МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук, фізики та математики

Кафедра комп'ютерних наук та програмної інженерії

РОЗРОБКА МОВИ ЗАГАЛЬНОГО ПРИЗНАЧЕННЯ ДЛЯ ПОБУДОВИ ДИНАМІЧНИХ ЗВІТІВ

Кваліфікаційна робота (проєкт)

на здобуття ступеня вищої освіти «магістр»

Виконав: студент 2 курсу 261М групи
Спеціальності
126 «Інформаційні системи та технології»
(шифр, назва)
Освітньо-професійної програми: «Інформаційні системи та технології»
(назва)
Яцюта Владислав Олександрович
Керівник: доктор економічних наук, професор Кобець В.М.
Рецензент: senior QA lead DataARt Асмолова В.С.

Івано-Франківськ – 2023

ЗМІСТ

ВСТУП

Актуальність дослідження. Створення звітів є невід'ємною складовою будь-якого бізнесу, від обчислення простих статистичних даних до складних математичних розрахунків, які впливають на розподіл майбутніх інвестицій. Наприклад, в маркетингу звіти активно використовуються для аналізу результативності різних рекламних кампаній на основі даних, таких як кількість переглядів, час перегляду реклами та відсоток відмов (bounce rate). Бізнеси, орієнтовані на продажі, розробляють свої стратегії на основі даних про продукти, регіони та цільову аудиторію. У фінансових проєктах аналіз валової прибутковості, чистої прибутковості та рентабельність інвестицій (ROI) є надзвичайно важливим для збільшення доходів та зменшення витрат. Відділи кадрів будь-якої компанії уважно вивчають звіти про продуктивність співробітників, щоб зрозуміти потребу у додаткових програмах навчання та наданні певних видів допомоги з використанням ключових показників ефективності (KPI). Зазвичай такі звіти ґрунтуються на критеріях, таких як рух кадрів, рейтинги задоволення співробітників та оцінки їх продуктивності.

Кожен напрямок бізнесу унікальний, і навіть компанії, що працюють у тій самій галузі зі схожими вхідними даними, зазвичай мають різний підхід до їх аналізу та управління. Проте усім бізнесам потрібно збирати, перетворювати та досліджувати дані, що представлені у форматі звітів. Виникає питання дослідження, чи можна використовувати сучасні методології та технології для створення єдиного способу створення звітів, який не залежить від їхньої сфери застосування. Щоб відповісти на це питання, спочатку потрібно сформулювати набір вимог до такого інструменту. З іншого боку, для правильного формулювання вимог потрібно дотримуватися загальних стандартів [1], і має бути чітке розуміння того, що об'єднує різні галузі бізнесу та в чому вони відрізняються одна від одної, оскільки цей інструмент працюватиме лише з певними наборами даних, порівняння має здійснюватися з точки зору саме даних.

Однією з важливих навичок бізнес-аналітика є підготовка звітів, які

становлять основу прийняття рішень [2]. З вхідних даних інвестори-робо-консультанти можуть створити свій профіль ризику [3] та створити інвестиційні портфелі у вигляді звіту з розподілом різних фінансових інструментів [4, 5]. Пошук відповідностей між компетентностями співробітників і вимогами роботодавців призводить до створення звіту з кандидатами на вакансії на ринку праці [6].

Крім того, для повноцінної функціональності цього інструменту необхідно створити універсальну мову GPL (мову загального призначення) для створення шаблонів звітів, яка має містити елементи, що дозволять її адаптувати до потреб будь-якого бізнесу.

В якості експерименту ми використовуємо створену мову GPL для створення програмного засобу для розрахунку бізнес-метрик та їх подальшого включення в звіт. Для цього не вистачає лише створеної мови GPL, додатково потрібно створити механізм її обробки. У цій роботі проведено дослідження можливості такої обробки за допомогою існуючих систем, а також алгоритмів для автономної побудови таких систем.

Об'єкт дослідження: процес побудови бізнес звітів

Предмет дослідження: мова загального призначення GPL для побудови динамічних звітів з різних бізнес сфер

Мета дослідження: розробити програмний прототип на основі мови загального призначення GPL для генерації звітів.

Завдання дослідження:

1) провести аналіз різних звітів та галузей бізнесу для визначення спільних рис та відмінностей;
2) сформулювати список вимог до мови загального призначення для побудови динамічних звітів;
3) описати та визначити мову загального призначення для побудови динамічних звітів;
4) побудувати прототип програми.

Методи і технології дослідження. контент та кореляційні аналізи різних

бізнес звітів, аналіз наукових робіт пов’язаних з тематикою, алгоритми побудови та опису мов загального призначення.

Робота має наступну структуру: розділ перший розглядає стан досліджень мов загального призначення при побудові звітів, розділ другий визначає методологічний підхід та вимоги; розділ третій представляє результати; розділ четвертий описує процес створення та роботи прототипу програмного модуля.

РОЗДІЛ 1.

ОЦІНКА ТА АНАЛІЗ НАУКОВИХ РОБІТ ПОВЯЗАНИХ З ПРОЦЕСАМИ АНАЛІЗУ ТА ОБРОБКИ ДАНИХ ТА МЕХАНІЗМІВ СТВОРЕНЯ ДИНАМІЧНИХ ФІНАНСОВВИХ ЗВІТІВ.

1.1. Отримання, збір та аналіз даних

Існує багато наукових робіт та досліджень, що аналізують можливості обробки даних та створення складних звітів [7]. "Такі техніки зазвичай спрямовані на рівень, що відповідає за доступ до даного інструменту, що використовується для маніпулювання їх змістом, з припущенням, що на цьому рівні достатньо інформації та контролю для автоматизації процесу" [8]. Як правило, програмні інструменти використовуватимуть різні технології інтерфейсу, що ускладнює їх взаємодію. Дослідники вказують, що інтеграція компенсує недоліки існуючих інструментів та технологій. Багато програмних інструментів не підтримують запит до ресурсів, що доступні [8]. Підхід до інтероперабельності (здатності до взаємодії) інструментів на основі принципів Linked Data (пов’язаних даних) може компенсувати виявлені недоліки. Лу [9] описує синтаксичні складнощі створення мов, включаючи мови побудови звітів [10], і піднімає питання якості даних [11]. "Безперервна комунікація даних на всіх етапах процесу розробки продукту є передумовою вартісного і успішного процесу співпраці, але головною проблемою у цьому ланцюжку процесу є недостатня якість даних" [11].

Отримання інформації має важливе значення для перетворення даних у бізнес-сценаріях цифрової економіки. Проте створення систем отримання інформації у реальних випадках має дві складнощі: обмежена доступність даних і їх деталізована класифікація [12]. Отримувач інформації може використовувати попередньо навчені мовні моделі, навчені за допомогою перетворювачів, для отримання інформації, результати зберігаються в базі

даних для обробки документів вручну (Рис. 1.1).

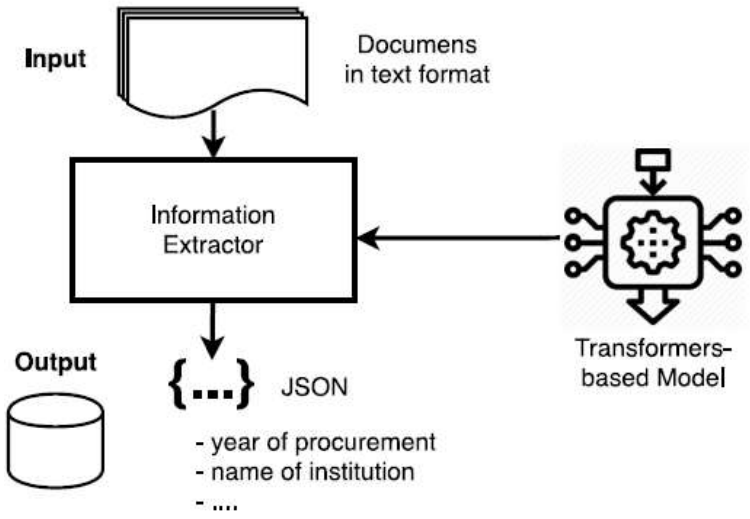


Рис. 1.1. Архітектура системи отримання даних [12]

Основна задача таких автоматизованих систем полягає у перетворенні неструктурованих документів на структуровані дані [12]. Оцінка бізнесу включає збір, аналіз і використання фінансової інтегральної інформації для оцінки вартості бізнесу. Оцінені результати використовуються при прийнятті рішень щодо ціноутворення, сегментації ринку, потужності бізнесу, витрат на одиницю тощо. "Існують конкретні відомості та події про оцінку бізнесу, збережені в розумних фінансових звітах, представлених у форматах HTML і PDF. Отже, існує попит на систему отримання інформації з фінансових даних для оцінки бізнесу з точки зору внутрішніх фінансових звітів бізнесів з різних джерел даних" [13].

