиці **ITERATION.**
- **RESOURCE_TYPE**: Тип ресурсу, до якого відноситься дія.
- **RESOURCE_ID**: Унікальний ідентифікатор конкретного ресурсу, до якого відноситься дія.
- **FIELD_NAME**: Назва поля (філда) ресурсу, до якого вноситься зміна.
- **OLD_VALUE**: Значення поля до синхронізації.

- **NEW_VALUE**: Значення поля після синхронізації.

Застосування Таблиці DATAPOOLACTION:

Ця таблиця використовується для відстеження змін та оновлень, які сталися в ході синхронізації між різними ресурсами. Інформація в цій таблиці дозволяє аналізувати, які поля були змінені, які значення вони мали до та після синхронізації, а також в який саме ітераційний цикл це відбулося.

Розглядаючи структуру бази даних для програмного засобу міграції та синхронізації даних між системами управління життєвим циклом програмного забезпечення (ALM/CPM) з клієнт-серверною архітектурою, ми визначили і розробили кілька ключових таблиць. Кожна таблиця відповідає за конкретний аспект функціональності програмного засобу, і всі вони взаємодіють для ефективної роботи системи. Давайте зробимо загальний висновок з усього наведеного вище:

1. **Таблиця LOGINUSER**: Забезпечує зберігання інформації про користувачів системи, їхні логіни та інші автентифікаційні дані.
2. **Таблиця SYNCSET**: Визначає синхронізаційні набори, які містять інформацію про конкретні групи даних для синхронізації.
3. **Таблиця SYNCRITERIA**: Відповідає за зберігання критеріїв синхронізації для визначення, які елементи обирати під час синхронізації.
4. **Таблиця ITERATION**: Зберігає інформацію про окремі ітерації синхронізації, включаючи статус, кількість синхронізованих елементів та інші параметри.
5. **Таблиця DATAPOOLACTION**: Фіксує зміни та оновлення в конкретних елементах та їх полях під час ітерацій синхронізації.

Кожна з цих таблиць грає важливу роль у визначенні, виборі та зберіганні даних під час процесу міграції та синхронізації. Загальна архітектура бази даних підтримує клієнт-серверну взаємодію, а ретельно продумана структу операцій та зручність роботи з системою. Важливо відзначити деякі ключові аспекти бази даних, що були визначені вище:

- **Зв'язки між таблицями**: Зовнішні ключі встановлені в таблицях **SYNCSET**, **SYNCRITERIA**, **ITERATION**, **DATAPOOLACTION** забезпечують ефективну організацію зв'язків між записами різних таблиць, що визначає логічну структуру бази даних.
- **JSON-поля**: Використання JSON-полів в таблицях, таких як **ITERATION_PARAMETERS** та **OTHER_SETTINGS**, надає гнучкість для зберігання додаткових параметрів чи налаштувань, які можуть змінюватися в залежності від потреб системи.
- **Ідентифікатори та Статуси**: У кожній таблиці передбачено унікальний ідентифікатор (наприклад, **ITERATION_ID**, **ACTION_ID**), а також поле для визначення статусу (наприклад, **STATUS**), що сприяє внутрішній логіці системи та відстеженню стану операцій.
- **Інтеграція з ітераційним процесом**: Таблиця **ITERATION** є ключовою для відстеження кожної ітерації синхронізації, надаючи інформацію про час початку та завершення, статус і кількість синхронізованих елементів.
- **Історія Змін в DATAPOOLACTION**: Збереження історії змін у таблиці **DATAPOOLACTION** надає можливість відстежувати еволюцію даних під час синхронізації.

Узагальнюючи, розроблена база даних відповідає вимогам проекту, надаючи ефективну структуру для взаємодії з різними аспектами міграції та синхронізації даних. Правильна організація таблиць дозволяє ефективно

виконувати операції зберігання, оновлення та вибору даних, що є критичним для успішного функціонування системи управління життєвим циклом програмного забезпечення. Повну структуру бази даних ми можемо побачити на рис.4.1.

