UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Integración de datos de procesos de negocio y organizacionales de bases de datos NoSQL

Informe de Trabajo Final de Licenciatura presentado por

Álvaro Vallvé

en cumplimiento parcial de los requerimientos para la graduación de la carrera de Licenciatura en Computación de Facultad de Ingeniería de la Universidad de la República

Supervisores

Dra. Ing. Andrea Delgado

Montevideo, 16 de junio de 2023

Resumen

Las empresas y organizaciones utilizan una gran variedad de procesos de negocio [1] para llevar a cabo sus operaciones diarias. Para gestionar estos procesos de manera efectiva, muchas organizaciones utilizan plataformas BPM (Business Process Management System, BPMS). Estas plataformas proporcionan soporte a lo largo de todo el ciclo de vida del proceso, incluyendo el análisis y diseño, la configuración, la ejecución y la evaluación.
A medida que las organizaciones y sus procesos de negocio se vuelven cada vez más complejos, es necesario integrar diferentes visiones, técnicas y herramientas para la gestión de la información, procesos y sistemas asociados. En este contexto, la visión compartimentada de procesos por un lado y datos por el otro no es adecuada para proporcionar a la organización la inteligencia organizacional basada en evidencia necesaria para mejorar su operativa diaria.

Adicionalmente las empresas y organizaciones generan grandes cantidades de datos a gran velocidad y con una gran variedad de formatos, lo que se conoce como las tres V de big data: volumen, velocidad y variedad. La gestión, análisis y descubrimiento de información a partir de estos datos es un desafío clave para las organizaciones. Es importante tener en cuenta también la veracidad de los datos, ya que estos pueden contener errores o ser incompletos.
Para enfrentar este desafío, es fundamental contar con un soporte integrado para el análisis de procesos y datos en las organizaciones. Esto incluye metodologías, técnicas y herramientas para la gestión y análisis de los datos, así como elementos para lidiar con las tres V de big data. La integración de los datos puede provenir de diversas fuentes, incluyendo bases de datos relacionales y no relacionales (NoSQL) [2], sistemas de información tradicionales, redes sociales, servicios internos y externos, entre otros.
Es importante destacar que la gestión y análisis de datos no debe ser un proceso aislado, sino que debe estar integrado con la gestión de procesos de negocio en la organización. Por lo tanto, es necesario contar con plataformas de gestión de procesos y datos que trabajen de manera integrada, proporcionando una vista holística de la organización y permitiendo tomar decisiones informadas basadas en datos concretos y en tiempo real.

En este proyecto, se propone estudiar la integración de datos de procesos y organizacionales de bases de datos NoSQL. El objetivo principal es analizar diferentes escenarios que combinen opciones tecnológicas posibles. El escenario general de integración incluye la ejecución de procesos en plataformas BPMS con bases de datos relacionales y NoSQL, y bases de datos organizacionales centrales de tipo relacional y NoSQL.

Contenido

1 Introducción

La complejidad de los procesos empresariales ha aumentado gracias a las nuevas tecnologías y a la capacidad de tener más interacciones entre los sistemas. Por lo tanto, las organizaciones han tenido que encontrar maneras de organizar sus procesos, lo que ha llevado a la adopción del Business Process Management (BPM). Con BPM, las organizaciones pueden usar diferentes reglas para modelar, implementar, ejecutar, evaluar y analizar los procesos empresariales. Para implementar y ejecutar estos procesos, se utilizan los sistemas de gestión de procesos empresariales (BPMS), que son sistemas de software genéricos basados en modelos de procesos explícitos.

Las empresas y organizaciones se enfrentan también al desafío de manejar grandes cantidades de datos, generados a gran velocidad y en diversos formatos. La gestión, análisis y descubrimiento de información a partir de estos datos es crucial para el éxito de la organización. Sin embargo, también es importante considerar la veracidad de los datos, ya que pueden contener errores o estar incompletos.
Para abordar este desafío, es esencial contar con un soporte integrado para el análisis de procesos y datos en las organizaciones. La integración de datos puede provenir de diversas fuentes, como bases de datos relacionales y no relacionales (NoSQL), sistemas de información tradicionales, redes sociales, servicios internos y externos, entre otros. La capacidad de integrar datos de múltiples fuentes en un solo sistema de información es crucial para obtener una visión completa y precisa de la organización y su entorno. De esta manera, las organizaciones pueden tomar decisiones informadas y basadas en datos para mejorar su rendimiento empresarial.

Conforme las organizaciones y sus procesos empresariales se vuelven más complejos, es fundamental integrar diversas perspectivas, técnicas y herramientas para una gestión eficaz de la información, procesos y sistemas asociados. En este sentido, la tradicional visión segmentada de procesos de un lado y datos del otro ya no es suficiente para proporcionar la inteligencia organizacional basada en evidencia necesaria para mejorar la operativa diaria de la organización.
En su lugar, se requiere una visión integral e integrada que permita analizar y comprender de manera holística el rendimiento y la eficacia de la organización en su conjunto. Esto implica la utilización de herramientas de análisis de datos, de modelado de procesos y de gestión de sistemas de información, que permitan una gestión más eficiente y efectiva de los procesos empresariales.

El objetivo general del proyecto es estudiar la integración de datos de procesos de negocio y organizacionales de bases de datos NoSQL, considerando distintos escenarios tecnológicos.

Los objetivos de este proyecto son:

1. Estudiar conceptos de BPM y bases de datos NoSQL, y herramientas asociadas (plataformas BPMS, bases de datos NoSQL) y seleccionar herramientas para el proyecto.
2. Analizar distintos escenarios tecnológicos de ejecución de procesos en plataformas BPMS con foco en bases de datos NoSQL, y bases de datos organizacionales NoSQL.
3. Aplicar la propuesta en un caso de estudio con base en procesos y datos reales.

Si bien la propuesta original incluía también prototipar la extracción de datos asociada hacia un metamodelo de integración de datos definido previamente, esto fue eliminado de los objetivos de implementación del prototipo, ya que fue integrado en un proyecto de grado que trabajó otras partes de la propuesta general de integración de datos. Este proyecto se enmarca en el proyecto de investigación "Minería de procesos y datos para la mejora de procesos en las organizaciones" financiado por Comisión Sectorial de Investigación Científica, UdelaR en colaboración con AGESIC (Agencia de Gobierno Electrónico y Sociedad de la Información y del Conocimiento.

El presente documento está organizado de la siguiente manera. En el Capítulo 2 se desarrolla un marco teórico con las definiciones de Gestión de Procesos de Negocio, y Bases de Datos NoSQL. En el Capítulo 3 se presenta el problema a resolver, presentando el estado actual y análisis de escenarios. En los Capítulos 4 y 5 se explica cómo se desarrolló la solución del problema que se expuso en el Capítulo 3, mientras que en el Capítulo 6 se aplica dicha solución para un caso particular llamado Student Mobility y se muestra cómo se puede explotar la solución. Finalmente, en el Capítulo 7 se presentan las conclusiones del proyecto y se mencionan trabajos futuros que se pueden desarrollar a partir de la solución alcanzada.

2 Marco Teórico

En esta sección se presenta el marco teórico de la tesis. En la subsección 2.1 se presenta la Gestión de Procesos de Negocio (BPM por sus siglas en inglés) y en la subsección 2.2 se presenta el marco teórico de bases de datos NoSQL.

2.1 Gestión de Procesos de Negocio

Los procesos de negocio se pueden definir como:
"Un proceso de negocio consiste en un conjunto de actividades que son realizadas en coordinación en un ambiente organizacional y técnico. Estas actividades juntas realizan un objetivo de negocio. Cada proceso de negocio es representado por una sola organización, pero puede interactuar con procesos de negocio ejecutados por otras organizaciones." [1]

Toda actividad en una organización es parte de un proceso de negocio y su ejecución impactará en la calidad final del producto o servicio que dicha organización brinda. La gestión de procesos de negocio es la disciplina, que incluye metodologías más tecnologías, orientada a mejorar el desempeño de las organizaciones mediante la optimización y la automatización de los procesos de negocios.
"El manejo de procesos de negocio incluye concepto, métodos y técnicas para soportar el diseño, administración, configuración, ejecución y análisis de los procesos." [1]

Existen diversos lenguajes de modelado y especificación de procesos de negocio, por ejemplo, Business Process Model and Notation (BPMN 2.0) [3], que permiten visualizar el flujo de control y los elementos involucrados (tareas, eventos, compuertas, etc.) en la realización del proceso. El estándar BPMN brinda a las organizaciones la capacidad de entender sus procedimientos internos en una notación gráfica y de comunicar estos procedimientos de manera estándar. Permite que las personas técnicas y comerciales se comuniquen sobre los procesos de negocios de una manera en que ambas partes entiendan. En la Figura 2.1 se puede ver un proceso de negocio de ejemplo simple especificado en BPMN 2.0.

BPMN 2.0 soporta tres tipos principales de procesos: orquestación, coreografía y colaboración.
En la orquestación de procesos, se modela el flujo de trabajo interno del participante, incluyendo las tareas que realiza y las decisiones que toma. En este tipo de proceso, hay un participante principal que coordina la ejecución del proceso.
La coreografía de procesos se centra en el flujo de mensajes entre los participantes más que en las tareas detalladas de un proceso. Se modela el flujo de trabajo entre los diferentes participantes, incluyendo las tareas que realizan y las interacciones que tienen entre ellos.
Una colaboración de procesos es el termino más amplio, describe cómo dos o más participantes trabajan juntos para llevar a cabo un proceso de negocio. A diferencia de la coreografía de procesos, en la colaboración de procesos, se modela el flujo de trabajo de cada participante individualmente, incluyendo las tareas que realiza y las decisiones que toma.
En resumen, la orquestación de procesos se enfoca en cómo un solo participante ejecuta el proceso, la coreografía de procesos se enfoca en cómo múltiples participantes interactúan entre sí para llevar a cabo el proceso, y la colaboración de procesos se enfoca en cómo dos o más participantes trabajan juntos para llevar a cabo el proceso.