"Застосування систем бізнес аналітики може розглядатися як стратегія та розвиток бізнесу, яка інтегрує в собі комплекс послуг для надання важливої корпоративної інформації у стратегічному та операційному прийнятті рішень та збільшення конкурентоздатності корпорації. Для успішної реалізації системи бізнес-аналітики в організації необхідно використовувати чітко визначені

процеси та бізнес-правила" [14].

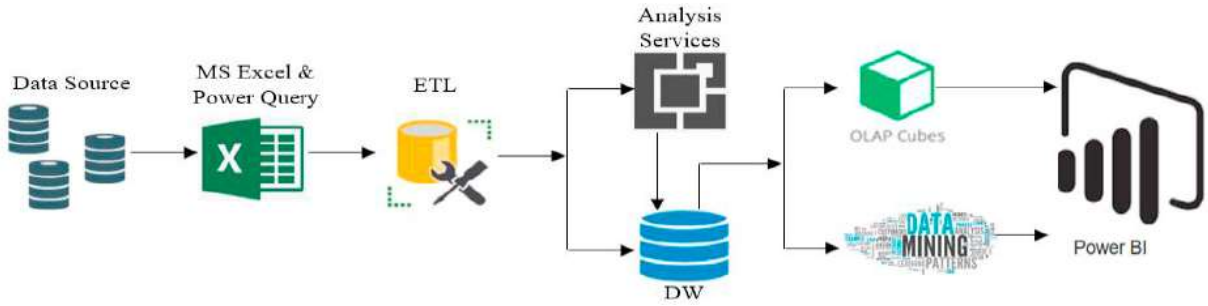


Рис. 1.2. Архітектура системи обробки даних [14]

"Комп'ютерні застосунки дозволяють організаціям забезпечувати контроль якості над даними, генерувати важливі показники для надання менеджерам інформації про бізнес організацій та готувати їх до більш ефективного прогнозування сценаріїв, використовуючи різні технологічні архітектури" (Рис. 1.2) [14].

1.2. Предметно орієнтовані мови та мови загального призначення

Існує багато предметно орієнтованих мов (DSL) для різних, але не загальних цілей. Наприклад, предметно орієнтована мова (DSL) може описувати набори даних машинного навчання з точки зору їх структури, контексту та соціальних питань (справедливість, відсутність упередженості), що може сприяти вибору найбільш відповідних наборів даних та їх використанню для нового проєкту або стартапу [15]. Квінтеро та ін. [16] пропонують "предметно орієнтовану мову для полегшення моделювання політик управління використанням та їх інтеграцію в процеси розробки на основі моделей", що полегшує політику безпеки.

Дослідження [17] пропонують предметно орієнтовану мову (DSL) створену для аналізу вимог. За допомогою цієї мови можна автоматично вказувати структурні незмінні вимоги до програмного забезпечення та розкривати їх неузгодженості між собою.

Завдяки високому рівню абстракції, DSL дозволяють створювати застосунки, що спрощують реалізацію програмного забезпечення. Розроблена архітектура моделі програмування [18] дозволяє визначати та специфікувати JSON-DSL, реалізовувати компоненти JavaScript, використовувати з'єднувачі для гетерогенних джерел інформації та інтегрувати їх з веб-компонентами та фреймворками JavaScript як на рівні веб-сервера, так і на рівні веб-клієнта. Ієрархії підходу багаторівневого моделювання можуть представляти мови моделювання специфічні для галузі, які можуть співіснувати та розвиватися незалежно між собою та одночасно, бути частиною більших систем [19].

Знання, зібрані в процесових моделях, повинні послідовно переноситися до визначених вимог. Для бізнес-орієнтованих програмних систем остаточні вимоги зазвичай все ще визначаються природною мовою, тому виникає проблема неузгодженості між процесними моделями та вимогами, вираженими природною мовою, в контексті розробки програмного забезпечення [20]. Для вирішення цієї проблеми, Айсолмаз та ін. пропонують напівавтоматизований підхід, результатом якого є створення документів вимог, що інтегрують дані з процесних моделей і дані щодо їх виконання в інтуїтивно зрозумілому вигляді [20]. Цей підхід включає три основні кроки:

1) аналіз процесних моделей, що відповідають системі,

2) збір даних, для подальшої обробки,

3) автоматичне створення документів вимог зі створених моделей за допомогою алгоритму генерації природною мовою на основі шаблонів.

Когнітивна теорія мультимедійного навчання вказує, що як текстові, так і візуальні дані мають бути представлені одночасно для отримання документів вимог високої якості з процесних моделей. Цей підхід включає дві фази: підготовчу фазу (визначення автоматизованих дій, аналіз вимог) та фазу генерації (генерація та вдосконалення речень, організація структури документу) [20]. Електронне поширення фінансової та бізнес-інформації суттєво розвинулося; онлайн-звітування перетворилося зі статичного формату PDF або

HTML у динамічний формат XBRL, виражений у трьох формах: концепції, зв'язки та ресурси. Розширювана мова обліку бізнесу (XBRL) полегшує обмін і стандартизацію фінансової інформації на різних платформах і системах. Використання стандарту XBRL для звітування стає більш реальним з різних точок зору: комунікації, збору даних, контролю та аналізу фінансової інформації. Документ-екземпляр XBRL - це бізнес-звіт, опублікований у спеціальному форматі, який включає період звітування; суму майна, об'єкти і обладнання; суму готівки та еквівалентів готівки; суму поточних запасів; річний прибуток тощо [21]. Іншою важливою концепцією XBRL як машинного "словника" є таксономія "словника концепцій та термінів, які компанія має використовувати для звітування своєї бізнес-інформації відповідно до набору стандартів або правил, які точно визначають ці терміни" [21]. Обмеження цього дослідження полягає в тому, що воно включає лише погляди бухгалтерів і не враховує думки інших стейкхолдерів.

1.3. NLP системи

Компанії досліджують потужність обробки природної мови (NLP) для автоматизації процесів та прийняття рішень на основі даних. Бехер та ін. пропонують фреймворк з формулюванням етичних принципів, щоб забезпечити, що NLP є безпечною та надійною технологією для відповідального прийняття рішень і призводить до суспільної користі [22]. NLP - це процес розуміння того, як комп'ютерні системи використовують текст, мову та інші ресурси, і як вони працюють на комп'ютерах. Основною метою NLP є досягнення обробки мови, схожої на людську, для різних задач, наприклад, створення різних звітів за допомогою обчислювальних методів для аналізу згенерованих текстів. Бізнес-менеджери можуть використовувати NLP для різних завдань організаційного аналізу, обробки комунікацій і текстового аналізу. NLP, як і будь-яка інша технологія, може бути шкідливою для приватності. Відповідальне прийняття рішень - це здатність систем NLP робити

конструктивний вибір щодо бізнес-поведінки на основі етичних стандартів.

Оскільки взаємозв'язок між продуктами, компаніями та клієнтами в бізнес-середовищі стає складнішим, а термін служби продуктів або послуг скорочується, традиційні методи, спрямовані на експертно-центричний підхід, стають витратними за часом і ресурсами. У результаті цього менеджери та галузеві практики все більше приймають підхід на основі даних для отримання надійних результатів щодо можливостей у бізнесі. Вплив аналізу бізнес-можливостей на основі даних відрізняється, в залежності від джерел даних (наприклад, стандартизовані та нестандартизовані дані) [23]. Мовна модель, адаптована Койя та ін. для ефективного аналізу численних та різноманітних бізнес-елементів, використовує метод локального фактору викиду для вимірювання рівня новизни, використовуючи численні шаблонами [23].

Аналіз даних процесів дозволяє організаціям отримувати цінні уявлення про те, як виконуються процеси і які можливості для їх поліпшення. Запити до даних процесів дозволяють аналітикам досліджувати дані з метою отримання уявлень про виконання бізнес-процесів [24]. Поточне покоління мов запитів до процесів спрямовано на аналітиків з глибоким рівнем розуміння даних. Проте існує потреба у мові запитів, для підтримки і допомоги аналітикам галузі, які можуть бути недосвідчені в технологіях баз даних, де інтерфейс приймає запити природною мовою від користувача та автоматично створює відповідний структурований запит для виконання над збереженими даними подій [24]. Аналітики галузі та навіть кінцеві користувачі повинні мати можливість використовувати можливості аналізу процесів для виконання своїх дій у цифрових процесах, оскільки основне обмеження технологій аналізу процесів полягає в тому, що вони не роблять дані процесів доступними для користувачів у природний спосіб. Кобейссі та ін. пропонують переваги машинного навчання та підходів на основі правил для створення запитів щодо вмісту, поведінки та продуктивності [24].

Результати цих матеріалів послужили відправною точкою для цієї роботи.

Розроблений програмний прототип унікальний тим, що його можна використовувати як бібліотеку та зручно інтегрувати в будь-яку існуючу систему з підтримкою Java. Наприклад, веб-сайт компанії може бути використаний не лише для створення звітів, але і для зручного обчислення різних метрик з подальшим збереженням у базі даних.