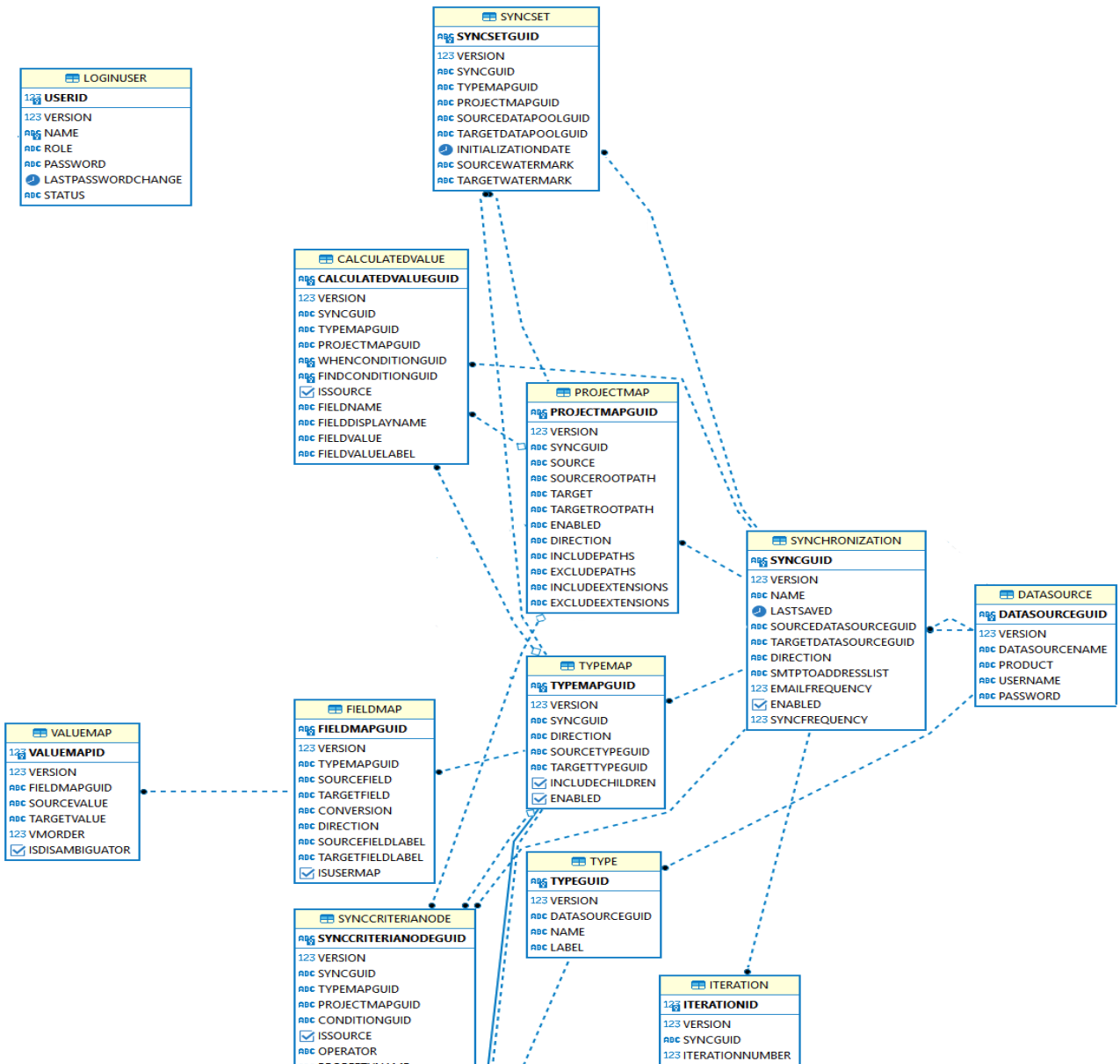

Рис. 4.1. Структура бази даних

4.3. Проектування програмного додатку

У цьому розділі ми розглянемо процес проєктування програмного додатку для розробки, міграції та синхронізації даних між системами управління життєвим циклом програмного забезпечення (АЛМ/СРМ). Наш додаток буде заснований на клієнт-серверній архітектурі і написаний на Java та Ext.js. Основний функціонал буде включати обробку команд для міграції, коннектори для різних АЛМ/СРМ систем, графічний інтерфейс для управління параметрами та валідацією доступів.

1. Системні Вимоги

Перед тим як перейти до деталей проєктування, визначимо системні вимоги для нашого додатку:

- Java Development Kit (JDK) 11 або вище.
- Apache Derby для локальної бази даних.
- Gradle для збірки та управління залежностями.
- Ext.js для графічного інтерфейсу.

2. Структура Проекту

Один із ключових аспектів проєктування - правильна структура проекту. Для нашого додатку ми будемо використовувати систему збірки Gradle, яка спростить управління залежностями та буде легко адаптовуватись до різних потреб.

- Файл Збірки Gradle

Файл збірки **build.gradle** буде містити конфігурації для нашого проекту, включаючи залежності, завдання збірки та інші налаштування.

gradle

```
plugins {
    id 'java'
}

repositories {
    jcenter()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter'
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.apache.derby:derby'
    // Додаткові залежності...
}

application {
    mainClassName = 'com.example.Application'
```

```
}
```

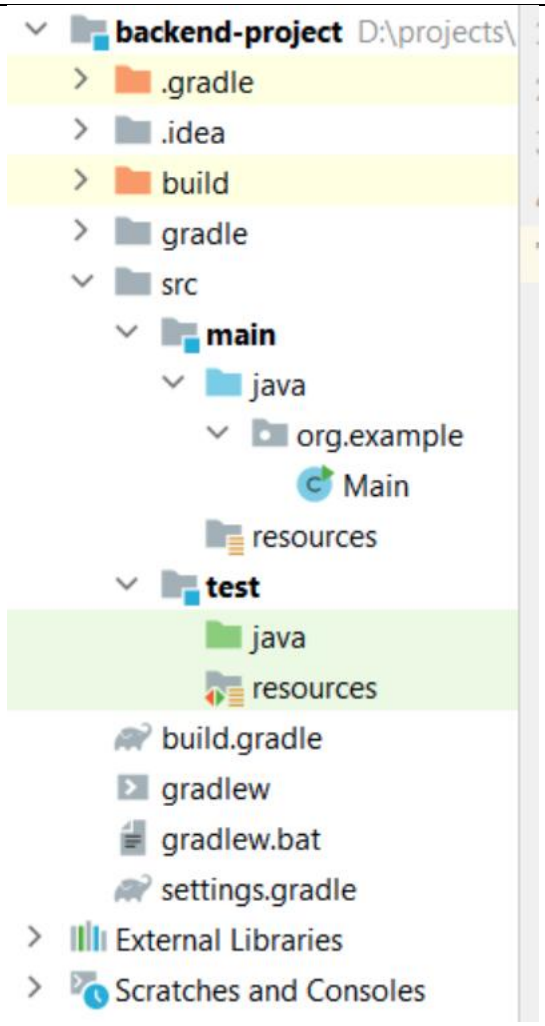

Рис.4.2. Структура каталогів Gradle

- **Модулі та Компоненти**

Наш додаток буде розділений на різні модулі та компоненти для покращення читабельності та підтримки високої зручності розширення.

- **Модуль Core -** Модуль **core** буде включати основний функціонал додатку, такий як обробка команд для міграції, коннектори та робота з базою даних.

- **Модуль GUI -** Модуль **gui** буде відповідати за графічний інтерфейс додатку та взаємодію з користувачем. Використовуючи Ext.js, ми створимо зручний та ефективний інтерфейс для управління налаштуваннями.

У цьому розділі ми визначили загальну структуру та вимоги для нашого проекту, а також розглянули основні модулі та компоненти. Наступні розділи будуть детально розглядати кожен модуль та компонент, розробляючи їх архітектуру та функціональність.

Реалізація UI Модулю в IntelliJ IDEA

У цьому розділі ми розглянемо реалізацію UI Модулю нашого додатку в середовищі розробки IntelliJ IDEA. Ми використаємо мову програмування JavaScript та фреймворк Ext.js для створення графічного інтерфейсу.