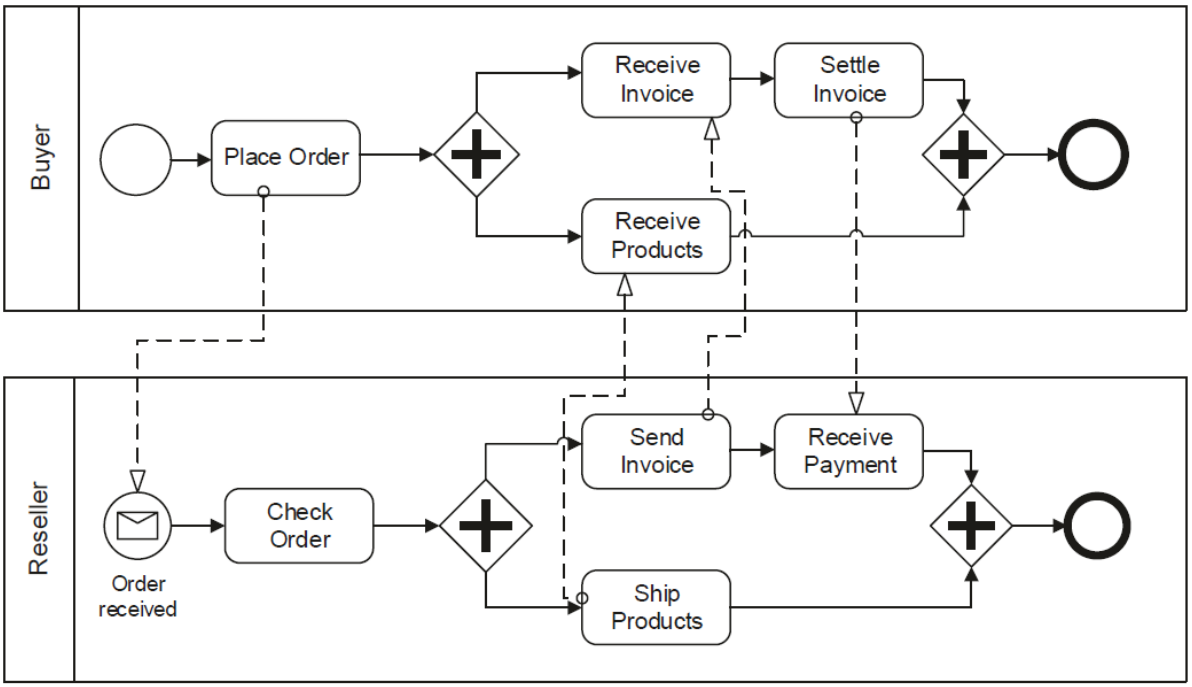


Figura 2.1: Proceso de negocio de ejemplo especificado en lenguaje BPMN 2.0 extraído de [1]

El proceso de pedido presentado en la Figura 2.1 presenta la interacción entre 2 participantes, el comprador y la empresa distribuidora. En el Pool Buyer se ve la orquestación del proceso que involucran las actividades del comprador, mientras que en el Pool Reseller vemos las de la empresa. La interacción entre los participantes se representa con la flecha punteada.

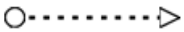
La orquestación del proceso del comprador inicia cuando el realiza un pedido. Una vez iniciado el pedido se ejecutan 2 tareas en paralelo, esperando recibir la factura y los productos. Una vez recibida la factura avanza a la siguiente actividad para abonar la misma. Luego de recibido los productos y abonado la factura finaliza el proceso.

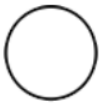
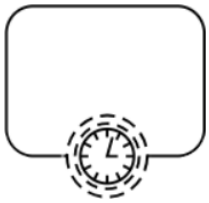
En la orquestación del proceso de la empresa distribuidora el primer elemento es el encargado de recibir la solicitud e iniciar el proceso. Luego de ser verificada el pedido se ejecutan dos actividades en paralelo las cuales se encargan por un lado manejar el pago del pedido y por el otro lado la entrega de los productos. Finalmente, cuando todas estas actividades terminan, el pedido termina y el proceso finaliza.

En la Tabla 2.1 se presenta de forma general los principales elementos definidos por el estándar BPMN. Se pueden dividir en cuatro tipos principales.

- Conectores: flujos de secuencia, flujos de mensajes, asociaciones
- Objetos de flujo: eventos, actividades, compuertas
- Carriles: piscina o carril
- Artefactos: objeto de datos, grupo, anotación

Tabla 2.1: Principales elementos definidos por el estándar BPMN

Conectores		
Sequence Flows		Define la secuencia de ejecución de dos elementos de proceso o tarea.
Message Flows		Indica el intercambio de información entre dos participantes en un proceso.
Association		Conecta un artefacto (como una etiqueta o un comentario) con un elemento de proceso o tarea.
Actividades		
Task		Representa una tarea específica que se realiza dentro de un proceso. Puede ser una tarea que tiene que realizar el usuario, un servicio, una regla de negocio.
Sub-Process		Representa un proceso completo que se puede incluir dentro de otro proceso.
Call Activity		Conceptualmente es igual al subproceso. La diferencia es que la actividad de llamada hace referencia a un proceso que es externo a la definición del proceso, mientras que el subproceso está incrustado en la definición del proceso original.
Compuertas		
Exclusive Gateway		Permite la selección de una única salida de varias entradas en un proceso.
Inclusive Gateway		Permite la selección de varias salidas de varias entradas en un proceso.
Parallel Gateway		Las compuertas paralelas permiten modelar situaciones en las que se ejecuta más de una ruta en paralelo. Luego para unir ramas paralelas se utiliza otra compuerta paralela.
Event Based Gateway		Se utiliza para modelar procesos que dependen de uno o más eventos.

Eventos		
Start Events		Representa el inicio de un proceso o subproceso. El evento de inicio puede ser desde un mensaje, un timer, entre otros. En la imagen se presenta el tipo evento de inicio más común.
End Events		Representa el final de un proceso o subproceso.
Intermediate Events		Representa un evento que ocurre dentro de un proceso o subproceso. En la imagen el evento de timer espera el tiempo configurado antes de continuar.
Boundary Events		Son eventos que se adjuntan a tareas, subprocesos o actividades de llamada. Si ocurre el evento, la ejecución del proceso sigue la secuencia de salida de ese evento límite. Pueden interrumpir o no interrumpir la actividad. En la imagen de ejemplo se muestra un timer que no interrumpe la actividad, es decir, no la marca como completada.
Carriles		
Pool	Un Pool es la representación gráfica de un participante en una Colaboración	
Lane	Los carriles son una buena manera de organizar las responsabilidades en un proceso.	
Artefactos		
Data Object	Los elementos del objeto de datos se muestran visualmente en un diagrama de proceso. Las referencias a objetos de datos son una forma de reutilizar objetos de datos en el mismo diagrama. Pueden especificar diferentes estados del mismo objeto de datos en diferentes puntos de un proceso.	
Annotation	Las anotaciones son un mecanismo para que un modelador proporcione información adicional para el lector de un diagrama BPMN.	
Group	Los grupos a menudo se usan para resaltar ciertas subcláusulas de un diagrama sin agregar restricciones adicionales para el rendimiento, como lo haría un subproceso. Los grupos no afectan el flujo del Proceso.	

2.1.1 Ciclo de vida de un proceso de negocio

El objetivo de esta sección es proporcionar una comprensión general de los conceptos que son relevantes en la gestión de procesos de negocio. Para la explicación se presenta el ciclo de vida de procesos de negocio definido en [1] en donde se distinguen cuatro grandes áreas: Diseño y Análisis, Configuración, Ejecución y Evaluación, que se puede ver en la Figura 2.2.

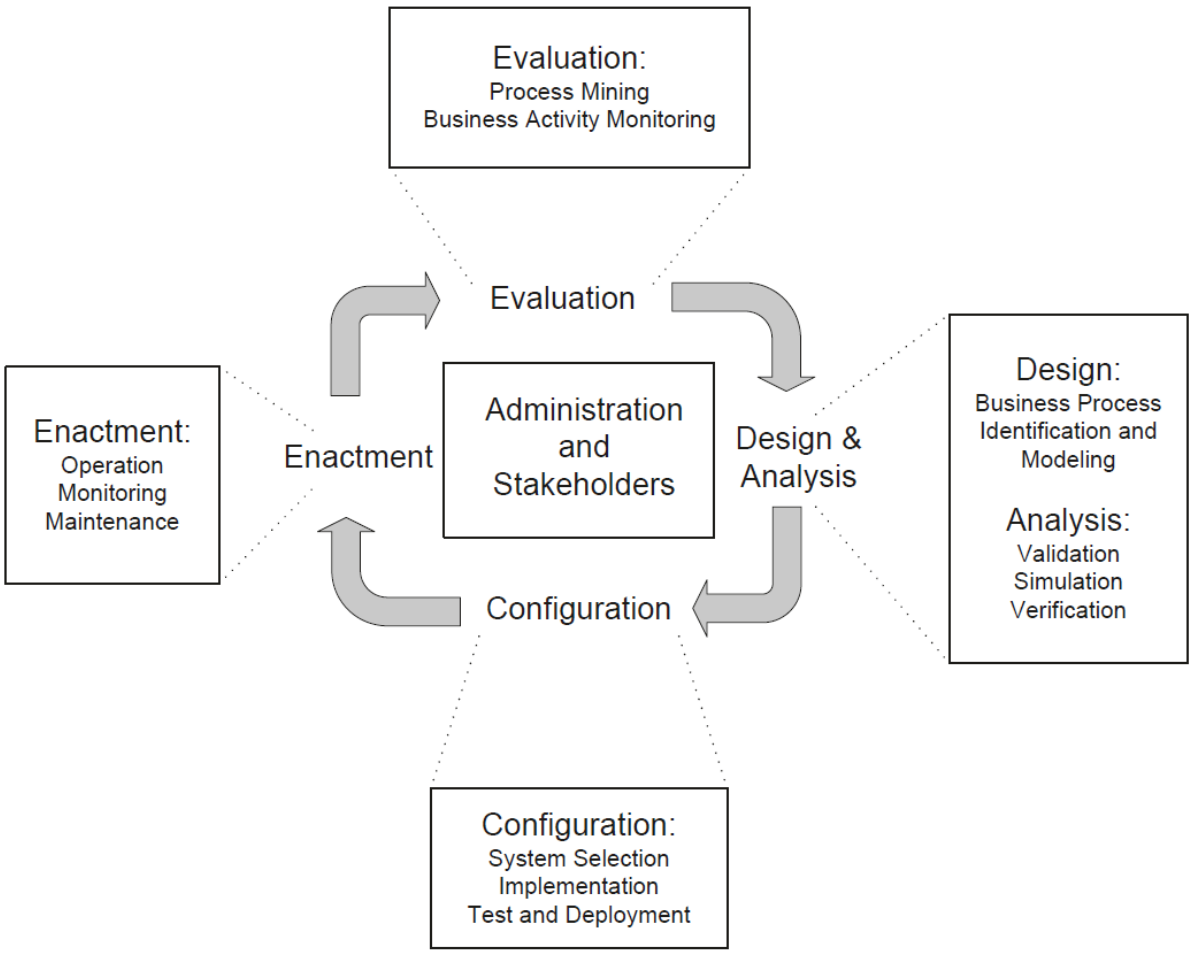


Figura 2.2: Ciclo de vida de los procesos de negocio extraído de [1]

Las fases se organizan en una estructura cíclica mostrando sus dependencias lógicas. Muchas actividades de diseño y desarrollo se llevan a cabo durante cada una de estas fases, y no son infrecuentes los enfoques incrementales y evolutivos que involucran actividades concurrentes en múltiples fases.