РОЗДІЛ 2

ФОРМУВАННЯ ВИМОГ МОВИ ЗАГАЛЬНОГО ПРИЗНАЧЕННЯ ДЛЯ ПОБУДОВИ ДИНАМІЧНИХ ЗВІТІВ

2.1. Аналіз бізнес звітів та результати їх порівняння

Основою цього розділу стали: аналіз різних бізнес-звітів з різних бізнес галузей (наприклад, фінанси: звіти про ризики у зв'язку з кредитними зобов'язаннями, подорожі: статистика бронювань в різні періоди, продажі: аналіз відвідуваності веб-сайту на основі рекламних кампаній тощо), дослідницькі матеріали з цієї теми та досвід роботи в галузі IT (15 років в різних бізнес-галузях). Для аналізу ми використовували, такий інструмент як "дерево рішень". Підхід, заснований на даних, - це підхід до управління бізнесом та побудови аналітичних звітів на основі використання великих обсягів даних. Дерево рішень - це інструмент підтримки прийняття рішень, який використовується в аналізі даних та статистиці.

Розпочнемо з порівняння різних бізнес-галузей з точки зору даних для визначення відмінностей між ними.

1) Модель. Кожен бізнес працює зі своїми власними унікальними даними. Наприклад, у галузі охорони здоров'я: інформація про пацієнтів, включаючи медичну історію, діагнози, результати аналізів і плани лікування, а також операційні дані про штат, інвентар та розрахунки. У виробництві: дані про процеси виробництва, рівні запасів, управління ланцюжком постачання та контроль якості, а також дані про продажі та попит споживачів. Ці дані мають власні типи та структуру. Все це є моделлю.
2) Зберігання та передача. Дані можна зберігати в базах даних, файлах, хмарах тощо Для їх передачі можна використовувати різні протоколи, такі як FTP, HTTP, SOAP тощо.

Далі визначимо, що об'єднує різні бізнес-галузі:

1) Структурованість. Незважаючи на те, що самі дані різняться, вони всі мають певну структуру і певні залежності.
2) Трансформація даних для різних видів обчислень є найбільш очевидною річчю.
3) Математичні формули є формою представлення абсолютної більшості обчислень.
4) Звіти створюються регулярно.

2.2 Формулювання та аналіз вимог

На основі цього ми можемо сформулювати наступний перелік вимог:

1) Підтримка отримання моделі з різних джерел: баз даних, файлів, онлайн-повідомлень тощо.
2) Можливість обробки даних. Оскільки моделі структуровані, їх можна представити у будь-якому структурованому форматі обміну даними, такому як JSON, XML, YAML тощо.
3) Можливість динамічно змінювати модель, змінюючи та додаючи нові дані, з підтримкою складних математичних формул і функцій для їх розширення.
4) Повернути змінену модель користувачеві.
5) Процес створення нової моделі має бути якомога більше автоматизованим та не вимагати постійного втручання людини.

У цій роботі ми розглянемо найцікавішу і, можливо, найважливішу вимогу: необхідність зміни моделі та додавання нових даних. Процес створення звіту, зазвичай, базується на шаблонах, створених людьми, а самі процеси є автоматизованими. Отже, всі правила для зміни моделі мають бути описані в шаблоні, який після застосування до моделі приведе до створення нової, модифікованої моделі, яка буде служити основою для звіту. Оскільки цей шаблон буде розроблятися людьми, всі правила для зміни моделі повинні бути описані у найзручнішій формі для людей. Для цього потрібно розробити мову

загального призначення (GPL). Враховуючи, що цей інструмент призначений для використання в абсолютно різних бізнес-сферах, розроблена мова повинна мати елементи предметно орієнтовані мови (DSL), щоб бути гнучкою і адаптованою до конкретних потреб бізнесу.

Тепер деталізуємо вимоги до GPL:

1) Лаконічність та зрозумілість для людей.
2) Адаптованість до потреб бізнесу.
3) Здатність до опису перетворень.
4) Підтримуваність у ході перетворень різних типів даних: чисел, тексту, дат, наборів даних тощо.

На сьогодні існує велика кількість програм і інструментів для створення звітів: від управління активами [25] до послуг з перевірки орендарів [26]. Більшість із них спеціалізуються і використовуються в конкретних галузях. Однак існують винятки, такі як Microsoft Excel, Tableau, SAP Crystal Reports, JasperReports, QlikView. Питання виникає щодо того, наскільки добре вони відповідають критеріям, які ми визначили для створення звітів і як вони описують перетворення. З точки зору передачі та отримання даних ці інструменти є досить гнучкими, але щодо підтримки різних моделей даних вони лише частково ефективні. У більшості випадків ці програми працюють з фіксованим набором табличних даних і інколи потребують втручання людини для їх конфігурації та генерації звітів, що додає значний рівень складності до автоматизації процесу. Крім того, табличні дані не є гнучкими і мають певні обмеження. Перетворення даних в основному виконуються за допомогою математичних функцій і формул. Різноманітність цих правил досить широка, але додавання нових функцій перетворення є складним і нетривіальним процесом і в більшості випадків потребує навичок програмування [27, 28], і не всі інструменти підтримують ці можливості.

2.3 Використання математичних формул як бази GPL

Розглянемо використання математичних формул різного рівня складності як основу для GPL з точки зору сформульованих нами вимог.

Лаконічність і зрозумілість для людей. Цей аспект суттєво залежить від складності формул і рівня знань конкретної особи. Наприклад, прості обчислювальні формули, такі як валова маржа (валовий прибуток поділений на дохід), поточний коефіцієнт (поточні активи поділені на поточні зобов'язання), чиста рентабельність (чистий прибуток поділений на дохід), досить зрозумілі. Проте формули для регресійного аналізу та розрахунків суми майбутнього доходу (значення інвестиції в майбутньому, на основі вказаної ставки відсотка та періоду часу) можуть бути досить складними для розуміння. Однак із підтримкою створення власних функцій всі складні розрахунки можуть бути упаковані в один виклик функції з набором параметрів.

Адаптивність до потреб бізнесу. У цьому пункті розкривається потреба в елементах DSL. Виділимо два критерії, які потрібно враховувати:

1) Перший - це інтеграція моделі в математичні формули, зокрема використання імен об'єктів як частину обчислень. Наприклад, замість передачі фактичних значень валового прибутку та доходу ми можемо передати їхні імена як змінні. Тоді формула виглядатиме так:

   ВаловаМаржа = ВаловийПрибуток /Дохід.

Це більш зрозуміло та лаконічно для людей. Оскільки вхідна модель зазвичай має складну ієрархічну структуру, можуть використовуватися символи, такі як '.' або '->', для отримання під-елементів.

2) Другий критерій - це підтримка створення власних функцій. Оскільки особливо складні розрахунки можуть перетворюватися на великі формули, які важко обслуговувати та оновлювати. Зручно пакувати такі формули у власні функціями, які приймають лише необхідні аргументи. Особливо це корисно, коли такі розрахунки повторюються в багатьох місцях.

   Опис перетворень може бути представлений як:

   <місце в моделі> = <формула перетворення>

Для позначення місця, куди дані мають бути розміщені після обробки,

можна використовувати той же підхід із використанням імен змінних. Якщо додати пріоритет до обчислень, отримані результати можуть бути використані в інших обчисленнях.

Підтримка різних типів даних. Математичні формули добре працюють із числовими типами даних, але стосовно інших типів: рядків, дат, грошей тощо, така підтримка має бути реалізована на семантичному рівні побудованої мови GPL.

РОЗДІЛ 3

СТВОРЕННЯ МОВИ ЗАГАЛЬНОГО ПРИЗНАЧЕННЯ ДЛЯ ПОБУДОВИ ДИНАМІЧНИХ ЗВІТІВ

3.1. Основана на формулах GPL з прикладами

На основі викладеного вище ми тепер маємо створити мову загального призначення (GPL), використовуючи математичні формули як основу. В якості прикладу розглянемо базові розрахунки для інвестиційного портфеля на основі кредитних угод. Ми розглянемо різні варіанти синтаксису, можливі проблеми і можливі способи їх вирішення. JSON-модель даних буде мати наступний вигляд (лістинг 3.1)

```
{"Deals":[
        {
        "DealID":"deal1",
        "Loans":[
                {
                "Price":65,
                "LoanId":"lx1",
                "Balance":10000
                },
                .... ]
         },
        .... ]
}
```

Лістинг 3.1. Вхідна модель для кредитних угод інвестиційного портфеля

Кожна угода містить набір позик з ціною та балансом. Завдання: побудувати модель звіту з наступними розрахунками:

1) Загальний баланс для всіх позик.
2) Загальний баланс для кожної угоди.
3) Зважена середня ціна за баланс для кожної угоди.
4) Зважена середня ціна за баланс для кожної угоди з критерієм ціна > 80.

5) Список назв позик, розділених ','.