Початкова Конфігурація

Перш ніж почати, переконайтеся, що ви встановили Ext.js та налаштували його у своєму проекті. Використаймо Sencha Cmd для створення початкової структури та налаштувань.

Sencha Ext JS - це високопродуктивний JavaScript фреймворк для розробки багатоплатформових веб-додатків з великою кількістю компонентів графічного інтерфейсу (GUI). Основною метою Sencha Ext JS є надання розробникам інструментів для швидкої та ефективної розробки складних веб-додатків з багатим функціоналом.

Основні особливості та компоненти Sencha Ext JS:

4. Компоненти інтерфейсу:

- Grids (таблиці): Великий набір функціональних таблиць для відображення та редагування даних.

- Forms (форми): Компоненти для створення введення та редагування даних.
- Trees (дерева): Компоненти для створення інтерактивних дерев структури даних.

5. MVC Архітектура: Використання моделі-вид-контролера для ефективного управління логікою програми та відображенням.
6. Data Packag - Забезпечує високорівневий API для взаємодії з даними, такий як набори даних, моделі, проксі, стори та інше.
7. Розширені Можливості Інтерфейсу - Анімації, перетягування та оптимізовані можливості взаємодії з користувачем.
8. Темизація та Стилізація: Легко змінювати вигляд додатків за допомогою CSS-стилів та вбудованого підтримки Sass.
9. Підтримка Мобільних Платформ: Sencha Ext JS надає можливості адаптації для створення мобільних додатків з використанням Sencha Touch.
10. Підтримка Плагінів та Розширень: Великий екосистема розширень та плагінів для розширення функціоналу.
11. Широкий Вибір Вбудованих Інструментів включає в себе інструменти для розробки, тестування та оптимізації коду.
12. Підтримка AJAX та Запитів на Сервер: Можливості взаємодії з сервером за допомогою AJAX-запитів.
13. Сумісність та Підтримка Браузерів: Працює в багатьох сучасних браузерах та має ефективні засоби роботи з декількома платформами.

Sencha Ext JS є комерційним продуктом, але також надається пробний період для ознайомлення. Він широко використовується для створення

корпоративних та бізнес-додатків з великим обсягом даних та складною логікою.

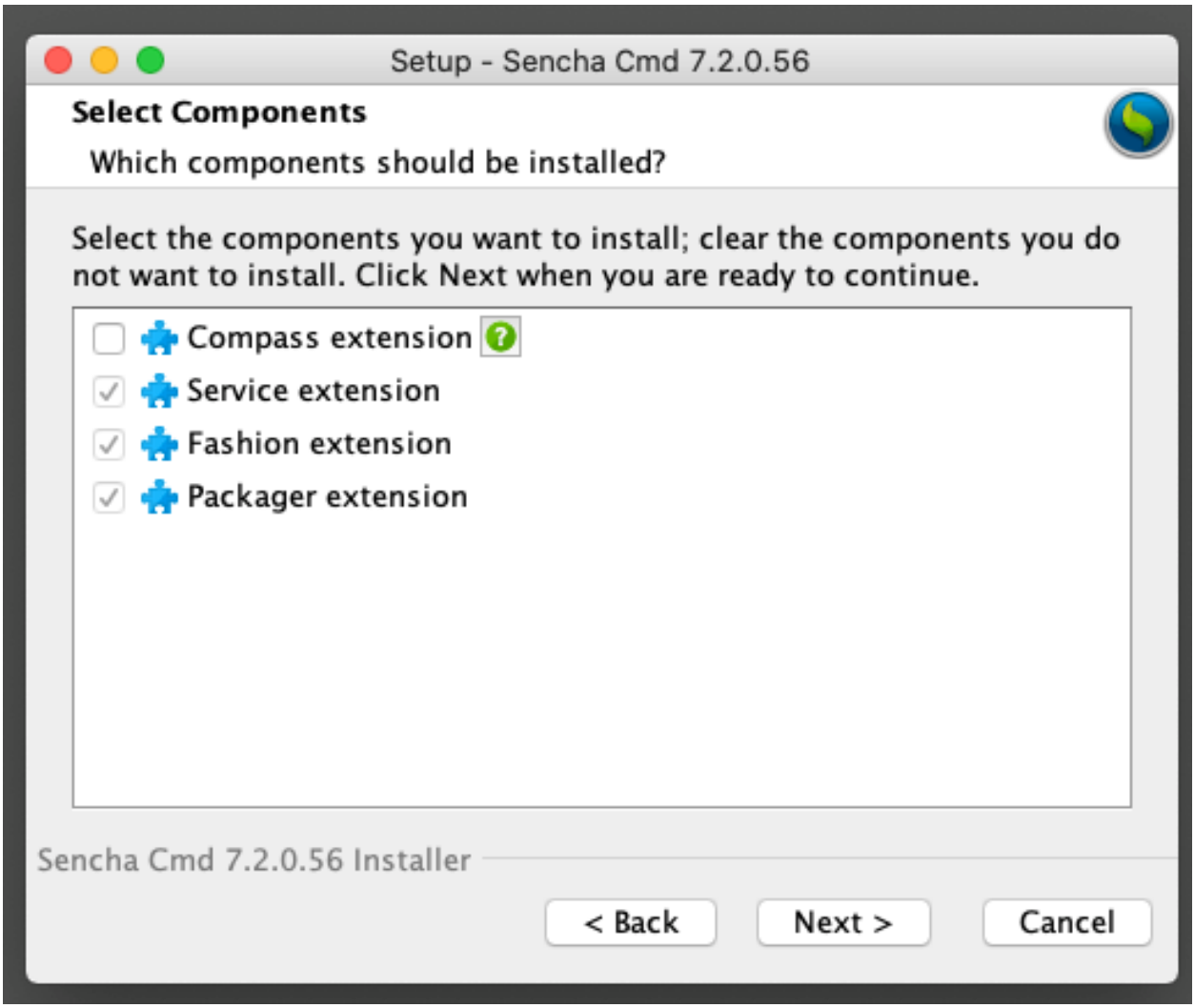

Рис.4.3. Інсталяція Sencha app.