- Diseño y Análisis: en esta fase se intenta en primer lugar identificar procesos de negocios. Luego de revisión y validación los procesos son expresados en una notación gráfica estándar para facilitar la comunicación, con los diferentes interesados, y así refinarlos y mejorarlos. Técnicas de modelado de procesos de negocio como de validación, simulación y verificación son usadas en esta fase.
- Configuración: una vez que el proceso de negocio está diseñado y verificado, es necesario implementarlo. Hay diferentes maneras de hacerlo. Puede implementarse mediante un conjunto de políticas y procedimientos que los empleados de la empresa deben cumplir. En este caso un proceso puede realizarse sin ningún tipo de software. Otra opción es utilizar un sistema de gestión de procesos de negocio explicado más adelante en el documento.
- Ejecución: en esta fase las instancias de procesos son ejecutadas. Incluye el seguimiento y control a las múltiples instancias de los procesos que son necesarias para el desarrollo de las actividades y propósitos organizacionales.
- Evaluación: se busca identificar fortalezas y debilidades del proceso con el propósito de desarrollar mejoras que puedan ser implementadas.

2.1.2 Sistemas de gestión de procesos de negocio

Es posible contar con un conjunto de herramientas que den el soporte necesario para cumplir con el ciclo de vida de los procesos de negocio. Este conjunto de herramientas es llamado Business Process Management Software (BPMS).

Se puede definir como [1]:
"Un BPMS es un sistema de software genérico que está impulsado por representaciones de procesos explícitos para coordinar la implementación de los procesos de negocio."

BPMN 2.0 se utiliza en conjunción con sistemas de gestión de procesos de negocio (BPMS). Estos sistemas pueden traducir los diagramas BPMN en código ejecutable.
Existen varias herramientas BPMS que permiten modelar y ejecutar procesos en BPMN 2.0, entre las más conocidas y usadas se encuentran Flowable [4], Activiti [5], Camunda [6], Bonita [7], Bizagi [8].
Para mostrar de forma general lo que usan/definen los motores de procesos que implementan BPMN 2.0 se elige presentar Flowable que es la plataforma seleccionada según diversas características que se explican en la siguiente sección.

2.1.2.1 Flowable BPM

Flowable es una herramienta completa para modelar, implementar y ejecutar cualquier tipo de procesos de negocio que la organización necesite. Se define cómo:
"Flowable es un motor de procesos de negocios liviano escrito en Java. El motor de procesos de Flowable permite implementar definiciones de procesos BPMN 2.0, crear instancias de procesos de esas definiciones de procesos, ejecutar consultas, acceder a instancias de procesos activos o históricos y datos relacionados, y mucho más." [4]

Flowable es extremadamente flexible, se puede integrar el motor en la aplicación (o servicio) con solo incluir la biblioteca Flowable disponible como JAR. Alternativamente se puede usar la API REST de Flowable para comunicarse a través de HTTP. También hay varias aplicaciones de Flowable (Flowable Modeler, Flowable Admin, Flowable IDM y Flowable Task) que ofrecen interfaces de usuario listas para usar y trabajar con procesos y tareas.

Se presenta en Listing 2.1 un ejemplo extraído del sitio de Flowable de un proceso de negocio para solicitar días de licencia. En el mismo se pueden ver los diferentes componentes utilizados como ser: process, startEvent, userTask, etc.

Listing 2.1: Ejemplo de proceso de negocio para solicitar días de licencia en formato XML

```
<?xml version=""1.0"" encoding=""UTF-8""?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
  xmlns:omgdc="http://www.omg.org/spec/DD/20100524/DC"
  xmlns:omgdi="http://www.omg.org/spec/DD/20100524/DI"
  xmlns:flowable="http://flowable.org/bpmn"
  typeLanguage="http://www.w3.org/2001/XMLSchema"
  expressionLanguage="http://www.w3.org/1999/XPath"
  targetNamespace="http://www.flowable.org/processdef">

  <process id="holidayRequest" name="Holiday Request" isExecutable="true">

    <startEvent id="startEvent"/>
    <sequenceFlow sourceRef="startEvent" targetRef="approveTask"/>

    <userTask id="approveTask" name="Approve or reject request"/>
    <sequenceFlow sourceRef="approveTask" targetRef="decision"/>

    <exclusiveGateway id="decision"/>
    <sequenceFlow sourceRef="decision" targetRef="externalSystemCall">
      <conditionExpression xsi:type="tFormalExpression">
        <![CDATA[
          ${approved}
        ]]>
      </conditionExpression>
    </sequenceFlow>
    <sequenceFlow  sourceRef="decision" targetRef="sendRejectionMail">
      <conditionExpression xsi:type="tFormalExpression">
        <![CDATA[
          ${!approved}
        ]]>
      </conditionExpression>
    </sequenceFlow>

    <serviceTask id="externalSystemCall" name="Enter holidays in external system"
        flowable:class="org.flowable.CallExternalSystemDelegate"/>
    <sequenceFlow sourceRef="externalSystemCall" targetRef="holidayApprovedTask"/>

    <userTask id="holidayApprovedTask" name="Holiday approved"/>
    <sequenceFlow sourceRef="holidayApprovedTask" targetRef="approveEnd"/>

    <serviceTask id="sendRejectionMail" name="Send out rejection email"
        flowable:class="org.flowable.SendRejectionMail"/>
    <sequenceFlow sourceRef="sendRejectionMail" targetRef="rejectEnd"/>

    <endEvent id="approveEnd"/>
    <endEvent id="rejectEnd"/>

  </process>
</definitions>
```

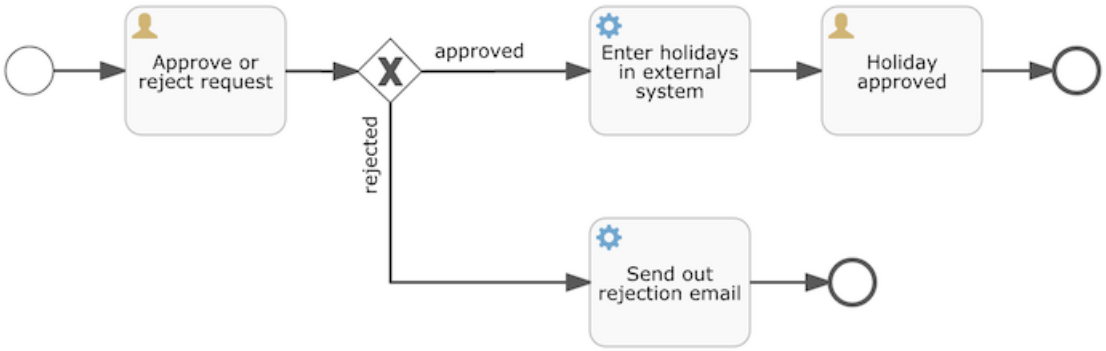


Figura 2.3: Representación gráfica del proceso anterior (Listing 2.1)

Advertir que, al utilizar una herramienta gráfica para el modelado, el archivo XML también contendrá la parte de "visualización" que describe la información gráfica, como las coordenadas de los diversos elementos de la definición del proceso (toda la información gráfica está contenida en otra etiqueta BPMNDiagram del XML).

Cada paso (nombrado como "actividad" en la terminología BPMN 2.0) tiene un atributo *id* que le da un identificador único en el archivo XML. Todas las actividades también pueden tener un nombre opcional para aumentar la legibilidad del diagrama visual.
Las actividades están conectadas por un flujo de secuencia (**sequence flow**), que es una flecha dirigida en el diagrama visual. Al ejecutar una instancia de proceso, la ejecución fluirá desde el evento de inicio hasta la siguiente actividad, siguiendo el flujo de secuencia.

Los flujos de secuencia que salen de la puerta exclusiva (***exclusive gateway***) son especiales: ambos tienen una condición definida en forma de expresión. Cuando la ejecución de la instancia de proceso llega a esta puerta de enlace se evalúan las condiciones y se toma la primera que se resuelve como verdadera.

La condición escrita como expresión tiene la forma ${aprobado}. La variable "aprobado" se denomina **variable de proceso**. Una variable de proceso es un dato que se almacena junto con la instancia de proceso y se puede utilizar durante la vida útil de la instancia de proceso.

El motor de procesos tiene integración con base de datos relacionales (Postgres, MySQL, Oracle, H2, entre otras) y es el único que tiene integración experimental con MongoDB. En la sección siguiente se presenta la integración con BD relacional, que es la tradicional que se ofrecen los BPMS, y luego en el caso de estudio se presentará la integración con BD NoSQL que es parte de esta tesis.

2.1.2.1.1 Motor del proceso Flowable en Base de datos relacional

Todas las tablas del motor Flowable comienzan con el prefijo ACT_. Las tablas que comienzan con ACT_RU_ contienen los datos de tiempo de ejecución (runtime) para la instancia de proceso y los datos relacionados. Una vez que finaliza una instancia de proceso, todos los datos de tiempo de ejecución se eliminan de estas tablas.
Las que comienzan con ACT_HI_ contienen datos históricos para las instancias en ejecución y completadas. Los nombres de estas tablas siguen los nombres de su contraparte en tiempo de ejecución.

Tabla 2.2: Descripción de las principales tablas utilizadas por Flowable

Relacional (H2)	**Descripción**
ACT_RU_EXECUTION	Almacena las instancias del proceso y los punteros (denominados *executions*) al estado actual de la instancia del proceso.
ACT_RU_ACTINST	Cada actividad en la instancia de proceso tiene una fila en esta tabla para indicar el estado actual de la actividad.
ACT_RU_JOB	Los motores Flowable utilizan las tablas de *jobs* para implementar lógica asíncrona, temporizadores o procesamiento de historial. Estas tablas almacenan los datos necesarios para cada job.
ACT_RU_SUSPENDED_JOB	
ACT_RU_TIMER_JOB	
ACT_RU_EVENT_SUBSCR	Cuando una definición de proceso usa eventos, el motor almacena una referencia a esa tabla. Esto simplifica la consulta de qué instancias están esperando un cierto tipo de evento.
ACT_RU_IDENTITYLINK	Almacena datos sobre usuarios o grupos y su rol con respecto a las instancias.
ACT_RU_TASK	Contiene una entrada para cada tarea de usuario de una instancia en ejecución que no ha finalizado. Luego, esta tabla se utiliza al consultar las listas de tareas de los usuarios.
ACT_RU_VARIABLE	Almacena variables relacionadas con una instancia.
ACT_HI_COMMENT	Mantienen el histórico de los datos.
ACT_HI_ACTINST	
ACT_HI_DETAIL	
ACT_HI_IDENTITYLINK	
ACT_HI_PROCINST	
ACT_HI_TASKINST	

