НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТімені ІГОРЯ СІКОРСЬКОГО»

Інститут прикладного системного аналізуКафедра системного проектування

До захисту допущено:

Завідувач кафедри

_____А.І. Петренко

«_____»_____20__р.

Дипломна робота

на здобуття ступеня бакалавра

**за освітньо-професійною програмою «Інтелектуальні сервіс-орієнтовані розподілені обчислювання»
спеціальності 122 «Комп'ютерні науки»на тему: «Порівняння засобів оркестрації мікросервісів Camunda та jBPM на прикладі каталогу сервісів інтернету речей FIWARE»**

Виконав:
студент IV курсу, групи ДА-72 Авдієнко Владислав Олександрович _
_____

Керівник:
ст. викладач, к. т. н.
Чкалов Олексій Валерійович _____

Консультант з економіки:доцент, к. е. н.
Рощина Надія Василівна _____

Рецензент:
Корнага Ярослав Ігорович, д.т.н., проф. каф. ТК ФІОТ__
Нормоконтроль:
ст. викладач Кирюша Богдан Анатолійович

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент Авдієнко В. О._____

Київ – 2021 рік

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут прикладного системного аналізу

Кафедра системного проектування

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 «Комп'ютерні науки»

Освітньо-професійна програма «Інтелектуальні сервіс-орієнтовані розподілені обчислювання»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____А.І. Петренко

«___»_____20___

р.

ЗАВДАННЯ

на дипломну роботу студенту

Авдієнку Владиславу Олександровичу

1. Тема роботи «Порівняння засобів оркестрації мікросервісів Camunda та jBPM на прикладі каталогу сервісів інтернету речей FIWARE», керівник роботи к.т.н ст. вик. Чкалов Олексій Валерійович, затверджена наказом по університету від «26» травня 2021 р. №1344-с

2. Термін подання студентом роботи 09.06.2021 р.

3. Вихідні дані до роботи:

Системи управління бізнес-процесами Camunda та jBPM, документаії до систем, мова програмування Java для реалізації сервісів та логіки бізнес-процесу, фреймворк - Spring,

4. Зміст роботи

1. Аналіз підходів управління бізнес процесами
2. Аналіз та порівняння Camunda BPM та jBPM
3. Дослідження сервісів Fiware
4. Аналіз підходів управління мікросервісами в BPM системах
5. Моделювання бізнес-процесу в Camunda та jBPM

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

1. Презентація.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н.В., к. е. н., доцент		

7. Дата видачі завдання 24.11.2020

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	24.11.2020	
2	Збір інформації	1.02.2021	
3	Ознайомлення з літературою і підготовка теоретичної частини роботи	10.03.2021	

4	Огляд підходів оркестрації мікросервісів в BPMS	12.04.2020	
5	Тестування можливостей оркестрації мікросервісів Camunda та jBPM	10.05.2020	
6	Оформлення дипломної роботи	30.05.2020	
7	Отримання допуску до захисту та подача роботи в ДЕК		

Студент В.О Авдієнко

Керівник О.В Чкалов

АНОТАЦІЯ

бакалаврської роботи Авдієнка Владислава Олександровича на тему:
“Порівняння засобів оркестрації мікросервісів Camunda та jBPM на прикладі каталогу сервісів інтернету речей FIWARE”

Дана дипломна робота присвячена вивченню можливостей систем управління бізнес-процесами Camunda та jBPM в якості оркестраторів мікросервісів.

В рамках дипломної роботи було досліджено сучасні підходи для моделювання та реалізації бізнес процесів та проведено аналіз вищезгаданих систем.

Було встановлено та налаштовано необіхдне програмне оточення для моделювання та реалізації бізнес-процесів, створено демонстраційні сервіси та налаштовано комунікацію між ними за подомогою функціоналу систем.

Дана робота може бути використана як навчальний посібник при ознайомленні з розглянутими в роботі BPM системами.

Загальний обсяг роботи: 103 сторінки, 55 рисунків, 7 таблиць, 15 посилань.

Ключові слова: BPMS, бізнес процес, оркестрація, мікросервіс, Camunda, jBPM, веб-сервіс, нотація, Fiware, IoT.

АННОТАЦИЯ

бакалаврской работы Авдиенко Владислава Александровича на тему:
"Сравнение средств оркестрации микросервисив Camunda и jBPM на примере каталога сервисов интернета вещей FIWARE"

Данная дипломная работа посвящена изучению возможностей систем управления бизнес-процессами Camunda и jBPM в качестве оркестраторов микросервисов.

В рамках дипломной работы были исследованы современные подходы для моделирования и реализации бизнес-процессов и проведен анализ вышеупомянутых систем.

Было установлено и настроено необходимое программное окружение для моделирования и реализации бизнес-процессов, создано демонстрационные сервисы и настроено коммуникацию между ними с помощью функционала систем.

Данная работа может быть использована как учебное пособие при ознакомлении с рассмотренными в работе BPM систем.

Общий объем работы: 103 страницы, 55 рисунков, 7 таблиц, 15 ссылок.

Ключевые слова: BPMS, бизнес-процесс, оркестрациия, микросервис, Camunda, jBPM, веб-сервис, нотация, Fiware, IoT.

ANNOTATION

to the Avdienko Vladyslav bachelor’s degree thesis on the topic: "Comparison of Camunda and jBPM microservice orchestration tools on

example of a directory of IoT services FIWARE"

Thesis is dedicated to exploring the capabilities of business process management systems Camunda and jBPM as a microservices orchestrator.

As part of the thesis, modern approaches to modeling and implementing business processes were investigated and the analysis of the above systems was carried out.

The necessary software environment for modeling and implementing business processes was installed and configured, demo services were created and communication between them was configured using the functionality of the systems.

This work can be used as a teaching aid in acquaintance with the BPM systems considered in the work.

Total volume of work: 103 pages, 55 figures, 7 tables, 15 references.

Keywords: BPMS, business process, orchestration, microservice, Camunda, jBPM, web service, notation, Fiware, IoT.

Зміст

Перелік умовних позначень і термінів

BPM - Business process management, управління бізнес-процесами

BPMS - Business process managemet system

SOA - Service Oriented Architecture

REST - Representational State Transfer

API - Application Programming Interface

HTTP - Hyper Text Transfer Protocol

DMN - Decision Model and Notation, нотація прийняття рішень

jBPM Assets - Все, що можна зберегти як версію у сховищі артефактів Business Central, є активом (asset). Сюди входять правила, пакети, бізнес-процеси тощо.

GE - Global Enabler

IoT - Internet of Things

Вступ

Бізнес-процеси - це невід’ємна частина будь-якої компанії. Ці процеси об’єднують системи, партнерів і співробітників для досягнення спільних стратегічних і тактичних цілей. Велика кількість компаній звертають увагу на веб-сервіси і сервіс-орієнтовану архітектуру для вирішення проблем інтеграції, що виникають при з'єднанні додатків. Стандарти в області Web-сервісів пропонують відкритий, переносимий і стандартизований спосіб вирішення типових проблем розвитку додатків. Вони дозволяють створювати рішення, які забезпечують гнучкість бізнесу при максимальному використанні вже задіяних ресурсів та мінімізації вартості розгортання нових додатків.

Бізнес-процес дозволяє змоделювати ваші бізнес-цілі, описуючи кроки, які необхідно виконати для досягнення вашої мети та вимог, використовуючи блок-схему. Це значно покращує видимість та продуктивність вашої бізнес логіки.

Бажання мати у своєму розпорядженні адаптивні бізнес-процеси, які можуть бути тонко налаштовані й оптимізовані, щоб відповідати постійно змінюваним умовам бізнесу, нормативним вимогам законодавства і тиску конкуренції, призвело до появи систем управління бізнес-процесами (Business Process Management systems).

Використовуючи BPMS, ми не просто малюємо абстрактні схеми по намальованій діаграмі, які буде виконувати наш бізнес-процес. Те, що ми бачимо на схемі, гарантовано корелює з тим, як працює процес, які мікросервіси використовуються, які параметри, за якими таблицями рішень відбувається вибір тієї чи іншої логіки.

В результаті процеси, сервіси та розробка стають:

- швидко читаємі;
- самодокументовані (працюють саме так, як намальовані, і немає дисинхронізації між документацією і реальною роботою коду);

- просто налагоджувані (легко подивитися, як проходить той чи інший процес, і зрозуміти, в чому помилка).

Метою цієї дипломної роботи є дослідження систем Camunda BPM та jBPM в якості оркестраторів мікросервісів. Вона має надати розуміння та продемонструвати, як ці системи можуть пришвидшити та спростити процес налаштування комунікації між мікросервісами. Робота має на меті аналіз та порівняння можливостей систем, які вони надають розробникам та аналітикам.

1. Аналіз підходів управління бізнес процесами

1.1 Business Process Management

BPM - концепція управління організацією, яка розглядає бізнес-процеси як особливі ресурси підприємства, безперервно адаптована до постійних змін, і яка покладається на такі принципи, як зрозумілість і наочність бізнес-процесів в організації за рахунок їх моделювання з використанням формальних нотацій, використання програмного забезпечення моделювання, симуляції, моніторингу та аналізу бізнес-процесів, можливості динамічного перестроювання моделей бізнес-процесів силами учасників і засобами програмних систем.

BPMS працюють за наступними правилами:

1. Розробка стає візуальної та процесною:

BPMS дозволяє створити бізнес-процес, в якому команда проекту (розробник або бізнес-користувач) визначає послідовність запуску мікросервісів, а також умов і гілок, за якими він рухається. При цьому один бізнес-процес (послідовність дій) може включатися в інший бізнес-процес.

2. Кожен сервіс має зрозумілі входи і виходи

BPMS спонукає писати більш абстрактно. Розробка ведеться саме процесно, з визначенням входу і виходу.

3. Паралелізація обробки екземплярів процесу

Якщо нам раптом буде потрібно прочитати логи по обробці конкретного замовлення, в Camunda, JBoss або будь-який інший BPMS, ви зможете повністю відновити всі дані і побачити, в якому екземплярі процесу це було і з якими параметрами входу / виходу.

4. Візуальна послідовність дій

У BPMS всі переривання і виключення - на боці BPMS. Ви не перевантажуєте код цією логікою (а саме наявність такої логіки в коді зробила б мікросервіси великими і незручними для перевикористання в інших процесах).

Існує безліч сценаріїв використання технологій BPM, однак наступні сценарії використання є основними:

- аналіз бізнес-процесів;
- моделювання процесів;
- проектування архітектури бізнес-процесів;
- імітаційне моделювання;
- управління даними;
- проектування і зберігання бізнес-правил;
- виконання бізнес-правил;
- інтеграція додатків;
- виконання процесу;
- вимірювання характеристик процесу.

1.2 Рівні бізнес-процесів

Бізнес-процеси компанії прийнято ділити на рівні:

Верхній рівень - відображає взаємодію основних напрямків діяльності компанії, наприклад, виробництва, маркетингу, управління фінансами і так далі. Тобто фактично показує роботу компанії наближено, не заглиблюючись в деталі. Нижній рівень - містить процеси, що відображають послідовність виконання більш конкретних операцій, наприклад, закупівлі матеріалів на склад, виготовлення будь-якої деталі, звільнення або підбір кадрів на посаду.

Бізнес-процеси верхнього рівня «декомпозуються» на процеси нижнього рівня, уточнюючи більш конкретно кожен напрямок діяльності, наприклад, процес верхнього рівня «Виробництво» може бути декомпозований на кілька процесів нижнього рівня: «Виробництво меблів на замовлення», « виробництво стандартних корпусних меблів »і т.д.

1.3 Цикл процесу управління бізнес процесами

Кожен процес в компанії повинен пройти через наступний цикл. В такому

випадку можна буде сказати, що процес керований і знаходиться на стадії «Удосконалення».

1. Проектування:

На даному етапі існує 2 основні завдання: визначити як бізнес процес існує на даний момент і зрозуміти яким ми хочемо його бачити в подальшому. Головне - створити наочну конструкцію або дизайн процесу. Потім вже можна переглянути основні моменти і позначити, що б нам хотілося змінити.

2. Моделювання бізнес процесів:

На підставі раніше складеного проекту створюємо модель. Модель - це та ж конструкція, що ми вже створили, але з додаванням різних змінних.

Припустимо, в моделі можна вказати, що певна операція займає не 1 годину, а тільки 10 хв. Як це вплине на інші операції? Що потрібно змінити, щоб це стало можливим? І т.д. Важливо не забути додати всілякі варіанти розвитку процесу. Таким чином ми створюємо оптимальне співвідношення витрат і результатів.

3. Виконання:

Якими б чудовими не були бізнес процеси на папері, в реальності завжди знайдуться «але». Ці «але» можна виявити тільки одним шляхом - виконати процес практично. А якщо точніше, то запустити його в роботу і уважно спостерігати. Без цього етапу ніяк не обійтися. Тому вкрай важливо приділити достатньо часу на етапах проектування і моделювання - це дозволить знизити ризики.

Ще важливо мати підстрахування на непередбачений випадок. Наприклад, якщо ви розробляєте нову систему CRM, то не впроваджуйте її відразу у всій компанії. Дайте для початку одному з фахівців попрацювати з нею. І нехай у нього буде можливість перейти в стару систему в разі виникнення непередбачених складнощів.

4. Моніторинг бізнес процесів

Це спостереження і відстеження процесів. Організувати моніторинг необхідно таким чином, щоб можна було легко отримати інформацію про хід та

результати процесу. При цьому необхідно вести статистику для можливості зміни показників у часі. Простий доступ до інформації забезпечує можливість виправити щось вчасно. Наприклад, якщо замовлення клієнта «застрягло» на одній зі стадій процесу, моніторинг дозволить не тільки побачити це, але і своєчасно втрутитися і внести поправки в процес. Те, наскільки детально відбувається відстеження процесів, залежить як від самих процесів, так і від ваших цілей. Якщо процес новий і тільки почав роботу, необхідний детальний моніторинг. Так само детальність залежить від можливостей всієї системи. Витрати, які ви несете на отримання інформації, не повинні перевищувати цінності цієї інформації.

5. Оптимізація бізнес процесів.

На даному етапі ви аналізуєте дані, отримані в результаті моніторингу, проектування і моделювання. Метою такого аналізу є отримання інформації про можливості поліпшення. Для того щоб зрозуміти, що необхідно покращувати, необхідно виявити «вузькі місця» процесу. Такі місця визначають продуктивність всього процесу, це найслабша ланка.

Наприклад, якщо ви робите 100 одиниць продукції, а продавати можете лише 80, продаж є цим вузьким місцем. Немає сенсу виготовляти більше, ніж ви продаєте.

1.4 Основні нотації моделювання бізнес процесів

1.4.1 BPMN

Як стандарт для моделювання бізнес-процесів отримує визнання BPMN, розроблена організацією Business Process Modeling Initiative.

Основне призначення BPMN полягає в наданні нотації, легкої у використанні і розумінні для бізнес-користувачів, включаючи бізнес-аналітиків, що моделюють бізнес-процеси, технічних розробників, які створюють системи для виконання цих процесів, і менеджерів різних рівнів, які повинні швидко читати і розуміти процесні діаграми, щоб приймати рішення.

BPMN прямо відображається на мови виконання бізнес-процесів, такі як BPEL і BPML. BPMN надає нотацію для моделювання, а BPEL є мовою опису виконання процесів.

BPMN надає нотацію моделювання, яка забезпечує перехід від бізнес-вимог до карти виконання процесу.

Нотація BPMN розширювана і дозволяє використовувати асоціації та анотації для встановлення взаємовідносин з іншими артефактами всередині або поза вашою системою. Наприклад, можна співвіднести бізнес-процеси з функціями, які вони виконують, з даними, які вони використовують, з системами, на яких вони розгорнуті, і т.д.

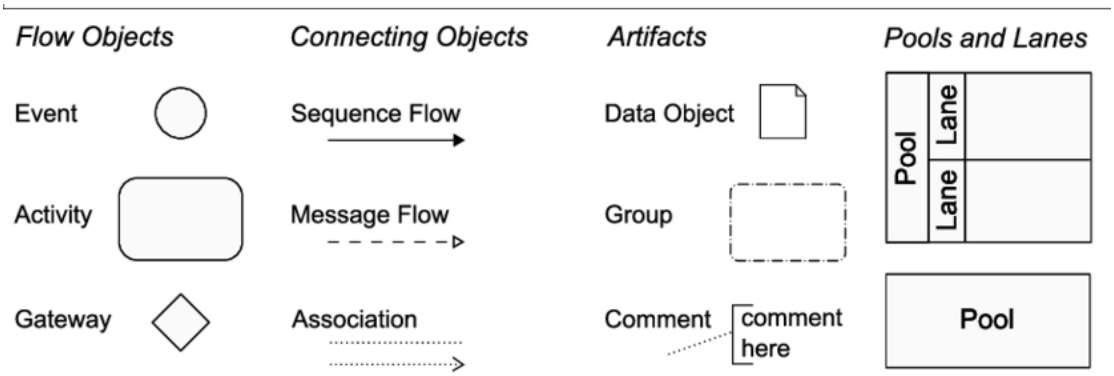

Рисунок 1.1 - Основні елементи BPMN

Об'єкти потоку: представляють основні елементи діаграми бізнес-процесів.

З’єднання об'єктів: використовуються для з'єднання основних об'єктів BPMN.

Артефакти: дозволяють розробникам процесів розширити базову нотацію BPMN, включивши додаткову інформацію про процес на схему процесу.

Доріжки: це механізм для організації діяльності та відповідальності за технологічною схемою.

Як вже було зазначено, об'єкти потоку - це фігури, що представляють основні елементи діаграми бізнес-процесів, включаючи:

- Діяльність: будь-яка робота, яка виконується в процесі.
- Подія: все, що "відбувається" під час бізнес-процесу.
- Шлюз: використовується для управління потоком процесу.

На наступній схемі зображено зразок “процедури купівлі” BPMN 2.0, що включає початок події, деякі діяльності, шлюз і кінцеву подію:

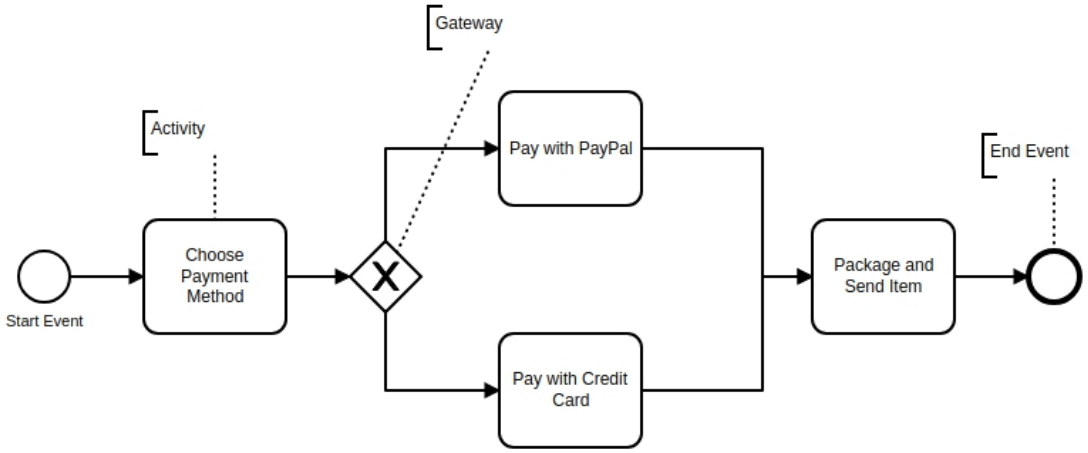

Рисунок 1.2 - BPMN Діаграма процесу купівлі

Якщо ви хочете бачити, яка роль відповідає за конкретну задачу, ви можете скористатися доріжками. Ось, наприклад, як зобразити вищезазначений процес з використанням доріжок:

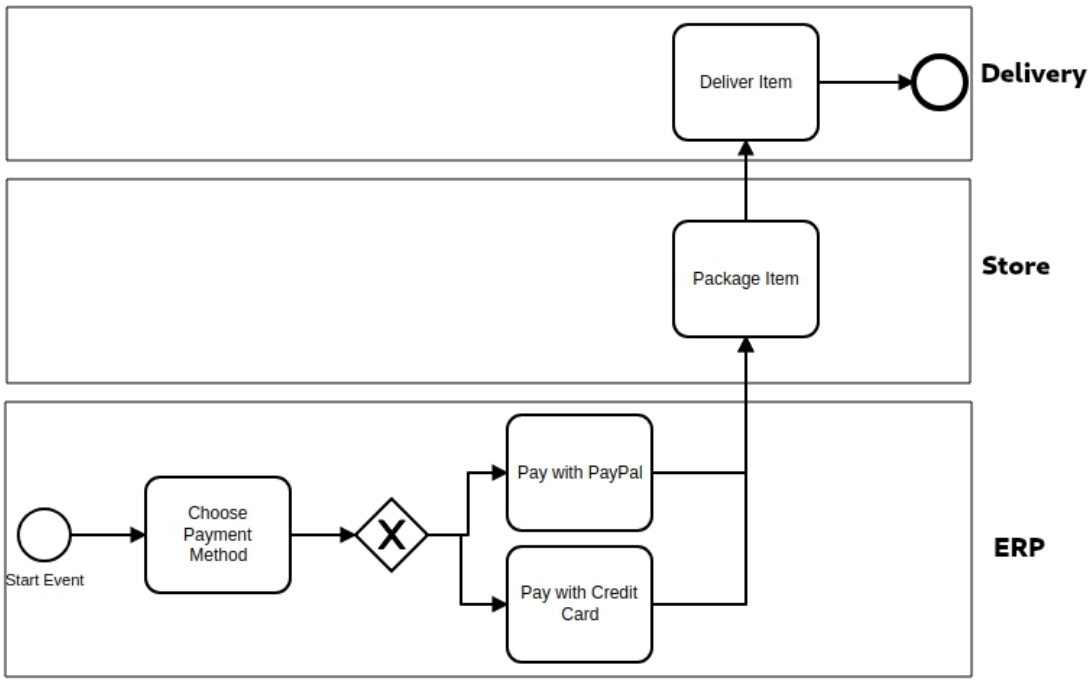

Рисунок 1.3 - BPMN діаграма процесу купівлі з використанням доріжок

1.4.2 DMN

DMN - нотація прийняття рішень. DMN можна використовувати, щоб явно описувати з яких міркувань приймаються рішення. Як і в BPMN, DMN має 2 способи застосування:

Аналітичний - коли ми просто складаємо таблиці рішень в наших програмах, а розробники їх програмують.

Автоматизований - коли наші таблиці виконуються DMN-движком в тому вигляді, в в якому намальовані.

DMN придумала в 2015 році та ж команда, що і BPMN - ці 2 нотації відмінно поєднуються один з одним. Але можуть використовуватись і окремо.

Основна ідея DMN полягає в тому, що рішеннями можна так само управляти, як і іншими частинами своїх додатків. Наприклад, у вас є рішення з розрахунку премії. Ви можете змінювати тільки його, коли змінюється спосіб розрахунку цієї премії, не чіпаючи інші елементи програми.

Рішення складається з:

1. назви;
2. методу спрацьовування. Якщо там стоїть U, то значить у нас політика спрацьовування - unique. Тобто логіка таблиці повинна завжди віддавати одне, унікальне рішення (перетин не допускається);
3. вхідних змінних;
4. вихідних даних - для кожної можливої вхідної змінної ми визначаємо вихідну змінну;
5. правила - це набір вхідних і вихідних даних, рядок таблиці. У кожного правила є номер, він вказаний в таблиці зліва;
6. анотацій - це текст праворуч, він використовується для пояснення правила і не автоматизується.

Beverages Show details

C	Input +		Output +	Annotation
	Dish	Guests with children	Beverages	
1	"Spareribs"	-	"Aecht Schlenkerla Rauchbier"	Tough Stuff
2	"Stew"	-	"Guinness"	-
3	"Roastbeef"	-	"Bordeaux"	-
4	"Steak", "Dry Aged Gourmet Steak", "Light Salad and a nice Steak"	-	"Pinot Noir"	-
5	-	true	"Apple Juice"	-
6	-	-	"Water"	-
+	-	-	-	-

Рисунок 1.4. - DMN таблиця рішень

BPMN без бізнес-правил може виглядати так:

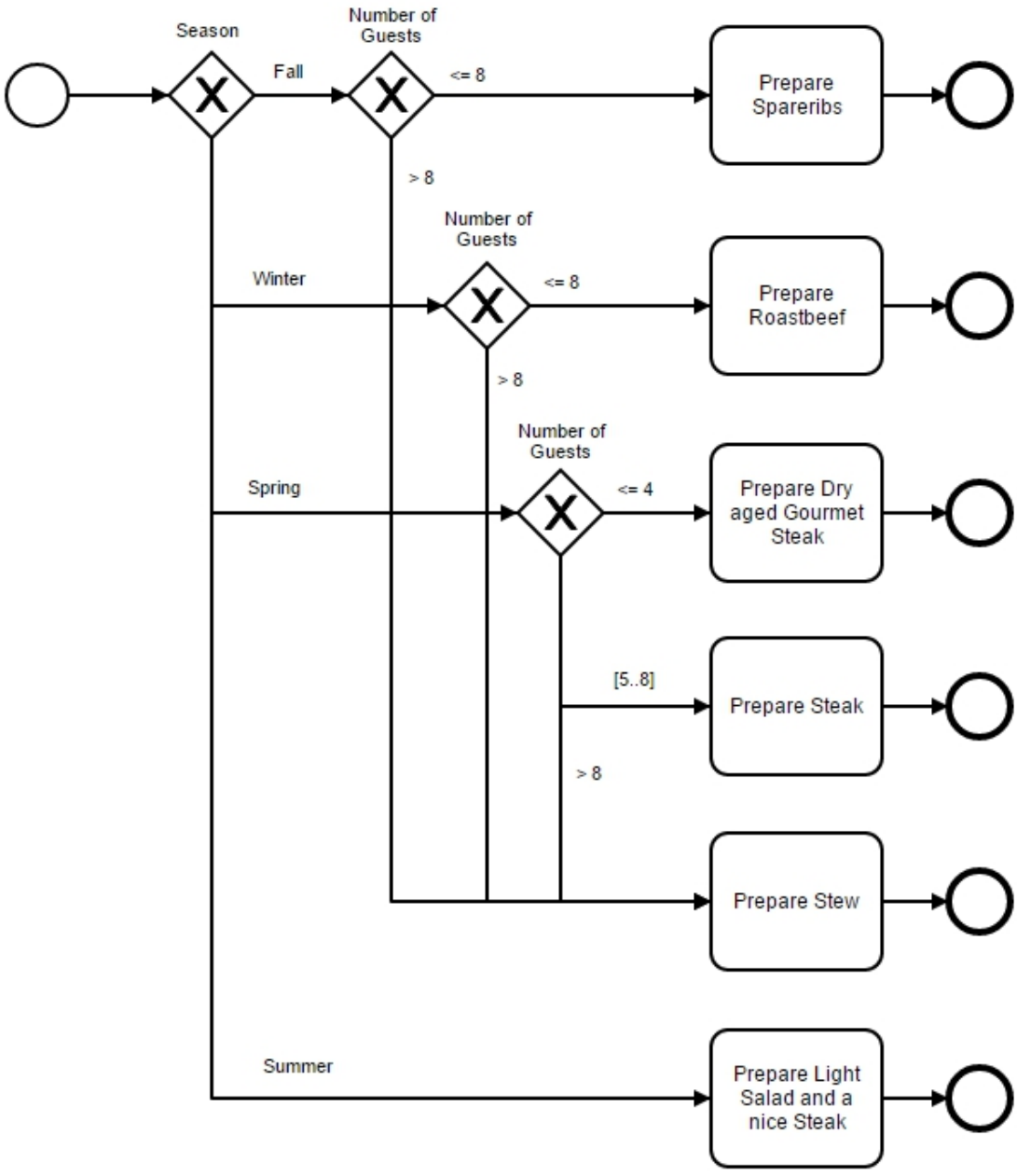

Рисунок 1.5 - BPMN процес без використання DMN

А з DMN може виглядати так:

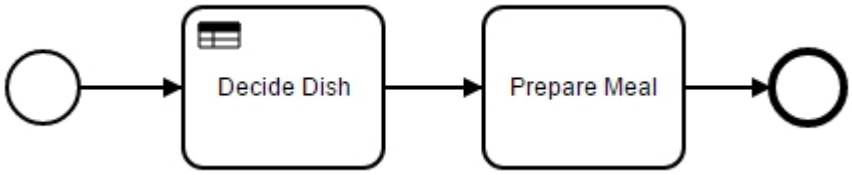

Рисунок. 1.6 - BPMN процес з використанням DMN

За допомогою таблиць рішень можна значно поліпшити ваші BPMN-схеми.

1.5 Порівняння BPMN з іншими нотаціями

1.5.1 Нотація типу IDEF

Для процесів верхнього рівня зазвичай використовується методологія IDEF. Це методологія моделювання, що дозволяє створити функціональну модель, яка відображає структуру і функції системи, а також потоки інформації і матеріальних об'єктів, що зв'язують ці функції.

Особливості нотації:

- можливість декомпозувати процеси на підпроцеси і, таким чином, будувати ієрархічні моделі бізнес-процесів.
- виділення чотирьох типів стрілок: три типи входів - вхідні дані, керуючі елементи і механізми (це дозволяє більш гнучко описувати логіку використання входів в процесі з метою подальшого аналізу), і вихід.

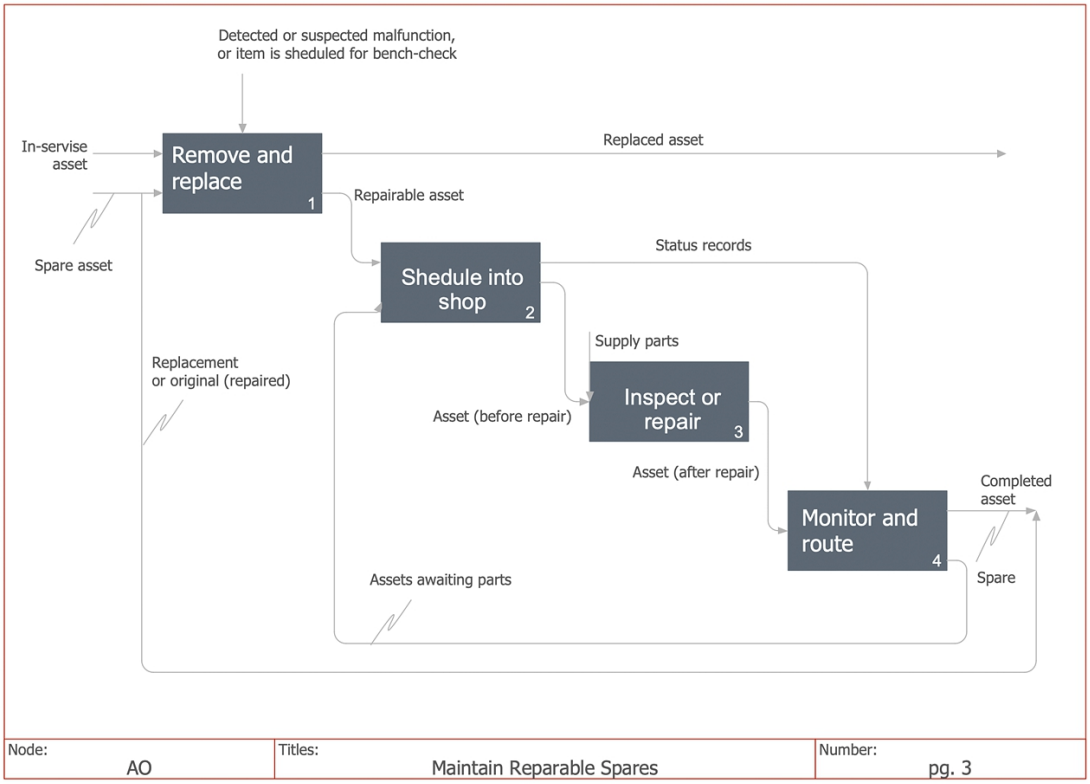

Рисунок 1.6 - IDEF нотація

1.5.2 Нотація EPC

Дана нотація використовується для представлення алгоритму виконання процесу (нотація класу workflow). Діаграма, описана в нотації EPC (ланцюжок подій процесів), являє собою впорядковану комбінацію подій і функцій. Для кожної функції можуть бути визначені початкові і кінцеві події, учасники, виконавці, матеріальні та документні потоки, які супроводжують її. В нотації EPC розгалуження стрілок здійснюється з використанням операторів.