Реалізація UI Модулю в IntelliJ IDEA

У цьому розділі ми розглянемо реалізацію UI Модулю нашого додатку в середовищі розробки IntelliJ IDEA. Ми використаємо мову програмування JavaScript та фреймворк Ext.js для створення графічного інтерфейсу.

Початкова Конфігурація

Перш ніж почати, переконайтеся, що ви встановили Ext.js та налаштували його у своєму проекті. Використаймо [Sencha Cmd](https://www.sencha.com/products/extjs/cmd-download/) для створення початкової структури та налаштувань.

Створення Головного Вікна

Створимо головне вікно для нашого додатку. Використаймо компоненти Ext.js для створення різних елементів інтерфейсу, таких як кнопки, поля введення, тощо.

javascript

```
Ext.application({
    name: 'MigrationApp',

    launch: function () {
        Ext.create('Ext.window.Window', {
            title: 'Migration App',
            width: 800,
            height: 600,
            layout: 'fit',
            items: [
                // Додайте сюди елементи інтерфейсу
            ]
        }).show();
    }
});
```

Реалізація Форм та Елементів Керування

Створимо різні форми для операцій міграції, налаштувань та інших функцій. Використовуйте елементи керування Ext.js для полегшення введення та вибору параметрів.

javascript

```
Ext.define('MigrationApp.view.MigrationForm', {
    extend: 'Ext.form.Panel',
    alias: 'widget.migrationform',
    title: 'Migration Form',

    items: [
        // Додайте сюди елементи форми для міграції
    ]
});
Ext.define('MigrationApp.view.SettingsForm', {
    extend: 'Ext.form.Panel',
    alias: 'widget.settingsform',
    title: 'Settings Form',

    items: [
        // Додайте сюди елементи форми для налаштувань
    ]
});
```

Модель та Відображення

Створимо моделі та відображення для ефективної роботи з даними та їх відображення в інтерфейсі.

javascript

```
Ext.define('MigrationApp.model.MigrationModel', {
    extend: 'Ext.data.Model',
    fields: [
        // Визначте поля моделі для міграції
    ]
});

Ext.define('MigrationApp.view.MigrationGrid', {
    extend: 'Ext.grid.Panel',
    alias: 'widget.migrationgrid',
    title: 'Migration Results',

    store: {
        // Використовуйте модель для зберігання та роботи з даними
    },

    columns: [
        // Визначте колонки для відображення результатів міграції
    ]
});
```

Взаємодія з Core Модулем

Застосуємо Ajax-запити або WebSocket для взаємодії з Core Модулем додатку та обміну даними.

```javascript
Ext.Ajax.request({
    url: 'core/migrateData',
    method: 'POST',
    jsonData: {
        // Відправка даних на сервер для міграції
    },
    success: function (response) {
        // Обробка успішної відповіді
    },
    failure: function (response) {
        // Обробка помилки
    }
});
```

Реалізуємо вікно логіну для нашого додатку. Використаємо Ext.js для створення форми входу та обробки введених користувачем даних.

```javascript
Ext.define('MigrationApp.view.LoginWindow', {
    extend: 'Ext.window.Window',
    alias: 'widget.loginwindow',
    title: 'Login',
    width: 400,
```

```
height: 200,
layout: 'fit',
closable: false,
modal: true,

items: [{
    xtype: 'form',
    bodyPadding: 10,
    defaults: {
        xtype: 'textfield',
        anchor: '100%',
        labelWidth: 120,
        allowBlank: false,
        margin: '10 0'
    },
    items: [{
        fieldLabel: 'Username',
        name: 'username'
    }, {
        fieldLabel: 'Password',
        inputType: 'password',
        name: 'password'
    }],
    buttons: [{
        text: 'Login',
        handler: 'onLoginClick' // Метод обробки натискання кнопки
```

```
    }]
  }]
});
```

У цьому коді ми створюємо вікно логіну, яке містить форму з полями для введення імені користувача та пароля. Також є кнопка "Login", яка буде викликати метод **onLoginClick** при натисканні.

Тепер створимо контролер, який буде обробляти події цього вікна:

javascript

```
Ext.define('MigrationApp.controller.LoginController', {
  extend: 'Ext.app.ViewController',
  alias: 'controller.login',

  onLoginClick: function () {
    var form = this.getView().down('form');
    if (form.isValid()) {
      var values = form.getValues();

      // Виконайте логіку перевірки логіна та пароля, наприклад, Ajax-запит на сервер

      // Якщо логін та пароль вірні, відкрийте головне вікно додатку
      Ext.create('MigrationApp.view.MainWindow').show();
      // Закрийте вікно логіну
      this.getView().close();
```

```
        } else {
            Ext.Msg.alert('Error', 'Please enter valid credentials.');
        }
    }
});
```

У цьому контролері ми визначаємо метод **onLoginClick**, який викликається при натисканні кнопки "Login". В цьому методі ми перевіряємо, чи введені дані коректні, і, якщо так, можна реалізувати логіку перевірки логіна та пароля. Як приклад, після введення вірних даних, ми відкриваємо головне вікно додатку та закриваємо вікно логіну.

Не забуваймо підключити цей контролер до вікна логіну та відповідну конфігурацію у вашому додатку.

javascript

```
Ext.define('MigrationApp.view.LoginWindow', {
    extend: 'Ext.window.Window',
    alias: 'widget.loginwindow',
    controller: 'login', // Підключення контролера
    // інші налаштування...
});
```

Це загальний шаблон, і нам може знадобитися додатково налаштовувати його відповідно до нашого додатку та логіки автентифікації.