ACT_HI_VARINST	
ACT_GE_BYTEARRAY	GE significa datos generales, se utiliza para varios casos de uso.
ACT_RE_DEPLOYMENT	RE indica repositorio. Las tablas con este prefijo contienen información estática sobre modelos y definiciones.
ACT_RE_MODEL	
ACT_RE_PROCDEF	
ACT_ID_PROPERTY	

En la Figura 2.4 se presenta el modelo generado en una base de datos H2 (solo se incluyen los atributos con referencias):

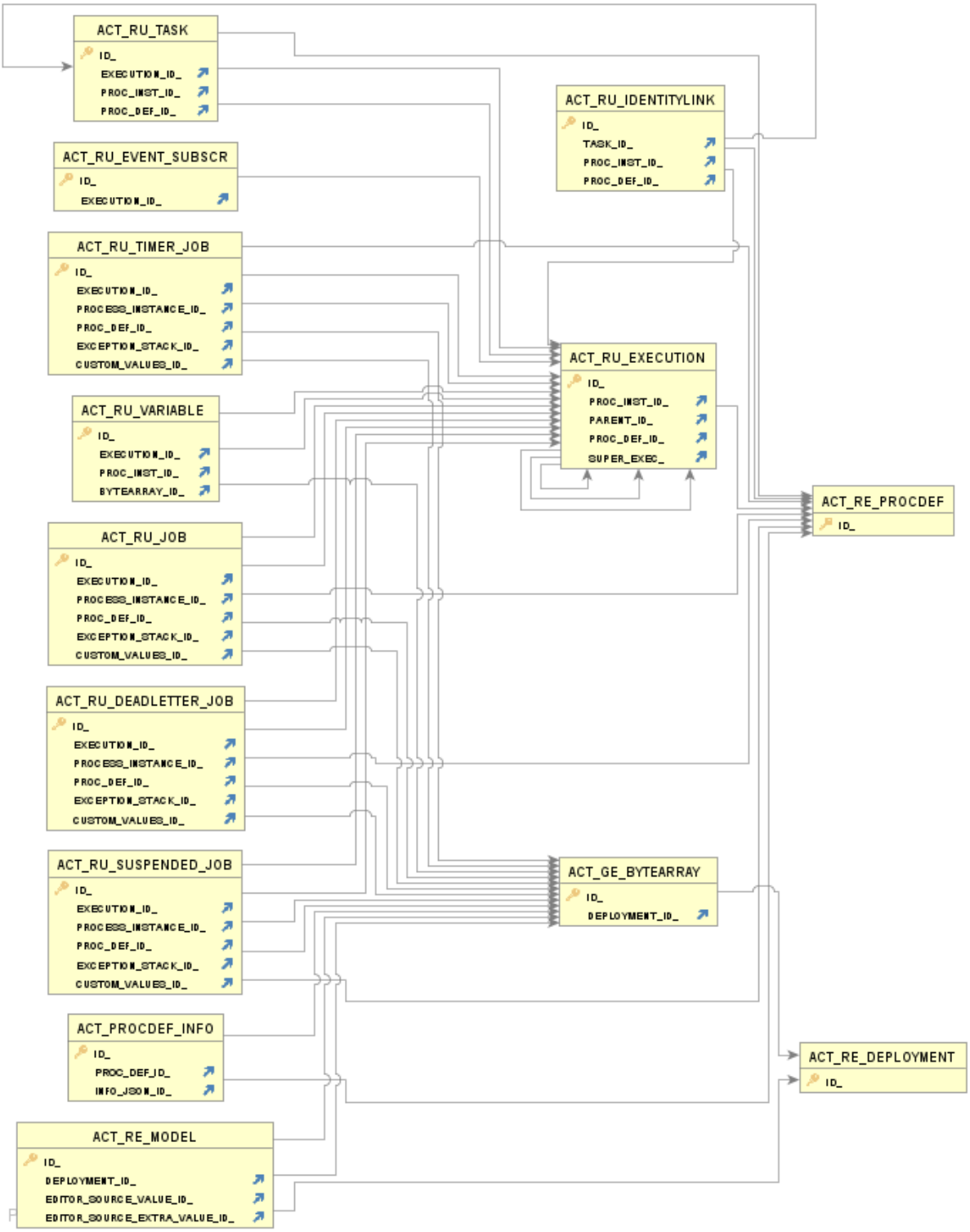


Figura 2.4: Modelo relacional generado en una base de datos H2

2.1.2.1.3 La API y los servicios del motor de procesos Flowable

La API del motor de procesos es la forma más común de interactuar con Flowable. El punto de partida es la creación del ProcessEngine que se puede realizar de varias formas. Una vez creado, desde el ProcessEngine se pueden obtener los distintos servicios que permite trabajar con el proceso de negocio.

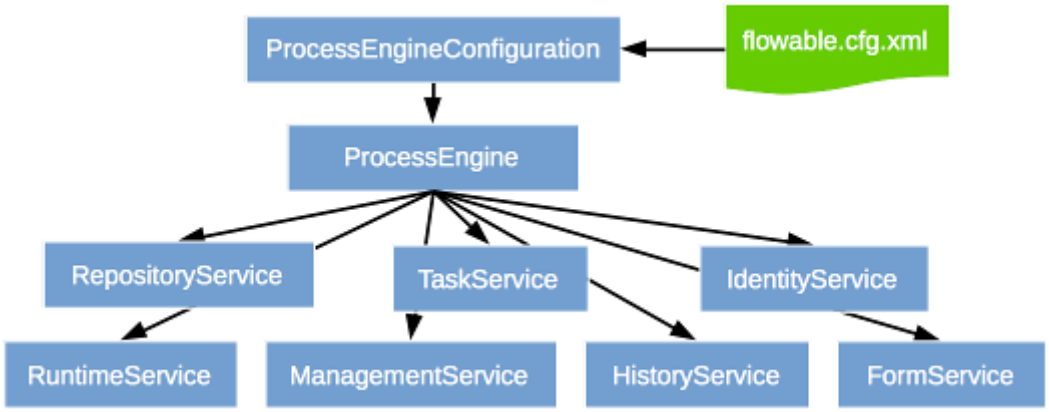


Figura 2.5: API y Servicio del motor de proceso de Flowable extraído de [4]

En el siguiente código se puede visualizar la creación del ProcesosEngine para una base de datos Postgres:

```
ProcessEngine processEngine = new StandaloneProcessEngineConfiguration()
        .setJdbcUrl("jdbc:postgresql://127.0.0.1:5432/flowable?sslmode=disable")
        .setJdbcUsername("postgres")
        .setJdbcPassword("sa")
        .setJdbcDriver("org.postgresql.Driver")
        .setDatabaseSchemaUpdate(ProcessEngineConfiguration.DB_SCHEMA_UPDATE_TRUE)
        .setAsyncExecutorActivate(true)
        .buildProcessEngine();
```

Una vez creado podemos obtener los diferentes servicios para interactuar con el motor de procesos, algunos ejemplos:

```
RuntimeService runtimeService = processEngine.getRuntimeService();
RepositoryService repositoryService = processEngine.getRepositoryService();
TaskService taskService = processEngine.getTaskService();
ManagementService managementService = processEngine.getManagementService();
IdentityService identityService = processEngine.getIdentityService();
HistoryService historyService = processEngine.getHistoryService();
FormService formService = processEngine.getFormService();
```

Para entender la idea de los servicios, sin entrar en muchos detalles, se explica **RepositoryService**. Es probablemente el primer servicio que se necesita cuando se trabaja con el motor de procesos de Flowable. Este servicio ofrece operaciones para administrar y manipular implementaciones y definiciones de procesos (es una contraparte en Java del proceso BPMN 2.0).

Este servicio permite:

- Consultar sobre implementaciones y definiciones de procesos conocidos por el motor.
- Suspender y activar deployment en su conjunto o definiciones de procesos específicos.
- Recuperar varios recursos. Como archivos contenidos en el deployment o diagramas de procesos (que el motor genera automáticamente al cargarlos).
- recuperar una versión POJO de la definición del proceso. Se puede utilizar para realizar una introspección del proceso utilizando Java en lugar de XML.

2.2 Bases NoSQL

NoSQL (“no solo SQL”) es un sistema de gestión de bases de datos no relacionales. Surge como una alternativa al sistema de base de datos relacional. Se utiliza con datos distribuidos, especialmente con aplicaciones que involucran big data y aplicaciones con datos en tiempo real. [9]

Los sistemas NoSQL comparten varias características clave incluyendo un modelo de datos más flexible, mayor escalabilidad y rendimiento superior.
Escalan fácilmente a medida que crece el volumen de datos; a diferencia de bases de datos relacionales en donde se degrada rápidamente el rendimiento a medida que aumenta el volumen de datos.
La flexibilidad en el esquema admite diferentes tipos de datos: datos estructurados, semiestructurados y no estructurados. Está diseñado para cubrir algunos de los inconvenientes de los sistemas de datos relacionales, específicamente cuando se trata de big data de alta variabilidad.

El modelo de datos NoSQL no siempre garantiza las propiedades ACID (Atomicity, Consistency, Isolation and Durability) de los sistemas relacionales, sino que garantiza las propiedades BASE (Basically Available, Soft state, Eventual consistency).

2.2.1 Propiedades BASE

Es un modelo de base de datos que podría considerarse como una alternativa del modelo clásico ACID.

Basic Availability: La principal preocupación de esta propiedad es proporcionar una solución tolerante a fallas para las bases de datos replicadas. Intenta eliminar la consistencia para proporcionar un estado máximo de disponibilidad al permitir una falla parcial en lugar de la falla total del sistema.

Soft state: Se prioriza la propagación de datos, delegando el control de inconsistencias a elementos externos. Los datos en las diferentes réplicas no tienen que ser mutuamente consistentes en todo momento.

Eventual consistency: se asegura la consistencia entre réplicas solo después de cierto tiempo.

2.2.2 Teorema CAP

Cuando se trabaja con bases NoSQL hay que considerar los siguientes criterios: Consistency, Availability, Partition tolerance. El teorema CAP afirma que en una arquitectura con un sistema distribuido no puede cumplir más de dos de esos criterios simultáneamente. [10]

Consistency: Esta propiedad asegura que las actualizaciones de los datos deben distribuirse entre los nodos para asegurar que todos los usuarios puedan recuperar y consultar los datos modificados. Cualquier lectura recibe como respuesta la escritura más reciente o un error.

Availability: La necesidad de al menos un nodo disponible en caso de falla. Cualquier solicitud recibe una respuesta no errónea, pero sin la garantía de que contenga la escritura más reciente.

Partition tolerant: El sistema permanece activo incluso si se corta la conexión entre los nodos.

El teorema CAP implica que cuando se usa un sistema distribuido, con el riesgo inherente de falla, se debe elegir entre consistencia y disponibilidad y no se pueden garantizar ambas al mismo tiempo.