Нотація EPC підтримує декомпозицію на більш низькі рівні. Діаграма декомпозованої функції EPC може бути описана тільки в нотаціях EPC або BPMN 2.0.

Нотацію EPC можна застосовувати для моделювання окремих процесів компанії, а також на нижньому рівні моделі бізнес-процесів, створеної в нотації IDEF.

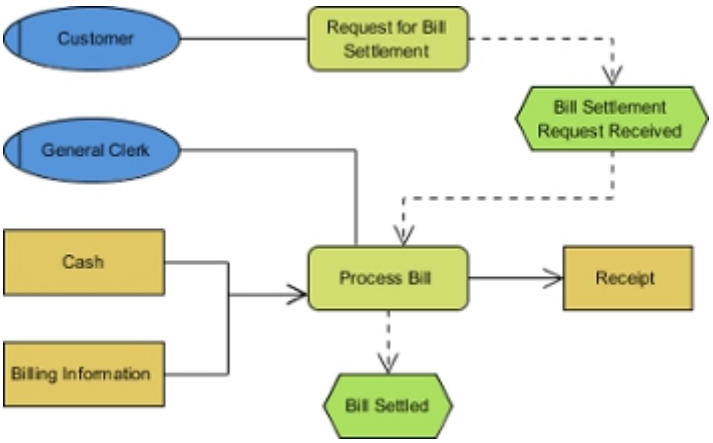

Рисунок 1.7 - EPC нотація

1.5.3 Нотація bpmn 2.0

Нотація bpmn 2.0 відрізняється тим, що:

- містить не тільки дії, а й події (причому виділяє різні типи подій);
- містить вказівку на документи і сховища даних;
- може використовуватись для "запуску", наприклад, в системах управління бізнес процесами;
- вона розрізняє, коли дія виконується людиною, а коли - системою, тобто її зручно використовувати при автоматизації.

Вона трохи складніше для сприйняття, але дозволяє висловити більше нюансів. Її рекомендується використовувати тоді, коли планується автоматизація процесів.

Таблиця 1.1 - Порівняння нотацій моделювання бізнес-процесів

Критерії	IDEF0	EPC	BPMN
Обмеження кількості блоків	+	-	-
Відображення часової послідовності робіт	-	+	+
Відображення входів і виходів процесу	+	+	+
Відображення організаційної	+	+	-

структури			
Складність вивчення нотації	Висока	Низька	Низька
Наочність діаграм	Низька	Висока	Висока

1.6 Висновки до розділу 1

У розділі було розглянуто основи моделювання бізнес-процесів, а саме: рівні бізнес-процесів, цикл робробки, нотації моделювання. Було проведено аналіз популярних нотацій та різниці між ними між ними за рядом критеріїв. На основі наведеної інформації, можна зауважити, що BPMN добре підходить для моделювання та автоматизації корпоративних бізнес-процесів.

2 Fiware та порівняння Camunda BPM та jBPM

2.1 Camunda

Camunda - це open-source-платформа для моделювання бізнес-процесів, яка написана на Java і в якості мови розробки використовує Java. Вона являє собою набір бібліотек, які дозволяють виконувати описані процеси. Для інтеграції Camunda в проект досить додати кілька залежностей. Для зберігання процесів можна вибрати in-memory або персистентну СУБД - в залежності від завдань. За замовчуванням платформа розгортається на H2.

Розробка складається з двох частин: створення моделі процесу в спеціальній утиліті Camunda Modeler і написання java-коду, який обробляє кроки процесу, описані на діаграмі. Щоб з процесу викликати java-код, досить реалізувати інтерфейс JavaDelegate, підняти цей Bean в контексті і вказати його id на потрібному етапі процесу. Логіка роботи делегатів досить проста: щось вичитали з контексту, виконали якісь дії і поклали назад в контекст.

2.1.1 Компоненти системи

Camunda Modeler - крос-платформенний додаток з графічним інтерфейсом, в якому бізнес-аналітик або архітектор процесу може моделювати і / або редагувати схеми процесів. Підтримує нотації BPMN 2.0, DMN 1.1, CMMN 1.1.

Camunda Tasklist - веб-додаток, через який кінцеві користувачі можуть зайти, подивитися які завдання на них призначені і виконати якісь дії.

Camunda Cockpit - веб-додаток, який створено для операторів або власників бізнес-процесів, що забезпечує моніторинг і управління бізнес-процесами.

Camunda Admin - веб-додаток, інтерфейс адміністратора системи, в якому йде управління ролями, користувачами, групами і т.д.

Центральна частина системи Camunda BPM - це движок, який керує процесом відповідно до намальованих схем і нотацій, і перетворює їх в код, а потім - виконує.

Над движком існує надбудова, яка реалізує REST або Java API. REST / Java API - програмний інтерфейс системи, що забезпечує взаємодію як із зовнішніми (REST), так і внутрішніми (Java) компонентами. REST API досить великий, зручний і дозволяє реалізовувати всілякі власні додатки.

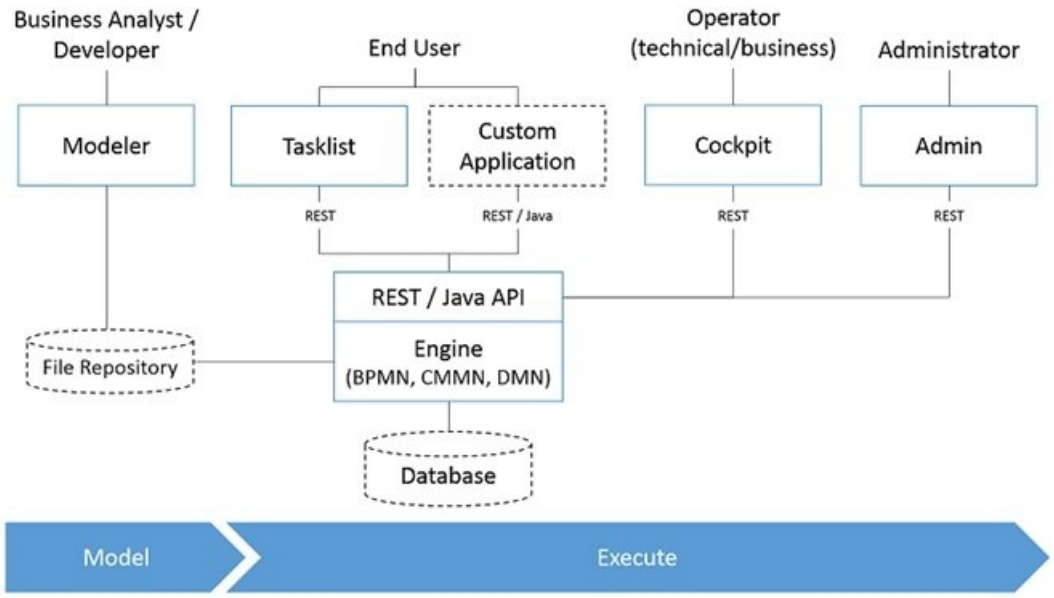

Рис. 2.1 - Архітектура Camunda Community

2.1.2 Способи розгортання Camunda BPM

Як згадувалося раніше, Camunda BPM написана на технологічному стеці Java, тому у неї є багато способів розгортання.

Перший і найпростіший - вбудована бібліотека в додатку.

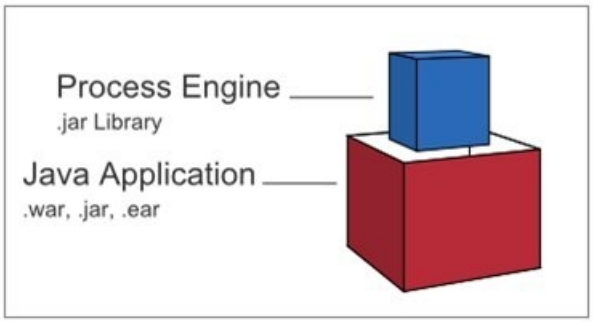

Рисунок 2.2 - Вбудована бібліотека

Другий спосіб теж підходить для Java додатків - сервіс всередині сервера додатків або контейнера сервлетів, який можуть використовувати інші додатки, розгорнуті в контейнері.

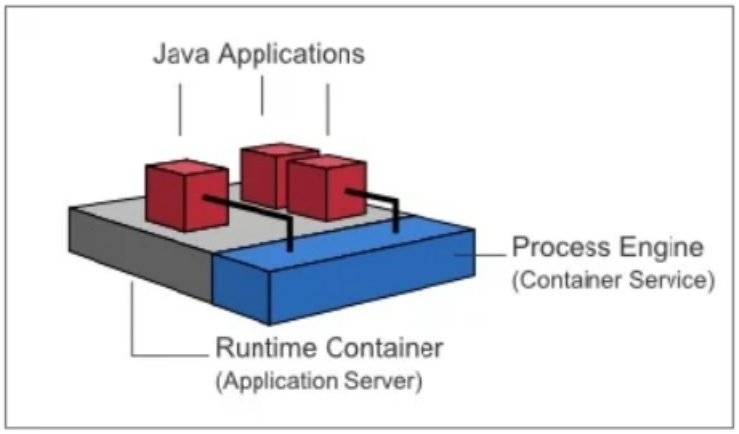

Рисунок 2.3 - Сервіс розгорнутий на сервері додатків

Третій спосіб, якщо додатки написані не на Java, розгорнути Camunda як окремий сервер і додатки будуть з ним контактувати через REST API віддалено.

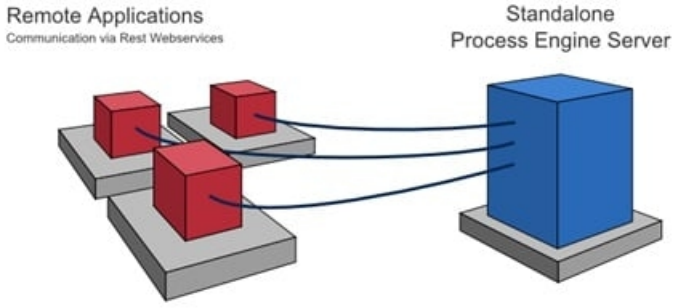

Рисунок 2.4 - Розгортання на окремому сервері

Четвертий спосіб для високонавантажених систем - кластерне розгортання, коли є загальна база даних і кілька нод з камундою.

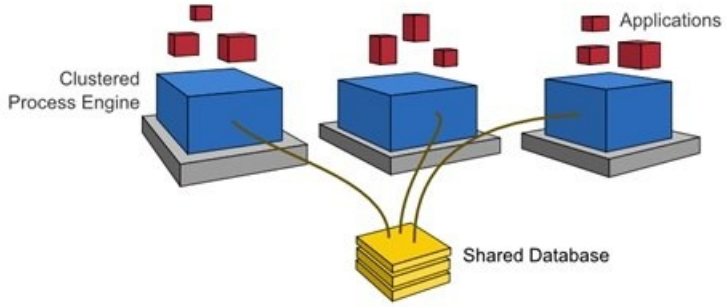

Рисунок 2.5 - Кластерне розгортання

2.2 JBPM

jBPM - це гнучкий пакет управління бізнес-процесами, легкий, повністю відкритий проект (поширюється за ліцензією Apache) і написаний на Java. Він дозволяє моделювати, виконувати та контролювати бізнес-процеси протягом усього їх життєвого циклу.

jBPM дозволяє ефективно взаємодіяти між бізнес аналітиками, розробниками, і кінцевими користувачами, пропонуючи функції і інструменти управління процесом таким чином, щоб ними було зручно користуватися бізнес-користувачам і розробникам.

Ядро jBPM легке, написано на Java, дозволяє запускати бізнес процеси використовуючи нотацію BPMN 2.0. Ядро jBPM можна запустити в будь-якому Java середовищі, вбудувати в додаток або запустити як сервіс.

На додаток до основного движку пропонується безліч функцій і інструментів для підтримки бізнес-процесів протягом всього життєвого циклу:

- підтримка Eclipse і веб редактора для графічного створення бізнес процесів;
- служба завдань потребуючих людського втручання;
- консоль управління, що підтримує управління екземпляром процесу, списки

завдань і управління формою завдання, а також звітність;

- додатковий репозиторій для розгортання вашого процесу;
- ведення журналу (для запитів / моніторингу / аналізу);
- інтеграція з Spring і тд.

jBPM пропонує середовище розробки, виконання, управління та моніторингу для підтримки повного життєвого циклу проектів для автоматизації бізнесу.

Ключовою існуючою функцією jBPM є проект Drools: можливість впровадження перевірки бізнес правил. Можна розробити традиційні правила і навіть реалізувати складні сценарії обробки подій (СОП). СОП - це можливість зіставлення фактів з подіями на основі обробки в режимі реального часу.

Цей інструмент автоматизації бізнес-процесів забезпечує гнучкість виконання програми. Архітектор може вибрати один із трьох режимів виконання для запуску бізнес-додатків:

- як незалежні служби поверх платформ сервера додатків, такі як, наприклад, WildFly та інші сервери додатків Java EE;
- як автономна програма Java;
- як вбудований та виконуваний формат uber jar за допомогою Spring Boot.

2.2.1 Компоненти системи

jBPM складається з двох основних компонентів, Business Central та Kie Server.

Business Central - це веб-інструмент, який використовується як IDE розробки з можливостями управління процесами, завданнями та моніторингу.

Kie Server - це механізм, написаний на Java, відповідаючий за виконання, збірку та розгортання бізнес-додатків (kjars). Це механізм прийняття рішень, вирішення проблем та механізм робочого процесу, який є надійним, масштабованим та готовим до хмарних додатків.

Хоча Business Central відповідає за створення та моніторинг бізнес-

додатків, Kie Server є движком, який їх виконує.

Business Central використовується для створення, управління та моніторингу бізнес активів (assets). На етапі розробки цей веб-інструмент може використовуватися розробниками, бізнес-аналітиками та спеціалістами з автоматизації бізнесу для створення проектів, моделей, процесів та правил.

Інтерфейс дозволяє з мінімальною кількістю коду створювати моделі даних, форми завдань, процеси та правила. Використовуючи цю стратегію розвитку, як Business Central, так і розробники, використовуючі IDE, можуть створювати нові версії активів у сховищі git та сховищі артефактів.

Починаючи з jBPM 5, коли проект Drools та проект jBPM почали взаємодіяти разом, було визначено нову назву групи проектів: Знання - це все (Knowledge - is everything) - KIE. Пізніше назва KIE поширилася на сховища git, архетипи Maven, назви класів та весь інший відповідний код, загальний для Drools та jBPM.

Проект kie - це проект Java, заснований на maven, який компілюється у kjar. Kjar - це простий файл jar, налаштований за допомогою <packaging> kjar </packaging> у налаштуваннях maven, всередині (pom.xml) проектів kie.

Цей розгортуваний пакет містить моделі Java, а також файл дескриптора метаданих kmodule.xml.

Виконання бізнес-процесу в jBPM відбувається наступним чином:

Коли будь-який компонент середовища виконання намагається виконати kjar, він спочатку перевіряє конфігурації всередині kmodule.xml. Необов’язково існують також інші дескриптори розгортання для визначення додаткових конфігурацій для механізму виконання.

Потім клієнтська програма зможе зв’язуватися та використовувати активи за допомогою API, наданого Kie Sessions. Kie Session відповідає за інформацію про виконання бізнес-активу.

Коли додаток хоче, наприклад, розпочати бізнес-процес, він взаємодіє з Kie Session, щоб сказати “Kie Session, запустити екземпляр процесу на основі визначення процесу з назвою “ProcessName” версії 1.0.0, розгорнутого у kie

контейнері з псевдонімом “NameApplication”.

Визначення процесу - це файл *.bpmn2, послідовність елементів, що представляє визначення. Використовується для створення багатьох екземплярів процесу. Екземпляр процесу - одне виконання процесу. Багато екземплярів процесу можна створити на основі визначення процесу.

Кожне розгортання kjar стає доступним через контейнер Kie. Кожен контейнер Kie має лише одне розгортання kjar. Зазвичай його ідентифікує розгорнутий проект GAV (група, ідентифікатор артефакту, версія) або псевдонім.

2.3 Порівняння Camunda та jBPM

Деякі продукти BPM мають власний механізм правил, тоді як інші забезпечують інтеграцію / плагіни для часто використовуваних механізмів правил. Drools - це загальноприйнятий механізм правил, який використовується в галузі. jBPM інтегрує Drools у свій проект, тоді як Camunda застосовує інший підхід та забезпечує інтеграцію з Drools. Обидві системи дозволяють використовувати задання бізнес-правила як частину робочого процесу, різниця лише в інтеграції, що стоїть за заданням бізнес-правила.

Продукти BPM, як правило, надають інтерфейс користувача для проектування / моделювання, що дозволяє користувачам створювати діаграми процесів. Деякі надають плагіни eclipse для розробників, які воліють працювати в IDE. Більшість продуктів BPM мають виконаний інтерфейс користувача, який дозволяє користувачам переглядати та обробляти призначені їм завдання. Camunda надає чотири окремі інтерфейси - моделювання, список завдань, адміністрування процесів та моніторинг. Це інший підхід, ніж у jBPM, оскільки вони пропонують єдиний веб-додаток для розробки та виконання - jBPM Workbench (Business Central). Додатковою відмінністю інтерфейсу Camunda modeler є те, що він наразі доступний лише як декстопна програма. З точки зору плагіна Eclipse, Camunda більше не розвиває його, оскільки вони вирішили від’єднатись від Eclipse, щоб усунути необхідність оновлення плагіна з кожним новим релізом.

На відміну від Camunda, jBPM пропонує однакові можливості в community та enterprise версіях.

Враховуючи важливість API в сучасній індустрії програмного забезпечення, можливість здійснення REST-викликів із додатків є критично важливою. Обидва продукти забезпечують можливість здійснення викликів REST API. jBPM надає нестандартне сервісне завдання (service task) REST, тоді як Camunda вимагає додаткової розробки для реалізації виклику REST (власні класи Java). Перевагою наявності готового сервісного завдання REST є те, що він скорочує час розробки, заснований на конфігурації, а не вимагаючи власної розробки.

Camunda та jBPM підтримують вбудовані та автономні режими розгортання. Вбудоване розгортання дозволяє запускати BPM Engine як частину існуючої програми (у межах тієї самої JVM). Це може забезпечити переваги продуктивності, оскільки дані передаються через пам’ять замість мережевих викликів. Автономне розгортання надає різні функції API, які клієнт може викликати через REST. Це може бути корисно, якщо ви хочете повторно використовувати один екземпляр робочого процесу в різних програмах. Це також може бути способом інтеграції програми в механізм робочого процесу, яка може бути написана мовою, яка відрізняється від API продукту BPM.

Camunda Cockpit - це розгалужена веб-програма для операційної діяльності над процесом. Завдяки модульній архітектурі додатку, ви може розробляти окремі плагіни, які можна інтегрувати як віджети в інтерфейс користувача. jBPM не надає подібних можливостей.

Таблиця 2.1 - Порівняння можливостей Camunda та jBPM

	Camunda	jBPM
Підтримка нотацій	підтримує нотації моделювання бізнес-процесів і світові стандарти BPMN 2.0, CMMN 1.1, DMN 1.1.	фокусується на специфікації BPMN 2.0 - глобальному стандарті в області моделювання бізнес-процесів. Підтримує CMMN 1.1, DMN 1.1, DRL.
Можливості	− моделювання бізнес-процесів; − управління бізнес-процесами; − інтеграція з електронною поштою; − управління завданнями; − BRMS; − відображення даних; − аналіз бізнес-процесів; − відображення показника ефективності бізнес-процесів.	
Компоненти системи	платформа Camunda складається з набору додатків (Modeler, Tasklist, BPMN Engine, DMN Engine, Cockpit, Admin) і включає в себе інструментарій для моделювання та	складається з двох основних компонентів, Business Central та Kie Server.

	виконання бізнес-процесів.	
Технічні особливості	підтримує останню версію Java і будь-яку з JVM-мов програмування: Kotlin, Scala і т. д.	jBPM написана на Java, випущена під ліцензією Apache License 2.0. Працює за допомогою веб-редактора або плагіна Eclipse.
Можливість масштабування	спроектована для роботи в кластері і тому легко масштабується.	jBPM також легко масштабується.
Продуктивність	має більш високу продуктивність і в цілому швидше за кількістю щойно розпочатих бізнес-процесів.	якщо говорити про масштабні впровадження, jBPM також підходить для високонавантажених проектів.

2.4 Fiware

FIWARE - набір стандартизованих API для підтримки та створення розумних додатків в різних сферах. FIWARE також рекламується як ідеальний вибір для створення розумних міст. Це сукупність публічних і безкоштовних специфікацій API, які додаються до реалізацій з відкритим кодом. Також існує ініціатива під назвою FIWARE Lab, яка пропонує хмарну платформу. Тоді як FIWARE Lab призначена лише для тестування, експериментів та оцінки, передбачається FIWARE iHub, який надає хмарні послуги, готові до виробництва, у майбутньому. Платформа FIWARE згрупована в сім основних частин, що називаються «основні активатори (general enablers - GE)». Кожен GE представляє певний аспект послуг FIWARE, а також надає один або кілька компонентів разом з реалізаціями, які підтримують зазначене API.

GE організовані наступним чином:

• Керування даними / контекстом: містить усі компоненти, необхідні для зберігання, доступу, обробки та аналізу даних як частини розумного додатку.

• Послуги Інтернету речей: усі необхідні компоненти для налаштування сенсорних мереж та датчику маршрутизації даних до інших GE.

• Розширений веб-інтерфейс користувача: компоненти для проектування користувальницьких інтерфейсів, включаючи географічну інформацію та інтерактивні 3D-діаграми.

• Безпека: компоненти для додавання та посилення захисту.

• Розширене проміжне програмне забезпечення та інтерфейси для мереж та пристроїв.

• Програми / послуги та доставка даних: компоненти та інструменти для візуалізації даних.

• Хмарний хостинг: компоненти та інструменти, що надають та керують хмарними послугами FIWARE через хмарну інфраструктуру.

Додатки потребують механізму для зберігання даних, як правило, цей рівень забезпечується базою даних. У випадку FIWARE існує послуга, яка

називається контекстний брокер. Він спирається на технологію MongoDB з відкритим кодом. Сутності можуть бути створені у брокері Orion Context і ними можна керувати за допомогою запитів API, за допомогою curl або Postman.

Контекстний брокер - це по суті REST API, заснований на відкритій службі Next Generation Service Interface (NGSI).

Він технічно складається з бази даних MongoDB з NGSI REST та API поверх неї. Це дозволяє створювати всі необхідні сутності та не вимагає жодної схеми бази даних. Усі сутності зберігаються в нормалізованому вигляді в одній колекції MongoDB. Кожна сутність має щонайменше два поля для його ідентифікації. Один називається «тип» (наприклад, "Квартира"), а інший називається "id" (наприклад, “Top12”).

Контекстний брокер дозволяє використовувати декількох орендарів. Використовуючи заголовок «Fiware-Service», що може бути унікальними в межах однієї програми або домену, дані можуть бути красиво відокремленими. Таким чином, якщо додатки використовують різні значення для поля заголовка «Fiware-Service», їх запити ніколи не будуть заважати один одному.

Брокер контексту підтримує всі необхідні CRUD операцій. Однак за замовчуванням усі запити базуються на так званому "нормалізованому" позначенні, що означає що кожен запит повертає свій результат разом з метаданими типу поля. Але існує варіант, який називається ‘KeyValues’, який перемикається на звичайний JSON та робить протокол NGSI зручним для клієнтської програми. Це значно полегшує створення додатків, оскільки вони можуть просто використовувати REST API.

Одна з найпотужніших особливостей брокеру контекста - це можливість підписки на події. Це дозволяє зручно реагувати на зміни в сховищі даних. Підписки можуть мати різний обсяг, типу "інформувати мене щоразу, коли створюється новий об'єкт" або “Повідомте мене, коли температура в кімнаті 102 перевищує 25 градусів Цельсія”. Це значно полегшує реалізацію розумних додатків.

Локально розгорнути брокер контексту можна, описавши образ в докерфайлі та виконавши команду: docker-compose -p fiware up -d.

```
version: "3.5"
services:
  orion:
    image: fiware/orion:${ORION_VERSION}
    hostname: orion
    container_name: fiware-orion
    depends_on:
      - mongo-db
    networks:
      - default
    ports:
      - "${ORION_PORT}:${ORION_PORT}"
    command: -dbhost mongo-db -logLevel DEBUG -noCache
    healthcheck:
      test: curl --fail -s http://orion:${ORION_PORT}/version || exit 1
      interval: 5s

  mongo-db:
    image: mongo:${MONGO_DB_VERSION}
    hostname: mongo-db
    container_name: db-mongo
    expose:
      - "${MONGO_DB_PORT}"
    ports:
      - "${MONGO_DB_PORT}:${MONGO_DB_PORT}"
    networks:
      - default
    volumes:
      - mongo-db:/data
    healthcheck:
      test: |
        host=`hostname --ip-address || echo '127.0.0.1'`;
        mongo --quiet $host/test --eval 'quit(db.runCommand({ ping: 1 }).ok ? 0 : 2)' && echo 0 || echo 1
      interval: 5s
```

```
networks:
  default:
    ipam:
      config:
        - subnet: 172.18.1.0/24
```

```
volumes:
  mongo-db: ~
```

```
# Orion variables
ORION_PORT=1026
ORION_VERSION=3.1.0

# MongoDB variables
MONGO_DB_PORT=27017
MONGO_DB_VERSION=4.4
```

Далі наведені можливі запити до брокеру контексту.