Для реалізації `restClient` з методами, які ви описали, нам знадобиться використати Ajax-запити для взаємодії з серверним REST API. Давайте

створимо простий приклад, який можна подальше розширити під ваші конкретні потреби.

javascript

```
Ext.define('MigrationApp.util.RestClient', {
    singleton: true,

    getAllItems: function () {
        return this.sendRequest('GET', '/core/getAllItems');
    },

    setItemField: function (itemId, fieldName, value) {
        var data = {
            itemId: itemId,
            fieldName: fieldName,
            value: value
        };
        return this.sendRequest('POST', '/core/setItemField', data);
    },

    getItemField: function (itemId, fieldName) {
        var params = {
            itemId: itemId,
            fieldName: fieldName
        };
        return this.sendRequest('GET', '/core/getItemField', null, params);
    },
```

```
    updateTime: function () {
        return this.sendRequest('POST', '/core/updateTime');
    },

    // Допоміжний метод для відправки Ajax-запиту
    sendRequest: function (method, url, jsonData, params) {
        return new Ext.Promise(function (resolve, reject) {
            Ext.Ajax.request({
                url: url,
                method: method,
                jsonData: jsonData,
                params: params,
                success: function (response) {
                    resolve(response.responseText);
                },
                failure: function (response) {
                    reject(response.statusText);
                }
            });
        });
    }
});
```

В цьому коді ми створюємо `RestClient`, який є синглтоном, тобто існує лише одна його копія в додатку. Використовується обіцянка (Promise), щоб забезпечити асинхронність при виклику методів.

Тепер, коли вам потрібно взаємодіяти з сервером, ви можете використовувати `RestClient`. Наприклад:

javascript

```
MigrationApp.util.RestClient.getAllItems()
  .then(function (response) {
    console.log('All items:', response);
  })
  .catch(function (error) {
    console.error('Error:', error);
  });
```

Це приклад взаємодії з методом `getAllItems`. Аналогічно ви можете викликати інші методи `RestClient` та обробляти результати асинхронних викликів.

Наголошую, що це лише заготовка, і нам може бути необхідно налаштувати код відповідно до особливостей вашого REST API та бізнес-логіки.

Далі - створимо connectorUtil, який буде взаємодіяти з API конкретного коннектора, наприклад, для Jira. Основною метою цього утилітного класу буде виконання запитів до API конкретного коннектора та обробка отриманих відповідей.

javascript

```
Ext.define('MigrationApp.util.ConnectorUtil', {
    singleton: true,

    jiraApiUrl: 'https://your-jira-instance/rest/api/2',

    authenticate: function (username, password) {
        // Реалізація логіки аутентифікації з використанням переданих ім'я користувача та пароля
        // Зазвичай для Jira використовується базова автентифікація через HTTP заголовок "Authorization"
    },

    getAllIssues: function () {
        var url = this.jiraApiUrl + '/search';
        var params = {
            jql: 'project = "Your Project"',
            maxResults: 50
        };

        return this.sendRequest('GET', url, null, params);
    },

    getIssueDetails: function (issueKey) {
        var url = this.jiraApiUrl + '/issue/' + issueKey;
```

```
    return this.sendRequest('GET', url);
},

createIssue: function (issueData) {
    var url = this.jiraApiUrl + '/issue';

    return this.sendRequest('POST', url, issueData);
},

// Допоміжний метод для відправки Ajax-запиту
sendRequest: function (method, url, jsonData, params) {
    // Додайте обробку автентифікації, якщо потрібно

    return new Ext.Promise(function (resolve, reject) {
        Ext.Ajax.request({
            url: url,
            method: method,
            jsonData: jsonData,
            params: params,
            success: function (response) {
                resolve(response.responseText);
            },
            failure: function (response) {
                reject(response.statusText);
            }
        });
```

```
    });
  }
});
```

У цьому коді `ConnectorUtil` має методи для аутентифікації, отримання всіх задач, отримання деталей задачі та створення нової задачі для Jira. Зверніть увагу, що `jiraApiUrl` повинен відповідати адресі API нашої Jira інстанції.

Ми можемо додати додаткові методи або модифікувати існуючі відповідно до особливостей нашого конкретного коннектора та вимог додатку.

4.4. Детальний опис та програмна реалізація

4.4.1. Login window

1. Вікно Входу:
    - Після запуску додатку відкриється вікно входу (login window).
2. Поля Введення:
    - У вікні входу будуть поля для введення інформації, такі як ім'я користувача та пароль.
3. Введення Даних:
    - Введіть коректні дані в поля введення. Можна також продемонструвати випадкові або невірні дані для прикладу неправильного входу.
4. Натискання Кнопки "Увійти":
    - Після введення інформації натисніть кнопку "Увійти" для відправлення даних на сервер та аутентифікації.
5. Відгук на Невірні Дані:

- Якщо були введені невірні дані, додаток може вивести повідомлення або виконати відповідні дії.

6. Успішний Вхід:
    - В разі успішної аутентифікації користувача демонструйте перехід до основного інтерфейсу додатку чи виведення повідомлення про успіх.