AP (Availability y Partition tolerant): cuando se elige la disponibilidad sobre la consistencia, el sistema siempre procesa la solicitud del cliente e intentará devolver la versión disponible más reciente de la información, incluso si no puede garantizar que esté actualizada debido a problemas en la red.

CP (Consistency y Partition tolerant): cuando se elige la consistencia sobre la disponibilidad, el sistema devolverá un error o un *timeout* (tiempo de espera agotado) si la información no se puede actualizar a otros nodos debido a problemas en la red.

El sistema de base de datos diseñado con ACID (RDBMS) generalmente elige la consistencia sobre la disponibilidad, mientras que el sistema diseñado con BASE elige la disponibilidad sobre la consistencia.

2.2.3 Atributos principales

En esta sección se realiza una explicación de los atributos principales a considerar cuando se habla de base de datos NoSQL [11].

2.2.3.1 Métodos de partición

Antes de explicar qué son los métodos de partición se detallan dos técnicas para abordar el crecimiento del sistema: escalado vertical y horizontal

El *escalado vertical* implica aumentar la capacidad de un solo servidor, como usar una CPU más potente, agregar más RAM o aumentar la cantidad de espacio de almacenamiento. Las limitaciones en la tecnología disponible pueden impedir que una sola máquina sea lo suficientemente potente para una carga de trabajo determinada. Como resultado, existe un máximo práctico para el escalado vertical.

El *escalado horizontal* implica dividir el conjunto de datos del sistema y cargarlo en varios servidores, agregando servidores adicionales para aumentar la capacidad según sea necesario. Si bien la velocidad o capacidad general de una sola máquina puede no ser alta, cada máquina maneja un subconjunto de la carga de trabajo general, lo que podría brindar una mayor eficiencia que un solo servidor de alta capacidad y velocidad. Ampliar la capacidad solo requiere agregar servidores adicionales según sea necesario, lo que puede tener un costo general más bajo que el hardware de alta gama para una sola máquina. La desventaja es una mayor complejidad en la infraestructura y el mantenimiento.

Los métodos de partición son métodos para almacenar diferentes datos en diferentes nodos (escalado horizontal).
Un método es el particionado (partitioning). Una partición es una división de una base de datos lógica o de sus elementos en distintas partes independientes.
Otro método es la fragmentación (sharding). Se dividen los datos por rangos de claves y se distribuyen entre dos o más instancias de bases de datos. Sharding es un tipo de particionamiento conocido como particionamiento horizontal.

2.2.3.2 Métodos de replicación

Son métodos para almacenar datos de forma redundante en varios nodos.
La replicación proporciona redundancia (*redundancy*) y aumenta la disponibilidad de datos (data availability). Con múltiples copias de datos en diferentes servidores de bases de datos, la replicación proporciona un nivel de tolerancia a fallas frente a la pérdida de un único servidor de bases de datos.

2.2.3.3 Consistencia

Métodos que aseguran la consistencia en un sistema distribuido:

- Consistencia eventual (*eventual consistency*): asegura la consistencia entre réplicas solo después de cierto tiempo.
- Consistencia fuerte o inmediata (strong/*immediate consistency*): los datos consultados inmediatamente después de una actualización serán consistentes para todos los observadores. Las bases de datos relacionales tradicionales han sido diseñadas con base a este concepto de consistencia.

2.2.3.4 Transacción

Soporte para garantizar la integridad de los datos después de manipulaciones no atómicas de datos.

2.2.3.5 Control de concurrencia

El control de concurrencia garantiza que las operaciones de la base de datos se puedan ejecutar simultáneamente sin comprometer el estado del sistema. El control de concurrencia pesimista bloqueará cualquier operación potencialmente conflictiva. El control de concurrencia optimista retrasa la verificación hasta después de que haya ocurrido un conflicto, abortando y volviendo a intentar una de las operaciones involucradas en cualquier conflicto de escritura que surja.

2.2.4 Clasificación

A continuación, se presenta una clasificación de sistemas NoSQL de acuerdo con su implementación (modelo de datos) [9]. Para cada categoría se presentan algunas herramientas; destacar que algunas pueden pertenecer a más de una categoría.

2.2.4.1 Key-Value

Es un tipo de base de datos NoSQL que utiliza un método simple de clave-valor para almacenar datos. Implementa una tabla hash para almacenar claves únicas que apuntan a los datos correspondientes.

Es adecuado en situaciones en las que desea almacenar por ejemplo la sesión de un usuario, el carrito de compras, o para obtener detalles como productos favoritos.

Algunas de las bases de datos Key-value más populares:

Redis [12]: Una base de datos de código abierto distribuida en memoria que admite una variedad de estructuras de datos, como listas, mapas, conjuntos, conjuntos ordenados y cadenas. Redis se centra en el rendimiento, por lo que la mayoría de sus decisiones de diseño priorizan un alto rendimiento y latencias muy bajas.

Amazon DynamoDB [13]: Es un servicio de base de datos NoSQL ofrecido por Amazon que proporciona un servicio de base de datos NOSQL rápido, altamente confiable y rentable. Ofrece latencias bajas y predecibles a cualquier escala. Almacena datos en discos de estado sólido (SSD) en lugar de discos duros tradicionales, lo que proporciona un acceso más rápido a los datos. Los datos se replican de forma síncrona en varias "zonas de disponibilidad AWS" en una "región AWS" para proporcionar alta disponibilidad. Replica datos en al menos tres centros de datos, lo que proporciona alta disponibilidad y durabilidad incluso en escenarios de falla complejos.

Riak KV [14]: Provee un almacén de clave-valor distribuido y tolerante a fallos. Ofrece tanto una versión de código abierto como una versión empresarial.

Tabla 2.3: Resumen comparación bases de datos de tipo Key-value

	Consistencia	**Transaccionalidad**	**Método consulta**
Redis	Eventual Consistency Strong eventual consistency with CRDTs	Bloqueo optimista. Ejecución atómica de scripts	Protocolo propietario REsP (REdis Serialization Protocol)
Amazon DynamoDB	Eventual Consistency Immediate Consistency	ACID (con restricciones)	RESTful HTTP API
Riak KV	Eventual Consistency	No	HTTP API

2.2.4.2 Wide column

Es una familia de arquitectura de base de datos NoSQL que almacena los datos por columna en lugar de por fila. Esta forma de columnas llega hasta el almacenamiento real, lo que permite un aumento del rendimiento en aplicaciones que requieren análisis en tiempo real.

Dado que los nombres de las columnas y las claves de registro no son fijos, y dado que un registro puede tener miles de columnas, los almacenes *wide column* pueden verse como almacenes de clave-valor bidimensionales.

Algunas de las bases de datos Wide column más populares:

Cassandra [15]: es un sistema de base de datos de código abierto basado en el modelo Dynamo de Amazon y también en BigTable de Google. Ofrece características como alta disponibilidad, tolerancia a la partición, persistencia, alta escalabilidad. Tiene un esquema dinámico. Se puede usar para una variedad de aplicaciones como sitios web de redes sociales, banca y finanzas, análisis de datos en tiempo real, venta minorista en línea, etc. La desventaja de Cassandra es que las lecturas son comparativamente más lentas que la escritura.

HBase [16]: es un sistema de base de datos también de código abierto que se basa en HDFS (Hadoop Distributed File System), con baja latencia porque permite el acceso aleatorio y admite el procesamiento de datos en tiempo real. Hbase está diseñado para escalar linealmente para datos con gran volumen. Soporta la consistencia inmediata como también la consistencia eventual.

Tabla 2.4: Resumen comparación base de datos de tipo Wide column

	Consistencia	**Transaccionalidad**	**Método consulta**
Cassandra	Eventual Consistency Immediate Consistency	No	Protocolo propietario CQL (Cassandra Query Language, an SQL-like language)
HBase	Eventual Consistency Immediate Consistency	ACID de una sola fila (en millones de columnas)	Java API RESTful HTTP API

2.2.4.3 Document

Mientras que las bases de datos relacionales almacenan datos en filas y columnas, las bases de datos de documentos almacenan datos en documentos.

Estos documentos son mucho más flexibles ya que no tienen esquema. En bases de datos relacionales, un registro dentro de la misma base de datos tendrá los mismos campos de datos y los campos de datos no utilizados se mantienen vacíos, pero en el caso de almacenes de documentos, cada documento puede tener datos similares y diferentes. Los documentos de la base de datos se direccionan mediante una clave única que representa ese documento.

Los almacenes de documentos suelen utilizar notaciones internas (JSON, XML, PDF, etc.) que pueden procesarse directamente en aplicaciones.
Los documentos JSON, por ejemplo, también se podrían almacenar como texto puro en almacenes de valores clave o sistemas de bases de datos relacionales. Sin embargo, eso requeriría el procesamiento del lado del cliente de las estructuras, lo que tiene la desventaja de que las características que ofrecen los almacenes de documentos (como los índices secundarios) no estén disponibles.

Las bases de datos de documentos son de uso general, útiles para una amplia variedad de aplicaciones debido a la flexibilidad del modelo de datos, la capacidad de realizar consultas en cualquier campo y la asignación natural del modelo de datos del documento a los objetos en los lenguajes de programación modernos. Deben evitarse los almacenes de documentos si la base de datos tendrá muchas relaciones y normalización.

Es una de las principales categorías de bases de datos NoSQL ya que los datos orientados a documentos constituyen una gran parte de los datos semiestructurados disponibles en línea.

Algunas de las bases de datos orientadas a documentos más populares:

MongoDB [17]: es uno de los almacenes de documentos de código abierto más populares. Está disponible como servicio en la nube o también para su utilización en una infraestructura autogestionada. Desde la versión 4 admite transacciones ACID de varios documentos. Además, incluye consultas ad hoc, indexación, replicación, balanceo de carga, almacenamiento de archivos (GridFS), realizar consultas utilizando JavaScript.

Utiliza JSON como notación para almacenar datos. La ventaja de JSON es que mapea naturalmente a objetos de cualquier lenguaje, lo que facilita las tareas de desarrollo. Internamente MongoDB convierte los documentos en un formato llamado BSON (Binary JSON). Esto logra un acceso más rápido internamente y soportar más tipos de datos que los que soporta JSON.

CouchDB [18]: es un almacén de documentos JSON nativa de código abierto. Es escalable desde servidores distribuidos globalmente hasta teléfonos móviles. Incluye una API HTTP RESTful para leer y actualizar (agregar, editar, eliminar) documentos.

Para las operaciones de lectura se utiliza un modelo de control de concurrencia de múltiples versiones (MVCC en inglés) en el que cada cliente ve un snapshot consistente de la base de datos desde el principio hasta el final de la operación de lectura. Esto significa que CouchDB puede garantizar la semántica transaccional por documento.