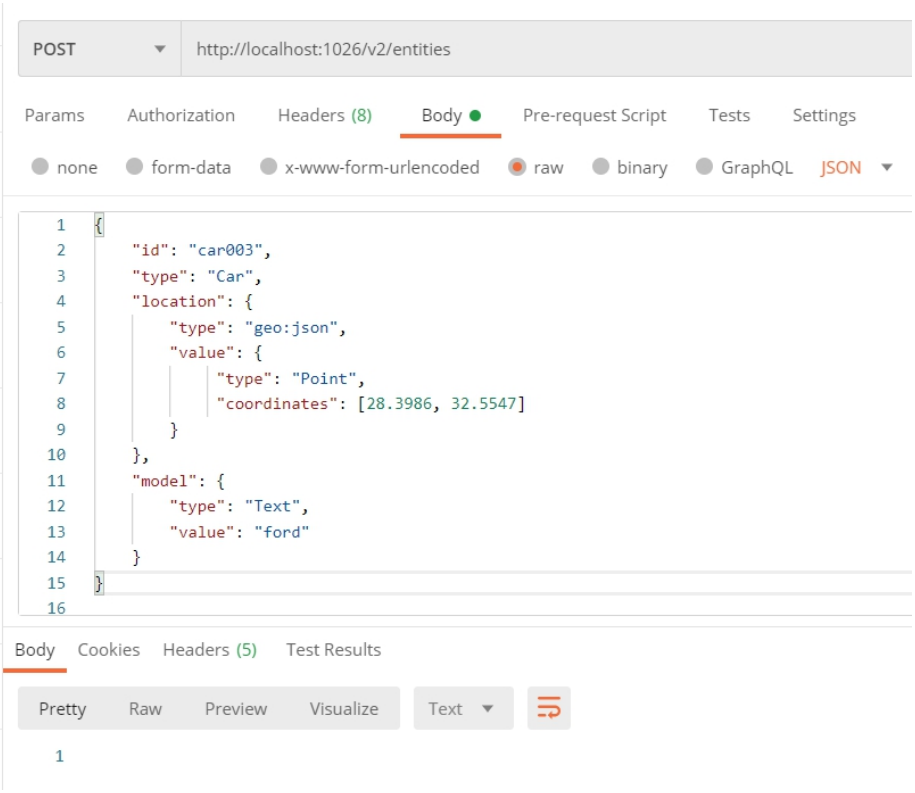

Рисунок 2.6 - Створення сутності

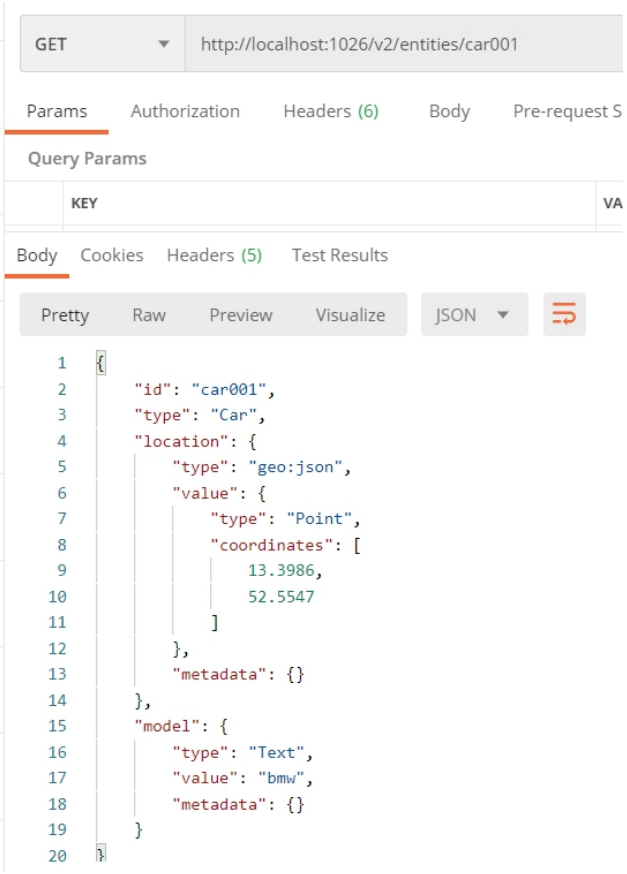

Рисунок 2.7 - Отримання сутності з контексту

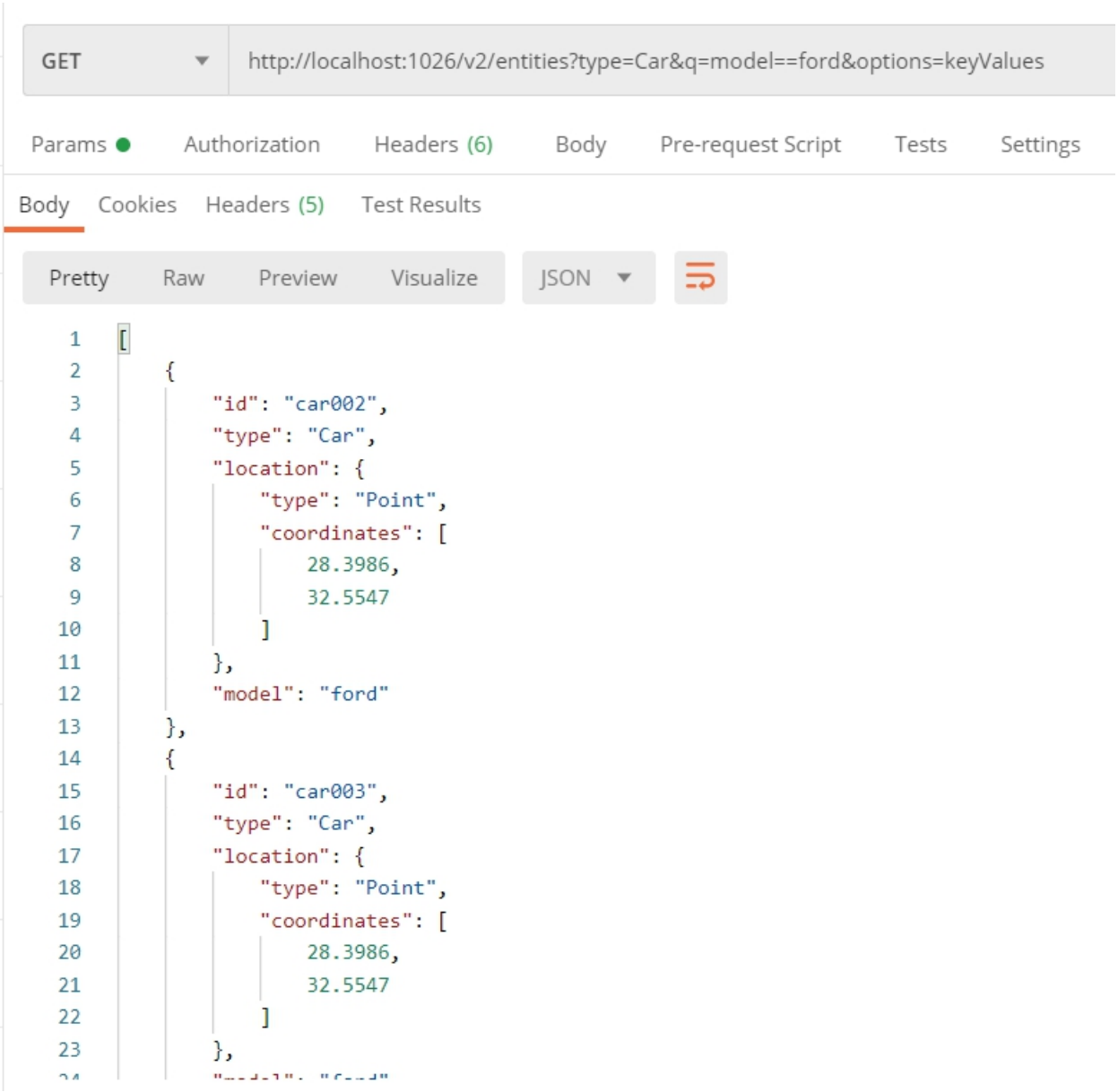

Рисунок 2.8 - Пошук за фільтрами

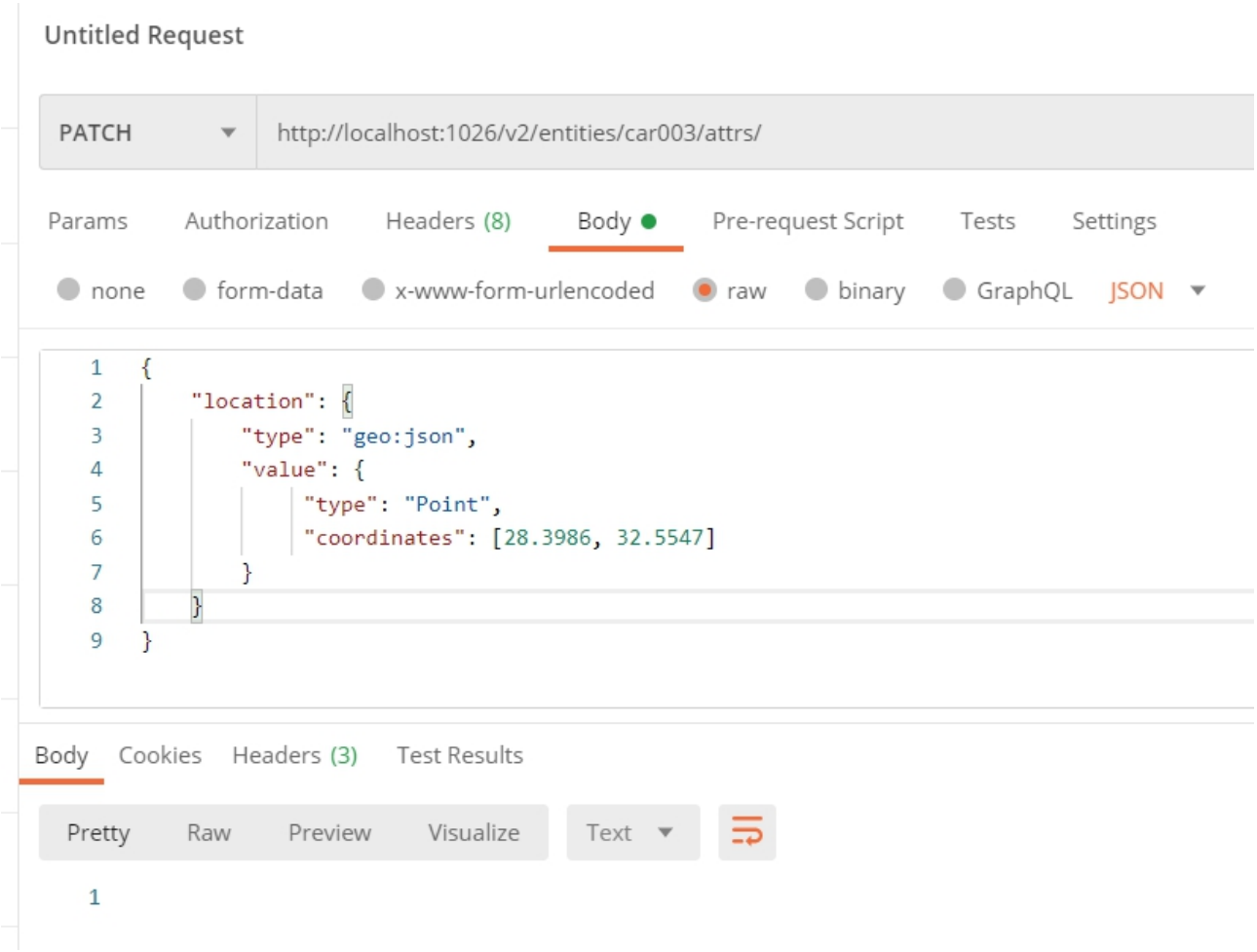

Рисунок 2.9 - Оновлення даних сутності

IoT Agent - це компонент, який дозволяє групі пристроїв надсилати свої дані та керувати ними з контекстного брокера за допомогою власних протоколів. Агенти IoT також повинні мати можливість вирішувати аспекти безпеки платформи FIWARE (аутентифікація та авторизація каналу) та надавати інші загальні послуги розробнику пристрою.

Контекстний брокер Orion використовує виключно запити NGSI-v2 для всіх своїх взаємодій. Кожен агент IoT надає інтерфейс NGSI-v2 North Port, який використовується для взаємодії посередника контексту, і всі взаємодії під цим портом відбуваються за допомогою власного протоколу підключених пристроїв.

По суті, це забезпечує стандартний інтерфейс для всіх взаємодій IoT на рівні управління контекстною інформацією. Кожна група пристроїв IoT може використовувати свої власні протоколи та різні транспортні механізми під капотом, тоді як пов'язаний агент IoT пропонує фасадний шаблон для вирішення цієї складності.

2.5 Висновки до розділу 2

У даному розділі було розглянуто системи управління бізнес-процесами Camunda BPM та jBPM, а саме: компоненти системи, способи розгортання та можливості, які вони надають аналітикам, розробникам та кінцевим користувачам.

Обидві системи надають схожі можлиловсті, перевага лише в швидкості налаштування необхідного функціоналу і зручності використання. Варто відмітити, що jBPM Community майже ні в чому вас не обмежує від Enterprise версії, лише в часі, ресурсах та підтримці. В той час, Camunda Enterprise має конкретний функціонал, відсутній в Open Source версії, наприклад, інструменти оптимізації, такі як HeatMap - кольорова карта процесу, яка показу розподілення навантаження на процес, що дозволяє побачити критичні місця процесу та можливість дивитись історію завершених екземпляроів процесу в користувацькому інтерфейсі.

Було розглянуто каталог сервісів інтернету речей Fiware, та проведено детальне ознайомлення з сервісами, використаними в роботі, а саме: Orion Context Broker та IoT Agent.

Важливими висновками є:

- Camunda BPM стратегічно спрямована на "зручність для розробників", мається на увазі, що система має обширний API, який дозволяє гнучко налаштовувати реалізацію процесу за допомогою коду. jBPM прагне до ідеї “Zero-Code-BPM”, тобто BPM з мінімальною кількістю коду.
- Camunda BPM пропонує деякі потужні функції, яких немає в jBPM.

3 Аналіз підходів управління мікросервісами в BPM системах

3.1 Переваги та недоліки мікросервісної архітектури

Мікросервіси - це підхід, при якому єдиний додаток будується як набір невеликих сервісів, кожен з яких працює у власному процесі і коммунікує з іншими використовуючи легкі механізми, як правило HTTP. Ці сервіси побудовані навколо бізнес-потреб і розгортаються незалежно з використанням повністю автоматизованого середовища. Існує абсолютний мінімум централізованого управління цими сервісами. Самі по собі ці сервіси можуть бути написані на різних мовах і використовувати різні технології зберігання даних.

Переваги:

- Висока відмовостійкість: при падінні одного із сервісів, всі інші залишаються в строю. Таким чином, несправності в окремих сервісах не завадять всьому робочому процесу.
- Гнучкість: можна спробувати впровадити нову технологію. Це буде значно швидше і, при невдачі, відкинути нові зміни просто, повернувшись до старої версії / технології. Змінюючи локально один з сервісів, ми не ризикуємо всією системою і час, що вимагається для змін, менше.
- Простота: чим менше коду (а кожен окремий сервіс являє собою цілісну систему, тому не потрібно розбиратися в безлічі деталей, які не стосуються даного конкретного функціоналу), тим простіше розробникам розібратися, що і як працює. До того ж, на це піде менше часу.
- Легкість введення написаного коду в роботу. Невелика кількість коду забезпечує швидкий деплоймент.
- Масштабованість. Найнеобхідніші послуги можна доповнити і розширити, коли з'явиться така необхідність. Вся система при цьому залишається незмінною.

Недоліки:

- Сполучення між самими сервісами складне. Так як кожен функціональний елемент ізольований, потрібна особлива ретельність при побудові між ними грамотної комунікації, адже їм в будь-якому випадку доведеться обмінюватися запитами і відповідями один з одним. Зрозуміло, що зі збільшенням кількості сервісів складність в їх побудові буде рости.
- Складність тестування виражається в тому, що спочатку потрібно окремо розбиратися з кожним сервісом, а потім тестувати коректну взаємодію його з іншими мікросервісамі.
- Мікросервіси гірше підходять для використання всередині окремих організацій, для них вони можуть виявитися невиправдано складними в застосуванні, в той час як для масових інтернет-сервісів вони підходять відмінно.

3.2 Моделі управління мікросервісами

В останні роки термін мікросервіси позиціонується як одна з основних тем, що стосуються архітектури програмного забезпечення. Ця тема, крім того, що приносить кілька переваг, також приносить ряд складностей для її розвитку. Розробка архітектури, заснованої на мікросервісах, додає одне із ключових питань для розробників та архітекторів: "Як ми можемо зробити архітектуру та організацію мікросервісів ефективними та багаторазовими?"

Серед усіх моделей композицій мікросервісів однією з найважливіших є оркестрація, яка дозволяє гармонійно інтегрувати мікросервіси, диктуючи ритм їх інтеграції та викликаючи кожний мікросервіс у необхідний час. Оркестрація описує те, як сервіси повинні взаємодіяти між собою, використовуючи для цього обмін повідомленнями, включаючи бізнес-логіку і послідовність дій.

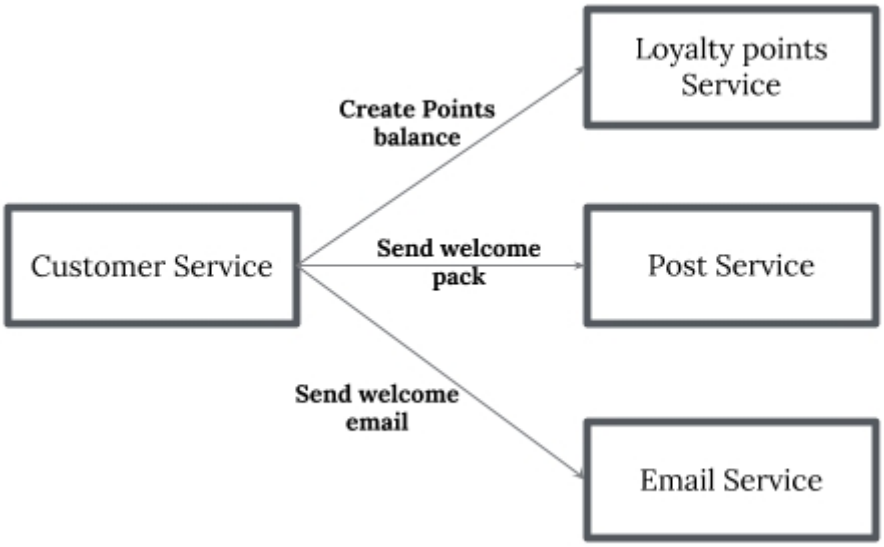

Рисунок 3.1 - Мікросервісна оркестрація

Ще однією моделлю, яка набула популярності, є мікросервісна хореографія. У цій моделі мікросервіси проектуються розподіленим чином, де кожен вузол знає, що він повинен робити для співпраці зі своїми сусідами. Оскільки у неї немає координатора виконання, ця модель вимагає більш глибоких знань технологій, щоб зрозуміти бізнес-потік, а також контролювати виконання процесу.

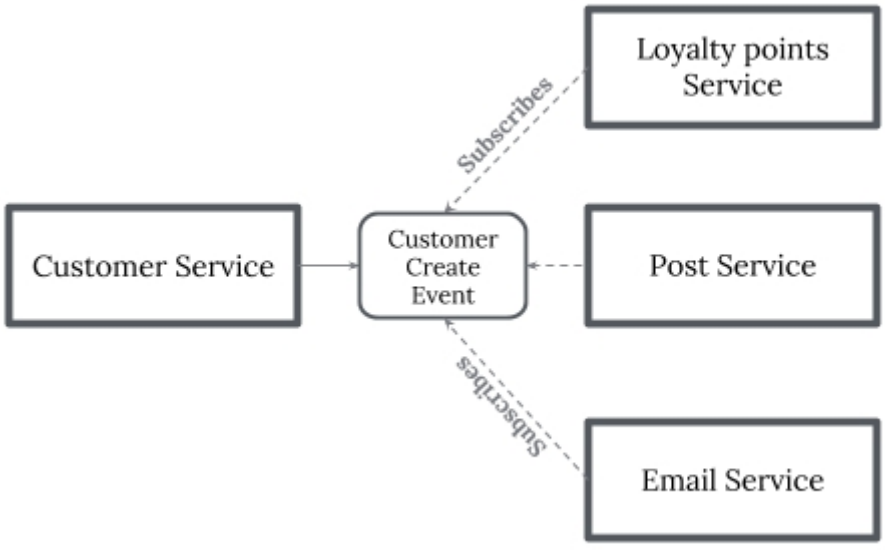

Рисунок 3.2 - Мікросервісна хореографія

При використанні цієї моделі, мікросервіси або викликають один одного безпосередньо (запит / шаблон відповіді), або спілкуються лише

опосередковано через повідомлення, або шини повідомлень (публікація / підписка).

Шаблон «Опублікувати / підписатись», зокрема, добре підходить для концепції незалежності окремих мікросервісів - мікросервісам не потрібно знати один про одного, тому їх можна розробляти, експлуатувати та масштабувати автономно.

У міру збільшення кількості мікросервісів, що беруть участь у бізнес-процесі, можуть виникати серйозні проблеми.

Загальний хід процесу стане важким для моніторингу та усунення несправностей у групі мікросервісів. Особливо турбує те, що ця проблема може бути не помітною при першій роботі з архітектурою мікросервісів, але вона поступово посилюється з часом, оскільки кількість мікросервісів зростає.

За замовчуванням хореографічний підхід не надає рішень для обробки помилок або таймаутів, і замість цього передає ці проблеми клієнту. Це ускладнює запобігання збоям у загальному потоці процесу, і в найгіршому випадку це може призвести до негативного досвіду споживачів - наприклад, коли веб-сайт відображає повідомлення про помилку, не пропонуючи рішення.

Одним із логічних виходів із цієї проблеми є використання моделювання оркестрації, заснованої на стандартизованій та функціональній мові, наприклад, моделювання та управління бізнес-процесів - BPMN.

За допомогою Camunda та jBPM ви можете уникнути цих проблем без шкоди для парадигми автономії сервісів та їх комунікації.

3.3 BPMS як оркестратор мікросервісів

Увійшовши до зони організації мікросервісів, існує загальна плутанина між організацією мікросервісів та організацією бізнесу. Ця плутанина зумовлена тим, що ціль програмного забезпечення, що використовується в огранізації бізнесу та мікросервісів, однакова. Цю двозначність можна легко вирішити, коли зрозуміло, що обидва види організації знаходяться на різних логічних архітектурних шарах.

Щоб розпочати вирішення цього, давайте уявимо дворівневу архітектурну модель: бізнес-оркестрація та мікросервісна оркестрація.

Як приклад, ми можемо матеріалізувати модель оплати рахунку, в якій є чотири дії з точки зору бізнесу, це заповнення форми, виконання платежу, друк ваучера (у випадку, якщо транзакція виконана) і повернення повідомлення про помилку (якщо виплата не здійснена). Для ілюстрації потоку, описаного в цьому абзаці, на наступному зображенні показано бізнес-процес, зіставлений у форматі BPMN:

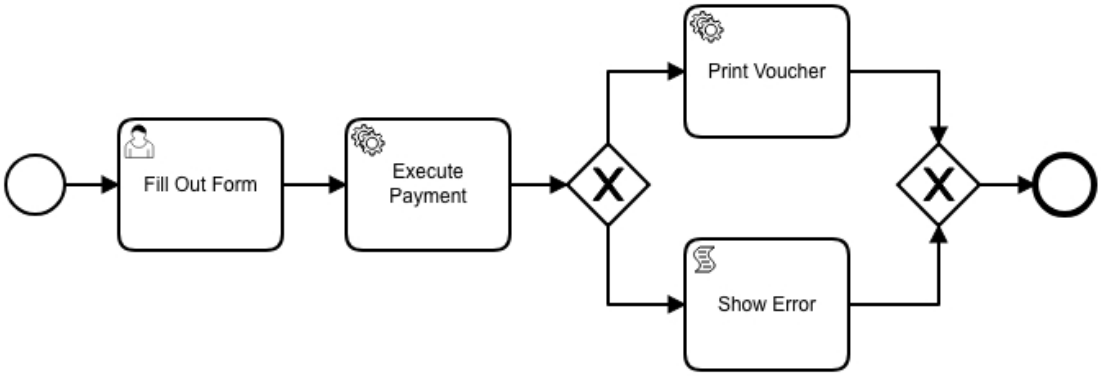

Рисунок 3.3 - Діаграма бізнес-процесу оплати рахунку

У цьому процесі першим видом діяльності є той, під час якого ви відкриваєте екран системи та заповнюєте необхідні дані для здійснення платежу. Після заповнення всіх даних користувач натискає кнопку для здійснення платежу.

Другий крок - здійснити платіж, викликавши мікросервіс Execute Payment. Однак для здійснення платежу необхідно зробити кілька перевірок, перш ніж платіж вважатиметься здійсненим. Наприклад, перевірка платіжних даних та перевірка баланса користувачів. У цьому контексті кожну дію можна вважати викликом до окремого сервісу.

3.4 Можливості BPMS для оркестрації мікросервісів

3.4.1 SOA-подібна оркестрація

SOA фокусується на віддаленому зв'язку між сервісами, побудованому на основі бізнес-можливостей. Центральний механізм процесів синхронно викликає віддалені розподілені сервіси. Інтеграція виконується між движком обробки процесів та сервісом, що не має стану.

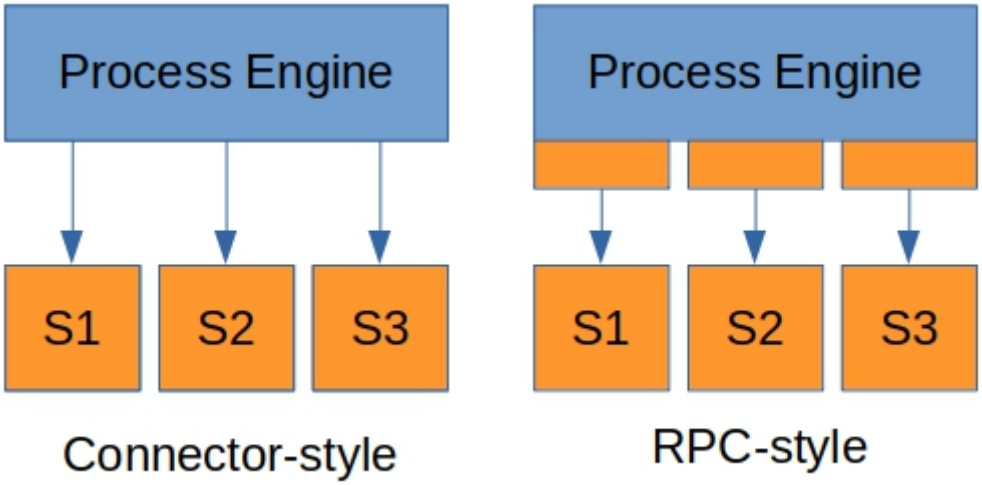

Рисунок 3.4 - Модель SOA-оркестрації

Існує два різних стилі реалізації цього класу систем.

Шаблон інтеграції Connector використовується, якщо механізм процесу викликає сервіс (S1, S2, S3), використовуючи безпосередньо обраний протокол (зазвичай HTTP).

Шаблон інтеграції RPC використовується, якщо механізм викликає локального делегата, і він викликає віддалений сервіс (S1, S2, S3) через обраний протокол (HTTP або будь-який інший синхронний протокол).

3.4.2 Повідомлення

Іншим популярним підходом для делегування виконання є використання повідомлень. На відміну від викликів, відправка повідомлень не блокується під час виконання процесу, який оброблює отримане повідомлення. Це створює

реальне одночасне виконання на концептуальному рівні, що може призвести до одночасного виконання на технічному рівні. Якщо ми прагнемо надати ту саму семантику, використовуючи повідомлення, як це передбачено в операціях викликів, необхідно використовувати пари throw (викинути) / catch (спіймати) подій. Приклад:

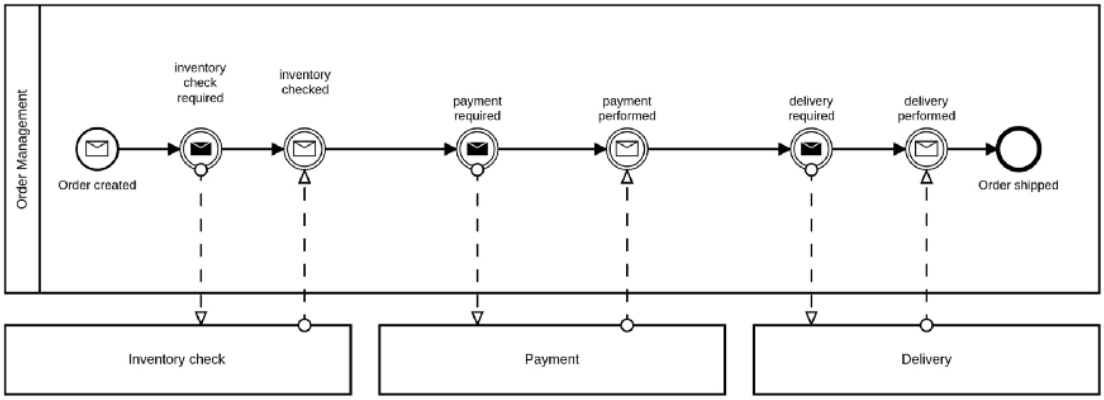

Рисунок 3.5 - Оркестрація з використанням повідомлень

3.4.3 Черги повідомлень

Замість синхронного виклику центральний механізм може надсилати повідомлення в черги або теми (topics), а сервіси підписуються на них. Одночасна доступність движка та сервісів не потрібна.

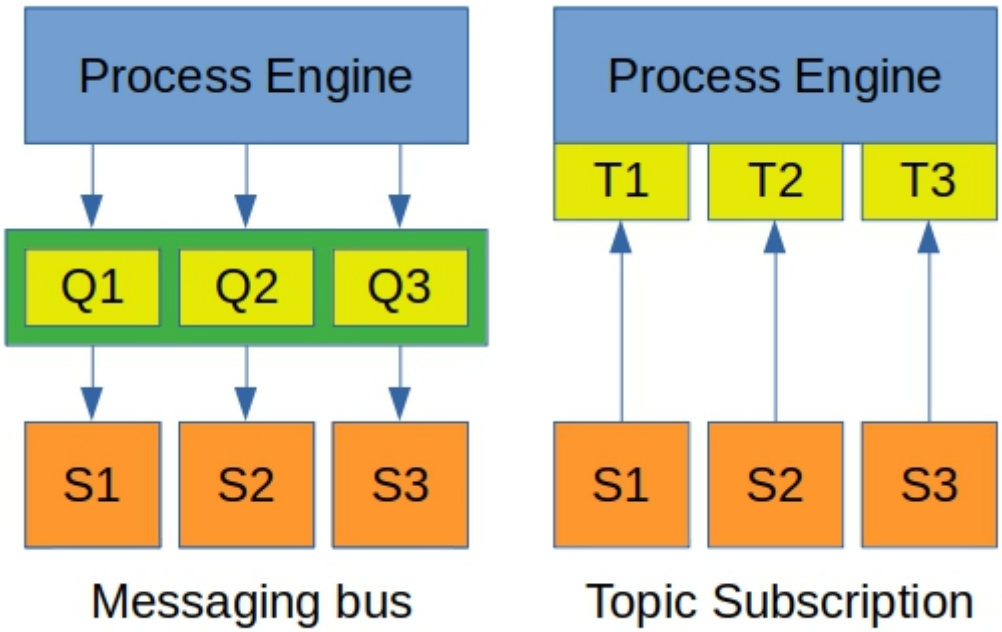

Рисунок 3.6 - Оркестрація з використанням черг повідомлень

Залежно від абстракції повідомлень, що використовуються, існує два типи реалізації:

Інфраструктура обміну повідомленнями може бути проміжним програмним забезпеченням (наприклад, за допомогою центральної шини обміну повідомленнями), що пропонує концепцію черг (Q1, Q2, Q3). Механізм надсилає асинхронні повідомлення до служб (S1, S2, S3) за допомогою черг.

Замість того, щоб використовувати черги, механізм обробки може публікувати інформацію в заздалегідь визначених темах (T1, T2, T3).

3.4.4 External Task / Service

Зовнішні завдання були представлені Camunda BPM у версії 7.4 і є однією з найважливіших функцій, щоб розірвати монолітний робочий процес і прямувати до розподіленого робочого процесу. Спочатку вони мали на меті забезпечити реалізацію service task, орієнтованої на підписку, на відміну від орієнтованої на виклик. Тобто, якщо механізм виконує службове завдання, він не викликає делегата для виклику (віддаленої) служби, а створює зовнішній запис завдання і чекає, поки (віддалений) зовнішній сервіс його отримає та виконає.

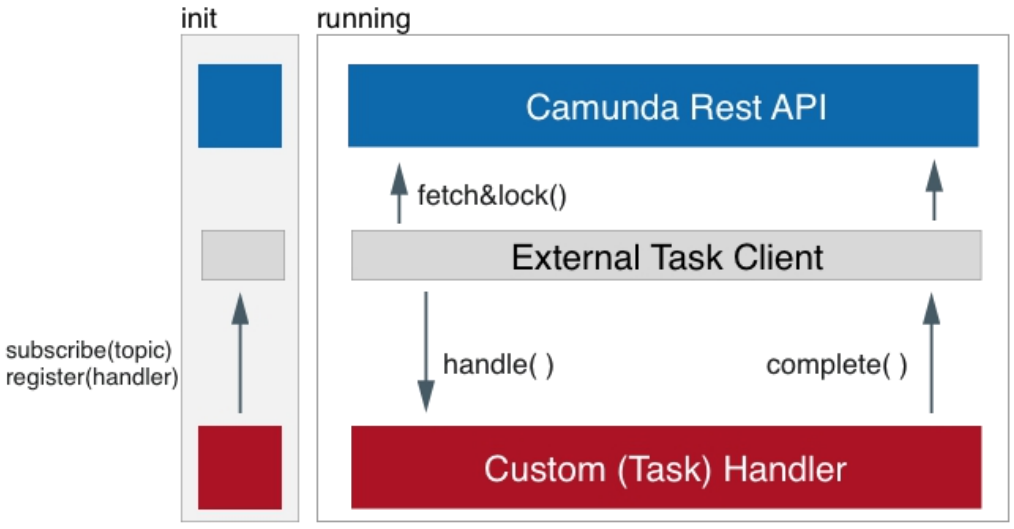

Рисунок 3.7 - Оркестрація за допомогою зовнішніх завдань

Шаблон зовнішнього завдання має кілька важливих властивостей:

Зовнішнє завдання - це завжди стан очікування в движку. Механізм процесу здійснить транзакцію після створення зовнішнього запису завдання.