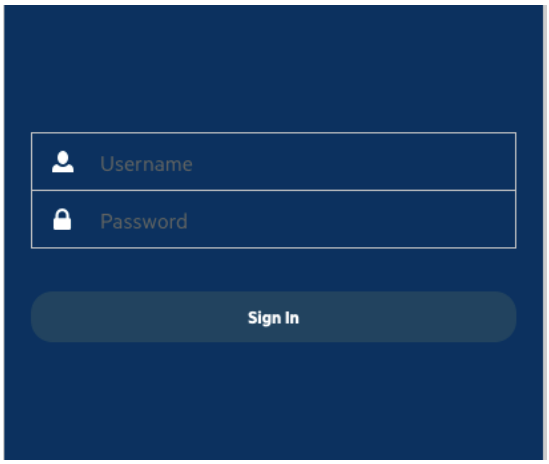

Рис.4.4. Вікно логіну

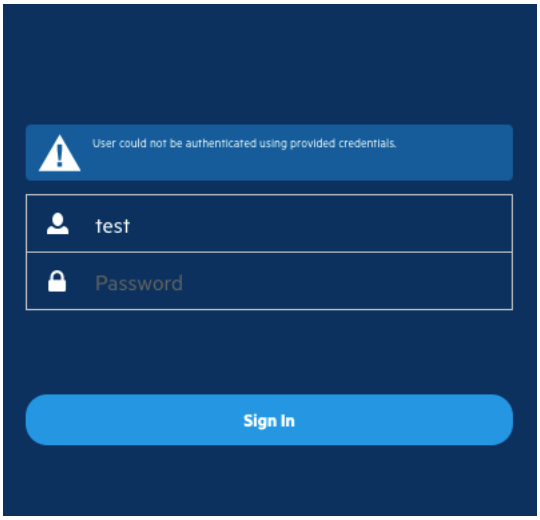

Рис.4.5. Вікно логіну з помилкою валідації

4.4.2. Main window

1. Заголовок та Меню:
    - У верхній частині вікна розташований заголовок "Керування Підключеннями". Меню надає доступ до різних опцій, таких як додавання нового підключення, редагування, видалення та інші.
2. Список Підключень:
    - Основна частина вікна відображає список усіх наявних підключень. Кожен рядок включає ім'я, тип, URL та статус підключення.
3. Стовпці Інформації:
    - Інформація відображається у зручних стовпцях для чіткості. Колонки включають ім'я, тип, URL та статус підключення.
4. Статусні Індикатори:
    - Кожен рядок обладнаний статусним індикатором, який вказує на поточний стан підключення. Кольорова гама та іконки використовуються для легкого розпізнавання.
5. Кнопки Дій:
    - Додавання, редагування, видалення та тестування підключення можна виконати за допомогою відповідних кнопок.
6. Контекстне Меню:
    - При натисканні правої кнопки миші на будь-якому рядку відображається контекстне меню з опціями для редагування, видалення та тестування підключення.
7. Функціональні Кнопки:
    - Вікно має додаткові кнопки, такі як "Оновити" для оновлення списку підключень та "Зберегти Зміни" для фіксації внесених змін.
8. Індикатор Оновлення Статусу:

- Встановлено анімаційний індикатор, який вказує на процес оновлення статусів підключень.

9. Завершення Графічної Реалізації:
    - Ваше вікно "Керування Підключеннями" готове до використання. Зручний та інтуїтивно зрозумілий інтерфейс допомагає користувачам легко управляти підключеннями та отримувати необхідну інформацію.

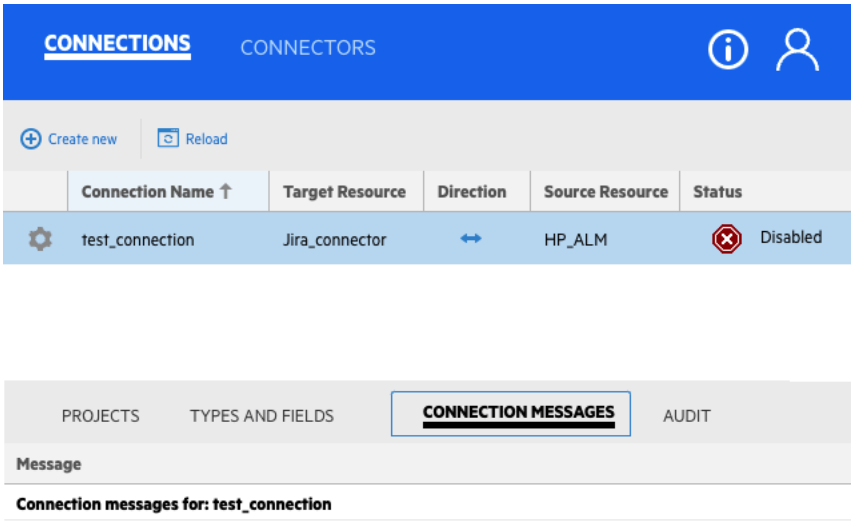

Рис.4.6. Вікно списку поточних коннекшенів

4.4.3. Connectors page

1. Заголовок та Меню:
    - У верхній частині вікна розташований заголовок "Управління Коннекторами". Меню надає доступ до опцій, таких як додавання нового коннектора, редагування, видалення та інші.
2. Список Коннекторів:

- Основна частина вікна відображає список всіх доступних коннекторів. Кожен рядок включає ім'я, тип та основні параметри коннектора.

3. Стовпці Інформації:
    - Інформація відображається в стовпцях для кращої чіткості. Колонки можуть включати ім'я, тип, параметри підключення та інші характеристики.
4. Редагування Параметрів:
    - При виборі коннектора користувач може редагувати його параметри. Додайте кнопку "Редагувати", щоб забезпечити легкий доступ до цієї функції.
5. Кнопки Дій:
    - Додавання нового коннектора, редагування, видалення та інші дії можна виконати за допомогою відповідних кнопок або іконок.
6. Додавання Нового Коннектора:
    - Забезпечте можливість додавати нові коннектори за допомогою кнопки "Додати Коннектор" або аналогічної опції.