Результатом має стати наступна модель (лістинг 3.2):

```
{
  "Totalbalance" : ***,
  "Deals":[
    {
      "DealID":"deal1",
      "Totalbalance" : ***,
      "WAP": ***,
      "WAP80": ***,
      "LoanNames":"lx1,..."
       "Loans":[
              {
               "Price":65,
              "LoanId":"lx1",
              "Balance":10000
               } .... ]
    }, ... ]
}
```

Лістинг 3.2. Модель звіту з описом трансформацій

Далі розглянемо, як буде виглядати наша мова загального призначення (GPL) за допомогою математичних формул:

1) Загальний баланс для всіх позик. Це, ймовірно, найпростіший розрахунок - це просто сума об'єктів у наборі. Припустимо, що в нашій системі реалізована функція SUM, яка приймає набір чисел як параметр і повертає результат у вигляді числа. У цьому разі правило перетворення буде виглядати таким чином:

```
{
   "Totalbalance" : "SUM(Deals.Loans.Balance)"
}
```

2) Загальний баланс для кожної угоди. Розрахунок, хоч і дуже схожий на попередній, але є складнішим з трьох причин:
   - Місце в моделі не є кореневим. Як ми раніше обговорювали, для

складної ієрархії можна використовувати символи "." або "->".

- Результат виконання буде не числом, а набором чисел. Далі спростимо задачу і припустимо, що функція SUM приймає набір наборів і повертає суму для кожного з них.
- З точки зору семантики, GPL необхідно однозначно визначити, що нам потрібен набір балансів, а не весь ряд. Крім того, цю проблему потрібно розглядати з точки зору можливих бізнес-потреб, зокрема додавання додаткових фільтрів і критеріїв. Наприклад, якщо для портфелю необхідно обчислити суму балансів лише для позик з вищою середньою ціною або для позик із ціною менше певного значення.

  Такі вимоги можна реалізувати кількома способами:

  - Можна змінити логіку Deals.Loans.Balance і замість одного списку отримувати список списків. У цьому випадку потрібно додати функцію для об'єднання списків (наприклад, для обчислення Totalbalance для всіх угод). Крім того, потрібно пам'ятати, що рівень вкладеності може бути набагато вищим, і тоді ми вже отримаємо список списків списків. У цьому випадку функція об'єднання повинна мати додаткові параметри, що описують рівень вкладеності.
  - У багатьох структурованих мовах "[]" часто використовується для фільтрації елементів. В цьому випадку, щоб показати, що саме на цьому рівні нам потрібен список елементів, ми можемо позначити "[i]", і тоді формула виглядатиме так:

    {

    "Deals.Totalbalance": "SUM(Deals[i].Loans.Balance)"

    }

    Також "[]" може бути використаний для визначення умов формування підмножин. Наприклад, Deals.Loans[price>80].Balance поверне нам список балансів позик з ціною більше 80.

- Варіант також може бути реалізований на функціональному рівні, написавши функцію getElements, яка буде повертати списки необхідних елементів. У цьому випадку формула перетворення буде такою...

```
{
    "Deals.Totalbalance":
    "SUM(getElements(Deals.Loans.Balance)"
}
```

За потреби до умови формування фільтра підмножини може бути додано додатковий параметр.

```
{
    "Deals.Totalbalance":
    "SUM(getElements(Deals.Loans.Balance, price>80)"
}
```

3) Зважена середня ціна за баланс для кожної угоди. Формула для знаходження зваженої середньої ціни - це сума всіх змінних, помножених на їх вагу, поділена на суму ваг. У нас вже є формула для обчислення суми списку, але для отримання списку нам потрібно помножити два списки, у цьому випадку нам потрібно реалізувати операцію множення наборів, або отримати список помножених елементів. Оскільки ми вже обчислюємо суму балансів, ми можемо використовувати пріоритет, якщо він існує в наших правилах, і використовувати наступне:

```
{
    "Deals.WAP":
    "SUM(getElements(Deals.Loans, Balance*Price)
    /getElements(Deals, Totalbalance)",
    "priority": 2
}
```

Ми також можемо написати функцію для обчислення зваженої середньої, яка приймає змінні та їх ваги як параметри:

{

"Deals.WAP":

"WtAvg(Deals[i].Loans.Price, Deals[i].Loans.Balance)"

}

4) Зважена середня ціна за баланс для кожної угоди з умовою price > 80 потребує реалізації умов. У результаті ми можемо отримати наступні варіанти:

- За допомогою функції SUM та getElements:

{

"Deals.WAP80":

*"SUM(getElements(Deals.Loans, Balance * Price, price > 80) / SUM(getElements(Deals.Loans, Balance))"*

}

- За допомогою фільтрації елементів в списку:

{

"Deals.WAP80":

*"SUM(Deals[i].Loans[price > 80].Price * Deals[i].Loans[price > 80].Balance) / SUM(Deals[i].Loans.Balance)"*

}

- За допомогою функції WtAvg та умови if:

{

"Deals.WAP80":

"WtAvg(if(Deals.Loans.Price > 80, Deals.Loans.Price, 0), Deals.Loans.Balance)"

}

5) Список назв кредитів, розділених ',', найкраще робити за допомогою функції конкатенації (Concat):

{

"Deals.WAP80": "Concat(getElements(Deals.Loans, LoanId), ',')"

}

3.2. Програмна реалізація GPL

Маючи базове розуміння GPL викликає питання, як його можна представити у програмній формі для зручного обчислення. Оскільки математичні формули мають різний пріоритет операцій (множення та ділення виконуються перед додаванням, параметри повинні бути обчислені перед функціями, пріоритети операцій можуть бути встановлені за допомогою дужок і таке інше), існує багато структур даних, які можна використовувати для таких обчислень. Оскільки обчислення відбуваються від операцій найвищого пріоритету до найнижчого, можна використовувати стек (список, який дозволяє доступ лише до останнього доданого елемента), де операції з найвищим пріоритетом будуть в кінці. Альтернативно, можна використовувати дерева (структуру даних, що складається з вузлів, які мають посилання на дочірні елементи), де обчислення відбуватимуться віднизу вгору, доки не досягнуть кореня.

Наступним кроком розглянемо, у якій формі наша створена GPL матиме таку структуру даних, як дерево. Для цього опишемо, як правила перетворення будуть перетворені у вузли:

1) Змінні (Deals.Loans.Balance, Deals.Loans.Price тощо), числові константи (80,0) і текстові константи: оскільки вони не потребують додаткових обчислень, їх можна представити як вузли без дочірніх елементів.
2) Бінарні операції: додавання, множення, ділення, віднімання та операції порівняння повинні бути представлені як вузли з двома дочірніми елементами.
3) Функції будуть перетворені у вузли з кількістю дочірніх елементів, що дорівнює кількості параметрів.
4) Унарний мінус можна розглядати як функцію з одним параметром.

Давайте розглянемо деякі приклади (рис. 3.1-3.4): З метою зрозуміння в формулах ми будемо використовувати спеціальну змінну (СЗ). Аналіз формул і їх перетворення в структури дерев є фундаментальною складовою розробки

інструменту, який використовує побудовану бібліотеку GSL для генерації звітів. Ця робота починається з визначення поставленої задачі, за яким іде ідентифікація вхідних і вихідних параметрів. Крім того, проводиться дослідження можливих альтернатив і алгоритмів для досягнення нашої мети.

Вхідні параметри включають текстові представлення формул. Ці формули можуть містити наступні компоненти:

1) Числові значення: 1, 3.14 ...
2) Логічні значення: true, false
3) Константи: Pi, E ...
4) Змінні та елементи моделі вхідних даних: TotalBalance, Deal.loans.balance
5) Текстові константи: 'Текст константи'
6) Булеві операції: &&, ||, ...
7) Операції порівняння: ==, !=, >, >= ...
8) Стандартні функції: SUM, MIN, SIN, а також спеціальні функції для конкретних бізнес-потреб з довільними параметрами
9) Дужки для зміни пріоритету операцій
10) Арифметичні операції з різними пріоритетами (множення та ділення мають вищий пріоритет): +, -, /, *