Спеціальний обробник отримує сповіщення, якщо зовнішнє завдання доступне, і він може використовувати API ExternalTaskService (complete (), handleFailure () та handleBPMNError ()) для інформування механізму процесу про результат.

Значення тайм-ауту fetchAndLock полягає в тому, щоб зарезервувати контекст для виконання робочого завдання на певний час. Якщо час очікування закінчується до передачі відповіді, завдання знову стає доступним для інших обробників завдань.

Для виконання робіт, які публікуються в топіки потрібен зовнішній клієнт завдань. Camunda пропонує власні реалізації на Java та JavaScript, які можна використовувати як бібліотеку.

3.4.5 Патерн Saga

Програми, що працюють в хмарі, часто змінюють дані. Ці дані можуть бути розподілені за різними джерелами даних, що зберігаються в різних географічних точках. Щоб уникнути конфліктів і підвищити продуктивність в розподіленій системі, додаток не повинен намагатися забезпечити високу узгодженість транзакцій. Швидше, додаток повинен реалізовувати "ймовірну узгодженість". У цій моделі типова бізнес-операція складається з серії автономних кроків. У той час як ці кроки виконуються, загальне уявлення про стан системи може бути непослідовним, але коли операція завершена і всі кроки виконані, система повинна знову стати узгодженою.

Серйозною проблемою в моделі ймовірної узгодженості є те, як обробити крок, який не вдалося виконати. У цьому випадку може знадобитися скасувати всю роботу, виконану попередніми кроками операції. Однак дані не

можна просто відкинути, тому що інші паралельні екземпляри додатку, можливо, вже змінили ці дані. Навіть в тих випадках, коли дані не були змінені паралельним екземпляром, скасування кроку може бути не просто питанням відновлення вихідного стану. Може виникнути необхідність застосувати різні правила, специфічні для бізнесу.

Рішенням проблеми описаної вище може стати Compensating Transaction Pattern. Таким шаблоном може виступити Saga pattern.

Saga виходить з усвідомлення того, що розподілені транзакції не можуть бути легко оброблені з використанням класичної моделі ACID з Two-Phase Commit принципом.

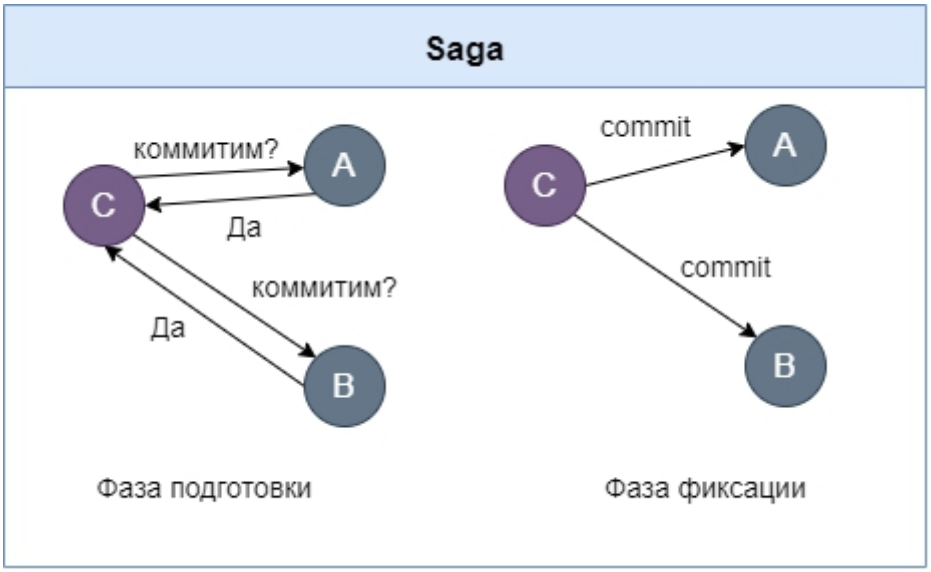

Рисунок 3.8 - Принцип двохфазового коміту

Фаза підготовки: На цьому етапі керуючий вузол запитує всі беручі участь вузли, чи готові вони прийняти транзакцію. Беручі участь вузли відповідають так чи ні.

Фаза фіксації: Потім, якщо всі вузли відповіли позитвно, керуючий вузол просить їх зафіксувати транзакцію, в іншому випадку, навіть якщо один вузол відповів негативно, він попросить відкотити назад всі зміни.

Saga замість цього розбиває роботу на окремі транзакції, наслідки яких можуть бути якимось чином скасовані після того, як робота була виконана і зафіксована.

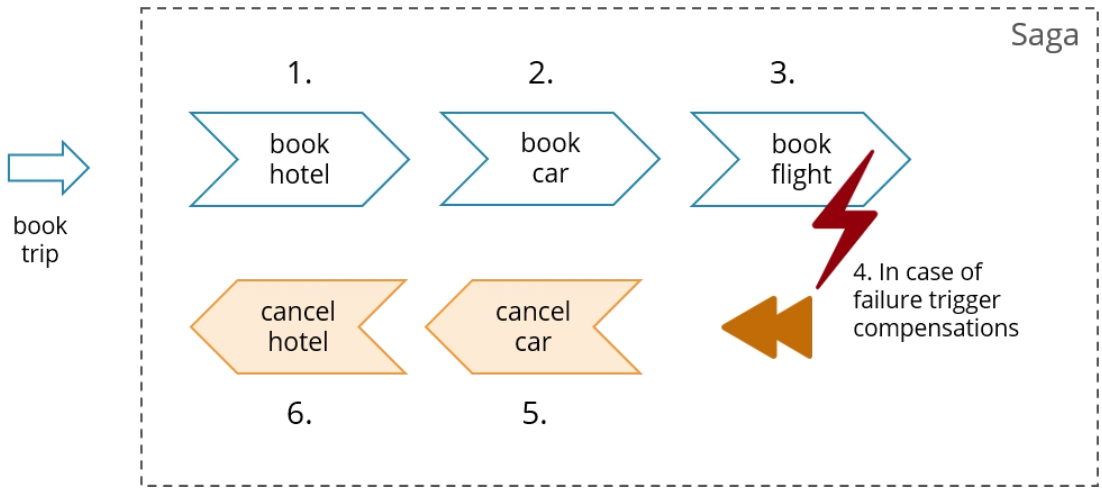

Рисунок 3.9 - Шаблон Saga

На малюнку вище показана проста реалізація Saga патерну. Якщо ви бронюєте маршрут подорожі, вам потрібен автомобіль, готель і рейс. Якщо ви не можете забронювати всі елементи, то, ймовірно, не варто їхати зовсім. Також цілком очевидно, що ви не можете підключити всі ці сервіси до розподіленої транзакції ACID. Замість цього у вас буде дія з бронювання прокату автомобілів, яка знає, як виконати бронювання, а також як скасувати його. Подібне для готелю і для рейсів.

Схема скасування транзакції буде виглядати наступним чином:

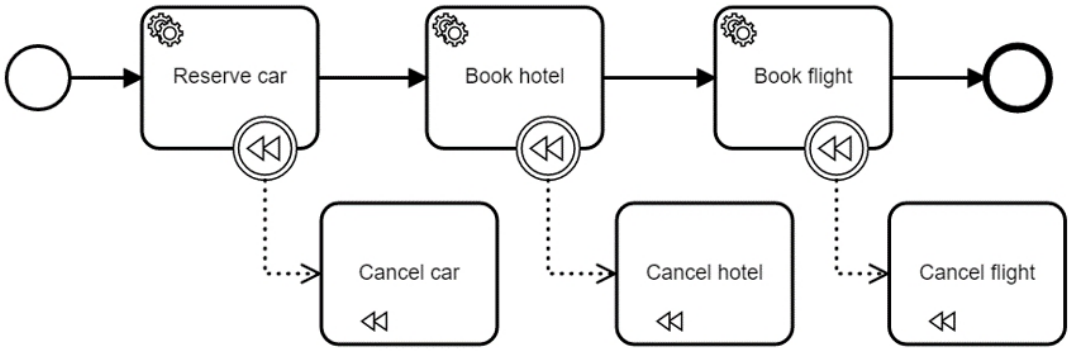

Рисунок 3.10 - BPMN діаграма патерну Saga

Приймаючи рішення про те, як реалізувати цей шаблон, врахуйте наступні моменти:

Може бути нелегко визначити, коли стався збій кроку в операції, що реалізує ймовірну узгодженість. Крок може не відразу завершитися невдачею, але замість цього він може блокуватися. Можливо, буде потрібно реалізувати деяку форму механізму тайм-ауту.

Логіка компенсації не легко узагальнюється. Компенсуюча транзакція залежить від програми. Вона спирається на те, що додаток володіє достатньою інформацією, щоб можна було скасувати "ефекти" кожного кроку невдалої операції.

Інфраструктура, яка обробляє етапи вихідної операції і компенсуючу транзакцію, повинна бути стійкою. Вона не повинна втрачати інформацію, необхідну для компенсації невдалого кроку, і повинна мати можливість надійно відслідковувати хід виконання логіки компенсації.

Компенсуюча транзакція не обов'язково повертає дані системи в стан, в якому вона перебувала на початку вихідної операції. Замість цього вона компенсує роботу, виконану кроками, які були успішно завершені до збою операції.

Порядок кроків в компенсуючій транзакції не обов'язково повинен бути дзеркальним відображенням кроків у вихідній операції. Наприклад, одне сховище даних може бути більш чутливим до невідповідностей, ніж інше, і тому спочатку необхідно виконати кроки в компенсуючій транзакції, яка скасовує зміни в цьому сховищі.

3.5 Висновки до розділу 3

У цьому розділі було наведено приклади використання бізнес-процесів у контексті мікросервісів. Присутні декілька моделей, які дозволяє побудувати BPMN, за допомогою яких можна досягти декомпозиції бізнес-процесу на більш дрібні частини та гнучко контролювати виконання:

- SOA-оркестрація;
- виклики;
- повідомлення;
- черги повідомлень;
- зовнішні завдання.

Camunda BPM та jBPM - це процесорні механізми, історично розроблені

як центральні оркестратори (модель функціонування «спільного движка»). Вони не підтримують та не вирішують більшість проблем, пов’язаних з мікросервісною архітектурою, але надають багато корисних можливостей для успішного використання у цьому середовищі.

4 Моделювання бізнес процесу в Camunda та jBPM

Реалізований в роботі бізнес-процес базується на наданні можливості користувачу регулювати температуру в кімнатах будинку. Використовуються сервіси платформи Fiware IoT Agent та Orion Context Broker, які мають багато можливостей гнучкого налаштування. Також реалізовний сервіс, який оновлює контекст кімнати, коли на датчиках відбуваються зміни показників. Сервіси обмінюються інформацією через HTTP виклики та оркеструються за допомогою BPMS.

Опис етапів процесу:

- користувач вказує номер кімнати та необхідну температуру;
- система дізнається поточну температуру в кімнаті;
- ящко температура співпадає з вказаною користувачем, робота завершується, інакше регулятор температури перемикається на необхідний режим;
- відбувається очікування, коли температура в кімнаті буде дорівнювати вказаній користувачем;
- в разі успіху, переведення регулятора температури в сплячий режим;
- інакше, регулятор працює далі, але повідомляє користувача, що довго не може досягти результату.

Детальний опис взаємодії сервісів наведено далі у цьому розділі.

Нижче наведена bpmn діаграма процесу:

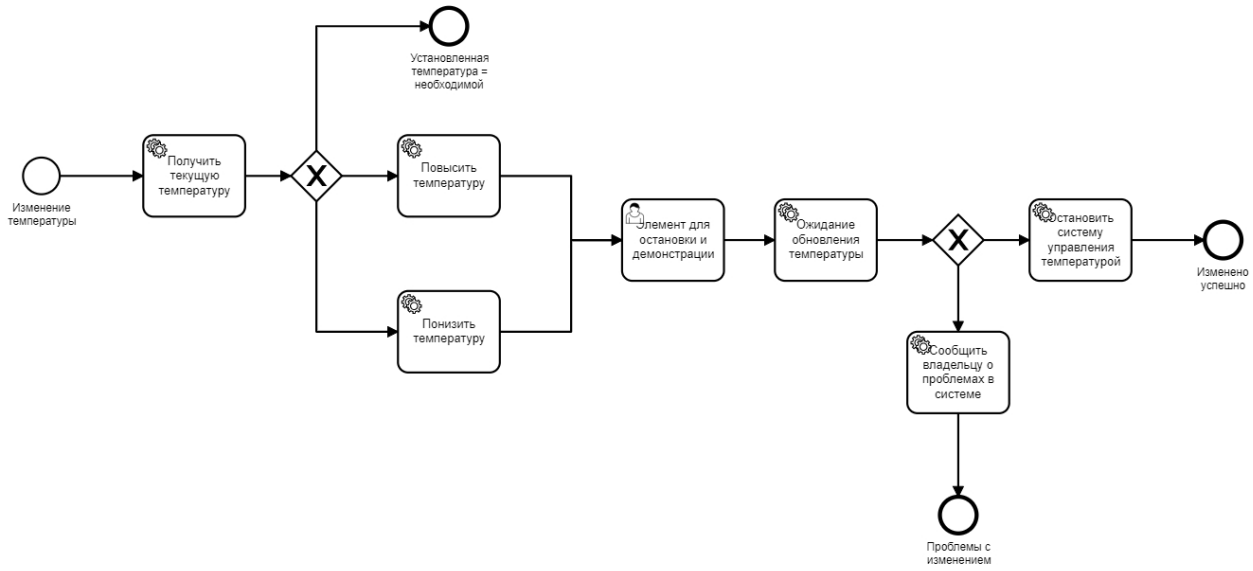

Рисунок 4.1 - Діаграма бізнес-процесу регуляції температури в домі

В системі використовується Orion Context Broker для зберігання контексту сутностей, в нашому випадку, кімнат, сенсорів, та регулятора температури. IoT Agent дозволяє комунікувати з пристроями наступним чином: User Request -> Context Broker -> IoT Agent -> Пристрій або датчики та у зворотньому напрямі. В роботі дані пристроїв та датчиків змінюються через http виклики. В ідеальному випадку, підключені до мережі пристрої будут самі відправляти зміни показників. Також IoT Agent дозволяє створювати групи пристроїв, що забезпечує зберігання контексту та стану в різних базах даних.

Оркестрація та сервіс для оновлення в реальному часі змін показників датчиків, реалізовані мовою програмування Java, та фреймворку Spring.

4.1 Налаштування сервісів Fiware

Для початку роботи з розглянутими сервісами Fiware, необхідно їх локально розгорнути, для цього використовуємо наступний docker-compose file:

```
version: "3.5"
services:
  # Orion is the context broker
  orion:
    image: fiware/orion:${ORION_VERSION}
    hostname: orion
    container_name: fiware-orion
```

```
    depends_on:
      - mongo-db
    networks:
      - default
    expose:
      - "${ORION_PORT}"
    ports:
      - "${ORION_PORT}:${ORION_PORT}" # localhost:1026
    command: -dbhost mongo-db -logLevel DEBUG
    healthcheck:
      test: curl --fail -s http://orion:${ORION_PORT}/version || exit 1
      interval: 5s

  iot-agent:
    image: fiware/iotagent-ul:${ULTRALIGHT_VERSION}
    hostname: iot-agent
    container_name: fiware-iot-agent
    depends_on:
      - mongo-db
    networks:
      - default
    expose:
      - "${IOTA_NORTH_PORT}"
      - "${IOTA_SOUTH_PORT}"
    ports:
      - "${IOTA_NORTH_PORT}:${IOTA_NORTH_PORT}" # localhost:4041
      - "${IOTA_SOUTH_PORT}:${IOTA_SOUTH_PORT}" # localhost:7896
    environment:
      - IOTA_CB_HOST=orion #t
      -
 IOTA_CB_PORT=${ORION_PORT} # port the context broker listens on to updat
e context
      - IOTA_NORTH_PORT=${IOTA_NORTH_PORT}
      -
 IOTA_REGISTRY_TYPE=mongodb #Whether to hold IoT device info in memory or
 in a database
      - IOTA_LOG_LEVEL=DEBUG # The log level of the IoT Agent
      -
 IOTA_TIMESTAMP=true # Supply timestamp information with each measurement
      -
 IOTA_CB_NGSI_VERSION=v2 # use NGSIv2 when sending updates for active att
ributes
      -
 IOTA_AUTOCAST=true # Ensure Ultralight number values are read as numbers
 not strings
      - IOTA_MONGO_HOST=mongo-db # The host name of MongoDB
      -
 IOTA_MONGO_PORT=${MONGO_DB_PORT} # The port mongoDB is listening on
      -
 IOTA_MONGO_DB=iotagentul # The name of the database used in mongoDB
```

```
      -
 IOTA_HTTP_PORT=${IOTA_SOUTH_PORT} # The port used for device traffic ove
r HTTP
      - IOTA_PROVIDER_URL=http://iot-agent:${IOTA_NORTH_PORT}
      - IOTA_DEFAULT_RESOURCE=/iot/d
    healthcheck:
      interval: 5s
```

```
mongo-db:
    image: mongo:${MONGO_DB_VERSION}
    hostname: mongo-db
    container_name: db-mongo
    expose:
      - "${MONGO_DB_PORT}"
    ports:
      - "${MONGO_DB_PORT}:${MONGO_DB_PORT}" # localhost:27017
    networks:
      - default
    volumes:
      - mongo-db:/data
    healthcheck:
      test: |
        host=`hostname --ip-address || echo '127.0.0.1'`;
        mongo --quiet $host/test --
eval 'quit(db.runCommand({ ping: 1 }).ok ? 0 : 2)' && echo 0 || echo 1
      interval: 5s

    healthcheck:
      test: |
        host=`hostname --ip-address || echo '127.0.0.1'`;
        mongo --quiet $host/test --
eval 'quit(db.runCommand({ ping: 1 }).ok ? 0 : 2)' && echo 0 || echo 1
      interval: 5s


networks:
  dockernet:
    external: true
  default:
    ipam:
      config:
        - subnet: 172.18.1.0/24
```

```
volumes:
  mongo-db: ~
```

4.1.1 Orion Context Broker

Необхідно створити в контексті сутності кімнат будинку, в яких

знаходяться датчики, та, температуру яких контролює регулятор. За допомогою наступного запиту до контекстного брокеру створюємо сутності 4-х кімнат:

```
curl -iX POST \
  'http://localhost:1026/v2/op/update' \
  -H 'Content-Type: application/json' \
  -d '{
  "actionType":"append_strict",
  "entities":[
    {
     "id":"urn:ngsi-ld:Room:unit001", "type":"Room",
     "temperature":{
       "type":"Integer", "value":20
     }
    },
    {
     "id":"urn:ngsi-ld:Room:unit002", "type":"Room",
     "temperature":{
       "type":"Integer", "value":18
     }
    },
    {
     "id":"urn:ngsi-ld:Room:unit003", "type":"Room",
     "temperature":{
       "type":"Integer", "value":20
     }
    },
    {
     "id":"urn:ngsi-ld:Room:unit004", "type":"Room",
     "temperature":{
       "type":"Integer", "value":22
     }
    }
  ]
}'
```

Подальше оновлення температури в кімнатах буде відбуватись програмно, при зміни даних на датчиках:

```
@PostMapping("/temperature/update")
public String handleTemperatureUpdate(@RequestBody
TemperatureUpdateRequest request) {
    log.info("Request={}", request);
    updateContextTemperature(request.getRoomNumber,
request.getData().get(0).getTemperature().getValue());
    return "OK";
}
```

```
@PutMapping("postTemperature/{roomNumber}/{temperature}")
public Data updateContextTemperature(@PathVariable String roomNumber,
@PathVariable String temperature) {

    RestTemplate restTemplate = new RestTemplate();
    String fooResourceUrl
            = "http://localhost:1026/v2/entities/urn:ngsi-ld:Room:unit" +
roomNumber + "/attrs/temperature/value?type=Room";
    restTemplate.exchange(fooResourceUrl, HttpMethod.PUT, new
HttpEntity<>(temperature), Void.class);

    return getCurrentTemperature(roomNumber);
}

@GetMapping("/getTemperature/{roomNumber}")
public Data getCurrentTemperature(@PathVariable String roomNumber) {
    RestTemplate restTemplate = new RestTemplate();
    String fooResourceUrl
            = "http://localhost:1026/v2/entities/urn:ngsi-ld:Room:unit" +
roomNumber + "?type=Room";
    ResponseEntity<Data> response
            = restTemplate.getForEntity(fooResourceUrl, Data.class);

    return response.getBody();
}
```

4.1.2 IoT Agent

Спочатку створюємо сервісну групу:

```
curl -iX POST \
  'http://localhost:4041/iot/services' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
 "services": [
   {
    "apikey":      "4jggokgpepnvsb2uv4s40d59ov",
    "cbroker":     "http://orion:1026",
    "entity_type": "Thing",
    "resource":    "/iot/d"
   }
 ]
}'
```

Після цього можемо створити 4 датчики та зв’язати їх з кімнатами для більш зручного виконання запитів в подальшій роботі:

```
curl -iX POST \
  'http://localhost:4041/iot/devices' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
 "devices": [
   {
    "device_id":   "temperatureSensor001",
    "entity_name": "urn:ngsi-ld:TemperatureSensor:001",
    "entity_type": "TemperatureSensor",
    "timezone":    "Europe/Berlin",
    "attributes": [
     { "object_id": "t", "name": "temperature", "type": "Integer" }
    ],
    "static_attributes": [
     {    "name":"refRoom",    "type":    "Relationship",    "value":    "urn:ngsi-
ld:Room:unit001"}
    ]
   }
 ]
}
```

Створимо підписку на кожен датчик. Це робиться для того, щоб, коли ми отримали зміни атрибута, який в нашому випадку ‘temperature’, orion відправив на вказаний нами url, оновлені значення показників. Їх, в свою, чергу приймає, згаданий ранішне сервіс, який оновить контексти кімнат, змінивши температуру на поточну:

```
curl -iX POST \
--url 'http://localhost:1026/v2/subscriptions' \
--header 'content-type: application/json' \
-H 'fiware-service: openiot'   -H 'fiware-servicepath: /' \
--data '{
"description": "Notify of temperature changes",
"subject": {
 "entities": [{"idPattern": ".*", "type": "TemperatureSensor"}],
 "condition": {
  "attrs": [ "temperature" ],
  "expression": {"q" : "refRoom==urn:ngsi-ld:Room:unit001"}
 }
```

```
  },
  "notification": {
    "http": {
      "url": "http://192.168.1.132:9001/temperature/update"
    }
  }
}'
```

Далі вже можна створити регулятори температури, які мають 3 стани: нагрівати (heat), охолоджувати (cool), та стан спокію (sleep):

```
curl -iX POST \
  'http://localhost:4041/iot/devices' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
  "devices": [
    {
      "device_id": "temperatureRegulator001",
      "entity_name": "urn:ngsi-ld:TemperatureRegulator:001",
      "entity_type": "TemperatureRegulator",
      "protocol": "PDI-IoTA-UltraLight",
      "transport": "HTTP",
      "endpoint": "http://iot-sensors:3001/iot/temperatureRegulator001",
      "commands": [
        {"name": "heat","type": "command"},
        {"name": "cool","type": "command"},
        {"name": "sleep","type": "command"}
       ],
       "attributes": [
        {"object_id": "s", "name": "state", "type":"Text"}
       ],
       "static_attributes": [
         {"name":"refStore", "type": "Relationship","value": "urn:ngsi-ld:Room:001"}
       ]
    }
  ]
}'
```

В наступному підрозділі наведено демонстрацію роботи реалізованого бізнес-процесу на прикладі Camunda BPM.

4.2 Camunda

Етапи процесу, зображені на діаграмі вище (Рис 4.1), реалізовані за використання Service Task з реалізацією Java Delegate, тобто розгортається сервіс з Camunda Engine, в якому описано декілька класів, реалізуючих інтерфейс JavaDelegate, які отримують необхідні дані з контексту конкретного екземпляру процесу, викликають необхідний мікросервіс і в разі позитивної відповіді, додають результат до контексту екземпляру процесу.

Реалізація делегату, що відповідає за початок процесу нагрівання кімнати:

```
    @Override
public void execute(DelegateExecution execution) {
    String processInstanceId = execution.getProcessInstanceId();
    Map<String, Object> processVariables =
runtimeService.getVariables(processInstanceId);

    String roomNumber =
String.valueOf(processVariables.get("roomNumber"));
    String temperature =
String.valueOf(processVariables.get("temperature"));

    heatRoom(roomNumber);
    updateTemperature(roomNumber, temperature);
    log.info("roomNumber={}, temperature={}", roomNumber, temperature);
}

public void updateTemperature(String roomNumber, String temperature) {
    String temperatureToHeat = "t|" + temperature;
    HttpHeaders headers = new HttpHeaders();
    headers.add(HttpHeaders.CONTENT_TYPE, "text/plain");
    String iotDeviceUrl
            = apiUrl +
"iot/d?k=4jggokgpepnvsb2uv4s40d59ov&i=temperatureSensor" + roomNumber;
    log.info("temperatureToHeat={}, url={}, headers={}",
temperatureToHeat, iotDeviceUrl, headers);
    restTemplate.exchange(iotDeviceUrl, HttpMethod.POST, new
HttpEntity<>(temperatureToHeat, headers), Void.class);
}

public void heatRoom(String roomNumber) {
    String temperatureRegulatorStatus = "s|heat";
    String iotDeviceUrl
            = apiUrl +
"iot/d?k=4jggokgpepnvsb2uv4s40d59ov&i=temperatureRegulator" + roomNumber;
    restTemplate.exchange(iotDeviceUrl, HttpMethod.POST, new
```

```
HttpEntity<>(temperatureRegulatorStatus), Void.class);
}
```

Співставлення java делегату до bpmn діаграми виглядає наступним чином, натиснувши на необхідний Service Task, заповнюємо спосіб реалізації та, в даному випадку, вказуємо назву Java Bean з маленької літери:

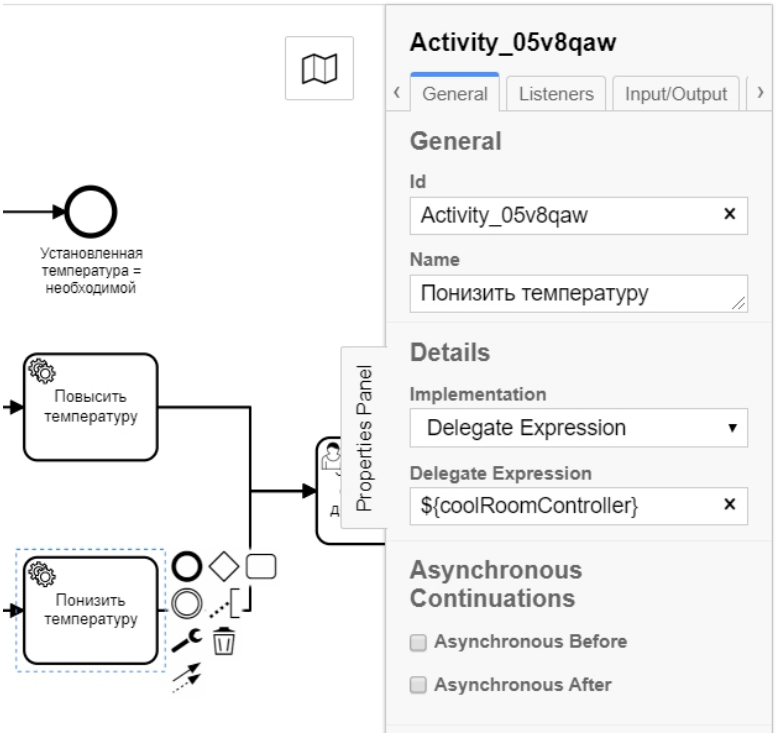

Рисунок 4.2 - співставлення java делегату сервісного завдання на діграмі

Спочатку подивимось на температуру в 1-й кімнаті. Зараз там 12 градусів.