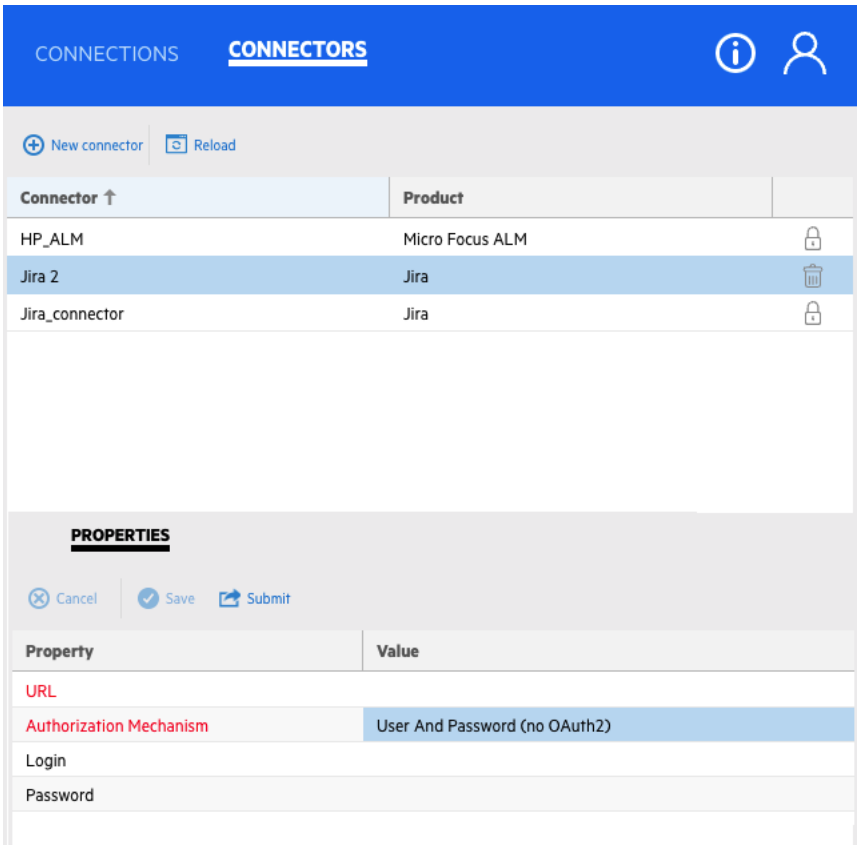

Рис.4.7. Вікно зі списком поточних коннекторів

4.4.4. Connection settings step 1

Графічна Реалізація Вікна "Налаштування Підключення" - Крок 1

1. Заголовок та Інструкції:
    - У верхній частині вікна розташований заголовок "Налаштування Підключення - Крок 1". Додається інструкція, яка надає користувачеві контекст для обрання коннекторів та параметрів.
2. Вибір Коннекторів:
    - На цьому кроці користувач може вибрати необхідні коннектори зі списку. Для зручності використовується багаторядковий список або інша графіка.
3. Інформація Про Коннектори:

- Для кожного коннектора відображається коротка інформація про його тип, функціональність та основні параметри.

4. Вибір Частоти Ітерацій:
    - Користувач може обрати частоту, з якою система виконуватиме ітерації синхронізації. Використовується елемент вибору або поле введення.
5. Кнопки Навігації:
    - Додайте кнопки для переходу до наступного кроку, наприклад, "Далі", а також кнопку "Скасувати" для відміни налаштувань.
6. Зручний Інтерфейс Вибору:
    - Забезпечте інтуїтивно зрозумілий та зручний інтерфейс вибору коннекторів. Можливість відзначати/скасовувати вибір для кожного коннектора.
7. Індикатор Прогресу:
    - Використовуйте індикатор прогресу або анімацію для позначення, що користувач робить крок до налаштувань.

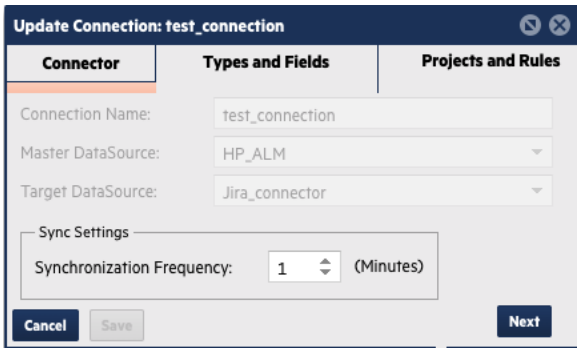

Рис.4.8.Перший крок створення/редагування коннекшена

4.4.5. Connection settings step 2

1. Заголовок та Інструкції:

- У верхній частині вікна розташований заголовок "Налаштування Підключення - Крок 2". Додається інструкція, що цей крок включає вибір полів та елементів для маппінгу.

2. Список Полів та Елементів:
    - Користувач має можливість вибрати конкретні поля та елементи даних для маппінгу. Використовується багаторядковий список або інтерактивна таблиця для вибору.
3. Зручна Візуалізація Маппінгу:
    - Вибрані елементи для маппінгу відображаються зручним способом, наприклад, за допомогою чекбоксів, кольорів або інших графічних елементів.
4. Кнопки Навігації:
    - Додайте кнопку "Назад" для можливості повернутися до попереднього кроку. Кнопка "Далі" виводить користувача до наступного кроку або завершає процес.
5. Індикатор Прогресу:
    - Використовуйте індикатор або анімацію для візуального підтвердження вибору та переходу між кроками.

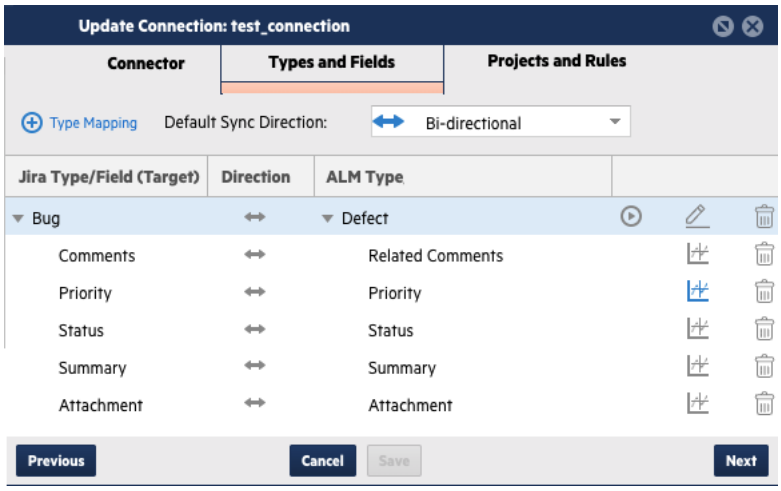

Рис.4.9.Другий крок створення/редагування конекшена

4.4.6. Connection settings step 3

1. Заголовок та Інструкції:
    - У верхній частині вікна розташований заголовок "Налаштування Підключення - Крок 3". Додається інструкція, що цей крок включає вибір вокспейсів для маппінгу.
2. Список Вокспейсів:
    - Користувач може вибрати вокспейси, які він бажає включити у процес маппінгу. Використовується багаторядковий список або інтерактивна таблиця для вибору.
3. Фільтрація за Проєктами:
    - Вокспейси можуть бути відсортовані за проєктами або іншими категоріями для полегшення вибору.
4. Зручна Візуалізація Вибору:
    - Вибрані вокспейси відображаються зручним способом, наприклад, за допомогою чекбоксів, кольорів або інших графічних елементів.
5. Кнопки Навігації:
    - Додайте кнопку "Назад" для можливості повернутися до попереднього кроку. Кнопка "Далі" виводить користувача до наступного кроку або завершає процес.
6. Індикатор Прогресу:
    - Використовуйте індикатор або анімацію для візуального підтвердження вибору та переходу між кроками.