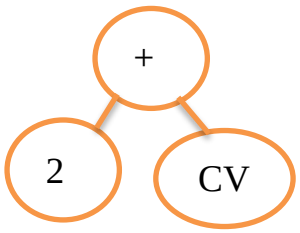


Рис. 3.1. Проста арифметична операція 2+CV

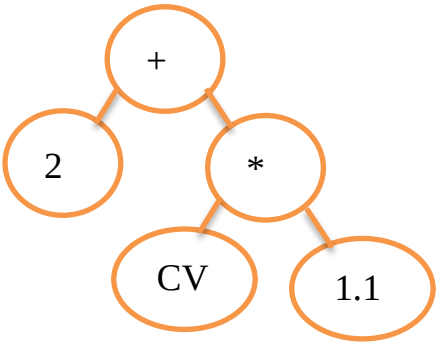


Рис. 3.2. Арифметична операція з пріоритетами за замовченням

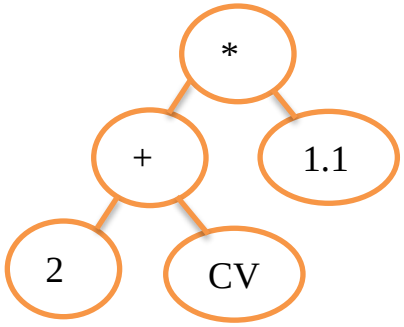


Рис. 3.3. Арифметична операція зі зміненими пріоритетами

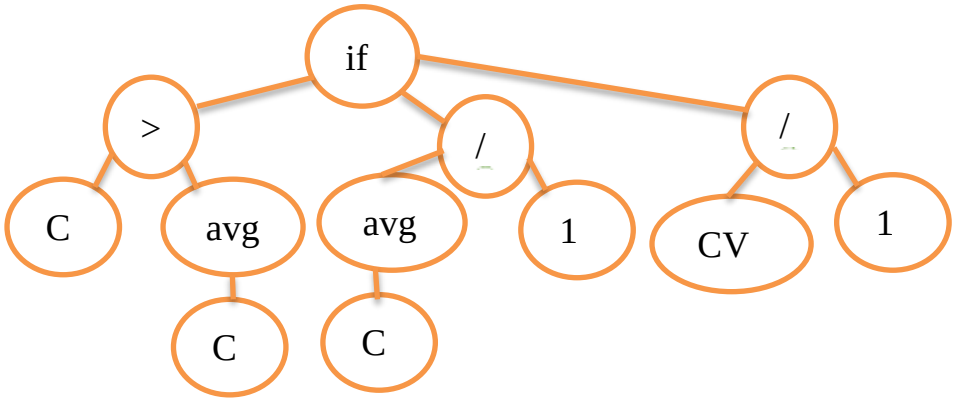


Рис. 3.4. Арифметична операція з функціями

Потрібний результат - це структура даних, що дозволить нам:

1) Виконувати вказані обчислення на основі вхідної моделі та попередніх обчислень.

2) Оновлювати модель результатами обчислень.

Для вирішення цієї задачі ми можемо використовувати бібліотеки та

системи для створення та обробки граматики:

1) ANTLR [29], відомий як ANother Tool for Language Recognition, виступає як потужний генератор парсерів, який дозволяє створювати обробники мови, такі як компілятори, інтерпретатори та транслятори. Використовуючи граматичний підхід, він спрощує специфікацію синтаксису мови та подальше створення парсера на обраній мові програмування [30].

2) JavaCC (Java Compiler Compiler) [31] - це інструмент для генерації парсерів, спеціально розроблений для створення парсерів на основі Java. Він дозволяє визначати синтаксис мови за допомогою специфікації граматики і генерує Java-парсер для цієї мови.

3) GNU Bison [32] - інструмент для генерації парсерів, який використовується для створення парсерів LALR(1). Він використовує граматичний підхід та генерує файли з вихідним кодом C або C++ для обробки вхідних даних на основі визначеної граматики.

Альтернативно, можемо аналізувати вирази самостійно і будувати дерево, використовуючи простий алгоритм (додатки 1-2). Цей метод може надати більше гнучкості та контролю над системою.

РОЗДІЛ 4
СТВОРЕННЯ ПРОГРАМНОГО ДОДАТКУ

4.1. Розробка архітектури

Для проведення експерименту ми поставили перед собою дві мети:

- Дослідити та знайти ефективний спосіб використання розробленої нами GPL для створення звітів.
- Розробити програмний прототип для використання GPL у створенні базового звіту.

Перша архітектура, яку ми одержали, зображена на рис. 4.1. У цій архітектурі ми вирішили розділити завантаження даних та обробку даних. На першому етапі ми збираємо дані з відповідних джерел (таких як бази даних, кеш, файли на FTP-сервері тощо) та структуруємо їх. Існує безліч бібліотек і фреймворків для завантаження даних, залежно від джерела даних (наприклад, Hibernate [33], Spring Data [34] для баз даних, Apache POI [35] для розбору файлів Excel та PDF тощо). Для структурування даних буде достатньо перетворити їх у формат "ключ-значення", де значення може бути складним об'єктом, що складається з кількох пар "ключ-значення" (каскадна мапа). Об'єкти без залежностей один від одного (наприклад, отримані з різних джерел даних) можуть бути представлені різними ключами у кореневому об'єкті.

На другому етапі модуль програмного забезпечення використовує шаблон перетворення, створений за допомогою розробленої нами GPL, для обробки та перетворення даних у модель звіту.

На третьому етапі ми перетворюємо отриману модель у необхідний формат (Excel, звичайний текст, електронну пошту тощо).

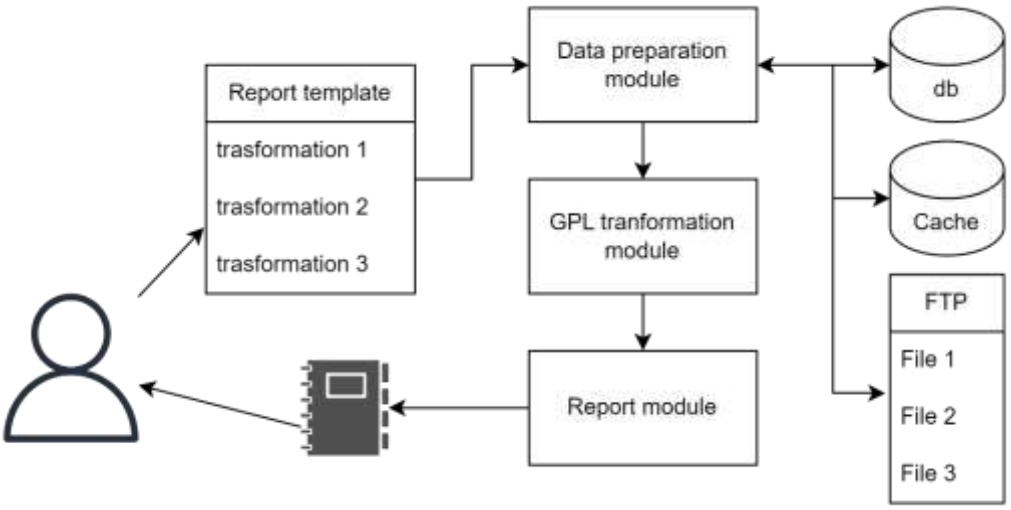


Рис. 4.1. Архітектура програмного модулю

Головною перевагою цієї архітектури є безпека. Оскільки збір та обробка даних відбуваються на сервері, користувачу не надається доступ до даних, які виходять за межі звіту. Крім того, якщо з певних причин шаблон потрапить до незаконних рук, він не надасть зловмиснику інформацію, яку можна використовувати для отримання конфіденційних даних.

Проте ця архітектура має значні недоліки:

- Низька гнучкість. Для кожного шаблону нам потрібно підтримувати список обов'язкових джерел даних, і якщо шаблон зазнав значних змін, то потрібно змінювати відповідні джерела даних.
- Складність створення звітів значно зростає, коли виникає потреба завантажити додаткову інформацію після певних перетворень. Наприклад, ми обчислили загальну статистику цін на кредити для конкретного портфеля. Якщо статика показує значне зниження прибутку, ми хочемо завантажити список додаткових кредитів, які можуть значно покращити результати і включити їх у звіт.

Як результат, ми прийшли до другої версії архітектури, зображеної на рис. 4.2. У цій версії ми вирішили об'єднати збір даних та їх обробку, дозволяючи користувачу визначити джерела даних, порядок отримання та обробку даних у

шаблоні. Для того, щоб ця архітектура була функціональною та ефективною, нам довелося відповісти на певні виклики:

- Безпека: Шаблон може потрапити до зловмисників, тому конфіденційні дані не можуть зберігатися в ньому.
- Стандартизація: Шаблон має бути написаний у нашій GPL та відповідати її семантичним та лексичним стандартам, включаючи використання математичних формул як основи.

Для вирішення цих питань ми вирішили використовувати динамічні функції. Оскільки вони є частиною GPL, вони відповідають її стандартам. Вони також достатньо безпечні, оскільки їх виконання розташоване на сервері, і звичайні користувачі мають доступ лише до назви функції та переданих параметрів. З правильним рівнем абстракції, якщо всі конфіденційні дані використовуються виключно в коді функції і загальна інформація передається через параметри, такий підхід можна безпечно використовувати для завантаження даних.

Отже, ми завершили перший етап експерименту, а саме знаходження ефективного способу використання розробленої нами GPL для створення звітів. Друга частина передбачає демонстрацію функціональності цього виконання, зокрема створення програмного модуля.