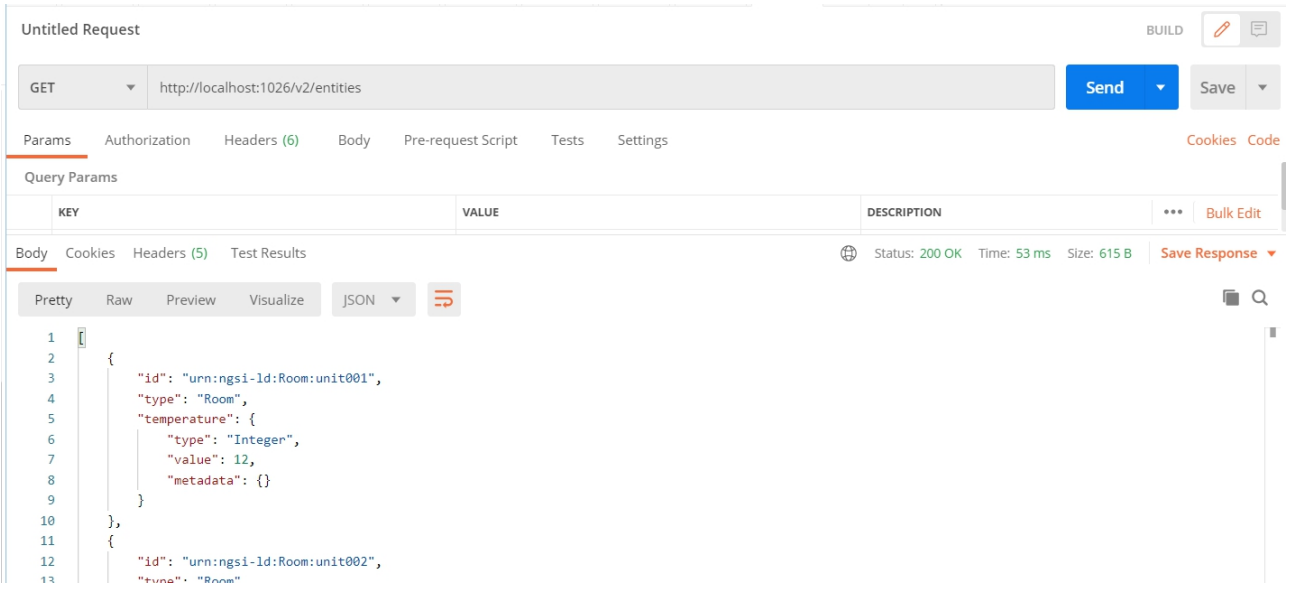

Рисунок 4.3 - запит для отримання даних сутності

Результат запиту всіх підписок:

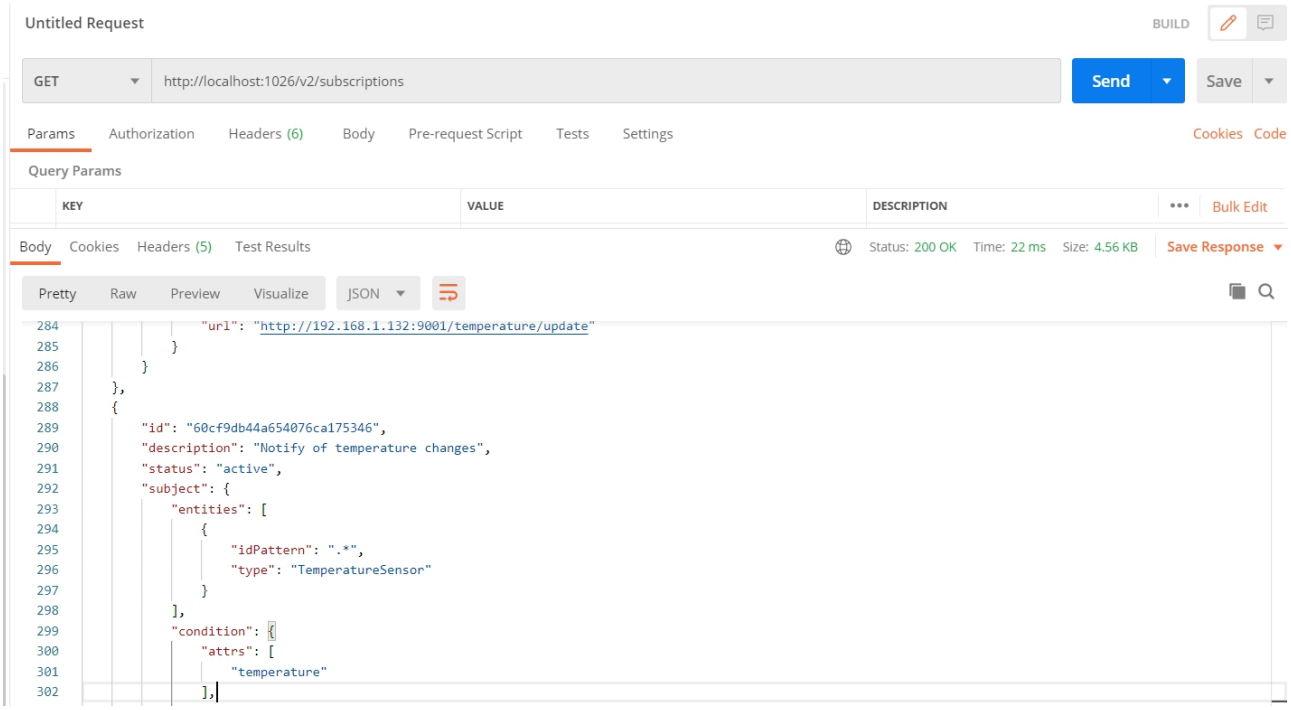

Рисунок 4.4 - запит для отримання створених підписок

Переконаємось, що регулятор температури знаходиться в стані спокою (sleep):

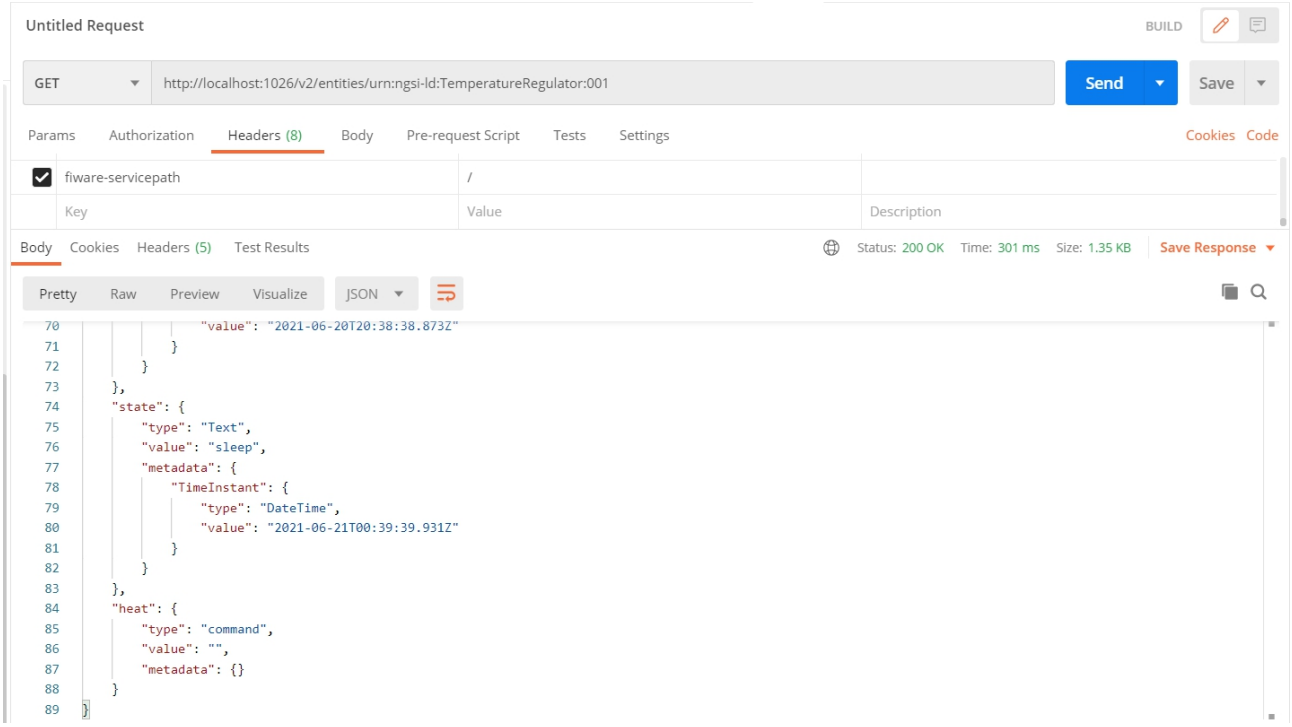

Рисунок 4.5 - запит для отримання інформації про регулятор температури

Також схожу інформацію можна побачити в документах mongodb, далі наведені сутності кімнати, датчику та регулятора температури, а також документ підписок:

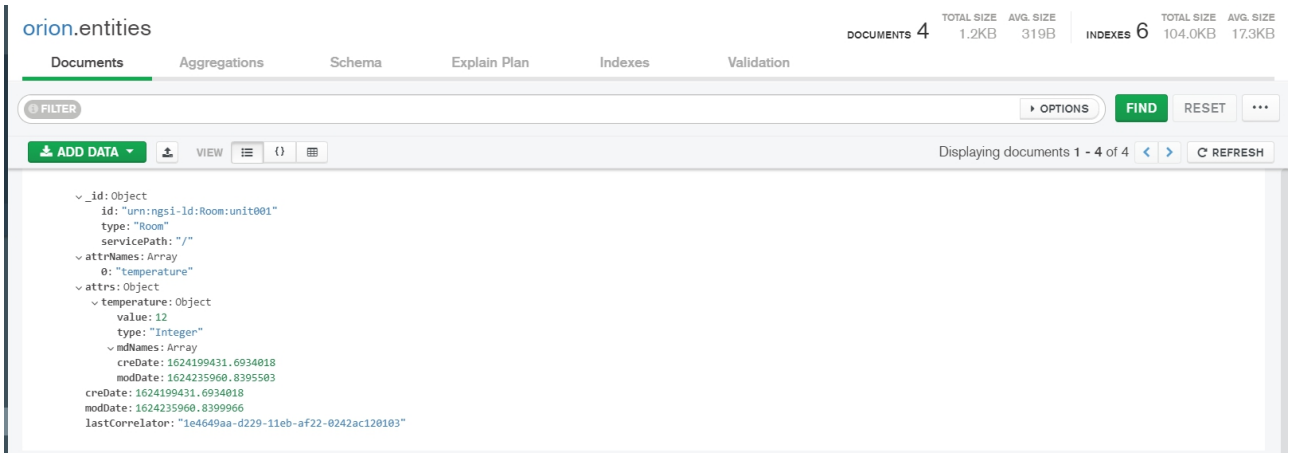

Рисунок 4.6 - документ з сутністю кімнати

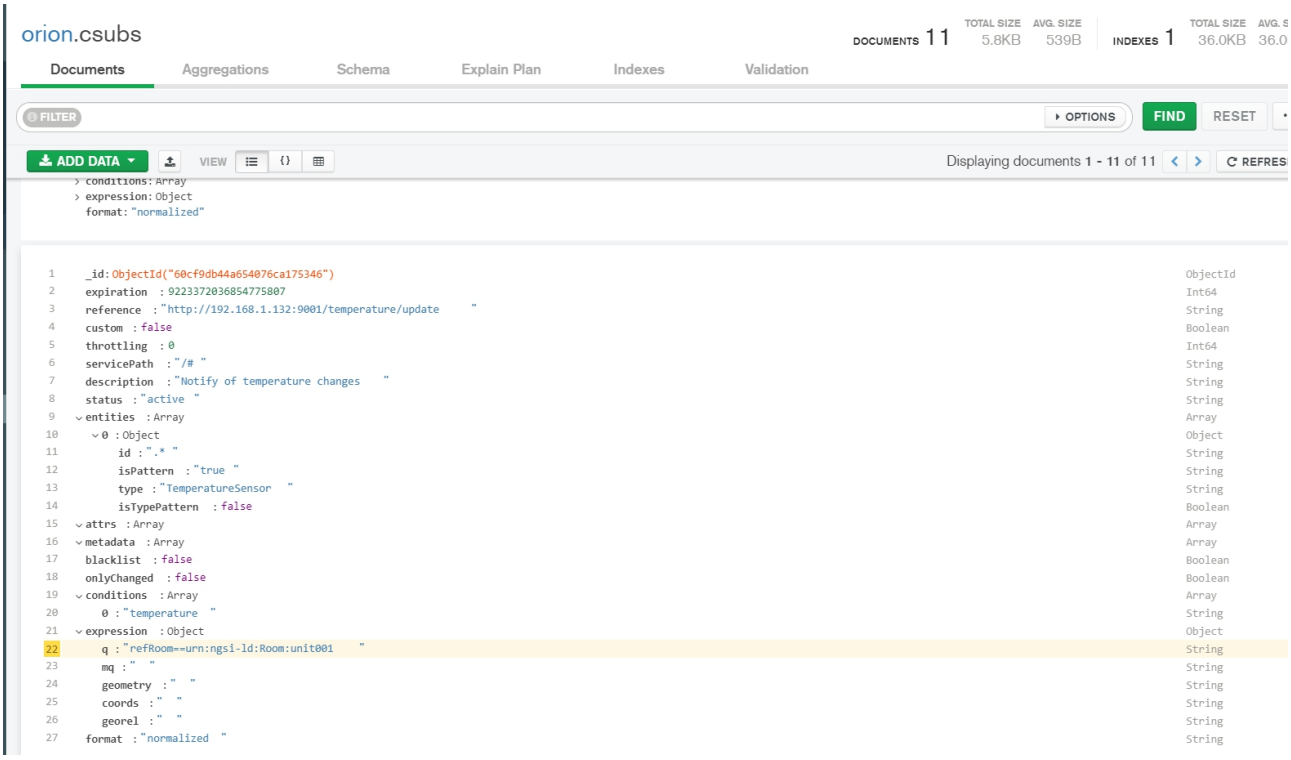

Рисунок 4.7 - документ з підпискою

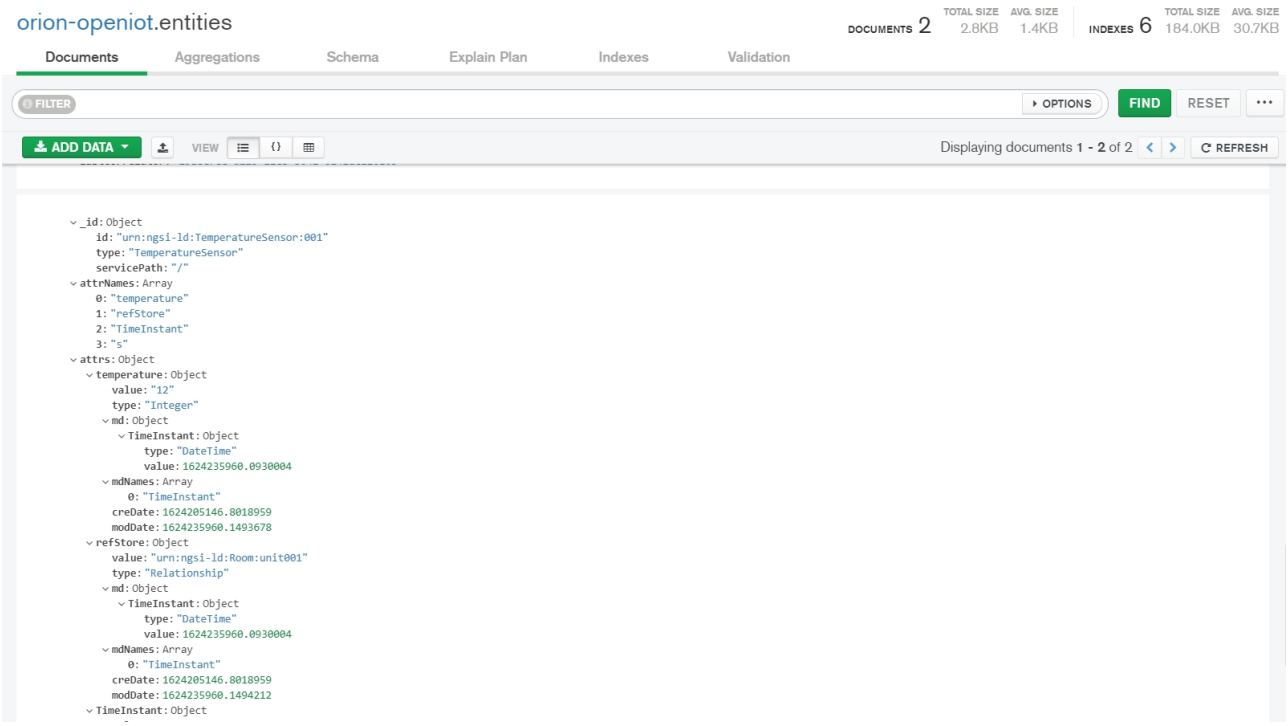

Рисунок 4.8 - документ з сутністю датчику температури

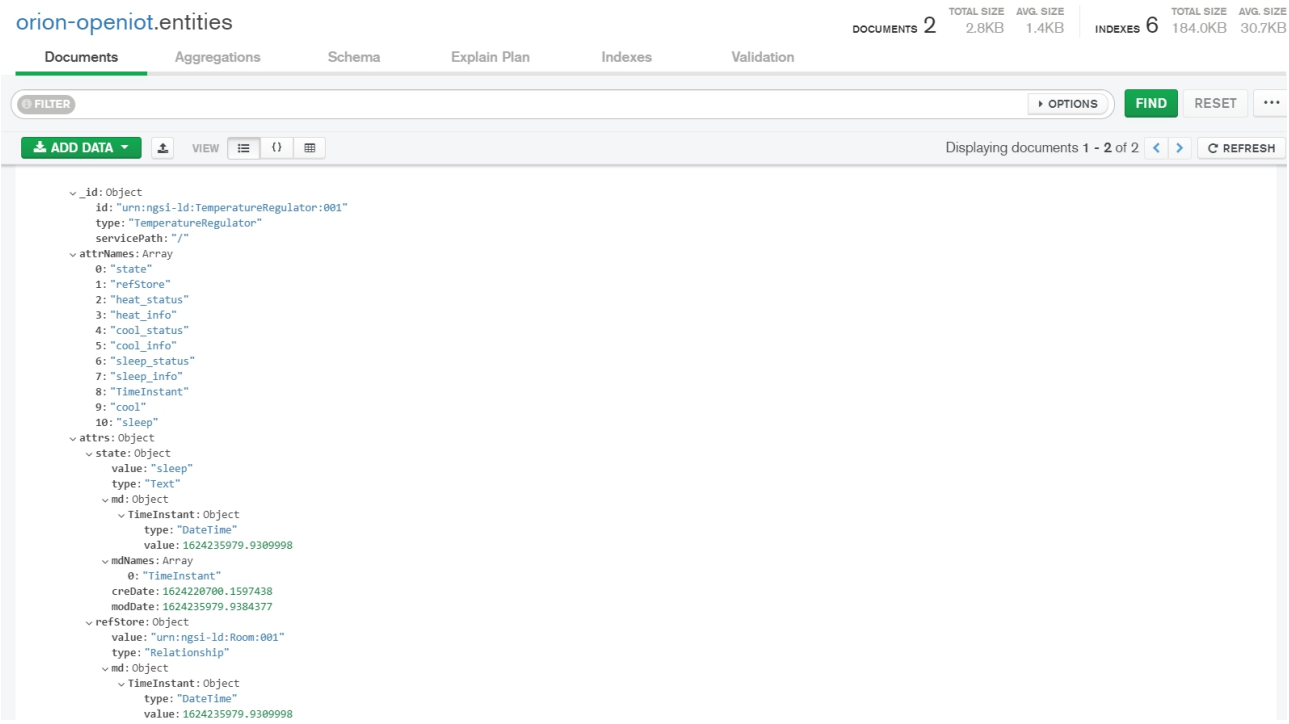

Рисунок 4.9 - документ з сутністю регулятора температури

Розгорнувши всі сервіси, які були описані до цього та побачивши поточні дані, можемо створити новий екземпляр процесу з заданими змінними:

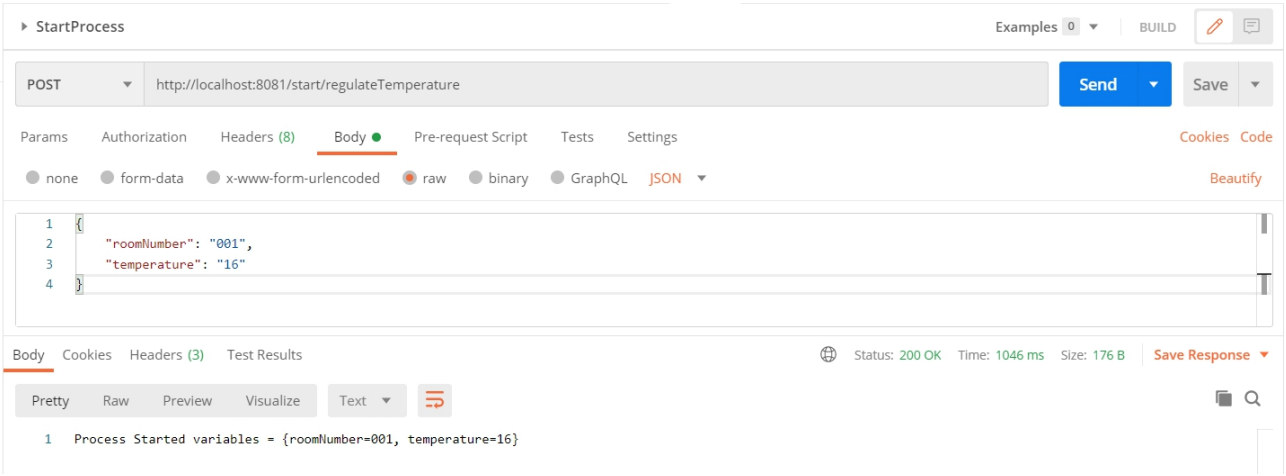

Рисунок 4.10 - запит для створення екземпляру процесу з вхідними змінними

Далі наведено клас, який відповідає за запуск екземпляру процесу з записом вхідних даних до контексту:

```
@PostMapping("/start/{processDefinitionKey}")
public String startProcess(@RequestBody ChangeTemperatureRequest
changeTemperatureRequest, @PathVariable String processDefinitionKey){
    try{
        Map<String,Object> variables =
objectMapper.convertValue(changeTemperatureRequest, new
TypeReference<Map<String, Object>>() {});

runtimeService.startProcessInstanceByKey(processDefinitionKey,variables);
        return String.format("Process Started variables = %s",
variables.toString());
    } catch (Exception e){
        return "Failed to start process Instance";
    }
}
```

Створивши екземпляр процесу, можна побачити його положення на карті процесу, тобто на якому етапі виконання він знаходиться. На даному етапі, бізнес-процес визначив, яку дію необхідно зробити регулятору температур, та оновив його стан:

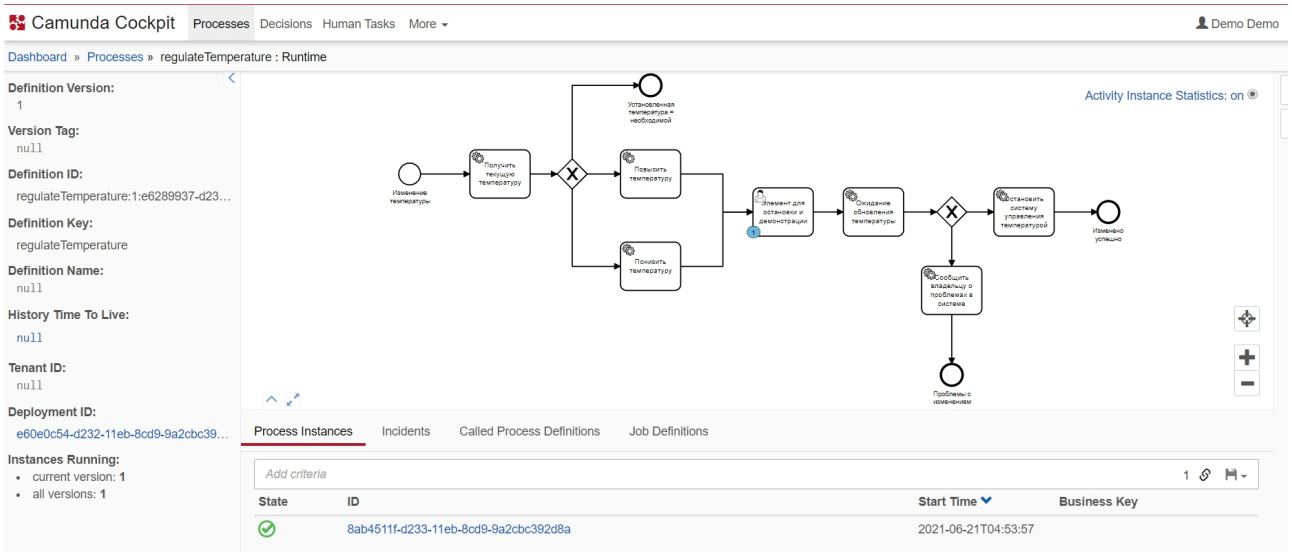

Рисунок 4.11 - Карту процесу

Є можливість фільтрувати та відкрити конкретний екземпляр процесу, побачити з можливістю редагування та видалення всі змінні екземпляру:

Рисунок 4.12 - Перелік змінних екземпляра процесу

Нажаль, в Community версії немає можливості переглянути завершені процеси за допомогою інтерфейсу користувача, але їх можна опрацювати використавши API bpm engine:

```
public List<HistoricProcessInstance> getAllFinishedProcessInstances(String
processDefinitionName) {
   ProcessEngine processEngine = BpmPlatform.getDefaultProcessEngine();
   HistoryService historyService = processEngine.getHistoryService();;
   RepositoryService repositoryService = processEngine.getRepositoryService();

   ProcessDefinition myProcessDefinition =
      repositoryService.createProcessDefinitionQuery()
         .processDefinitionName(processDefinitionName)
         .latestVersion()
         .singleResult();

   List<HistoricProcessInstance> processInstances =
      historyService.createHistoricProcessInstanceQuery()
         .processDefinitionId(myProcessDefinition.getId())
```

```
            .finished()
            .list();

    return processInstances;
}
```

Створено людське завдання для зупинки процесу і демонстрації проміжних результатів:

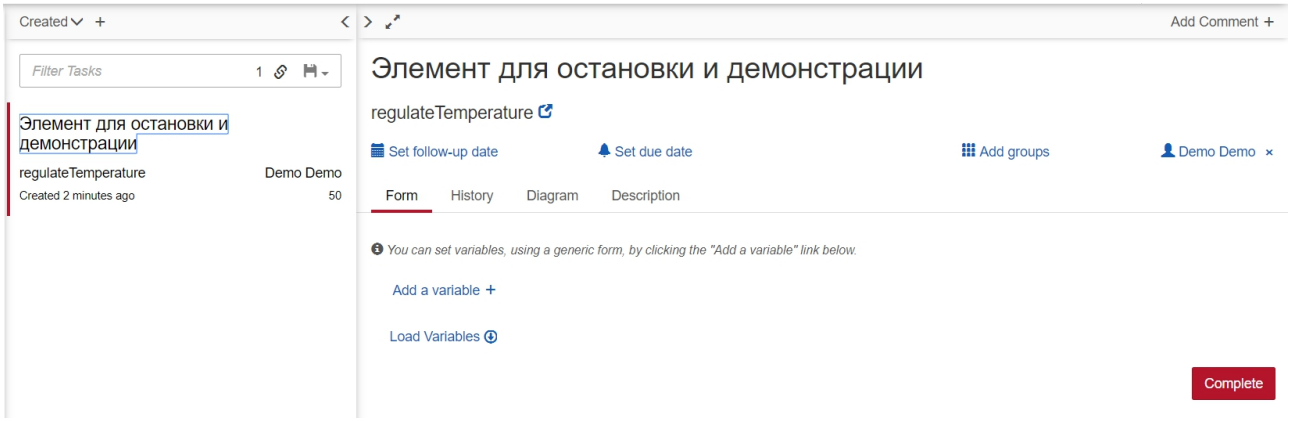

Рисунок 4.13 - Людське завдання

До підтвердження людського завдання, бачимо стан регулятора температури, як нагрівання (heat):

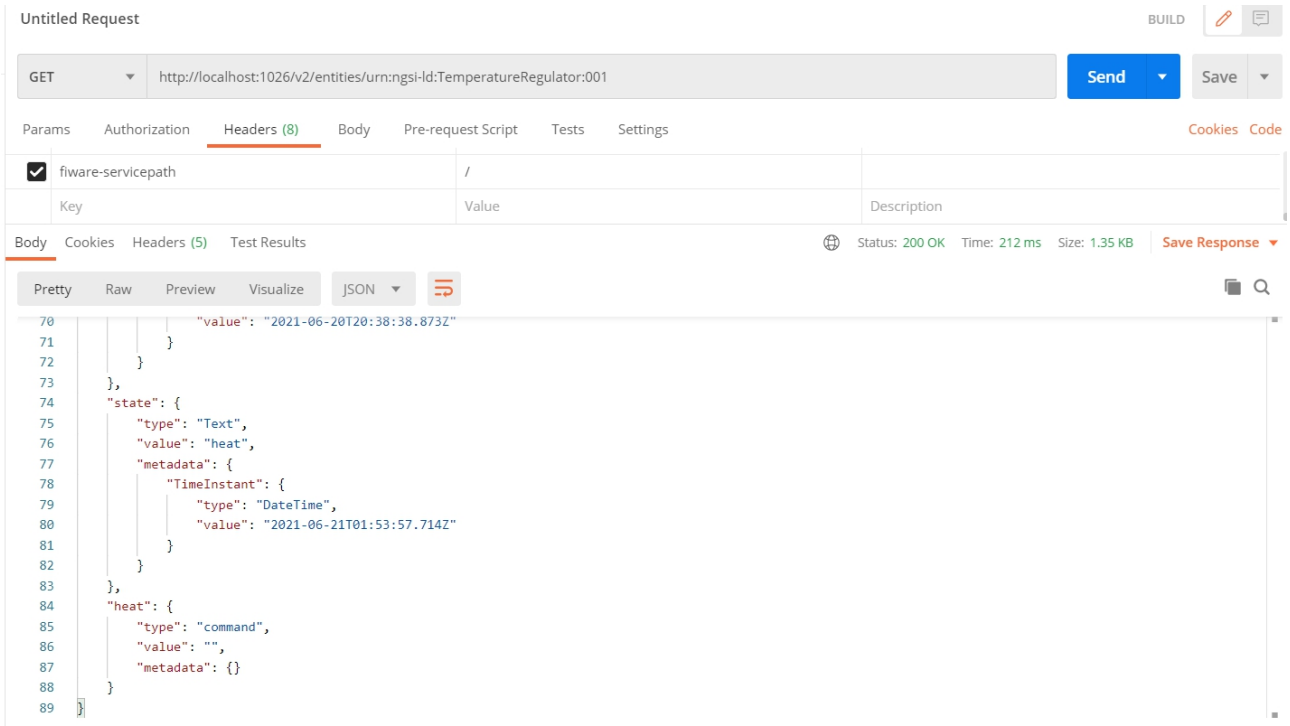

Рисунок 4.14 - Проміжний статус регулятора температури

Підтвердимо людське завдання. Після виконання наступного запиту, можна переконатись, що після завершення роботи, регулятор перейшов до стану спокою:

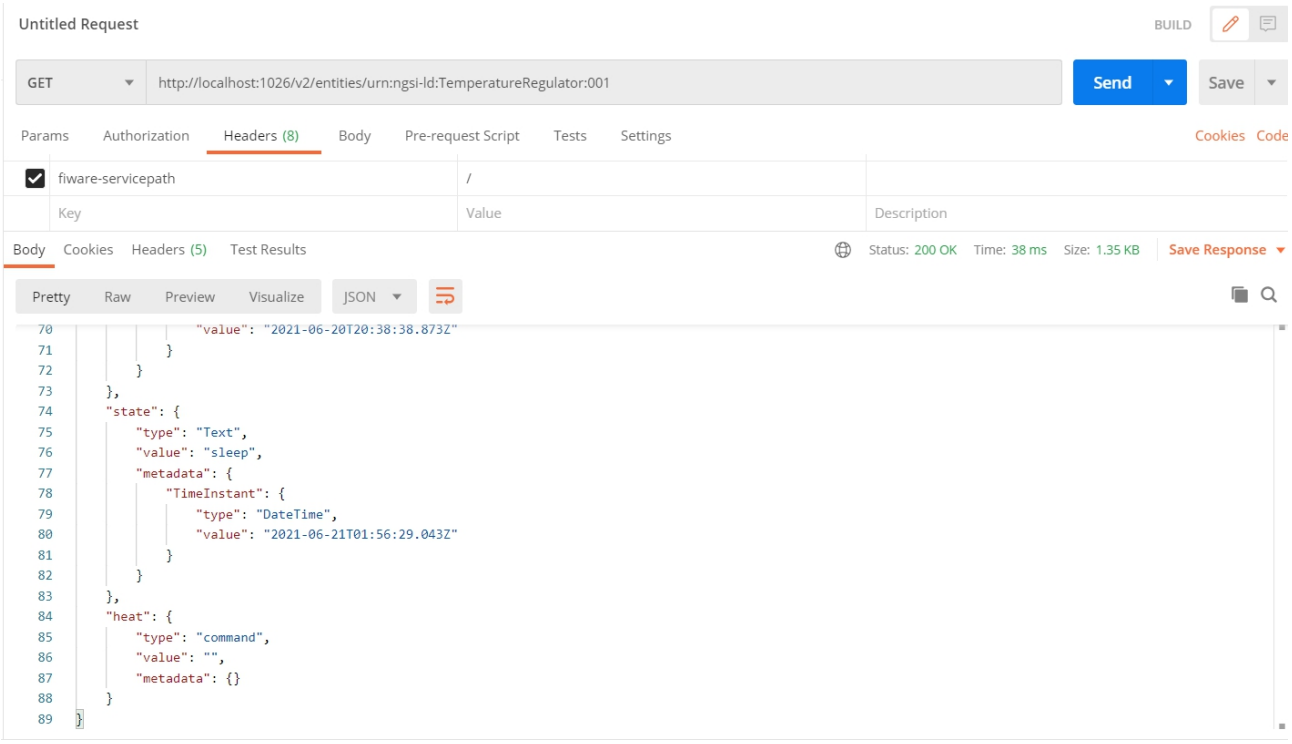

Рисунок 4.15 - Статус регулятора температури після завершення процесу

Також бачимо оновлену температуру в кімнаті:

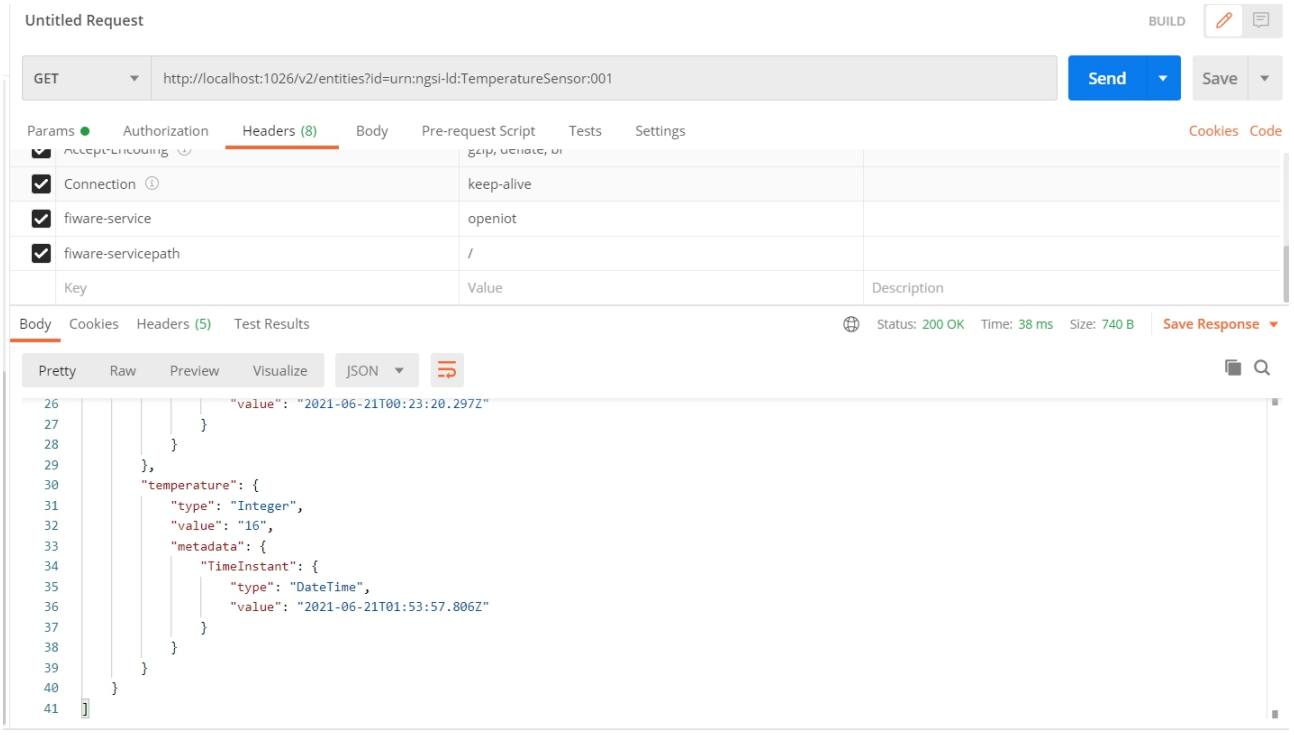

Рисунок 4.16 - Результат температури в кімнаті після завершення процесу

Далі наведено інші можливості Camunda BPM як оркестратора

мікросервісів на прикладі процесу замовлення товару користувачем.