4.4.7. Крок 3 створення або редагування коннекшена

- Крок 3 відображається зрозуміло та інтуїтивно, допомагаючи користувачам визначити, які вокспейси вони хочуть включити у процес маппінгу для міграції.

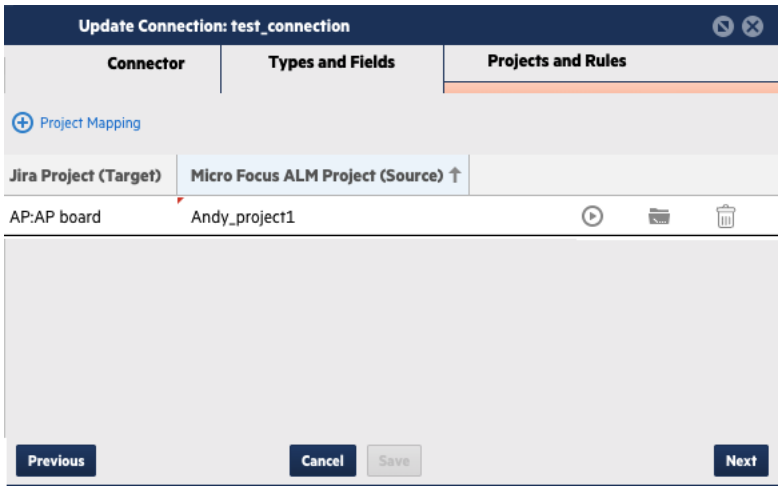

Рис.4.10.Третій крок створення/редагування конекшена

Висновок

Під час розробки та налагодження програмного продукту, спрямованого на міграцію та синхронізацію даних між ALM/CPM системами, було виконано комплексну роботу з проектування та імплементації клієнт-серверної архітектури. Розроблений програмний засіб складається з основної частини, написаної на Java, та графічного інтерфейсу, реалізованого за допомогою бібліотеки Ext.js. Дана робота включала в себе розгортання бази даних Apache Derby для збереження і управління конфігураційними та іншими даними, пов'язаними з процесом міграції.

У розділі "Загальна Схема Роботи Додатку" було представлено високорівневий огляд архітектури програмного продукту. Детально розглянуто взаємодію між клієнтом та сервером, а також основні модулі, такі як REST-клієнт, утиліти коннекторів, та важливі таблиці в базі даних.

У главі "Проектування Функціональних Можливостей Додатку" описано процес визначення та конфігурації міграційних опцій, вибору полів та елементів для маппінгу, а також вибору вокспейсів. Кожен крок налаштувань ілюструється відповідними зображеннями графічного інтерфейсу, що полегшує взаєморозуміння та користування додатком.

Загальна реалізація програмного продукту дозволяє здійснювати міграцію та синхронізацію даних між різними АЛМ/СРМ системами, забезпечуючи вибірковий та гнучкий підхід до конфігурації процесу. Крім того, графічний інтерфейс, розроблений з використанням Ext.js, надає користувачу зручний та інтуїтивно зрозумілий інтерфейс для налаштування всіх необхідних параметрів.

У підсумку, розроблений програмний засіб є потужним інструментом для команд, які використовують різні системи управління проектами, дозволяючи їм ефективно обмінюватися та синхронізувати дані, забезпечуючи високу гнучкість та конфігуруємість в управлінні міграційними процесами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мельник, Г. Управління проектами і розвиток продуктів. Центр учбової літератури, 2016. - 194 с.
2. Дзяк, І. Агільний розробник. Як стати професіоналом в ІТ. Фактор, 2018. – 53с
3. Державний стандарт України. Інформаційні технології. Методологія управління проектами та програмами з інформаційних технологій. ДСТУ ISO/IEC 20000-4:2014. – 8с.
4. Черняк, Л., & Мельничук, І. Програмний менеджмент проектів. Видавничий дім "Інтелсект", 2017. – 19 с.
5. Державний стандарт України. Інформаційні технології. Керування якістю та веденням проектів програмного забезпечення. ДСТУ ISO 9001:2015. – 9 с.
6. Закон України "Про захист персональних даних". № 2297-VI від 01.06.2010.
7. Sommerville, I. Software Engineering: A Practitioner's Approach. (9th ed.). Addison-Wesley, 2011. – 32с.
8. Ambler, S. W. "Introduction to UML 2 Activity Diagrams." Retrieved from http://www.agilemodeling.com/artifacts/activityDiagram.htm, 2004.
9. Boehm, B. Software Engineering Economics. Prentice-Hall, 1981.- 20 с.
10. Cockburn, A. Agile Software Development. Addison-Wesley, 2002. – 17 с.
11. Pressman, R. S. Software Engineering: A Practitioner's Approach. (8th ed.). McGraw-Hill, 2014. 76 с.
12. Fowler, M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002.

13. Kruchten, P. "The 4+1 View Model of Architecture." IEEE Software, 12(6), 42-50, 1995. 25 с.
14. Cohn, M. Agile Estimating and Planning. Prentice Hall, 2005.
15. Bass, L., Clements, P., & Kazman, R. Software Architecture in Practice. (3rd ed.). Addison-Wesley, 2012.
16. Beck, K. Extreme Programming Explained: Embrace Change. Addison-Wesley, 1999.
17. ISO/IEC/IEEE 42010:2011. Systems and software engineering – Architecture Description. Retrieved from https://www.iso.org/standard/50508.html.
18. IEEE Std 830-1998. IEEE Recommended Practice for Software Requirements Specifications. Retrieved from https://ieeexplore.ieee.org/document/720574.