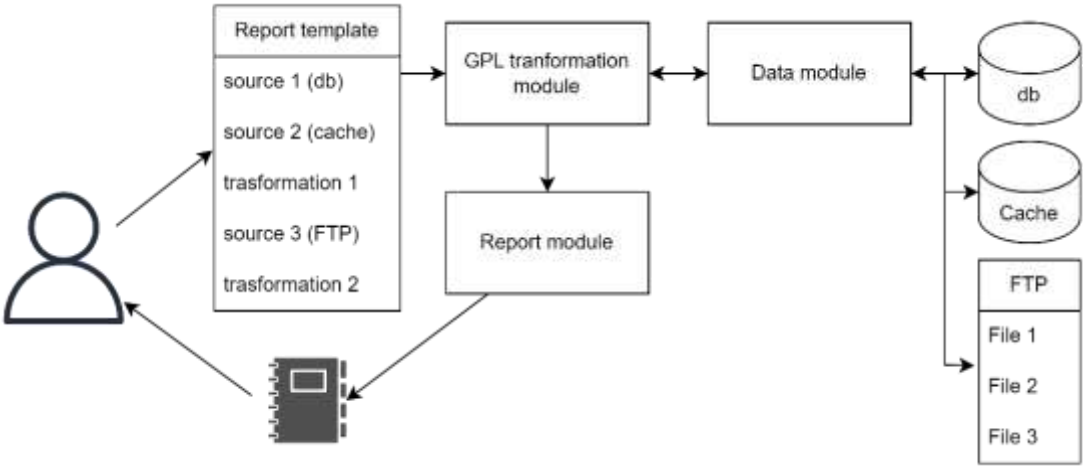


Рис. 4.2. Архітектура програмного модулю

4.2. Програмний модуль

На вхід програмний модуль приймає шаблон генерації звітів, створений за допомогою визначеної GPL та представлений у форматі JSON. У шаблоні вказуються джерела даних та список перетворень даних (Рис 4.3), порядок команд у шаблоні є пріоритетом.

```
{
    "Loans" : "GenerateTestData('Loans','lx1','lx2','lx3','lx4')",
    "TotalBalance" : "SUM(Loans.Balance)",
    "WAP" : "SUM(Loans.Balance*Loans.Price)/TotalBalance"
}
```

Рис. 4.3. Шаблон побудови репорту

З огляду на презентаційні цілі, ми використовували невеликий JSON як тестові дані, але створений інструмент може обробляти великі обсяги даних. Щоб більш чітко продемонструвати процес, ми перетворили шаблон генератора звітів в список команд (Рис 4.4).

```
@Test
void testReport() throws IOException {

    List<FormulaColumn> formulas = new ArrayList<>();
    formulas.add(new FormulaColumn( field: "Loans", formula: "GenerateTestData('Loans','lx1','lx2','lx3','lx4')"));
    formulas.add(new FormulaColumn( field: "TotalBalance", formula: "SUM(Loans.Balance)"));
    formulas.add(new FormulaColumn( field: "WAP", formula: "SUM(Loans.Balance* Loans.Price)/TotalBalance"));
    FormulaProcessor processor = new FormulaProcessor();
    print(processor.update(formulas).toString());

}
```

Рис. 4.4. Шаблон побудови репорту у вигляді команд

Пропонуємо розглянути шаблон, представлений на Рис. 4.3-4.4. Він складається з трьох команд:

1. Генерація тестових даних про кредити з ідентифікаторами: lx1, lx2, lx3, lx4.

2. Обчислення загального балансу згенерованих кредитів: *SUM(Loans.Balance)*.
3. Обчислення зваженого середнього значення ціни за баланс:
   *SUM(Loans.Price * Loans.Balance) / TotalBalace*,
   де TotalBalace - це значення, обчислене на попередньому етапі.

Усі команди з шаблону будуть перевірені, оброблені і перетворені в дерева, як показано на Рис 3.1-3.4.

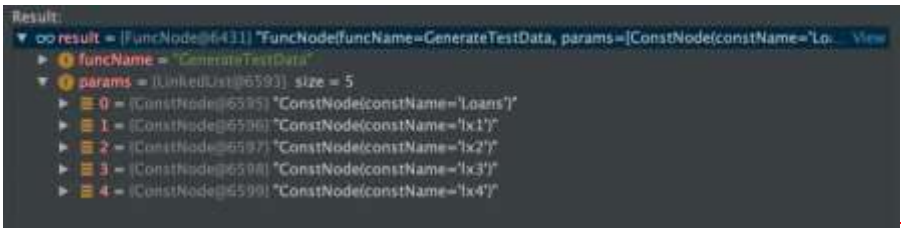


Рис. 4.5. GenerateTestData('Loans','lx1','lx2','lx3','lx4') у вигляді дерева

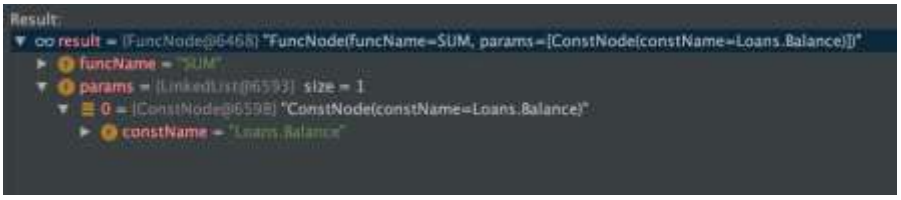


Рис. 4.6. SUM(Loan.Balance) у вигляді дерева

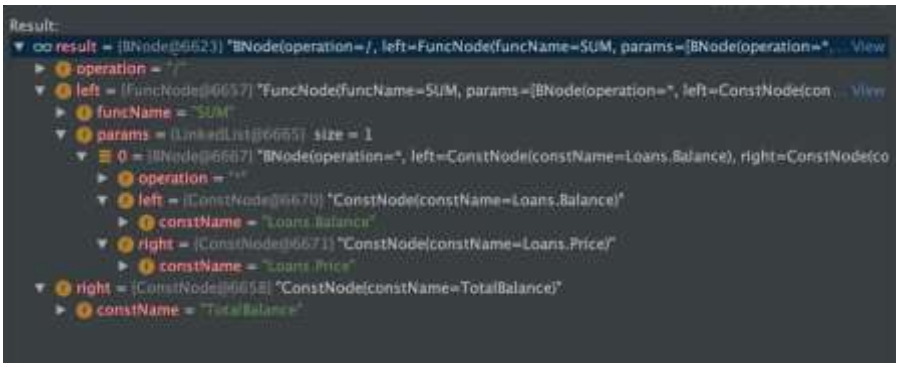


Рис. 4.6. SUM(Loan.Price*Loan.Balance)/TotalBalance у вигляді дерева

Після валідації ми будемо виконувати команди по черзі. Результат виконання першої команди буде списком кредитів, представлених у форматі "ключ-значення", як показано на Рис 4.7.

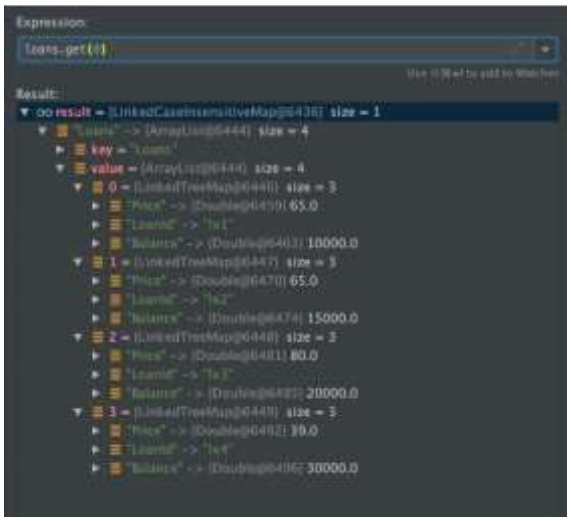


Рис. 4.7. Згенеровані тестові дані

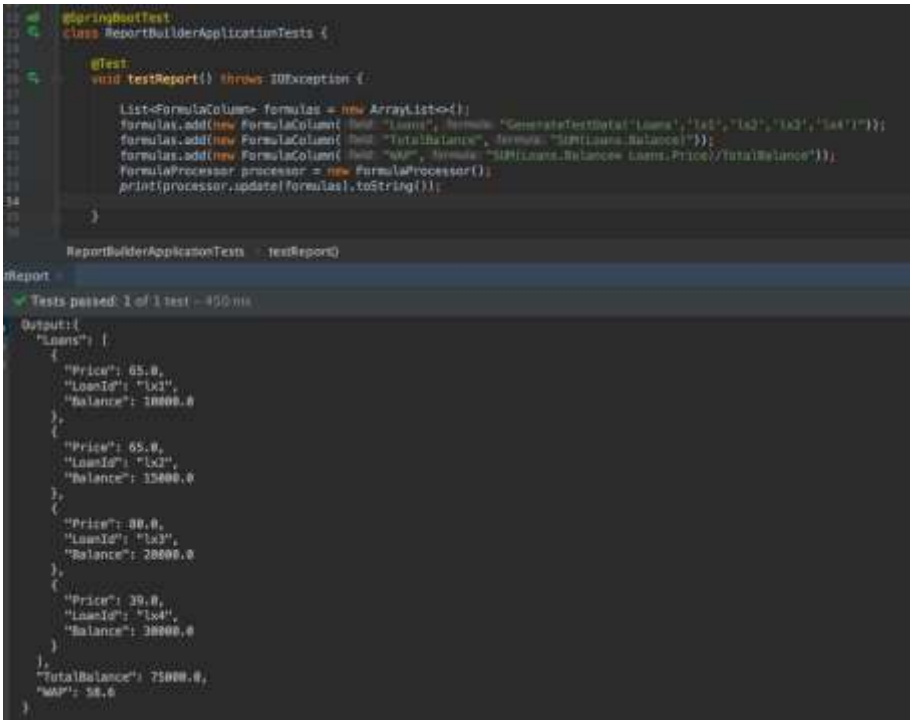
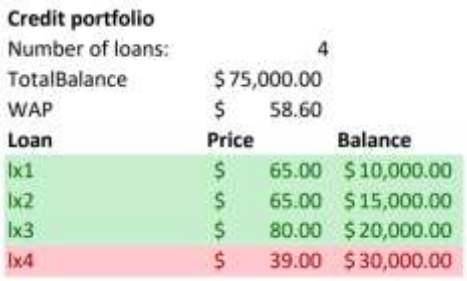


Рис. 4.8. Результати роботи програмного модулю

Виконання другої та третьої команди надасть результати обчислень на

запит. Після виконання всіх команд буде створений вихідний JSON, який може бути використаний для генерації електронного звіту (Рис 4.9).