Tabla 2.5: Resumen comparación base de datos de tipo Document

	Consistencia	**Transaccionalidad**	**Método consulta**
MongoDB	Eventual Consistency y Immediate Consistency	Multi-document ACID	HTTP y JSON
CouchDB	Eventual Consistency	Operaciones atómicas para un solo documento	RESTful HTTP/JSON API

2.2.4.4 Graph

En las bases de datos NoSQL orientadas a grafos los datos se representan utilizando estructuras de grafos: nodos, aristas y propiedades. Un grafo es una representación abstracta de un conjunto de objetos.

El modelo en grafo es útil cuando lo importante es la interrelación que existe entre los datos, más que en los propios datos. Los datos se modelan como una red de relaciones entre elementos específicos.

Una de las características principales de este tipo de bases de datos es que las interrelaciones se representan de forma explícita en la base de datos. Eso quiere decir que las interrelaciones entre los distintos nodos del grafo se almacenan en el disco como punteros entre los dos nodos relacionados. Eso suele simplificar la recuperación de elementos relacionados.

Como contrapartida al tener datos altamente relacionados, particionarlos entre diferentes nodos es un problema en este tipo de base de datos por lo que no son tan fácilmente escalables como otros modelos NoSQL.

Las bases de datos grafos son útiles en los casos en los que las relaciones son fundamentales para la aplicación, como navegar por conexiones de redes sociales, topologías de red o cadenas de suministro.

Algunas de las bases de datos orientadas a grafos más populares:

Neo4j [19]: es una base de datos de grafos nativa de código abierto disponible desde 2007. Proporciona características completas de base de datos como transacciones ACID y soporte de clusters.

Utiliza el lenguaje Cypher para consultas. Es un lenguaje declarativo similar a SQL pero optimizado para grafos.

JanusGraph [20]: es una base de datos de grafos de código abierto optimizada para clusters distribuidos. Implementa interfaces robustas y modulares para la persistencia de datos, indexación de datos y acceso de clientes. Presenta una arquitectura modular que permite interoperar con una amplia gama de tecnologías de almacenamiento (Cassandra, HBase, BerkeleyDB), índices (Elasticsearch, Lucene) y clientes. Las transacciones no necesariamente son ACID va a depender del tipo de almacenamiento utilizado.

Tabla 2.6: Resumen comparación base de datos de tipo Graph

	Consistencia	**Transaccionalidad**	**Método consulta**
Neo4j	Eventual Consistency Immediate Consistency (con restricciones)	ACID	Cypher, RESTful HTTP API, Java API, entre otros
JanusGraph	Eventual Consistency Immediate Consistency	ACID (con restricciones)	Java API, Apache TinkerPop

3 Análisis de problema

En esta sección se analizará el problema objeto de esta tesis. En la subsección 3.1 se realiza una introducción al problema y en la subsección 3.2 se presenta, de manear simplificada, el estado actual con relación al problema planteado. Por último, en la subsección 3.3, se presentan diferentes escenarios.

3.1 Introducción al problema

Los procesos de negocio que realizan las organizaciones pueden ser implementados y ejecutados en plataformas BPM (Business Process Management System, BPMS), proveen soporte a las etapas del ciclo de vida de los procesos Análisis & Diseño, Configuración, Ejecución y Evaluación, que guía las actividades y tecnologías para su gestión. Las organizaciones, sus procesos de negocio y los sistemas de software que soportan sus procesos y datos, son cada vez más complejos, definiendo ecosistemas en los que se hace necesario integrar diferentes visiones, técnicas y herramientas para la gestión de la información, procesos y sistemas asociados. En este contexto, la visión compartimentada de procesos por un lado y datos por el otro no es adecuada para proveer a la organización de la inteligencia organizacional basada en evidencia necesaria para mejorar su operativa diaria. [21]

En este proyecto se propone estudiar la integración de datos de procesos y organizacionales en bases de datos NoSQL. El escenario general de integración incluye la ejecución de procesos en plataformas BPMS con bases de datos relacionales y NoSQL, y bases de datos organizacionales centrales de tipo relacional y NoSQL.

Como parte del análisis del problema interesaba identificar plataformas BPMS que usaran BPMN 2.0 y además tuviera capacidades de integración con BDs NoSQL, con lo cual se realizó un relevamiento del estado actual de plataformas BPMS y BDs soportadas, que se presenta a continuación.

3.2 Estado actual plataformas BPMS y Bases de datos soportadas

A continuación, se lista el estado actual de las diferentes plataformas BPMS seleccionadas y el soporte que presentan de bases de datos para almacenar la información del motor (definiciones de procesos, configuraciones de procesos, historial de ejecución de procesos, usuarios, etc.).

Tabla 3.1: Plataformas BPMS y soporte base de datos

BPMS	**BD Relacional**	**NoSQL**
Activiti	h2, mysql, oracle, postgres, db2, mssql	No tiene soporte
Camunda	MySQL, MariaDB, Oracle, DB2, PostgreSQL, mssql, H2, CockroachDB	No tiene soporte
Flowable	h2, mysql, oracle, postgres,	Soporte experimental con

	db2, mssql	MongoDB [22]
jBPM	db2, derby, h2, hsqldb, mysql5, mysqlinnodb, oracle, sqlserver, sybase	No tiene soporte
Bizagi	Oracle, mssql	No tiene soporte
Bonita	MySQL, PostgreSQL, mssql, Oracle	No tiene soporte

El soporte que se indica como experimental no se encuentra en la documentación de la plataforma, sino en blogs con pruebas de conceptos para lograr la integración.

3.3 Análisis de escenarios

En esta sección, se realizaron comparaciones y análisis de escenarios relevantes para combinaciones de BPMS y bases de datos relacionales y no relacionales. Como resultado de este análisis, se identificó un escenario (#3) particularmente interesante para la tesis, el cual se presenta en detalle en los próximos capítulos.

Tabla 3.2: Escenarios considerados de interés

#	**BPMS**	**Datos del motor de proceso**	**Datos organizacionales**
1	Flowable	MongoDB	Postgres
2	Flowable	Postgres	Cassandra
3	Flowable	MongoDB	Redis KV
4	Flowable	MongoDB	Neo4J

4 Solución datos del motor de procesos en NoSQL

En esta sección se presenta la solución al problema planteado en el capítulo anterior con respecto a almacenar los datos del motor del proceso en bases NoSQL. En la subsección 4.1 se indican las herramientas seleccionadas para presentar la solución, y en la subsección 4.2 se detalla la integración de estas herramientas.

4.1 Selección de herramientas

Como se presentó previamente el soporte de BPMS, para los datos del motor de proceso, en bases NoSQL todavía es experimental. La selección de la herramienta se basó en el post Running Flowable on MongoDB, de agosto 2018, publicado en el blog oficial de Flowable [22].

4.2 Flowable con MongoDB

4.2.1 Diseño del esquema

El cambio más importante al pasar de un base de datos relacional a MongoDB es la forma en que se modelan los datos. En esta sección se presentan las diferentes consideraciones que existen en la librería de MongoDB para poder integrarse con Flowable. Este análisis fue necesario ya que se tuvieron que hacer modificación a la librería para poder lograr la integración.

4.2.1.1 Mapeo entre tablas y documentos

Uno de los desafíos es el mapeo entre Tablas del modelo relacional y Documentos JSON. A primera vista se puede visualizar una relación bastante directa (analógica) entre una fila de la tabla relacional y un documento. Cada columna de la tabla es un campo (*field*) del documento JSON. Finalmente, una colección de documentos JSON es una tabla.

4.2.1.2 Atributos nulos

Al tener MongoDB (y bases NoSQL en general) un esquema flexible los campos entre distintos documentos pueden no existir. Esto es importante tener en cuenta a la hora de mapear objetos a documentos.

4.2.1.3 Claves primarias

En Flowable todas las tablas tienen IDs autogenerados como claves primarias. En MongoDB los documentos almacenados en colecciones requieren un único campo _id que actúa como clave primaria.
Flowable no utiliza las secuencias que ofrecen las bases de datos relacionales. Utiliza una técnica en la que cada motor de proceso (*process engine*) tiene un generador de IDs. El generador de ID que tiene por defecto reserva un bloque de IDs en la base de datos, utilizando una tabla específica para esto, de modo que ningún otro motor podrá usar IDs del mismo bloque. Durante las operaciones del motor, cuando el generador de ID advierte que el bloque de ID está agotado, se inicia una nueva transacción

para obtener un nuevo bloque.
La alternativa al generador de IDs que se utiliza por defecto es *org.flowable.engine.impl.persistence.StrongUuidGenerator*, el cual genera un UUID (*Universally Unique IDentifier*) único localmente y lo usa como identificador para todas las entidades. Flowable presenta esta alternativa para los casos de uso con concurrencia muy alta, dado que el UUID se genera sin necesidad de acceder a la base de datos.
Para la integración con MongoDB se hace beneficio de esta alternativa. Al momento de iniciar y configurar el motor de procesos Flowable se selecciona *StrongUuidGenerator* como generador de IDs.

4.2.1.4 Índices

MongoDB soporta índices similares a los que se encuentran en base de datos relacionales. Si existe un índice apropiado para una consulta, MongoDB puede utilizar el índice para limitar la cantidad de documentos que debe inspeccionar.
MongoDB crea un índice único (*unique index*) en el campo _id durante la creación de una colección. Este índice evita que se inserten dos documentos con el mismo valor para el campo _id.

4.2.1.5 JOINs

Para realizar el JOINs entre documentos se utiliza principalmente la referencia por atributo a documentos. Hay que aclarar que esta referencia no es utilizando técnicas de agregación que ofrece MongoDB para consultar varios documentos.

En la siguiente figura se presenta un ejemplo con otra estrategia en una relación simple entre estudiantes que cursan asignaturas. En este ejemplo puntual para la relación entre estudiantes y asignaturas se embeben las asignaturas como una lista en la colección de documentos de estudiantes.

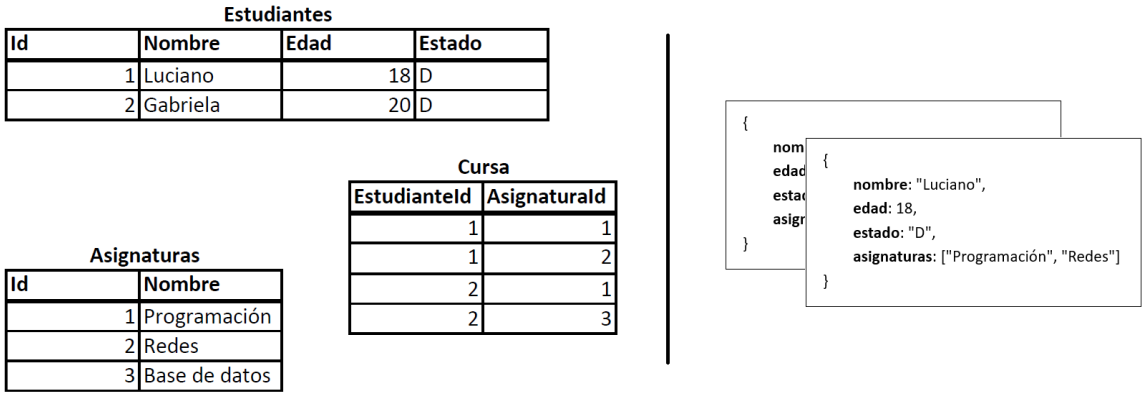


Figura 4.1: Ejemplo de mapeo entre tablas y colecciones de documentos JSON

4.2.1.6 Tipos de datos

Otro elemento para considerar son los tipos de datos de cada atributo. MongoDB almacena los datos como documentos BSON [23]. BSON es una representación binaria de documentos JSON, y además contiene más tipos de datos que JSON. Con los tipos de datos que presenta BSON se realiza el mapeo directamente: string, integer, date, boolean, long, double, binary.