Ще один схожий спосіб виклику зовнішнього сервісу - це External Task. Необхідно вказати топік і ті сервіси, які прослуховують цей топік, зможуть виконати певну роботу, коли отримають повідомлення.

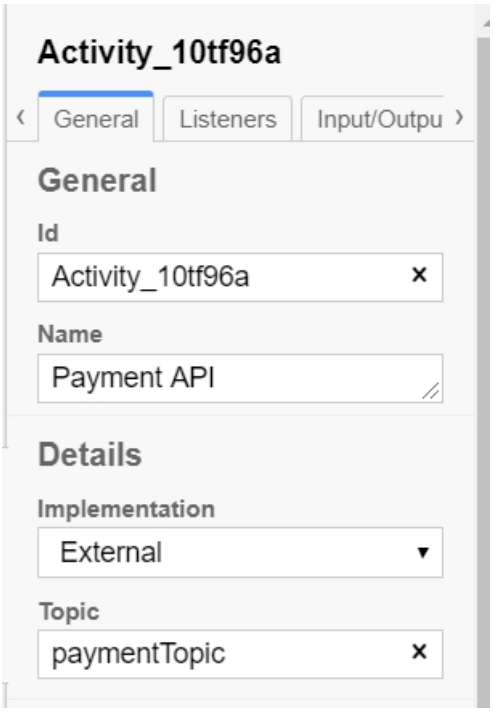

Рисунок 4.17 - Налаштування зовнішнього завдання

```
  @Scheduled(initialDelay = 5000, fixedRate = 5000)
    public void periodicalTaskExecutor() {
        FetchRequest fetchRequest = FetchRequest.builder().
            maxTasks(1).
            workerId(workerId).
            topics(new FetchTopicRequest[]{
                FetchTopicRequest.builder().
                    lockDuration(10000).
                    topicName("paymentTopic").
                    variables(new String[]{"customerId",
"productId"}).
                    build()
            }).
            build();

        try {
            String fetchUrl = "http://localhost:" + port +
"/engine-rest/external-task/fetchAndLock";
```

```
            PaymentRequest[] paymentRequests =
restTemplate.postForObject(fetchUrl, fetchRequest,
PaymentRequest[].class);
            log.info("Got {} external tasks to execute",
paymentRequests.length);
            for (PaymentRequest paymentRequest :
paymentRequests) {
                log.info("Marking {} as complete",
paymentRequest.getId());
                String completeUrl = "http://localhost:" +
port + "/engine-rest/external-task/" + paymentRequest.getId()
+ "/complete";
                CompleteRequest completeRequest =
CompleteRequest.builder().
                    workerId(workerId).
                    variables(new HashMap<>()).
                    build();
                completeRequest.variables.put("approved",
VariableReference.builder().value("payed").build());
                restTemplate.postForEntity(completeUrl,
completeRequest, null);
            }
        } catch (Exception e) {
            log.warn("Failed too check for tasks: {}",
e.getMessage(), e);
        }
    }
```

Також наявна можливість реалізувати паттерн Saga, створивши події відмови у всіх місцях, де потрібна компенсація роботи сервісів. Це актуально для реалізації зворотньої операції у разі неуспішного виконання транзакції.

Наприклад, на наступній діаграмі, якщо з’явились якісь проблеми на етапі доставки замовлення, то ми попадемо в гілку завершаючої компенсації і вона спровокує виконання всіх відмов. В даному випадку PaymentCompensation, яка поверне гроші звову на баланс клієнта.

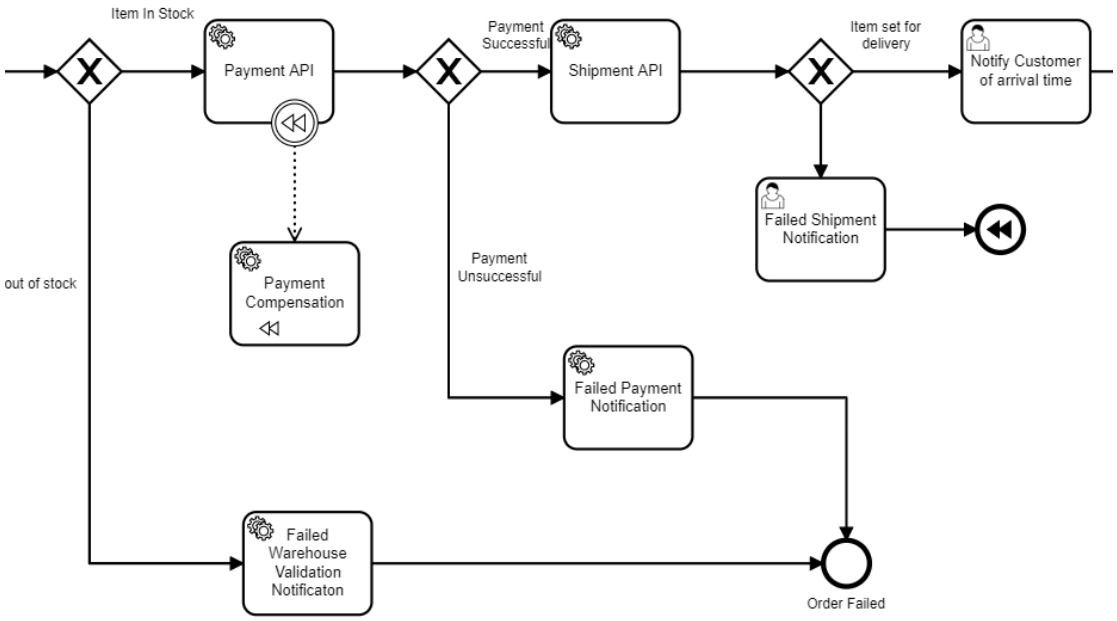

Рисунок 4.18 - Діаграма процесу з використанням компенсацій

Реалізувати логіку відмови можна будь яким зручним чином, який можливий в системі. Це дуже спрощує кодову базу, як для реалізації, так і для розуміння.

4.3 jBPM

Для локальної роботи з jbpm необхідно запустити kie-server та business central:

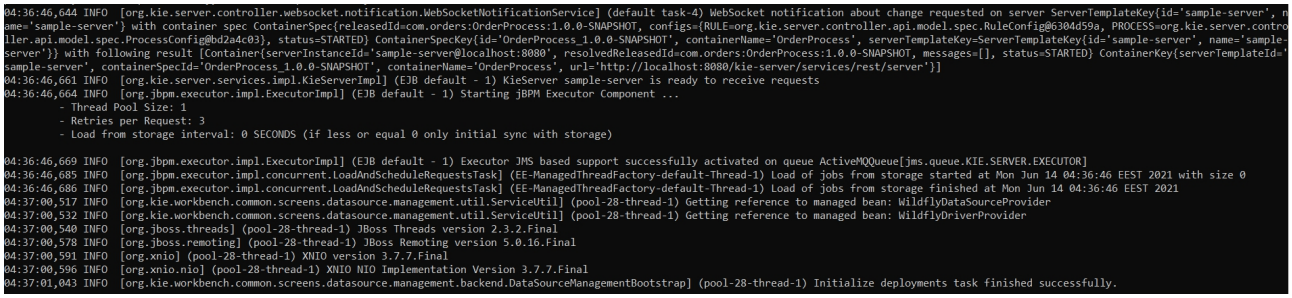

Рисунок 4.19 - Запуск kie-server та business central

Після входу в аккаунт, необхідно створити простір та проект всередині нього. В одному просторі можна створювати декілька проектів. В проекті доступна бібліотека ассетів (елементи проекту, діаграма процесу, моделі даних, обробники, тощо).

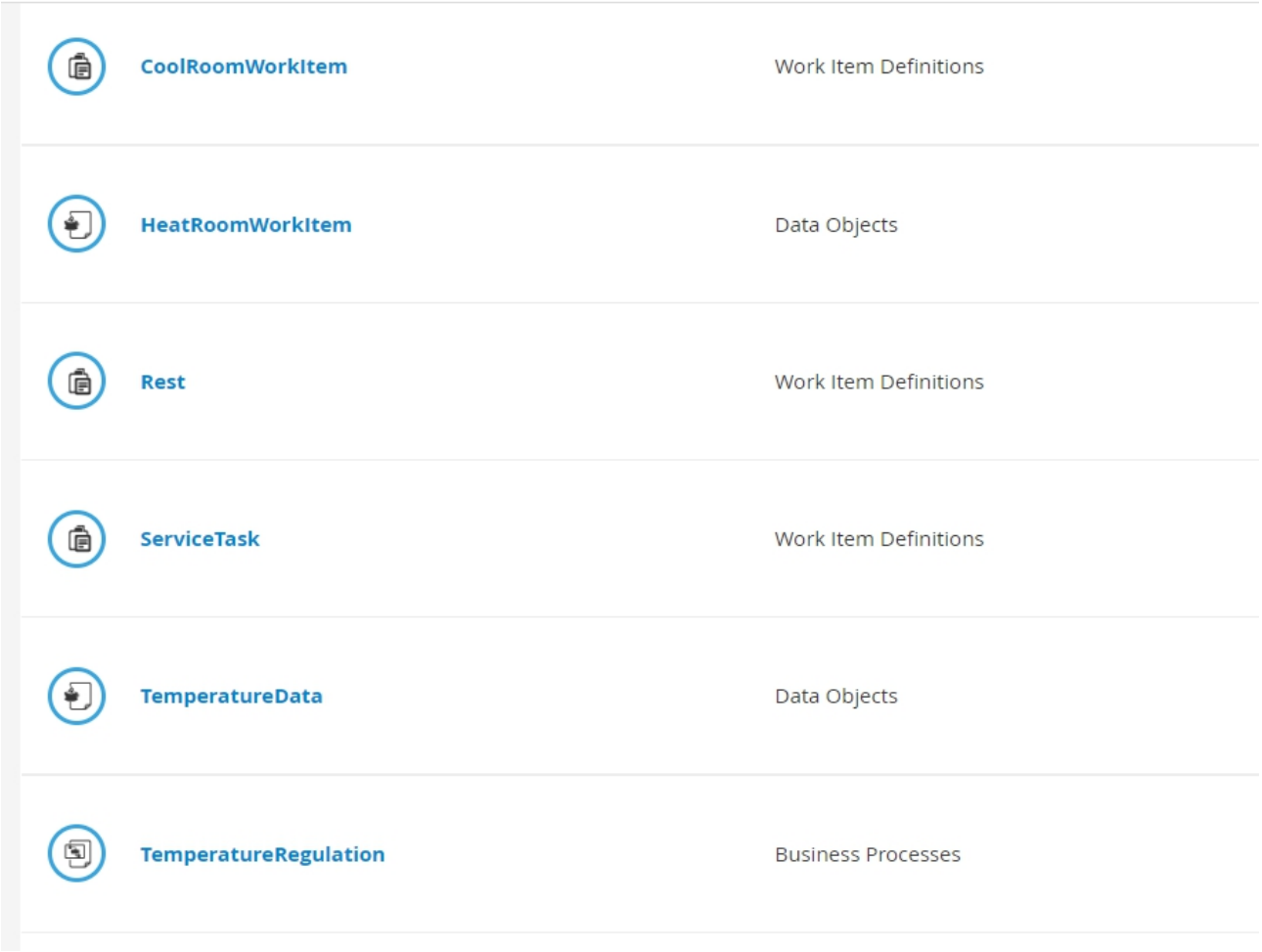

Рисунок 4.20 - Перелік ассетів проекту

Інтерфейс користувача business central надає багато можливостей для розробки. Якщо є необхідність, можна склонувати проект локально і по досягненню цілі, вносити зміни до серверу. Можна створити власний обробник наступним чином:

```
public class HeatRoomWorkItemHandler extends
AbstractLogOrThrowWorkItemHandler {

    @Inject
    ProcessService processService;

    public void executeWorkItem(WorkItem workItem, WorkItemManager
manager) {
        try {
```

```
            String roomNumber =
processService.getProcessInstanceVariable(workItem.getProcessInstanceId(),
"roomNumber");

            String temperatureRegulatorStatus = "s|heat";
            String iotDeviceUrl
                    = apiUrl +
"iot/d?k=4jggokgpepnvsb2uv4s40d59ov&i=temperatureRegulator" + roomNumber;
            restTemplate.exchange(iotDeviceUrl, HttpMethod.POST, new
HttpEntity<>(temperatureRegulatorStatus), Void.class);

            manager.completeWorkItem(workItem.getId(), "");
        } catch (Throwable cause) {
            handleException(cause);
        }
        System.out.println("Work finished");
    }

    public void abortWorkItem(WorkItem workItem, WorkItemManager manager)
{
    }
}
```

Після реалізації логіки, для того, щоб користуватися обробником в bpmn діаграмі проекту, необхідно в ассеті WorkDefinitions.wid створити запис з бажаними вхідними параметрами та метаінформацією про оформлення та кодову реалізацію. Також можна завантажити вже реалізовані обробники, дублювати їх об’явлення та налаштовувати під себе, де це має сенс.

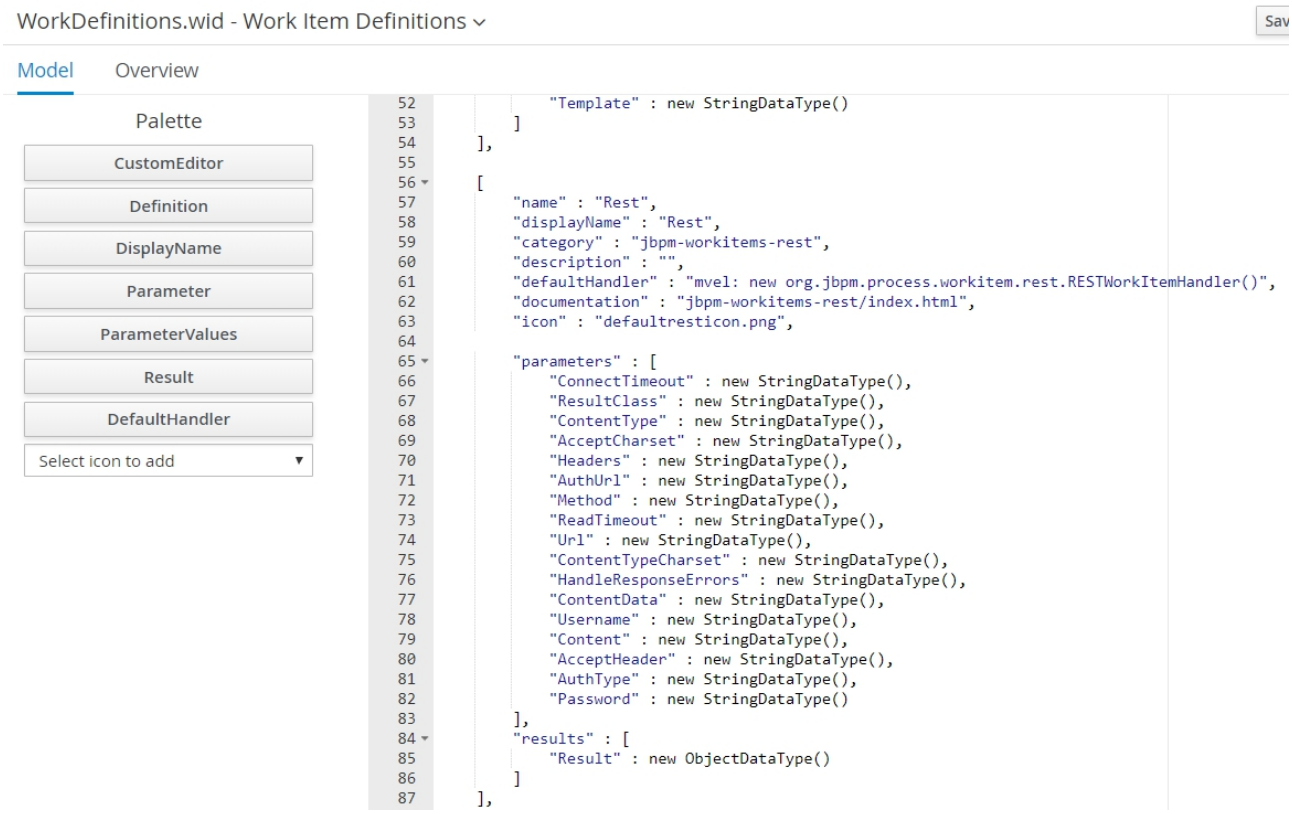

Рисунок 4.21 - Файл з об’явленням власних обробників

Аналогічно як і в Camunda, можна подивитись статистичні дані по процесу та екземплярам процеса.

Наявна реалізація Rest Service Task, яка надає можливість зробити запит до зовнішнього сервісу з мінімальними вимогами до конфігурації. Відповідь можна записати в змінну конктексту екземпляру процесу.

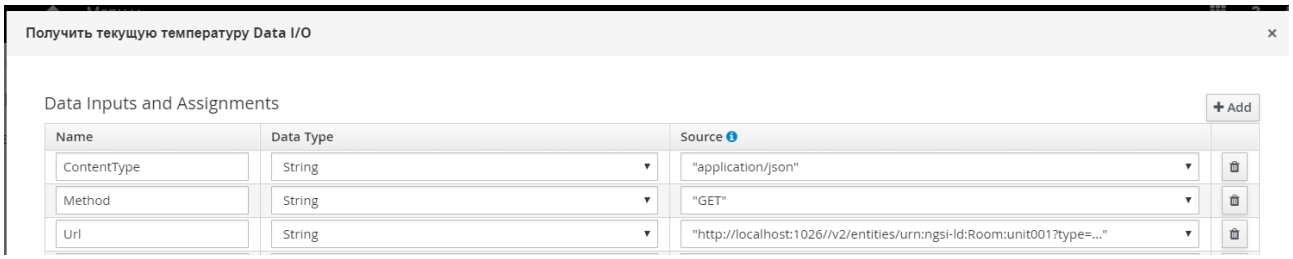

Рисунок 4.22 - Вхідні змінні Rest Service Task

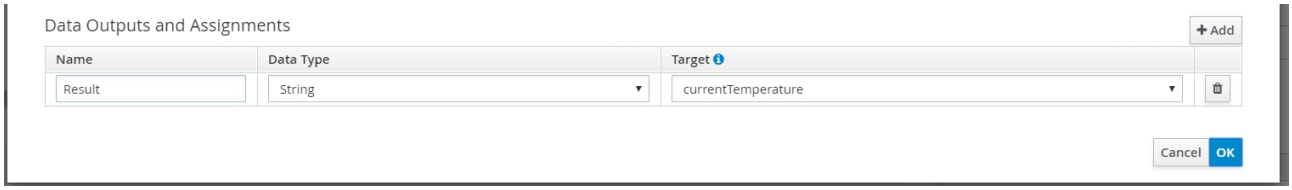

Рисунок 4.23 - Вихідні змінні Rest Service Task

В jbpm в сервісних завдань присутня можливість задати логіку, яка

виконається перед початком роботи обробника та після завершення його роботи. Це дозволяє виконувати необхідні дії, не створюючи нових блоків на діаграмі.

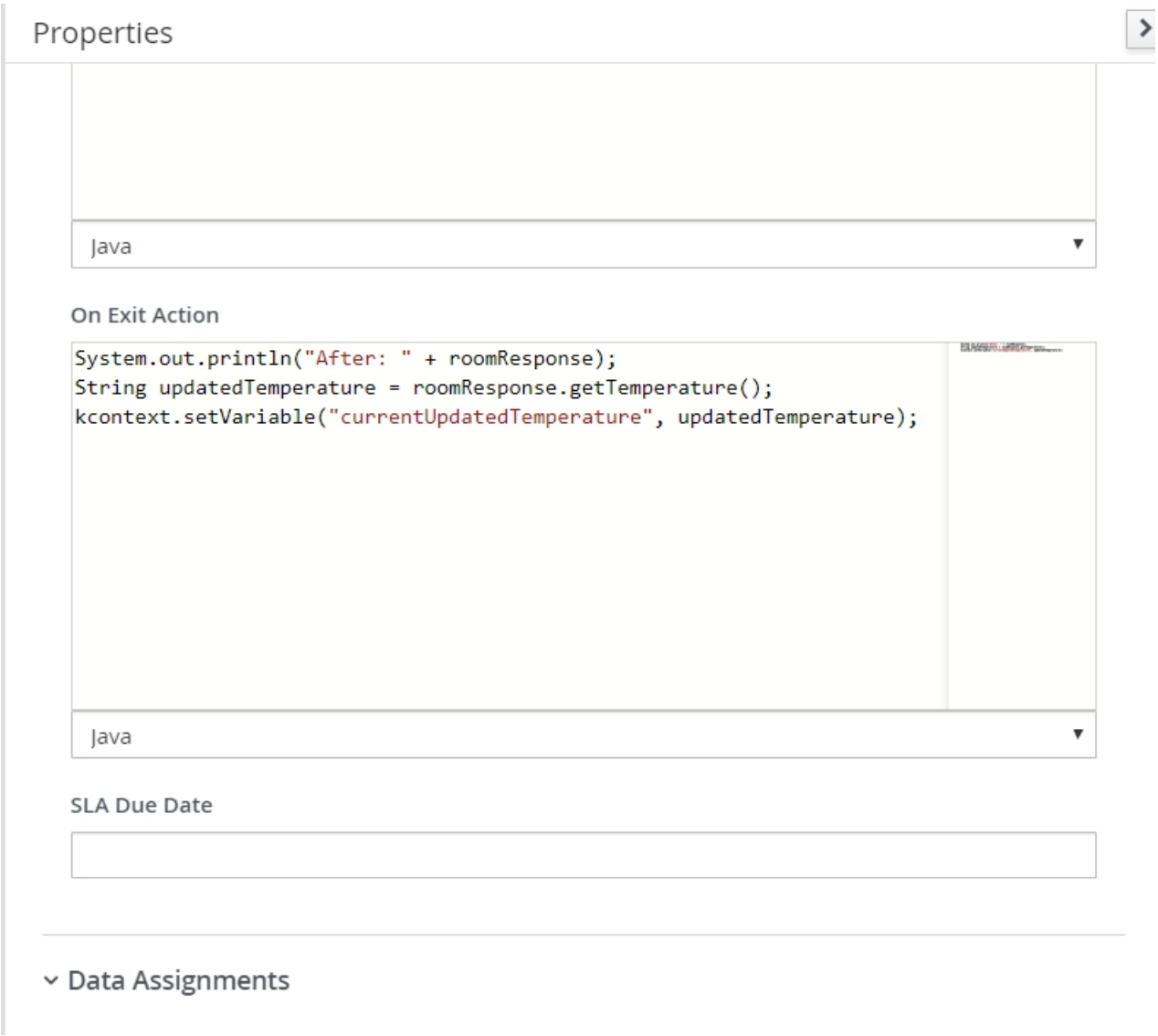

Рисунок 4.24 - Реалізація дій перед початком виконяння сервісного завдання та після його виконання

4.4 Висновки до розділу 4

У цьому розділі було встановлено і налаштовано движки CamundaBPM та jBPM, розглянуто основні можливості систем для оркестрації мікросервісів. Обидві системи надають схожий функціонал. jBPM дозволяє склонувати собі проект, підключившись до сервера через ssh, тобто не є обов'язковим проводити моделювання виключно через інтерфейс користувача чи середовище розробки. Також jBPM має реалізацію Rest service task, що дозволяє швидко налаштувати

виклики до зовнішніх сервісів з необхідними параметрами без зайвої розробки.

Camunda, на відміну від jBPM, надає для реалізації зовнішні завдання, що дозволяє створити запис у чергу завдань і продовжити виконання процесу, коли завдання буде виконано. Це вирішує проблему збоїв або недоступності зовнішнього сервісу.

Деякі переваги Fiware:

Легко впроваджувати, забезпечувати сумісність, легко масштабовані, відкриті стандарти API. Наявність GE, що надають можливості, які допомагають підключатися до пристроїв IoT, обробки даних та носіїв даних у реальному часі у великих масштабах.

5 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

5.1 Вступ

Функціонально-вартісний аналіз - це технологія, що дозволяє оцінити реальну вартість продукту або послуги відносно до організаційної структури компанії. Як прямі, так і непрямі витрати розподіляються за продуктами та послугами в залежності від обсягу ресурсів, необхідних на кожному з етапів виробництва. Дії, виконані на цих етапах, в контексті методу ФВА називаються функціями.

Мета ФВА полягає в забезпеченні правильного розподілу коштів, що виділяються на виробництво продукції або надання послуг, за прямими і непрямими витратам. Це дозволяє найбільш реалістично оцінювати витрати компанії.

По суті, метод ФВА працює за наступним алгоритмом:

Визначається послідовність функцій, необхідних для виробництва товару або послуги. Спочатку виявляються всі можливі функції. Вони розподіляються за двома групами: ті, які впливають на цінність товару / послуги і які не впливають. Далі на цьому етапі проводиться оптимізація послідовності: усуваються або скорочуються кроки, які не впливають на цінність, і скорочуються витрати. Для кожної функції визначаються повні річні витрати і кількість робочих годин. Для кожної функції визначається кількісна характеристика джерела витрат.

Після того як для всіх функцій будуть їх джерела витрат, проводиться остаточний розрахунок витрат на виробництво конкретного продукту або послуги.

Таким чином, використовуючи цей метод можна швидко оцінити обсяг

прибутку, очікуваний від виробництва того чи іншого товару або послуги.

Якщо вихідна оцінка витрат виконана правильно, то дохід (до виплати податків) буде дорівнює різниці між продажною ціною і витратами, розрахованими за методом ФВА. Крім того, відразу стане ясно, виробництво яких продуктів або послуг виявиться збитковим (їх ціна при реалізації буде нижче розрахункових витрат). На основі цих даних можна швидко прийняти коригуючі заходи, в тому числі переглянути цілі і стратегії бізнесу на найближчі періоди.

5.2 Постановка задачі техніко-економічного аналізу

У роботі застосовується метод ФВА для проведення технікоекономічного аналізу розробки. Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

- швидкість обробки даних, доступ до інформації екземплярів бізнес-процесу в реальному часі;
- зручність та зрозумілість для користувачів та розробників;
- передбачати мінімальні витрати на впровадження програмного продукту.

5.2.1 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка програмного продукту, який моделює простий бізнес-процес, утворений декількома мікросервісами з оркестрацією в BPMS. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір мови програмування;

F_2 – вибір протоколу та методу спілкування сервісів;

F_3 – вибір інтерфейсу програмного продукту.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1:

а) Java;

б) Golang;

Функція F_2:

а) HTTP;

б) Брокер повідомлень Kafka;

Функція F_3:

а) REST API;

б) Web UI.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 5.1).