Credit portfolio		
Number of loans:	4	
TotalBalance	$ 75,000.00	
WAP	$ 58.60	
Loan	**Price**	**Balance**
lx1	$ 65.00	$ 10,000.00
lx2	$ 65.00	$ 15,000.00
lx3	$ 80.00	$ 20,000.00
lx4	$ 39.00	$ 30,000.00

Рис. 4.9. Приклад згенерованого звіту

Результати експерименту показують ефективний спосіб використання розробленої нами GPL для створення власних звітів. Ми створили програмний прототип, щоб продемонструвати один із прикладів такої реалізації.

ВИСНОВКИ

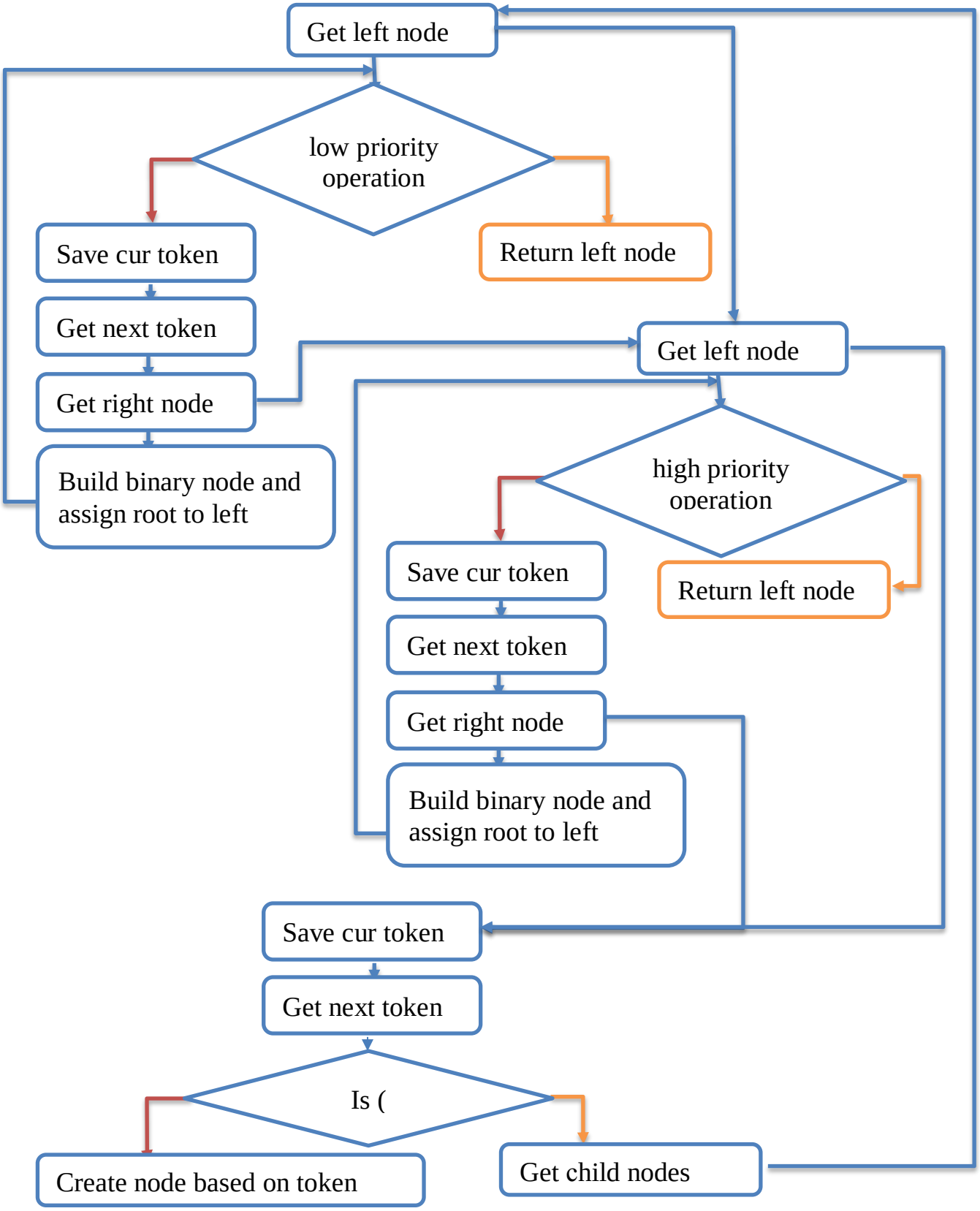
У цій роботі ми сформулювали основу GPL для створення динамічних звітів. Крім того, ми розглянули механізми обробки такої GPL для подальшого використання при обчисленні різних бізнес-метрик.

Результати експерименту показують, що, використовуючи підхід, який ми описали у даній роботі, дає можливість створити програмний модуль, який легко інтегрується з будь-якою бізнес-системою. Цей модуль може бути використаний для створення бізнес-звітів, а можливість швидко змінювати шаблон робить цей процес динамічним. Варіанти використання цього підходу не обмежуються лише генерацією звітів. Складні математичні обчислення можуть бути застосовані в різних галузях, таких як динамічний розрахунок ресурсів для зменшення навантаження на систему, або динамічні розрахунки для визначення цін та знижок на продукти та послуги бізнесу тощо.

Побудова GPL для створення звітів у різних бізнес-галузях є дуже цікавим і перспективним напрямком. Використання математичних підходів може зробити процес більш гнучким і адаптованим до потреб бізнесу. Великою перевагою такого підходу є лаконічність та зрозумілість GPL для людини і його легкість використання. Серед недоліків можна виділити складність впровадження такої мови через численні критерії, які потрібно враховувати.

ДОДАТКИ

Додаток. 1. Схема алгоритму побудови дерев на основі виразів



Додаток. 2. Опис алгоритму побудови дерев на основі виразів

Увесь процес складається з двох основних частин:

1. Розбір виразу: Розбиття виразу на частини.
2. Побудова вузлів дерева.

В розділі 3.2 ми вже ідентифікували елементи, що складають формулу. Для розбору виразу ми створимо об'єкт, який складається з трьох частин:

1. Нерозібрана частина формули.
2. Буфер: Зберігає елемент формули, який в даний момент обробляється.
3. Тип даних у буфері: Ця інформація допоможе нам зручно побудувати вузли.

Спочатку нам потрібно створити просту функцію getToken(), яка визначатиме тип елементу на основі першого символу в нерозібраній частині формули. Функція послідовно переміщуватиме дані в буфер, доки символ задовольняє умови для відповідного елементу. Ми визначимо умови та типи повернення для кожного ідентифікованого елементу:

1. Числові значення: Починаються з цифри, можуть містити цифри та '.', тип елементу - NUMBER (ЧИСЛО).
2. Булеві значення, константи, змінні та функції: Починаються з алфавітного символу, можуть містити буквено-цифрові символи, тип елементу - NAME (ІМ'Я).
3. Текстові константи: Починаються і закінчуються """, тип елементу - TEXT (ТЕКСТ).
4. Арифметичні операції та дужки: Представлені відповідними символами, кожен має свій власний тип елементу.

5. Булеві операції та порівняльні операції: Представлені одним чи двома символами, кожен має свій власний тип елементу.

Цей підхід дозволить нам розібрати вирази і побудувати дерево, надаючи нам гнучку та керовану систему. Давайте докладніше розглянемо процес розбору. Ми використовуватимемо рекурсивний метод для побудови дерева. Всі елементи в формулі можна розділити на три пріоритети:

1. Найвищий пріоритет: Унарні вузли (числові значення, текстові значення, змінні, константи тощо) та функції.
2. Вищий пріоритет другого порядку: Бінарні вузли з більшим пріоритетом операцій (множення, ділення, піднесення до степеню тощо).
3. Найнижчий пріоритет: Бінарні вузли з меншим пріоритетом операцій (додавання, віднімання, ...).