4.2.1.7 Transacciones

Las transacciones en Flowable juegan un rol fundamental para garantizar la consistencia de los datos y resolver problemas de concurrencia. Toda llamada a la API de Flowable ocurre dentro de una transacción. El proceso pasa de un estado de espera a otro estado de espera. Por ejemplo, cuando se inicia una instancia de proceso habrá una transacción de base de datos desde el inicio de la instancia de proceso hasta el siguiente estado de espera.

En MongoDB las operaciones sobre un único documento son atómicas. Desde la versión 4.0 MongoDB soporta transacciones ACID sobre múltiples documentos.
Las transacciones de múltiples documentos permiten abordar una gama completa de casos de uso. Sin esta característica sería mucho más complejo poder utilizar MongoDB como base de datos para el motor de procesos de Flowable.

4.2.1.8 Resumen de terminología y conceptos

En la Tabla 4.1 se presenta el mapeo entre los elementos del modelo relacional y su correspondencia en MongoDB [17] utilizado en este proyecto.

Tabla 4.1: Mapeo modelo relacional y MongoDB

SQL términos y conceptos	**MongoDB términos y conceptos**
Base de datos	Base de datos
Tabla	Colección de documentos
Fila	Documento (BSON documento)
Columna (atributo)	Campo del documento (*field*)
Tipos de datos	Tipos BSON
Id (clave primaria)	UUID
Índice	Índice
JOINs	Documentos embebidos, referencias a documentos
Transacciones ACID	Transacciones ACID sobre varios documentos

4.2.2 Soporte experimental a MongoDB en Flowable

El soporte provisto por Flowable para integración con MongoDB se realiza a través de la librería flowable-process-engine-mongodb (6.4.0.alpha1). No hay una liberación final de esta librería utilizada para la integración entre Flowable y MongoDB. La versión actual es alpha1 [24] ya que no todas las características han sido implementadas. A continuación, se detallan los principales componentes incluidos en la librería para poder utilizar MongoDB para los datos del motor BPMN de Flowable.

Esta librería hace uso de características del motor BPMN Flowable que permite establecer otra implementación en la capa de persistencia. Los *Data Manager* son estos componentes que manejan la lógica de la capa de persistencia. Son componentes conectables en donde las implementaciones predeterminadas se pueden intercambiar con otra implementación.

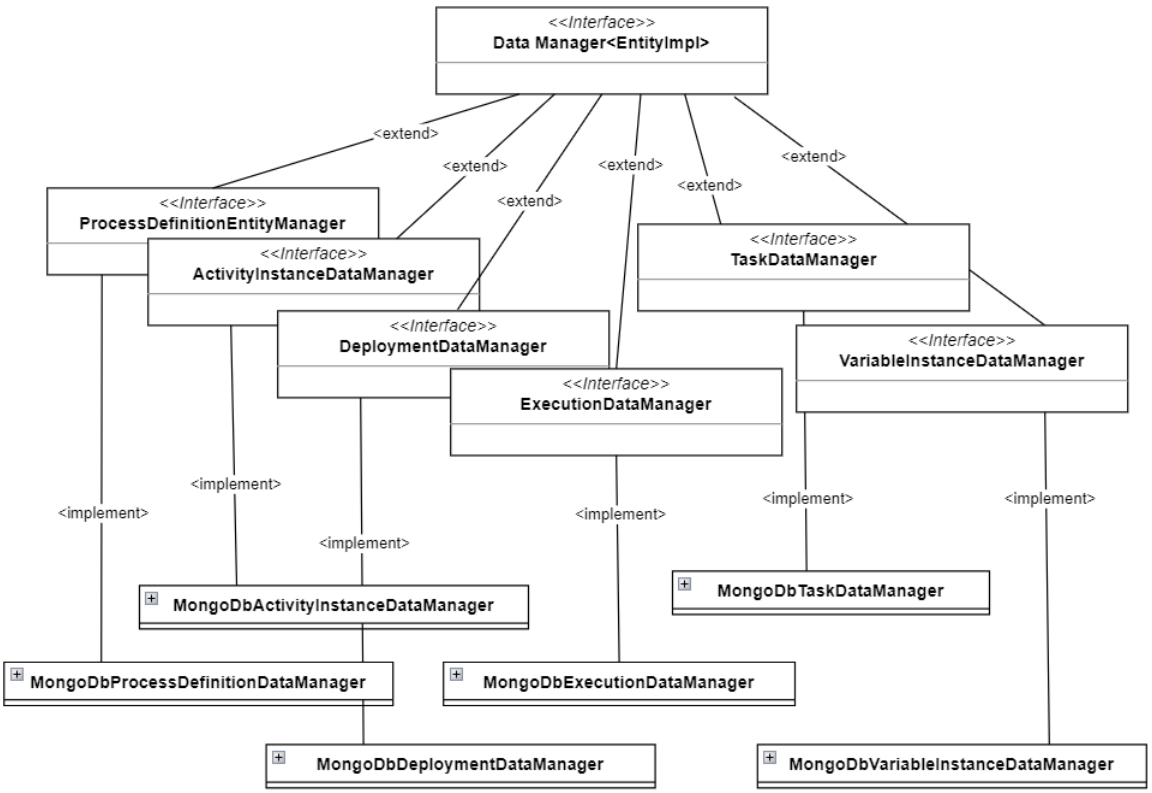


Figura 4.2: Ejemplos de Data Managers y su implementación

En el siguiente fragmento de código vemos la interfaz más básica de un *Data Manager*:

```
public interface DataManager<EntityImpl extends Entity> {

        EntityImpl create();
        EntityImpl findById(String entityId);
        void insert(EntityImpl entity);
        EntityImpl update(EntityImpl entity);
        void delete(String id);
        void delete(EntityImpl entity);

}
```

Después cada entidad de Flowable implementa la interfaz de *Data Manager* según corresponda. Estas interfaces extienden de esta interfaz básica. Por ejemplo, la interfaz de Flowable *org.flowable.engine.impl.persistence.entity.data.ProcessDefinitionDataManager* define el *Data Manager* encargado de trabajar con la definición de procesos. Para base de datos relacionales se utiliza, viene por defecto en el código común del motor de Flowable, el *Data Manager MyBatisProcessDefinitionDataManager* que implementa *ProcessDefinitionDataManager*. Para MongoDB se define otro *Data Manager MongoDbProcessDefinitionDataManager* que implementa también *ProcessDefinitionDataManager*.

En el siguiente fragmento de código se presenta alguno de los métodos que se deben implementar según la interfaz *org.flowable.engine.impl.persistence.entity.data.ProcessDefinitionDataManager:*

```
public interface ProcessDefinitionDataManager extends DataManager<ProcessDefinitionEntity> {

   ProcessDefinitionEntity findLatestProcessDefinitionByKey(String processDefinitionKey);

   ProcessDefinitionEntity findLatestProcessDefinitionByKeyAndTenantId(String
processDefinitionKey, String tenantId);

   ProcessDefinitionEntity findLatestDerivedProcessDefinitionByKey(String processDefinitionKey);

   ProcessDefinitionEntity findLatestDerivedProcessDefinitionByKeyAndTenantId(String
processDefinitionKey, String tenantId);

   void deleteProcessDefinitionsByDeploymentId(String deploymentId);

   List<ProcessDefinition> findProcessDefinitionsByQueryCriteria(ProcessDefinitionQueryImpl
processDefinitionQuery);

   long findProcessDefinitionCountByQueryCriteria(ProcessDefinitionQueryImpl
processDefinitionQuery);

   .
   .
   .

   long findProcessDefinitionCountByNativeQuery(Map<String, Object> parameterMap);

   void updateProcessDefinitionTenantIdForDeployment(String deploymentId, String newTenantId);

   void updateProcessDefinitionVersionForProcessDefinitionId(String processDefinitionId, int
version);

}
```

Otro elemento importante es la interfaz *org.flowable.common.engine.impl.interceptor.Session*. La lógica de persistencia específica de bajo nivel se realiza en la clase que implemente esta interfaz. En la librería *MongoDbSession* implementa la interfaz y es creada por *MongoDbSessionFactory*. Esto es similar a las clases predeterminadas *MyBatisDbSqlSession* y *DbSqlSessionFactory* que son parte del código común del motor de Flowable y se encargan de la lógica de persistencia de la base de datos relacional.

MongoDbSession utiliza un driver de Java para MongoDB [25] que permite comunicarse de forma sincrónica con MongoDB. Incluye todas las operaciones necesarias para trabajar con MongoDB, tales como realizar una conexión, acceder a colecciones, operaciones de escritura y lectura, agregaciones, filtros, trabajar con índices, etc.