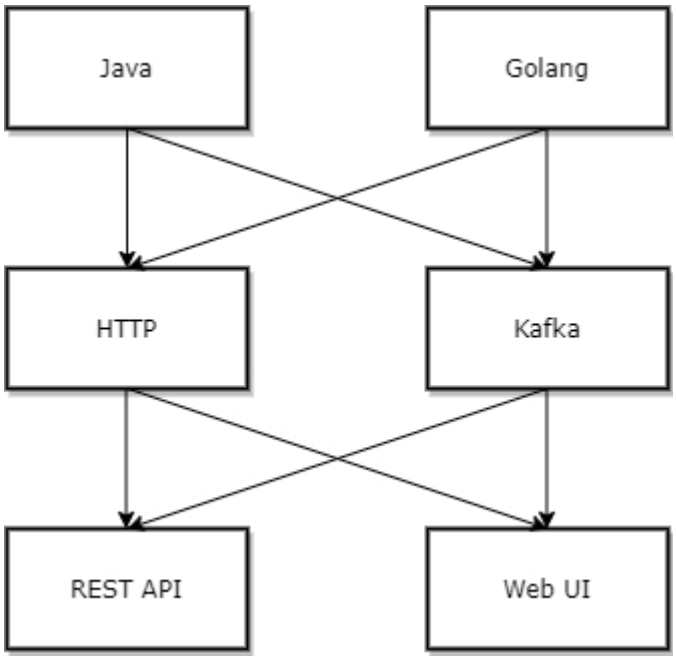

Рисунок 5.1 - Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіанти основних функцій.

Таблиця 5.1 - Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	*А*	Доступність готових рішень, легка інтеграція з розглянутими BPMS	Багато часу на розробку прототипу сервісу
	Б	Код швидко виконується, легко орієнтуватися в сервісах	Менше реалізованих залежностей, більше часу на гнучку обробку моделей даних
F_2	*А*	Легкість налаштування	Синхронність
	Б	Асинхронність, надійність	Додатковий час на конфігурацію та вивчення
F_3	*А*	Необхідність лише документування готових API	Незручність використання кінцевими користувачами
	Б	Легкість у використанні кінцевими користувачами	Більше часу на розробку

На основі цієї карти будуємо позитивно-негативну матрицю варіантів основних функцій (Таб.5.1). Робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1:

Перевагу віддаємо швидкості вивчення, простоті використання та наявності стандартних бібліотек для обчислення. Для спрощення роботи по написанню коду варіант Б має бути відкинутий.

Функція F_2:

Аналогічно до F_1.

Функція F_3:

Можна розглянути обидва варінти.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

F_1а – F_2а – F_3а

F_1а – F_2*а* – F_3б

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

5.3 Обґрунтування системи параметрів ПП

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- *X1* – час обробки запиту клієнта сервісами;
- *X2* – кількість сервісів;
- *X3* – об’єм даних, що передаються;
- *X4* – потенційний об’єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у таблиці 5.2.

Таблиця 5.2 - Основні параметри ПП

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Час обробки запиту клієнта сервісами	X1	мс	1000	500	100
Кількість сервісів	X2	К-ть	2	3	4
Об’єм даних, що передаються	X3	Кб	2000	800	500
Потенційний об’єм програмного коду	X4	кількість рядків коду	5000	3500	2000

За даними таблиці 5.2 будуються графічні характеристики параметрів – рис. 5.2 – рис. 5.5.

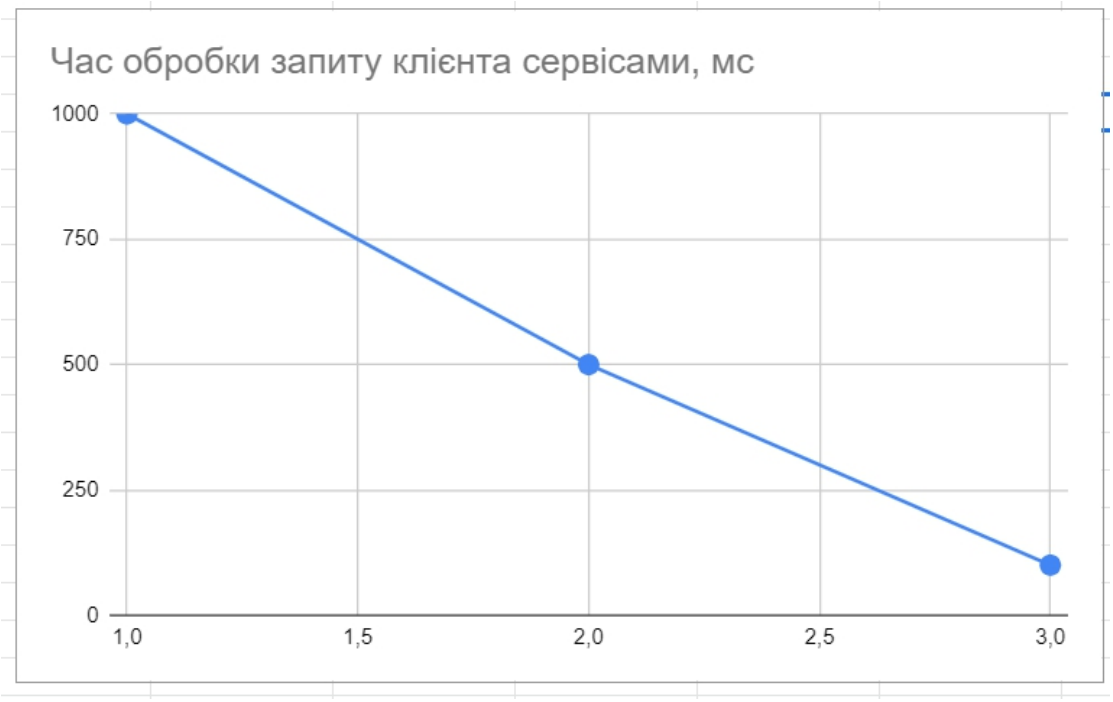

Рисунок 5.2 - X1, час обробки запиту клієнта сервісами

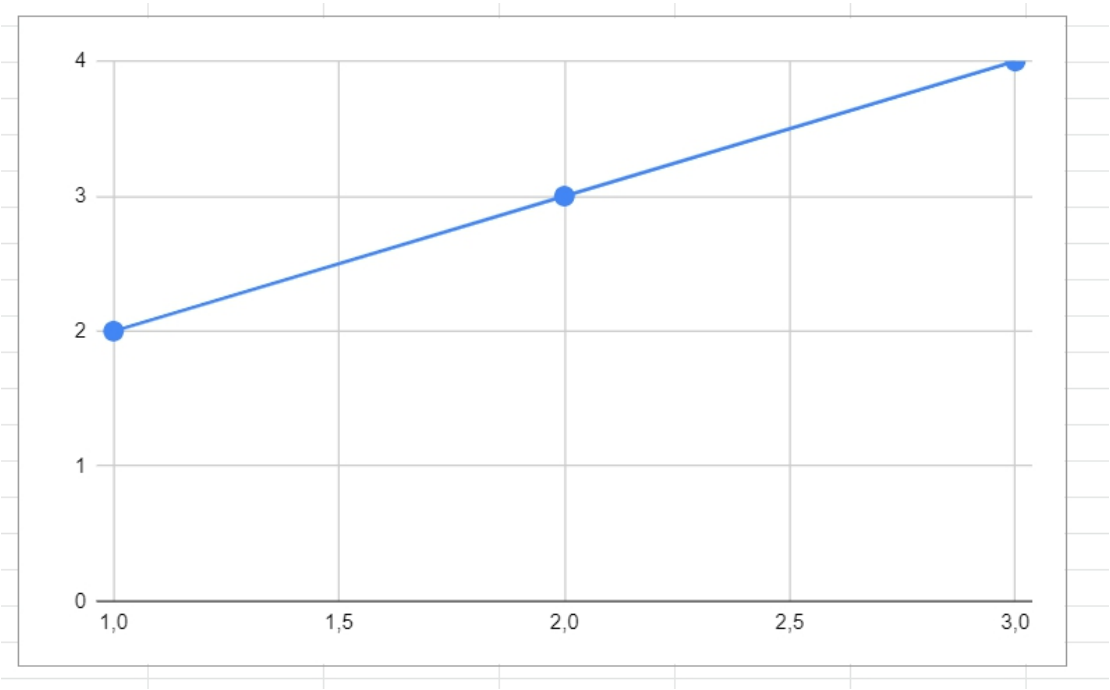

Рисунок 5.3 - X2, кількість сервісів

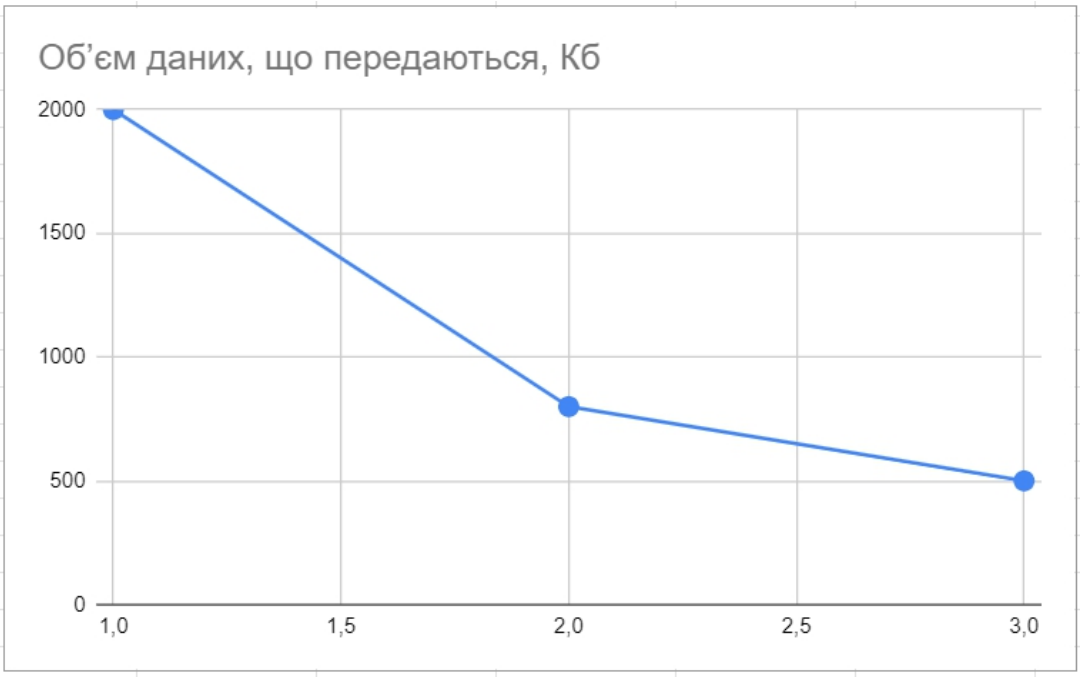

Рисунок 5.4 - X3, об’єм даних, що передаються

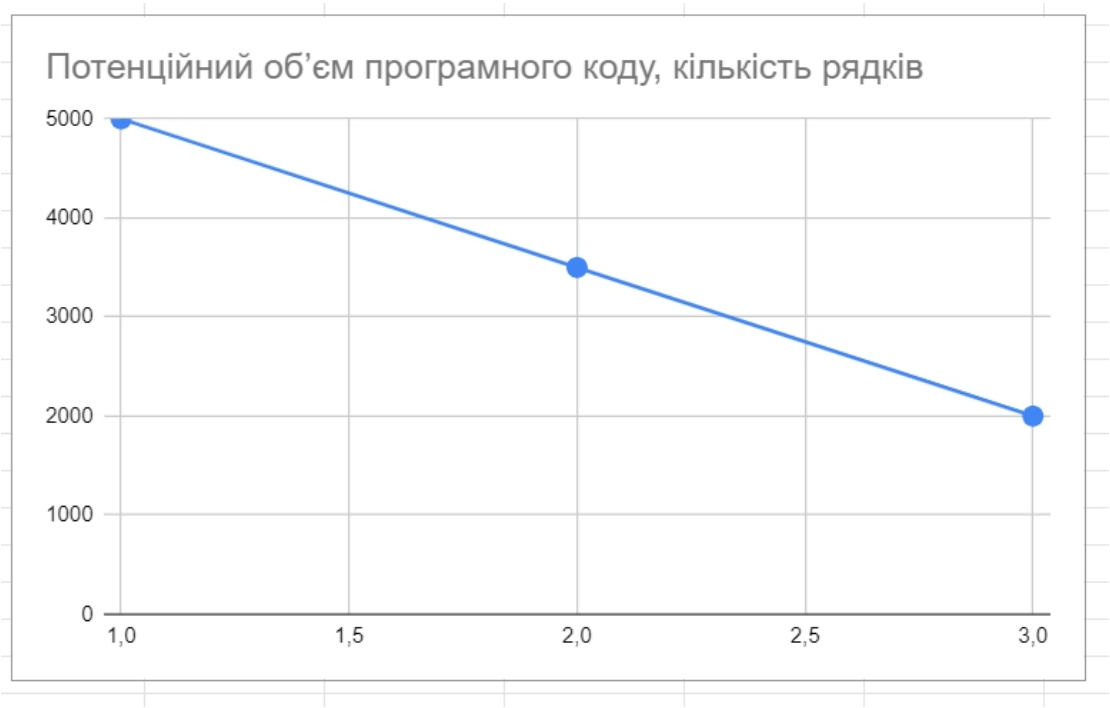

Рисунок 5.5 - X4, потенційний об’єм програмного коду

5.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 5.4.

Таблиця 5.3 - Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Час обробки запиту клієнта сервісами	мс	4	4	4	4	4	4	4	28	10,5	110,25
X2	Кількість сервісів	К-ть	2	3	3	3	3	3	3	20	2,5	6,25
X3	Об’єм даних, що передаються	Кб	1	2	1	1	1	1	1	8	-9,5	90,25
X4	Потенційний об’єм програмного коду	кількість рядків коду	3	1	2	2	2	2	2	14	-3,5	12,25
	Разом		10	10	10	10	10	10	19	70	0	219

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів:

$$R_i = \sum_{j=1}^{N} r_{ij} \qquad (5.1)$$

б) загальна сума рангів:

$$R_i = \frac{Nn(n+1)}{2} = 70 \qquad (5.2)$$

де N – число експертів,

n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5 \qquad (5.3)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T \qquad (5.4)$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 219 \qquad (5.5)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3-n)} = \frac{12 \cdot 219}{7^2(4^3-4)} = 0{,}89 > W_k = 0{,}67 \qquad (5.6)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 5.5.

Таблиця 5.4 - Попарне порівняння параметрів.

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	>	>	>	>	>	>	>	1,5
X1 і X3	>	>	>	>	>	>	>	>	1,5
X1 і X4	>	>	>	>	>	>	>	>	1,5
X2 і X3	>	>	>	>	>	>	>	>	1,5
X2 і X4	<	>	>	>	>	>	>	>	1,5
X3 і X4	<	>	<	<	<	<	<	<	0,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5 \ при \ X_i > X_j \\ 1.0 \ при \ X_i = X_j \\ 0.5 \ при \ X_i < X_j \end{cases} \quad (5.7)$$

З отриманих числових оцінок переваги складемо матрицю А=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{ві} = \frac{b_i}{\sum_{i=1}^{n} b_i} \quad (5.8)$$

$$b_i = \sum_{i=1}^{N} a_{ij} \quad (5.9)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{ві} = \frac{b_i^{'}}{\sum_{i=1}^{n} b_i^{'}} \quad (5.10)$$

$$b_i^{'} = \sum_{i=1}^{N} a_{ij} b_j \quad (5.11)$$

.Як видно з таблиці 5.6, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 5.5 - Розрахунок вагомості параметрів

Параметриx_i	Параметриx_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{вi}$	b_i^1	$K_{вi}^1$	b_i^2	$K_{вi}^2$
X1	1,0	1,5	1,5	1,5	5,5	0,344	21,25	0,36	77,875	0,36
X2	0,5	1,0	1,5	1,5	4,5	0,281	16,25	0,275	59,125	0,274
X3	0,5	0,5	1,0	0,5	2,5	0,156	9,25	0,157	34,125	0,158
X4	0,5	0,5	1,5	1,0	3,5	0,218	12,25	0,208	44,875	0,208
Всього:					16	1	59	1	216	1

5.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X1* (час обробки запиту клієнта сервісами), *X2* (кількість сервісів) та *X3* (об’єм даних, що передаються) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X4* (потенційний об’єм програмного коду) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 5.7):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j} \qquad (5.11)$$

де n – кількість параметрів;

$K_{вi}$– коефіцієнт вагомості i–го параметра;

B_i – оцінка i–го параметра в балах.

Таблиця 5.7 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X2	3	5	0,274	1,37
F2	А	X1	500	7	0,36	2,52
		X3	800	4	0,158	0,632
F3	А	X4	2000	6	0,208	1,248
	Б	X4	2500	5	0,208	1,04

За даними з таблиці 5.7 за формулою:

$$K_K = K_{ТУ}[F_{1k}] + K_{ТУ}[F_{2k}] + \ldots + K_{ТУ}[F_{zk}] \quad (5.12)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 1{,}37 + 2{,}52 + 0{,}632 + 1{,}248 = 5{,}77,$$

$$K_{K2} = 1{,}37 + 2{,}52 + 0{,}632 + 1{,}04 = 5{,}562.$$

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

5.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Технічна реалізація бізнес процесу;

2. Моделювання бізнес процесу.

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_М \cdot K_{СТ} \cdot K_{СТ.М} \qquad (5.13)$$

де T_P – трудомісткість розробки ПП;

$K_П$ – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

$K_М$ – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: T_P =90 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $K_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{СТ}$ = 0.8. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 90 \cdot 1.7 \cdot 0.8 = 122.4 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_Р$ =27 людино-днів, $К_П$ =0.9,$К_{СК}$ = 1,$К_{СТ}$ =0.8:

$$T_2 = 27 \cdot 0.9 \cdot 0.8 = 19.44 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = (122.4 + 19.44 + 4.8 + 19.44) \cdot 8 = 1328{,}64 \text{ людино-годин.}$$

$$T_{II} = (122.4 + 19.44 + 6.91 + 19.44) \cdot 8 = 1345.52 \text{ людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 20000 грн., один проектувальник процесів з окладом 15000. Визначимо середню зарплату за годину за формулою:

$$С_Ч = \frac{М}{T_m \cdot t}\ грн. \qquad (5.14)$$

де М – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$С_Ч = \frac{20000+20000+15000}{3 \cdot 21 \cdot 8} = 109{,}13\ грн. \qquad (5.15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{ЗП} = C_ч \cdot T_i \cdot К_Д \quad (5.16)$$

де $С_ч$– величина погодинної оплати праці програміста;

T_i – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 109{,}13 \cdot 1328.64 \cdot 1.2 = 173993{,}38$ грн.

II. $С_{ЗП} = 109{,}13 \cdot 1345.52 \cdot 1.2 = 176203{,}92$ грн.

Відрахування на єдиний соціальний внесок становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 173993{,}38 \cdot 0.22 = 38278{,}54$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 176203{,}92 \cdot 0.22 = 38764{,}86$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 20000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 20000 \cdot 0{,}2 = 48000 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 48000 \cdot (1 + 0.2) = 57600 \text{ грн.}$$

Відрахування на соціальний внесок:

$$С_{ВІД}= С_{ЗП} \cdot 0.22 = 57600 \cdot 0,22 =12672грн.$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 40000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 40000 = 11500 грн.,$$

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$– річна норма амортизації;

$Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 40000 \cdot 0.05 = 2300 грн.,$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} =(Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 12 – 16) \cdot 8 \cdot 0.9 =$$

$$= 1677.6 годин,$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t –кількість робочих годин в день;

$К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1677.6 \cdot 0,3 \cdot 3,51 \cdot 0,4 = 706,61 \text{ грн.},$$

де $N_С$ – середньо-споживча потужність приладу;

$К_З$– коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0.67 = 40000 \cdot 0,67 = 26800 \text{грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н \qquad (5.17)$$

$$С_{ЕКС} = 57600 + 12672 + 2300 + 11500 + 1150 + 856,48 + 26800 = 112728,61 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 112878,48 / 1677.6 = 67,2 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_М = C_{М-Г} \cdot Т \quad (5.18)$$

I. $C_М = 67,29 \cdot 1328,64 = 89284,61$ грн.

II. $C_М = 67,29 \cdot 1345.52 = 90418,94$ грн.

Накладні витрати складають 67% від заробітної плати:

$$C_Н = C_{ЗП} \cdot 0,67 \quad (5.19)$$

I. $C_Н = 173993,38 \cdot 0,67 = 116575,57$ грн.

II. $C_Н = 176203,92 \cdot 0,67 = 118056,63$ грн.

Отже, вартість розробки ПП за варіантами становить:

$$C_{ПП} = C_{ЗП} + C_{ВІД} + C_М + C_Н \quad (5.20)$$

I. $C_{ПП} = 173993,38 + 38278,54 + 89404,19 + 116575,57 = 418132,2$ грн.

II. $C_{ПП} = 176203,92 + 38764,86 + 90540,04 + 118056,63 = 423444,55$ грн.

5.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР_j} = К_{Кj}/С_{Фj}, \quad (5.21)$$

$$К_{ТЕР1} = 5,77/ 418132,2 = 1,38 \cdot 10^{-5},$$

$$К_{ТЕР2} = 5,562/ 423444,55 = 1,31 \cdot 10^{-5}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1}= 1,38 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР} = 1,38 \cdot 10^{-5}$.

Цей варіант реалізації програмного продукту має такі параметри:

- мова програмування – Java;
- сервіси обмінюються даними черз HTTP виклики;
- API як вихідний інтерфейс.

5.8 Висновки до розділу 5

Проведено повний функціонально-вартісний аналіз програмного продукту. Визначено та проведено оцінку основних функцій програмного продукту.

Визначено параметри, які характеризують програмний продукт. Проведено експертне оцінювання параметрів та аналіз якості варіантів реалізації функцій.

Проведено економічний аналіз варіантів розробки – трудомісткість, витрати на заробітну плату та інші витрати.

На основі аналізу вибрано варіант реалізації програмного продукту.

Висновки

В ході виконання роботи було налаштовано системи управління бізнес процесами Camunda та jBPM, які надають широкі можливості для розробки, підтримки, автоматизації та реалізації бізнес-процесів, такі як: моніторинг, адміністрування, інтерфейс користувача для проектування та збору статистичних даних, які допомогають приймати рішення для оптимізації бізнес-процесів.

Метою дипломної роботи було дослідити системи в якості оркестраторів мікросервісів.

Однією з найпоширеніших технологій для опису бізнес-процесу є стандарт BPMN 2.0. Цей стандарт спочатку був розроблений для опису бізнес-процесу без усіх технічних деталей програмної системи. Діаграму BPMN легко зрозуміти. Окрім загального опису бізнес-процесу, модель BPMN може бути виконана за допомогою движку процесу. Система управління робочим процесом контролює кожне завдання від початкової точки до його закінчення.

Програми BPM (управління бізнес-процесами), як правило, дуже складні, оскільки їм часто потрібно вмістити дуже різні інтерфейси користувача, підтримку декількох баз даних, сховищ документів. Ще більше ускладнює справу той факт, що програма BPM повинна постійно розвиватися для підтримки змін у бізнес-середовищі.

Говорячи конкретно про Camunda та jBPM, то вони мають дуже подібний функціонал. jBPM надає реалізацю Rest виклику до зовнішнього сервісу та слідує концепції “bpm з мінімальною кількістю коду”, тобто, якщо присутня кодова реалізація бізнес-логіки, то ми можемо прив’язати її до діаграми бізнес-процесу без зайвої розробки.

Camunda, на відміну від jBPM, надає для реалізації зовнішні завдання, що дозволяє створити запис у чергу завдань і продовжити виконання процесу, коли завдання буде виконано. Це вирішує проблему збоїв або недоступності

зовнішнього сервісу.

Тобто, якщо постає питання, яку вибрати BPMS, то тут немає однозначної відповіді, оскільки за домогою підтримуваних нотацій, ми маємо однакові можливості для опису бізнес-процесу. Слід відштовхуватись від експертизи команди розробників, щоб вони могли швидко і зручно підтримувати бізнес-зміни. Якщо одна з систем надає виключний функціонал для зв'язку із зовнішніми сервісами, то це не привід одразу обирати її, оскільки можна створити власну реалізацію для вирішення проблеми або використати вже готові рішення.

В цілому ж, використовуючи розглянуті BPMS, ми можемо достатньо швидко описати оркестрацію мікросервісів, які задіяні в процесі.

З очевидних недоліків, слід виділити, що розробка прототипу, якщо ви вперше працюєте з системами, займає дуже багато часу, тому слід правильно оцінювати переваги, які ви хочете отримати від використання систем. Також з jBPM дуже важко у відкритому доступі отримати актуальні практичні навички. Більшість джерел, які допомагають розпочати розробку, не оновлювались кілька років.

Підводячи підсумок, Camunda BPM та jBPM можуть бути використані як движок робочого процесу в мікросервісній архітектурі. Очевидні переваги: зменшення розриву між бізнесом та розробниками. Зберігання бізнес-логіки в мікросервісі робить робочий процес дуже легким та керованим.

Перелік використаних джерел

1.https://www.happyfox.com/what-is-business-process-management/ (дата звертання: 17.04.2021)

2.https://pure.aston.ac.uk/ws/files/26737118/1.4707613.pdf (дата звертання: 17.04.2021)

3.https://www.modernanalyst.com/Resources/Articles/tabid/115/ID/3189/Integrating-BPMN-and-DMN.aspx (дата звертання: 17.04.2021)

4.https://elib.bsu.by/bitstream/123456789/224346/1/tsalko_dip.pdf (дата звертання: 17.04.2021)

5.https://medium.com/capital-one-tech/comparing-and-contrasting-open-source-bpm-projects-196833f23391 (дата звертання: 25.04.2021)

6.https://docs.camunda.org/manual/7.15/user-guide/ (дата звертання: 25.04.2021)

7.https://docs.jboss.org/jbpm/release/7.3.0.Final/jbpm-docs/html_single/#_jbpmoverview (дата звертання: 25.04.2021)

8.https://assets.ctfassets.net/vpidbgnakfvf/1U7qZ7K1mMe2KC2yc6SKS4/baae9bf53217557638bee90ebba13442/Camunda_vs_JBoss_jBPM_EN.pdf (дата звертання: 25.04.2021)

9.https://tproger.ru/articles/modelirovanie-biznes-processov-praktika-ispolzovanija-camunda-bpm-v-java-razrabotke/ (дата звертання: 25.04.2021)

10.https://docs.jboss.org/jbpm/v6.0/userguide/jBPMOverview.html (дата звертання: 30.04.2021)

11.https://www.linkedin.com/pulse/bpmn-based-microservices-orchestration-denys-g-santos/ (дата звертання: 30.04.2021)

12.https://blog.bernd-ruecker.com/the-microservice-workflow-automation-cheat-sheet-fc0a80dc25aa (дата звертання: 07.05.2021)

13.https://onesaitplatform.atlassian.net/wiki/spaces/OP/pages/273612801/Microservice+Orchestration+with+BPM+Engine+Camunda+and+Saga+pattern#BPM-Engine-as-orchestrator (дата звертання: 07.05.2021)

14.https://access.redhat.com/documentation/en-us/reference_architectures/2017/html-single/business_process_management_with_red_hat_jboss_bpm_suite_6.3/index (дата звертання: 11.05.2021)

15. https://fiware-tutorials.readthedocs.io/en/latest/index.html (дата звертання: 17.06.2021)

Додаток А

StartProcess.java

```
@RestController
@RequiredArgsConstructor
public class StartProcess {

    private final RuntimeService runtimeService;
    private final ObjectMapper objectMapper;

    @PostMapping("/start/{processDefinitionKey}")
    public String startProcess(@RequestBody ChangeTemperatureRequest
changeTemperatureRequest, @PathVariable String processDefinitionKey){
        try{
            Map<String,Object> variables =
objectMapper.convertValue(changeTemperatureRequest, new
TypeReference<Map<String, Object>>() {});

runtimeService.startProcessInstanceByKey(processDefinitionKey,variables);
            return String.format("Process Started variables = %s",
variables.toString());
        } catch (Exception e){
            return "Failed to start process Instance";
        }
    }
}
```