Кожен пріоритет представлений окремим блоком, який викликає один одного. Якщо потрібно додати додаткові пріоритети, ви просто створюєте новий блок для обробки і поміщаєте його на відповідне місце в ланцюгу обробки. Побудова дерева завжди розпочинається з найнижчого пріоритету та рухається вгору, поки дерево не буде повністю побудоване.

Додаток 1 ілюструє діаграму алгоритму розбору формули, де показані блоки кожного пріоритету.

Кожен пріоритет представлений окремим блоком, які викликають один одного. Оскільки два найнижчих пріоритети включають бінарні операції, блоки виглядають однаково, і єдиний відмінок полягає в перевірці бінарної операції. Давайте ретельніше розглянемо ці блоки:

Найнижчий та вищий другорядний пріоритет:

1. Створіть лівий вузол, викликавши наступний блок вищого пріоритету.
2. У нескінченному циклі перевірте, чи елемент у буфері є операцією

потрібного пріоритету.

3. Якщо ні, поверніть лівий вузол з блоку.
4. Якщо так:
    - Збережіть поточний елемент.
    - Отримайте наступний елемент.
    - Створіть правий вузол, викликавши наступний блок вищого пріоритету
    - Перетворіть лівий вузол, використовуючи збережений елемент як корінь і лівий та правий вузли як дітей.

Найвищий пріоритет:

1. Збережіть поточний елемент.
2. Отримайте наступний елемент.
    - Якщо збережений елемент має тип NUMBER (ЧИСЛО) або TEXT (ТЕКСТ), створіть та поверніть термінальний вузол (вузол без дітей).
    - Якщо збережений елемент - '-' (унарний мінус), створіть унарний вузол (має одного дитину), викликавши блок ще раз для отримання дочірнього елемента.
    - Якщо збережений елемент має тип NAME (ІМ'Я), перевірте поточний елемент. Якщо це не відкриваюча дужка "(", створіть та поверніть термінальний вузол. Якщо це відкриваюча дужка, маємо справу з функцією, і нам потрібно отримати всі її параметри, викликавши блок ще раз.
    - Якщо збережений елемент - відкриваюча дужка, у нас можуть бути

параметри функції або дужки для зміни пріоритету. Викличте блок найнижчого пріоритету, щоб отримати дочірній елемент.

- Якщо поточний елемент - це кома ',', у нас є параметри функції. Викликайте блок найнижчого пріоритету до тих пір, поки отримаєте всі дочірні елементи, і поточний елемент більше не є комою.

- Якщо поточний елемент не є закриваючою дужкою, у формулі є помилка. Створіть та поверніть вузол на основі кількості дочірніх елементів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Palmer, B.: What Are International Financial Reporting Standards (IFRS)? 2022. URL: https://www.investopedia.com/terms/i/ifrs.asp/
2. Kobets, V., Yatsenko, V., Buiak, L.: Bridging Business Analysts Competence Gaps: Labor Market Needs Versus Education Standards. Communications in Computer and Information Science 1308 (2021) 22-45. doi: 10.1007/978-3-030-77592-6_2.
3. Kobets, V., Yatsenko, V., Mazur, A., Zubrii, M.: Data analysis of personalized investment decision making using robo-advisers. Science and Innovation 16 (2) (2020) 80-93. doi: 10.15407/SCINE16.02.080.
4. Savchenko, S., Kobets, V.: Development of Robo-Advisor System for Personalized Investment and Insurance Portfolio Generation. Communications in Computer and Information Science 1635 (2022) 213-228. doi: 10.1007/978-3-031-14841-5_14.
5. Kobets, V., Petrov, O., Koval, S. Sustainable Robo-Advisor Bot and Investment Advice-Taking Behavior. Lecture Notes in Business Information Processing 465 (2022) 15-35. doi: 10.1007/978-3-031-23012-7_2.
6. Kobets, V., Tsiuriuta, N., Lytvynenko, V., Novikov, M., Chizhik, S. Recruitment web-service management system using competence-based approach for manufacturing enterprises. Lecture Notes in Mechanical Engineering (2020) 138-148. doi: 10.1007/978-3-030-22365-6_14.
7. Kenton, W.: Business Segment Reporting Definition, Importance, Example, 2021. URL: https://www.investopedia.com/terms/b/business-segment-reporting.asp.
8. El-khoury, J., Berezovskyi, A., Nyberg, M.: An industrial evaluation of data access techniques for the interoperability of engineering software tools. Journal of Industrial Information Integration 15 (2019) 58-68. doi: 10.1016/j.jii.2019.04.004.

9. Lu, X.: Automatic analysis of syntactic complexity in second language writing. International Journal of Corpus Linguistics 15:4 (2010), 474–496. doi: 10.1075/ijcl.15.4.02lu.
10. Hayes, A.: eXtensible Business Reporting Language (XBRL): Investor's Guide, 2022 URL: https://www.investopedia.com/terms/x/xbrl.asp.
11. Bondar, S., Ruppert, C., Stjepandić, J.: Ensuring data quality beyond change management in virtual enterprise. International Journal of Agile Systems and Management 7 (3-4) (2014) 304 – 323. doi: 10.1504/IJASM.2014.065346.
12. Nguyen, M.-T., Le, D.T., Le, L.: Transformers-based information extraction with limited data for domain-specific business documents. Engineering Applications of Artificial Intelligence 97 (2021) 104100.
13. Seng, J.-L., Lai, J.T.: An Intelligent information segmentation approach to extract financial data for business valuation. Expert Systems with Applications 37 (2010) 6515–6530. doi:10.1016/j.eswa.2010.02.134.
14. Duque, J., Godinhob, A., Vasconceloscd, J.: Knowledge data extraction for business intelligence. Procedia Computer Science 204 (2022) 131–139.
15. Giner-Miguelez, J., Gómez, A., Cabot, J.: A domain-specific language for describing machine learning datasets. Journal of Computer Languages 76 (2023) 101209.
16. Quintero, A.M.R., Pérez, S.M., Varela-Vaca, A.J., López, M.T.G., Cabot, J.: A domain-specific language for the specification of UCON policies. Journal of Information Security and Applications 64 (2022) 103006.
17. Vidal, M., Massoni, T., Ramalho, F.: A domain-specific language for verifying software requirement constraints. Science of Computer Programming 197 (2020) 102509.
18. Chavarriaga, E., Jurado, F., Rodríguez, F.D.: An approach to build JSON-based Domain Specific Languages solutions for web applications. Journal of Computer Languages 75 (2023) 101203.
19. Rodrígueza, A., Macíasd, F., Duránc, F., Rutle, A., Wolter, U.:

Composition of multilevel domain-specific modelling languages. Journal of Logical and Algebraic Methods in Programming 130 (2023) 100831.

20. Aysolmaz, B., Leopold, H., Reijers, H.A., Demirörs, O.: A semi-automated approach for generating natural language requirements documents based on business process models. Information and Software Technology 93 (2018) 14–29. doi: 10.1016/j.infsof.2017.08.009.
21. Enia, L.C.: Empirical research: exploring extensible business reporting language and views of Romanian accountants. Procedia Economics and Finance 32 (2015) 1675-1699. doi: 10.1016/S2212-5671(15)01495-1.
22. Behera, R.K., Bala, P.K., Rana, N.P., Irani, Z.: Responsible natural language processing: A principlist framework for social benefits. Technological Forecasting & Social Change 188 (2023) 122306. doi: 10.1016/j.techfore.2022.122306.
23. Choia, J., Jeong, B., Yoonc, J.: Identification of emerging business areas for business opportunity analysis: An approach based on language model and local outlier factor. Computers in Industry 140 (2022) 103677. doi: 10.1016/j.compind.2022.103677.
24. Kobeissi, M., Assy, N., Gaaloul, W., Defude, B., Benatallah, B., Haidar, B.: Natural language querying of process execution data. Information Systems 116 (2023) 102227. doi: 10.1016/j.is.2023.102227.
25. Best, R.: Best Asset Management Software, 2023. URL: https://www.investopedia.com/best-asset-management-software-5090064.\
26. Carmody, B.: Best Tenant Screening Services, 2023. URL: https://www.investopedia.com/best-tenant-screening-services-5070361.
27. Kenton, W.: Visual Basic for Applications (VBA): Definition, Uses, Examples, 2022. URL: https://www.investopedia.com/terms/v/visual-basic-for-applications-vba.asp.
28. Hicks, M., Levin, D.: CMSC 330: Organization of Programming Languages, 2013. URL: https://www.coursehero.com/file/178765173/org-of-Progpdf/

29. ANother Tool for Language Recognition, https://www.antlr.org/documentation.html, Accessed 29 May 2023.
30. ANTLR, https://github.com/antlr/antlr4, Accessed 29 May 2023.
31. JAVACC, https://javacc.github.io/javacc/documentation/, Accessed 29 May 2023.
32. GNU Bison, https://www.gnu.org/software/bison/, Accessed 29 May 2023.
33. Hibernate, https://hibernate.org/, Accessed 29 May 2023.
34. Spring Data, https://spring.io/projects/spring-data, Accessed 29 May 2023.
35. Apache POI, https://poi.apache.org/, Accessed 29 May 2023.