CamundaApplication.java

```
@SpringBootApplication
@EnableProcessApplication
@EnableScheduling
public class CamundaApplication {

    public static void main(String... args) {
        SpringApplication.run(CamundaApplication.class, args);
    }

    @Bean
    public RestTemplate restTemplate(RestTemplateBuilder builder) {
        return builder.build();
    }
}
```

temperatureRegulation.bpmn

```
<?xml version="1.0" encoding="UTF-8"?>
<bpmn:definitions
xmlns:bpmn="http://www.omg.org/spec/BPMN/20100524/MODEL"
```

```
xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
xmlns:camunda="http://camunda.org/schema/1.0/bpmn"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
id="Definitions_1kqd77s" targetNamespace="http://bpmn.io/schema/bpmn"
exporter="Camunda Modeler" exporterVersion="4.7.0">
  <bpmn:process id="regulateTemperature" isExecutable="true">
    <bpmn:startEvent id="StartEvent_1" name="Изменение температуры">
      <bpmn:outgoing>Flow_1iqlerf</bpmn:outgoing>
    </bpmn:startEvent>
    <bpmn:endEvent id="Event_1rbrg4v" name="Проблемы с изменением">
      <bpmn:incoming>Flow_13fp5sx</bpmn:incoming>
    </bpmn:endEvent>
    <bpmn:sequenceFlow id="Flow_1iqlerf" sourceRef="StartEvent_1"
targetRef="Activity_06pdvle" />
    <bpmn:sequenceFlow id="Flow_1jksmcm" sourceRef="Activity_06pdvle"
targetRef="Gateway_0gdisyp" />
    <bpmn:sequenceFlow id="Flow_11g7o3f" sourceRef="Activity_10tf96a"
targetRef="Activity_0a5jixl" />
    <bpmn:serviceTask id="Activity_06pdvle" name="Получить текущую
температуру" camunda:asyncAfter="true"
camunda:delegateExpression="${orionContextController}">
      <bpmn:incoming>Flow_1iqlerf</bpmn:incoming>
      <bpmn:outgoing>Flow_1jksmcm</bpmn:outgoing>
    </bpmn:serviceTask>
    <bpmn:serviceTask id="Activity_10tf96a" name="Повысить температуру"
camunda:asyncAfter="true"
camunda:delegateExpression="${heatRoomController}">
      <bpmn:incoming>Flow_0k5uszl</bpmn:incoming>
      <bpmn:outgoing>Flow_11g7o3f</bpmn:outgoing>
    </bpmn:serviceTask>
    <bpmn:exclusiveGateway id="Gateway_0qfkxe0">
      <bpmn:incoming>Flow_04cetpe</bpmn:incoming>
      <bpmn:outgoing>Flow_1c4op3d</bpmn:outgoing>
      <bpmn:outgoing>Flow_1mv0w60</bpmn:outgoing>
    </bpmn:exclusiveGateway>
    <bpmn:exclusiveGateway id="Gateway_0gdisyp">
      <bpmn:incoming>Flow_1jksmcm</bpmn:incoming>
      <bpmn:outgoing>Flow_0k5uszl</bpmn:outgoing>
      <bpmn:outgoing>Flow_1lcswc2</bpmn:outgoing>
      <bpmn:outgoing>Flow_0af3r55</bpmn:outgoing>
    </bpmn:exclusiveGateway>
    <bpmn:sequenceFlow id="Flow_0k5uszl" sourceRef="Gateway_0gdisyp"
targetRef="Activity_10tf96a">
      <bpmn:conditionExpression
xsi:type="bpmn:tFormalExpression">${currentTemperature &lt;
temperature}</bpmn:conditionExpression>
    </bpmn:sequenceFlow>
    <bpmn:sequenceFlow id="Flow_1lcswc2" sourceRef="Gateway_0gdisyp"
targetRef="Activity_05v8qaw">
```

```
      <bpmn:conditionExpression
xsi:type="bpmn:tFormalExpression">${currentTemperature &gt;
temperature}</bpmn:conditionExpression>
    </bpmn:sequenceFlow>
    <bpmn:serviceTask id="Activity_05v8qaw" name="Понизить температуру"
camunda:delegateExpression="${coolRoomController}">
      <bpmn:incoming>Flow_1lcswc2</bpmn:incoming>
      <bpmn:outgoing>Flow_0pqxlil</bpmn:outgoing>
    </bpmn:serviceTask>
    <bpmn:endEvent id="Event_001smcp" name="Установленная температура =
необходимой">
      <bpmn:incoming>Flow_0af3r55</bpmn:incoming>
    </bpmn:endEvent>
    <bpmn:sequenceFlow id="Flow_0af3r55" sourceRef="Gateway_0gdisyp"
targetRef="Event_001smcp">
      <bpmn:conditionExpression
xsi:type="bpmn:tFormalExpression">${currentTemperature==
temperature}</bpmn:conditionExpression>
    </bpmn:sequenceFlow>
    <bpmn:sequenceFlow id="Flow_0pqxlil" sourceRef="Activity_05v8qaw"
targetRef="Activity_0a5jixl" />
    <bpmn:sequenceFlow id="Flow_09u1thy" sourceRef="Activity_0a5jixl"
targetRef="Activity_19dm5k3" />
    <bpmn:endEvent id="Event_0m5u7hn" name="Изменено успешно">
      <bpmn:incoming>Flow_0daxlk0</bpmn:incoming>
    </bpmn:endEvent>
    <bpmn:sequenceFlow id="Flow_1c4op3d" sourceRef="Gateway_0qfkxe0"
targetRef="Activity_1xrz8z8">
      <bpmn:conditionExpression
xsi:type="bpmn:tFormalExpression">${currentUpdatedTemperature==
temperature}</bpmn:conditionExpression>
    </bpmn:sequenceFlow>
    <bpmn:userTask id="Activity_0a5jixl" name="Элемент для остановки и
демонстрации" camunda:assignee="demo">
      <bpmn:incoming>Flow_11g7o3f</bpmn:incoming>
      <bpmn:incoming>Flow_0pqxlil</bpmn:incoming>
      <bpmn:outgoing>Flow_09u1thy</bpmn:outgoing>
    </bpmn:userTask>
    <bpmn:sequenceFlow id="Flow_1mv0w60" sourceRef="Gateway_0qfkxe0"
targetRef="Activity_0ulzs8f">
      <bpmn:conditionExpression
xsi:type="bpmn:tFormalExpression">${currentUpdatedTemperature !=
temperature}</bpmn:conditionExpression>
    </bpmn:sequenceFlow>
    <bpmn:sequenceFlow id="Flow_13fp5sx" sourceRef="Activity_0ulzs8f"
targetRef="Event_1rbrg4v" />
    <bpmn:sequenceFlow id="Flow_0daxlk0" sourceRef="Activity_1xrz8z8"
targetRef="Event_0m5u7hn" />
    <bpmn:serviceTask id="Activity_1xrz8z8" name="Остановить систему
управления температурой"
camunda:delegateExpression="${stopSystemController}">
```

```
      <bpmn:incoming>Flow_1c4op3d</bpmn:incoming>
      <bpmn:outgoing>Flow_0daxlk0</bpmn:outgoing>
    </bpmn:serviceTask>
    <bpmn:serviceTask id="Activity_0ulzs8f" name="Сообщить владельцу о
проблемах в системе"
camunda:delegateExpression="${sendMessageController}">
      <bpmn:incoming>Flow_1mv0w60</bpmn:incoming>
      <bpmn:outgoing>Flow_13fp5sx</bpmn:outgoing>
    </bpmn:serviceTask>
    <bpmn:sequenceFlow id="Flow_04cetpe" sourceRef="Activity_19dm5k3"
targetRef="Gateway_0qfkxe0" />
    <bpmn:serviceTask id="Activity_19dm5k3" name="Ожидание обновления
температуры"
camunda:delegateExpression="${temperatureUpdateWaitController}">
      <bpmn:incoming>Flow_09u1thy</bpmn:incoming>
      <bpmn:outgoing>Flow_04cetpe</bpmn:outgoing>
    </bpmn:serviceTask>
  </bpmn:process>
  <bpmndi:BPMNDiagram id="BPMNDiagram_1">
    <bpmndi:BPMNPlane id="BPMNPlane_1" bpmnElement="regulateTemperature">
      <bpmndi:BPMNEdge id="Flow_1lcswc2_di" bpmnElement="Flow_1lcswc2">
        <di:waypoint x="460" y="272" />
        <di:waypoint x="460" y="400" />
        <di:waypoint x="540" y="400" />
        <bpmndi:BPMNLabel>
          <dc:Bounds x="471" y="356" width="64" height="27" />
        </bpmndi:BPMNLabel>
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="Flow_0k5uszl_di" bpmnElement="Flow_0k5uszl">
        <di:waypoint x="485" y="247" />
        <di:waypoint x="540" y="247" />
        <bpmndi:BPMNLabel>
          <dc:Bounds x="508" y="216" width="64" height="27" />
        </bpmndi:BPMNLabel>
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="Flow_11g7o3f_di" bpmnElement="Flow_11g7o3f">
        <di:waypoint x="640" y="247" />
        <di:waypoint x="710" y="247" />
        <di:waypoint x="710" y="310" />
        <di:waypoint x="760" y="310" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="Flow_1jksmcm_di" bpmnElement="Flow_1jksmcm">
        <di:waypoint x="390" y="247" />
        <di:waypoint x="435" y="247" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="Flow_1iqlerf_di" bpmnElement="Flow_1iqlerf">
        <di:waypoint x="208" y="247" />
        <di:waypoint x="290" y="247" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="Flow_0af3r55_di" bpmnElement="Flow_0af3r55">
        <di:waypoint x="460" y="222" />
```

```
        <di:waypoint x="460" y="100" />
        <di:waypoint x="572" y="100" />
        <bpmndi:BPMNLabel>
          <dc:Bounds x="464" y="110" width="77" height="40" />
        </bpmndi:BPMNLabel>
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="Flow_0pqxlil_di" bpmnElement="Flow_0pqxlil">
        <di:waypoint x="640" y="400" />
        <di:waypoint x="710" y="400" />
        <di:waypoint x="710" y="310" />
        <di:waypoint x="760" y="310" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="Flow_09u1thy_di" bpmnElement="Flow_09u1thy">
        <di:waypoint x="860" y="310" />
        <di:waypoint x="910" y="310" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="Flow_1c4op3d_di" bpmnElement="Flow_1c4op3d">
        <di:waypoint x="1115" y="310" />
        <di:waypoint x="1160" y="310" />
        <bpmndi:BPMNLabel>
          <dc:Bounds x="1113" y="272" width="52" height="27" />
        </bpmndi:BPMNLabel>
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="Flow_1mv0w60_di" bpmnElement="Flow_1mv0w60">
        <di:waypoint x="1090" y="335" />
        <di:waypoint x="1090" y="400" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="Flow_13fp5sx_di" bpmnElement="Flow_13fp5sx">
        <di:waypoint x="1090" y="480" />
        <di:waypoint x="1090" y="562" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="Flow_0daxlk0_di" bpmnElement="Flow_0daxlk0">
        <di:waypoint x="1260" y="310" />
        <di:waypoint x="1332" y="310" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge id="Flow_04cetpe_di" bpmnElement="Flow_04cetpe">
        <di:waypoint x="1010" y="310" />
        <di:waypoint x="1065" y="310" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNShape id="Activity_1v2wrwu_di"
bpmnElement="Activity_06pdvle">
        <dc:Bounds x="290" y="207" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="Gateway_0gdisyp_di"
bpmnElement="Gateway_0gdisyp" isMarkerVisible="true">
        <dc:Bounds x="435" y="222" width="50" height="50" />
        <bpmndi:BPMNLabel>
          <dc:Bounds x="657" y="126" width="66" height="14" />
        </bpmndi:BPMNLabel>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="_BPMNShape_StartEvent_2"
```

```
bpmnElement="StartEvent_1">
        <dc:Bounds x="172" y="229" width="36" height="36" />
        <bpmndi:BPMNLabel>
          <dc:Bounds x="157" y="272" width="67" height="27" />
        </bpmndi:BPMNLabel>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="Event_001smcp_di"
bpmnElement="Event_001smcp">
        <dc:Bounds x="572" y="82" width="36" height="36" />
        <bpmndi:BPMNLabel>
          <dc:Bounds x="552" y="125" width="77" height="40" />
        </bpmndi:BPMNLabel>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="Activity_0dacida_di"
bpmnElement="Activity_10tf96a">
        <dc:Bounds x="540" y="207" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="Activity_0sw09iz_di"
bpmnElement="Activity_05v8qaw">
        <dc:Bounds x="540" y="360" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="Activity_0aq9yac_di"
bpmnElement="Activity_0a5jixl">
        <dc:Bounds x="760" y="270" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="Event_0m5u7hn_di"
bpmnElement="Event_0m5u7hn">
        <dc:Bounds x="1332" y="292" width="36" height="36" />
        <bpmndi:BPMNLabel>
          <dc:Bounds x="1324" y="335" width="52" height="27" />
        </bpmndi:BPMNLabel>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="Activity_0hdqm8t_di"
bpmnElement="Activity_1xrz8z8">
        <dc:Bounds x="1160" y="270" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="Gateway_0qfkxe0_di"
bpmnElement="Gateway_0qfkxe0" isMarkerVisible="true">
        <dc:Bounds x="1065" y="285" width="50" height="50" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="Activity_1qmm15c_di"
bpmnElement="Activity_0ulzs8f">
        <dc:Bounds x="1040" y="400" width="100" height="80" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape id="Event_1rbrg4v_di"
bpmnElement="Event_1rbrg4v">
        <dc:Bounds x="1072" y="562" width="36" height="36" />
        <bpmndi:BPMNLabel>
          <dc:Bounds x="1059" y="605" width="64" height="27" />
        </bpmndi:BPMNLabel>
      </bpmndi:BPMNShape>
```

```
      <bpmndi:BPMNShape id="Activity_1607gxc_di"
bpmnElement="Activity_19dm5k3">
        <dc:Bounds x="910" y="270" width="100" height="80" />
      </bpmndi:BPMNShape>
    </bpmndi:BPMNPlane>
  </bpmndi:BPMNDiagram>
</bpmn:definitions>
```

Application.properties

```
server.servlet.context-path=/
server.port=8081
server.servlet.session.timeout=7200

management.endpoints.web.exposure.include=health,info
management.endpoints.web.base-path=/management

camunda.bpm.webapp.index-redirect-enabled = false
camunda.bpm.admin-user.id = demo
camunda.bpm.admin-user.password = demo
camunda.bpm.history-level = full
camunda.bpm.process-engine-name = default
camunda.bpm.database.schema-update = true
camunda.bpm.id-generator = strong
camunda.bpm.job-execution.active = true
camunda.bpm.job-execution.lockTimeInMillis = 1200000
camunda.bpm.job-execution.maximum-pool-size = 5
camunda.bpm.job-execution.core-pool-size = 5
spring.jersey.application-path=/engine-rest

customer.service.url = http://localhost:8083/

url.orion.context = http://localhost:1026/
url.iot.agent = http://localhost:7896/
```

ChangeTemperatureRequest.java

```
@Getter
@Setter
@ToString
@NoArgsConstructor
public class ChangeTemperatureRequest {
    private String roomNumber;
    private String temperature;
}
```

Data.java

```
@Getter
@Setter
@JsonIgnoreProperties(ignoreUnknown = true)
@ToString
public class Data {
    private String id;
    private String type;
```

```
    private Temperature temperature;
}
```

Temperature.java

```
@Getter
@Setter
@JsonIgnoreProperties(ignoreUnknown = true)
@ToString
public class Temperature {
    private String value;
}
```

CoolRoomController.java

```
@Component
@RequiredArgsConstructor
@Slf4j
public class CoolRoomController implements JavaDelegate {
    private final RuntimeService runtimeService;
    private final RestTemplate restTemplate;
    @Value("${url.iot.agent}")
    private String apiUrl;

    @Override
    public void execute(DelegateExecution execution) {
        String processInstanceId = execution.getProcessInstanceId();
        Map<String, Object> processVariables =
runtimeService.getVariables(processInstanceId);

        String roomNumber =
String.valueOf(processVariables.get("roomNumber"));
        String temperature =
String.valueOf(processVariables.get("temperature"));

        heatRoom(roomNumber);
        updateTemperature(roomNumber, temperature);
    }

    public void updateTemperature(String roomNumber, String temperature)
{
        String temperatureToCool = "t|" + temperature;
        HttpHeaders headers = new HttpHeaders();
        headers.add(HttpHeaders.CONTENT_TYPE, "text/plain");
        String iotDeviceUrl
                = apiUrl +
"iot/d?k=4jggokgpepnvsb2uv4s40d59ov&i=temperatureSensor" + roomNumber;
        log.info("temperatureToHeat={}, url={}, headers={}",
temperatureToCool, iotDeviceUrl, headers);
        restTemplate.exchange(iotDeviceUrl, HttpMethod.POST, new
HttpEntity<>(temperatureToCool, headers), Void.class);
    }

    public void heatRoom(String roomNumber) {
```

```
        String temperatureRegulatorStatus = "s|cool";
        String iotDeviceUrl
                = apiUrl +
"iot/d?k=4jggokgpepnvsb2uv4s40d59ov&i=temperatureRegulator" + roomNumber;
        restTemplate.exchange(iotDeviceUrl, HttpMethod.POST, new
HttpEntity<>(temperatureRegulatorStatus), Void.class);
    }
}
```

OrionContextController.java

```
@Component
@RequiredArgsConstructor
public class OrionContextController implements JavaDelegate {
    private final RuntimeService runtimeService;
    private final RestTemplate restTemplate;
    @Value("${url.orion.context}")
    private String apiUrl;

    @Override
    public void execute(DelegateExecution execution) {
        String processInstanceId = execution.getProcessInstanceId();
        Map<String, Object> processVariables =
runtimeService.getVariables(processInstanceId);
        String roomNumber =
String.valueOf(processVariables.get("roomNumber"));
        Temperature currentTemperature =
getCurrentTemperature(roomNumber);
        execution.setVariables(Map.of("currentTemperature",
currentTemperature.getValue()));
    }

    public Temperature getCurrentTemperature(String roomNumber) {
        String orionContextUrl
                = apiUrl + "/v2/entities/urn:ngsi-ld:Room:unit" +
roomNumber + "?type=Room";
        ResponseEntity<Data> response
                = restTemplate.getForEntity(orionContextUrl, Data.class);

        return response.getBody().getTemperature();
    }
}
```

SendMessageController.java

```
@Component
@RequiredArgsConstructor
@Slf4j
public class SendMessageController implements JavaDelegate {

    private final RuntimeService runtimeService;

    @Override
    public void execute(DelegateExecution delegateExecution) throws
```

```
Exception {
        String processInstanceId =
delegateExecution.getProcessInstanceId();
        Map<String, Object> processVariables =
runtimeService.getVariables(processInstanceId);
        String roomNumber =
String.valueOf(processVariables.get("roomNumber"));
        String temperature =
String.valueOf(processVariables.get("temperature"));
        delegateExecution.setVariables(Map.of("errorMessage", "Unable to
change temperature. System problem."));
        log.info("Unable to change temperature in roomNumber={} to={}",
roomNumber, temperature);
    }
}
```

HeatRoomController.java

```
@Component
@RequiredArgsConstructor
@Slf4j
public class HeatRoomController implements JavaDelegate {
    private final RuntimeService runtimeService;
    private final RestTemplate restTemplate;
    @Value("${url.iot.agent}")
    private String apiUrl;

    @Override
    public void execute(DelegateExecution execution) {
        String processInstanceId = execution.getProcessInstanceId();
        Map<String, Object> processVariables =
runtimeService.getVariables(processInstanceId);

        String roomNumber =
String.valueOf(processVariables.get("roomNumber"));
        String temperature =
String.valueOf(processVariables.get("temperature"));

        heatRoom(roomNumber);
        updateTemperature(roomNumber, temperature);
        log.info("roomNumber={}, temperature={}", roomNumber,
temperature);
    }

    public void updateTemperature(String roomNumber, String temperature)
{
        String temperatureToHeat = "t|" + temperature;
        HttpHeaders headers = new HttpHeaders();
        headers.add(HttpHeaders.CONTENT_TYPE, "text/plain");
        String iotDeviceUrl
                = apiUrl +
"iot/d?k=4jggokgpepnvsb2uv4s40d59ov&i=temperatureSensor" + roomNumber;
        log.info("temperatureToHeat={}, url={}, headers={}",
```

```
temperatureToHeat, iotDeviceUrl, headers);
        restTemplate.exchange(iotDeviceUrl, HttpMethod.POST, new
HttpEntity<>(temperatureToHeat, headers), Void.class);
    }

    public void heatRoom(String roomNumber) {
        String temperatureRegulatorStatus = "s|heat";
        String iotDeviceUrl
                = apiUrl +
"iot/d?k=4jggokgpepnvsb2uv4s40d59ov&i=temperatureRegulator" + roomNumber;
        restTemplate.exchange(iotDeviceUrl, HttpMethod.POST, new
HttpEntity<>(temperatureRegulatorStatus), Void.class);
    }
}
```

StopSystemController.java

```
@Component
@RequiredArgsConstructor
@Slf4j
public class StopSystemController implements JavaDelegate {

    private final RuntimeService runtimeService;
    private final RestTemplate restTemplate;
    @Value("${url.iot.agent}")
    private String apiUrl;

    @Override
    public void execute(DelegateExecution delegateExecution) throws
Exception {

        String processInstanceId =
delegateExecution.getProcessInstanceId();
        Map<String, Object> processVariables =
runtimeService.getVariables(processInstanceId);

        String roomNumber =
String.valueOf(processVariables.get("roomNumber"));

        String temperatureRegulatorStatus = "s|sleep";
        String iotDeviceUrl
                = apiUrl +
"iot/d?k=4jggokgpepnvsb2uv4s40d59ov&i=temperatureRegulator" + roomNumber;
        restTemplate.exchange(iotDeviceUrl, HttpMethod.POST, new
HttpEntity<>(temperatureRegulatorStatus), Void.class);
    }
}
```

TemperatureUpdateWaitController.java

```
@Component
@RequiredArgsConstructor
public class TemperatureUpdateWaitController implements JavaDelegate {
    private final RuntimeService runtimeService;
```

```
    private final RestTemplate restTemplate;
    @Value("${url.orion.context}")
    private String apiUrl;

    @Override
    public void execute(DelegateExecution execution) {
        String processInstanceId = execution.getProcessInstanceId();
        Map<String, Object> processVariables =
runtimeService.getVariables(processInstanceId);
        String roomNumber =
String.valueOf(processVariables.get("roomNumber"));
        Temperature currentTemperature =
getCurrentTemperature(roomNumber);
        execution.setVariables(Map.of("currentUpdatedTemperature",
currentTemperature.getValue()));
    }

    public Temperature getCurrentTemperature(String roomNumber) {
        String orionContextUrl
                = apiUrl + "/v2/entities/urn:ngsi-ld:Room:unit" +
roomNumber + "?type=Room";
        ResponseEntity<Data> response
                = restTemplate.getForEntity(orionContextUrl, Data.class);

        return response.getBody().getTemperature();
    }
}
```

SubscriptionController.java

```
@RestController
@Slf4j
public class RoomSubscriptionController {

    @PostMapping("/temperature/update")
    public String handleTemperatureUpdate(@RequestBody
TemperatureUpdateRequest request) {
        log.info("Request={}", request);
        updateContextTemperature("001",
request.getData().get(0).getTemperature().getValue());
        return "OK";
    }

    @PutMapping("postTemperature/{roomNumber}/{temperature}")
    public Data updateContextTemperature(@PathVariable String roomNumber,
@PathVariable String temperature) {

        RestTemplate restTemplate = new RestTemplate();
        String fooResourceUrl
                = "http://localhost:1026/v2/entities/urn:ngsi-
ld:Room:unit" + roomNumber + "/attrs/temperature/value?type=Room";
        restTemplate.exchange(fooResourceUrl, HttpMethod.PUT, new
HttpEntity<>(temperature), Void.class);
```

```
        return getCurrentTemperature(roomNumber);
    }

    @GetMapping("/getTemperature/{roomNumber}")
    public Data getCurrentTemperature(@PathVariable String roomNumber) {
        RestTemplate restTemplate = new RestTemplate();
        String fooResourceUrl
                = "http://localhost:1026/v2/entities/urn:ngsi-
ld:Room:unit" + roomNumber + "?type=Room";
        ResponseEntity<Data> response
                = restTemplate.getForEntity(fooResourceUrl, Data.class);

        return response.getBody();
    }
}
```

TemperatureUpdateRequest.java

```
@Getter
@Setter
@JsonIgnoreProperties(ignoreUnknown = true)
@ToString
public class TemperatureUpdateRequest {
    private String subscriptionId;
    private List<Data> data;
}
```

```
# WARNING: Do not deploy this tutorial configuration directly to a production environment
#
# The tutorial docker-compose files have not been written for production deployment and will not
# scale. A proper architecture has been sacrificed to keep the narrative focused on the learning
# goals, they are just used to deploy everything onto a single Docker machine. All FIWARE components
# are running at full debug and extra ports have been exposed to allow for direct calls to services.
# They also contain various obvious security flaws - passwords in plain text, no load balancing,
# no use of HTTPS and so on.
#
# This is all to avoid the need of multiple machines, generating certificates, encrypting secrets
# and so on, purely so that a single docker-compose file can be read as an example to build on,
# not use directly.
#
# When deploying to a production environment, please refer to the Helm Repository
# for FIWARE Components in order to scale up to a proper architecture:
#
# see: https://github.com/FIWARE/helm-charts/
#
version: "3.5"
services:
  # Orion is the context broker
  orion:
    image: fiware/orion:${ORION_VERSION}
    hostname: orion
    container_name: fiware-orion
    depends_on:
      - mongo-db
    networks:
      - default
    expose:
      - "${ORION_PORT}"
      # - 9001
```

```
    ports:
      - "${ORION_PORT}:${ORION_PORT}" # localhost:1026
      # - 9001:9001
    command: -dbhost mongo-db -logLevel DEBUG
    healthcheck:
      test: curl --fail -s http://orion:${ORION_PORT}/version || exit 1
      interval: 5s
    # extra_hosts:
      # - "localhost:192.168.1.132"

  # IoT-Agent is configured for the UltraLight Protocol
  iot-agent:
    image: fiware/iotagent-ul:${ULTRALIGHT_VERSION}
    hostname: iot-agent
    container_name: fiware-iot-agent
    depends_on:
      - mongo-db
    networks:
      - default
    expose:
      - "${IOTA_NORTH_PORT}"
      - "${IOTA_SOUTH_PORT}"
      # - 9001
    ports:
      - "${IOTA_NORTH_PORT}:${IOTA_NORTH_PORT}" # localhost:4041
      - "${IOTA_SOUTH_PORT}:${IOTA_SOUTH_PORT}" # localhost:7896
      # - 9001:9001
    environment:
      - IOTA_CB_HOST=orion # name of the context broker to update context
      - IOTA_CB_PORT=${ORION_PORT} # port the context broker listens on to update context
      - IOTA_NORTH_PORT=${IOTA_NORTH_PORT}
      - IOTA_REGISTRY_TYPE=mongodb #Whether to hold IoT device info in memory or in a database
      - IOTA_LOG_LEVEL=DEBUG # The log level of the IoT Agent
      - IOTA_TIMESTAMP=true # Supply timestamp information with each measurement
      - IOTA_CB_NGSI_VERSION=v2 # use NGSIv2 when sending updates for active attributes
      - IOTA_AUTOCAST=true # Ensure Ultralight number values are read as numbers not strings
      - IOTA_MONGO_HOST=mongo-db # The host name of MongoDB
      - IOTA_MONGO_PORT=${MONGO_DB_PORT} # The port mongoDB is listening on
      - IOTA_MONGO_DB=iotagentul # The name of the database used in mongoDB
      - IOTA_HTTP_PORT=${IOTA_SOUTH_PORT} # The port used for device traffic over HTTP
      - IOTA_PROVIDER_URL=http://iot-agent:${IOTA_NORTH_PORT}
      - IOTA_DEFAULT_RESOURCE=/iot/d
    healthcheck:
      interval: 5s

  # Database
  mongo-db:
    image: mongo:${MONGO_DB_VERSION}
    hostname: mongo-db
    container_name: db-mongo
    expose:
      - "${MONGO_DB_PORT}"
    ports:
      - "${MONGO_DB_PORT}:${MONGO_DB_PORT}" # localhost:27017
    networks:
      - default
    volumes:
      - mongo-db:/data
    healthcheck:
      test: |
        host=`hostname --ip-address || echo '127.0.0.1'`;
        mongo --quiet $host/test --eval 'quit(db.runCommand({ ping: 1 }).ok ? 0 : 2)' && echo 0 || echo 1
      interval: 5s
```

```
    healthcheck:
      test: |
        host=`hostname --ip-address || echo '127.0.0.1'`;
        mongo --quiet $host/test --eval 'quit(db.runCommand({ ping: 1 }).ok ? 0 : 2)' && echo 0 || echo 1
      interval: 5s
```

```
networks:
  dockernet:
    external: true
  default:
    ipam:
      config:
        - subnet: 172.18.1.0/24
```

```
volumes:
  mongo-db: ~
```

Main curl requests

#create multi room entity

curl -iX POST \

'http://localhost:1026/v2/op/update' \

-H 'Content-Type: application/json' \

-d '{

"actionType":"append_strict",

"entities":[

{

"id":"urn:ngsi-ld:Room:unit001", "type":"Room",

"temperature":{

"type":"Integer", "value":20

}

},

{

"id":"urn:ngsi-ld:Room:unit002", "type":"Room",

"temperature":{

"type":"Integer", "value":18

}

},

{

```
    "id":"urn:ngsi-ld:Room:unit003", "type":"Room",
    "temperature":{
      "type":"Integer", "value":20
    }
   },
   {
    "id":"urn:ngsi-ld:Room:unit004", "type":"Room",
    "temperature":{
      "type":"Integer", "value":22
    }
   }
 ]
}'

#create subscription
curl -iX POST \
  --url 'http://localhost:1026/v2/subscriptions' \
  --header 'content-type: application/json' \
  --data '{
  "description": "Notify me of temperature changes",
  "subject": {
    "entities": [{"idPattern": ".*", "type": "Room"}],
    "condition": {
      "attrs": [ "temperature" ],
      "expression": {"q" : "refRoom==urn:ngsi-ld:Room:unit001"}
    }
  },
  "notification": {
    "http": {
      "url": "http://192.168.1.132:9001/temperature/update"
```

```
    }
  }
}'

#update temperature value
curl -iX PUT \
  --url 'http://localhost:1026/v2/entities/urn:ngsi-
ld:Room:unit001/attrs/temperature/value' \
  --header 'Content-Type: text/plain' \
  --data 30

#create service group
curl -iX POST \
  'http://localhost:4041/iot/services' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
 "services": [
   {
     "apikey":      "4jggokgpepnvsb2uv4s40d59ov",
     "cbroker":     "http://orion:1026",
     "entity_type": "Thing",
     "resource":    "/iot/d"
   }
 ]
}'

#create sensor
curl -iX POST \
```

```
 'http://localhost:4041/iot/devices' \
 -H 'Content-Type: application/json' \
 -H 'fiware-service: openiot' \
 -H 'fiware-servicepath: /' \
 -d '{
"devices": [
  {
    "device_id":   "temperatureSensor001",
    "entity_name": "urn:ngsi-ld:TemperatureSensor:001",
    "entity_type": "TemperatureSensor",
    "timezone":    "Europe/Berlin",
    "attributes": [
      { "object_id": "t", "name": "temperature", "type": "Integer" }
    ],
    "static_attributes": [
      { "name":"refRoom", "type": "Relationship", "value": "urn:ngsi-
ld:Room:unit001"}
    ]
  }
 ]
}
'

#set value to iot sensor
curl -iX POST \

'http://localhost:7896/iot/d?k=4jggokgpepnvsb2uv4s40d59ov&i=temperatureSensor0
01' \
 -H 'Content-Type: text/plain' \
 -d 't|42'
```

```
# get
curl -G -X GET \
  'http://localhost:1026/v2/entities/urn:ngsi-ld:TemperatureSensor:001' \
  -d 'type=TemperatureSensor' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /'

#create subscription for sensor
  curl -iX POST \
  --url 'http://localhost:1026/v2/subscriptions' \
  --header 'content-type: application/json' \
  -H 'fiware-service: openiot'   -H 'fiware-servicepath: /' \
  --data '{
  "description": "Notify of temperature changes",
  "subject": {
    "entities": [{"idPattern": ".*", "type": "TemperatureSensor"}],
    "condition": {
      "attrs": [ "temperature" ],
      "expression": {"q" : "refStore==urn:ngsi-ld:Room:unit001"}
    }
  },
  "notification": {
    "http": {
      "url": "http://192.168.1.132:9001/temperature/update"
    }
  }
}'

curl -iX POST \
```

```
 'http://localhost:4041/iot/devices' \
 -H 'Content-Type: application/json' \
 -H 'fiware-service: openiot' \
 -H 'fiware-servicepath: /' \
 -d '{
 "devices": [
  {
   "device_id": "temperatureRegulator001",
   "entity_name": "urn:ngsi-ld:TemperatureRegulator:001",
   "entity_type": "TemperatureRegulator",
   "protocol": "PDI-IoTA-UltraLight",
   "transport": "HTTP",
   "endpoint": "http://iot-sensors:3001/iot/temperatureRegulator001",
   "commands": [
    {"name": "heat","type": "command"},
    {"name": "cool","type": "command"},
    {"name": "sleep","type": "command"}
   ],
   "attributes": [
    {"object_id": "s", "name": "state", "type":"Text"}
   ],
   "static_attributes": [
    {"name":"refStore", "type": "Relationship","value": "urn:ngsi-ld:Room:001"}
   ]
  }
 ]
}'

curl -iX PATCH \
 'http://localhost:1026/v2/entities/urn:ngsi-ld:TemperatureRegulator:001/attrs' \
```

```
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
  "heat": {
     "type" : "command",
     "value" : ""
  }
}'

curl -G -X GET \
  'http://localhost:1026/v2/entities/urn:ngsi-ld:TemperatureRegulator:001' \
  -d 'type=TemperatureRegulator' \
  -d 'options=keyValues' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /'

curl -iX POST \

'http://localhost:7896/iot/d?k=4jggokgpepnvsb2uv4s40d59ov&i=temperatureRegulator001' \
  -H 'Content-Type: text/plain' \
  -d 's|start'
```

Docker.compose.yml