Cuando se obtiene una entidad de la base de datos, utilizando esta nueva implementación de MongoDB de la capa de persistencia, se obtiene un documento JSON de tipo *org.bson.Document*. Las capas superiores del motor flowable trabajan con Entidades, entonces es necesario realizar la conversión entre las estructuras de datos Java y MongoDB en ambos sentidos. En la librería se crean clases (llamadas *EntityMapper)* que se encargan de hacer todo el mapeado de forma explícita para cada tipo de entidad (Task, ProcessDefinition, Execution, etc.).
El siguiente fragmento de código es de la clase ProcessDefinitionEntityMapper que se encarga del mapeo de ProcessDefinitionEntity:

```
public class ProcessDefinitionEntityMapper extends
AbstractEntityToDocumentMapper<MongoDbProcessDefinitionEntityImpl> {

    @Override
    public MongoDbProcessDefinitionEntityImpl fromDocument(Document document) {
        MongoDbProcessDefinitionEntityImpl processDefinitionEntity = new
MongoDbProcessDefinitionEntityImpl();
        processDefinitionEntity.setId(document.getString("_id"));
        processDefinitionEntity.setName(document.getString("name"));
        processDefinitionEntity.setDescription(document.getString("description"));
        processDefinitionEntity.setKey(document.getString("key"));
        .
        .
        .
        processDefinitionEntity.setEngineVersion(document.getString("engineVersion"));

        // Mongo impl specific
        processDefinitionEntity.setLatest(document.getBoolean("latest"));

        return processDefinitionEntity;
    }

    @Override
    public Document toDocument(MongoDbProcessDefinitionEntityImpl processDefinitionEntity) {
        Document processDefinitionDocument = new Document();
        appendIfNotNull(processDefinitionDocument, "_id", processDefinitionEntity.getId());
        appendIfNotNull(processDefinitionDocument, "name", processDefinitionEntity.getName());
        appendIfNotNull(processDefinitionDocument, "description",
processDefinitionEntity.getDescription());
        appendIfNotNull(processDefinitionDocument, "key", processDefinitionEntity.getKey());
        appendIfNotNull(processDefinitionDocument, "version",
processDefinitionEntity.getVersion());
        .
        .
        .
        appendIfNotNull(processDefinitionDocument, "engineVersion",
processDefinitionEntity.getEngineVersion());

        // Mongo impl specific
```

```
        appendIfNotNull(processDefinitionDocument, "latest",
processDefinitionEntity.isLatest());

        return processDefinitionDocument;
    }

}
```

Otro elemento importante a implementar es *org.flowable.engine.ProcessEngineConfiguration*. Esta clase contiene la información de configuración a partir de la cual se puede construir el motor de procesos; expone por ejemplo los *Data Manager* implementados. La librería implementa la clase *MongoDbProcessEngineConfiguration* que extiende, de la clase de configuración predeterminada, *ProcessEngineConfigurationImpl*.

En el siguiente fragmento de código se puede visualizar la implementación del método *initDataManagers* (sobreescrito):

```
@Override
public void initDataManagers() {
       MongoDbDeploymentDataManager mongoDeploymentDataManager = new
MongoDbDeploymentDataManager();
       mongoDbSessionFactory.registerDataManager(MongoDbDeploymentDataManager.COLLECTION_DEPLOY
MENT, mongoDeploymentDataManager);
       this.deploymentDataManager = mongoDeploymentDataManager;

       MongoDbProcessDefinitionDataManager mongoDbProcessDefinitionDataManager = new
MongoDbProcessDefinitionDataManager();
       mongoDbSessionFactory.registerDataManager(MongoDbProcessDefinitionDataManager.COLLECTION
_PROCESS_DEFINITIONS, mongoDbProcessDefinitionDataManager);
       this.processDefinitionDataManager = mongoDbProcessDefinitionDataManager;
       .....
}
```

Al momento de construir el motor de procesos de Flowable también es necesario inicializar un manejador de esquema. Para cumplir con esta configuración en la librería se crea la clase *org.flowable.mongodb.schema.MongoProcessSchemaManager* que implemente la interfaz *org.flowable.common.engine.impl.db.SchemaManager*. En una base de datos relacional la implementación de esta clase crearía la estructura de tablas. El concepto de esquema que se maneja en esta clase no es el mismo que se utiliza en una base de datos relacional ya que MongoDB no tiene un esquema que fuerce los datos en una estructura estricta. Lo que se realiza en esta es: asegurarse de que se creen todas las colecciones que necesita el motor de Flowable y asegurarse de que se han creado los índices necesarios sobre esas colecciones.

Tabla 4.2: Colecciones generadas por la librería para MongoDB

Relacional	**MongoDB**
ACT_RU_EXECUTION	executions
ACT_RU_ACTINST	activityInstances
ACT_RU_JOB	jobs
ACT_RU_SUSPENDED_JOB	suspendedJobs
ACT_RU_TIMER_JOB	timerJobs
ACT_RU_EVENT_SUBSCR	eventSubscriptions
ACT_RU_IDENTITYLINK	identityLinks
ACT_RU_TASK	tasks
ACT_RU_VARIABLE	variables
ACT_HI_COMMENT	comments
ACT_HI_ACTINST	historicActivityInstances
ACT_HI_DETAIL	historicDetails
ACT_HI_IDENTITYLINK	historicIdentityLinks
ACT_HI_PROCINST	historicProcessInstances
ACT_HI_TASKINST	historicTaskInstances
ACT_HI_VARINST	historicVariableInstances
ACT_GE_BYTEARRAY	byteArrays
ACT_RE_DEPLOYMENT	deployments
ACT_RE_MODEL	models
ACT_RE_PROCDEF	processDefinitions
ACT_ID_PROPERTY	flowableProperties

4.2.2.1 Ajustes realizados a la librería

No todas las características de las librerías estaban implementadas y algunas no funcionaban correctamente. En el Anexo B se detallan los principales ajustes realizados.

5 Solución datos organizacionales en NoSQL

En esta sección se presenta la solución al problema de almacenar los datos organizacionales en un tipo de base de datos NoSQL. En la subsección 5.1 se indica la herramienta seleccionada para presentar la solución, y en la subsección 5.2 se presenta como modelar la información partiendo de los elementos utilizados en base de datos relacionales.

5.1 Selección de herramienta

Para la solución se utiliza Redis como base de datos NoSQL. Se eligió una base de datos clave-valor (KV) para utilizar otro tipo de base de datos a la utilizada para los datos del motor del proceso, y distinta a la utilizada en otras implementaciones en el contexto del proyecto de investigación (grafos con Neo4j). Redis maneja varios tipos de datos [12]. Para este proyecto se eligió string por ser el tipo de dato más simple que se puede asociar con una clave (key) de Redis.
Redis tiene módulos con soporte a JSON [12]. De esta forma se podría simplificar el acceso y almacenamiento de datos estructurados. La intención es agregar complejidad a la hora de modelar los datos organizacionales y por eso no se utilizan estos tipos de módulos.

5.1.1 Manejo de datos: string

SET y GET son los comandos utilizados para almacenar y obtener un string.
El comando SET, en su utilización por defecto, sobreescribe el valor si ya existe la clave. Si no existe crea la clave. Opcionalmente se permite indicar, con un parámetro adicional, que se actualice la clave solamente si existe, o no existe, la clave.
Otro comando utilizado es INCR. El comando incrementa el número almacenado en la clave en 1. Si la clave no existe o contiene un valor de un tipo incorrecto, establezca la clave en el valor de "0" antes de realizar la operación de incremento. INCR también tiene la ventaja de ser atómico, es decir, si dos clientes ejecutan el INCR al mismo tiempo no van a obtener el mismo valor.
Notar que en Redis no hay tipos de datos integer. Lo que hace el comando es analizar el string almacenado en la clave como un entero de base 10 de 64 bits con signo, se incrementa, y luego se vuelve a convertir a string.
Uno de los problemas al utilizar una base de datos NoSQL clave valor es la performance al obtener todas las keys por prefijo o patrón. La opción que sugiere Redis para esto es iterar sobre las claves de Redis que coincidan con algún patrón usando el comando SCAN [12]. El comando es un iterador basado en un cursor. Esto significa que en cada llamada del comando el servidor devuelve un cursor actualizado que el usuario necesita usar como argumento de cursor en la próxima llamada. Una iteración comienza cuando el cursor se establece en “0” y finaliza cuando el cursor devuelto por el servidor es “0”.

5.2 Modelado de datos

El principal desafío al almacenar información “relacional” en base de datos clave-valor es con el modelado de datos. Se tomaron ciertas decisiones, explicadas a continuación, para poder almacenar y obtener la información. Hay que aclarar que solo se explican las decisiones tomadas para este proyecto, no intenta ser una guía general de cómo migrar de un modelo relacional a Redis.

5.2.1 Representando entidades

Es una buena práctica usar namespaces (espacios de nombres) al definir claves en Redis para evitar colisiones de nombres de claves y organizar las claves según el área de la aplicación. En las bases de datos SQL un espacio de nombres se puede representar mediante el esquema de la base de datos. Para explicar la solución consideremos la entidad Teacher. Al momento de definir la clave es necesario incluir la entidad (namespace), el identificador y también el atributo. Se podría generalizar en tres componentes principales para especificar: prefijo, identificador y sufijo. Se utiliza un prefijo para describir la entidad. En este caso, como la entidad es Teacher, el prefijo debe definirse como teacher. Cada fila de la tabla se identifica de forma única a partir de la clave principal idTeacher por lo tanto debe usarse como el identificador de la clave. Luego se debe definir un sufijo para especificar el atributo relacionado.
Advertir que Redis no admite namespacing, se utiliza esta convención de nombre de clave para imitar namespaces.
En general, cualquier tabla de un modelo relacional puede representarse en clave valor de la siguiente manera:

```
$table_name:$primary_key_value:$attribute_name = $value
```

Para la entidad Teacher el esquema clave valor queda definido:

```
teacher:$idteacher:name = $name
teacher:$idteacher:idinstitute = $idinstitute
```

5.2.2 Representando relaciones

En el modelo relacional se usan claves foráneas en las tablas para representar relaciones entre entidades. De manera similar la clave externa se puede usar como un identificador de la clave para representar las relaciones en un esquema de clave valor.
Para explicar la solución consideremos la entidad Application fuertemente relacionada con Program. Existen en la aplicación varias consultas de Application utilizando idProgram. Por este motivo se decidió incluir en la clave de Application también idProgram. Quedando la solución:

```
application:$idappliaction:$idprogram:idstate = $idstate
application:$idapplication:$idprogram:idstudent = $idstudent
application:$idapplication:$idprogram:orden = $orden
application:$idapplication:$idprogram:notified = $notified
application:$idapplication:$idprogram:date = $date
application:$idapplication:$idprogram:approved = $approved
application:$idapplication:$idprogram:amount = $amount
```

Notar que las otras relaciones (Student y State) se manejan como cualquier otro atributo en la clave. Esta decisión depende principalmente del uso de esta relación en la aplicación.

5.2.3 Filtros y orden

Cuando se consulta información en el SQL se pueden filtrar y ordenar los resultados según atributos indicados. Para la solución se decidió, por simplicidad del trabajo que había que realizar, filtrar y ordenar en la lógica de la aplicación.

5.2.4 Secuencias

Para las secuencias se utiliza una clave especial que mantiene la secuencia de la entidad. Para incrementar el valor de la secuencia se utiliza el comando INCR. Las siguientes secuencias son utilizadas en la solución:

- application_seq
- mobility_seq
- program_seq
- validation_seq

5.2.5 Solución entidades

A continuación, se presentan las soluciones utilizadas en las entidades.

```
### Entidad Teacher
teacher:$idteacher:name = $name
teacher:$idteacher:idinstitute = $idinstitute

### Entidad Program
program:$idprogram:name = $name
program:$idprogram:year = $year
program:$idprogram:callnumber = $callnumber
program:$idprogram:amount = $amount
program:$idprogram:date = $date

### Entidad Course
course:$idcourse:name = $name
course:$idcourse:credits = $credits
course:$idcourse:idinstitute = $idinstitute
course:$idcourse:idcareer = $idcareer

### Entidad Application
application:$idappliaction:$idprogram:idstate = $idstate
application:$idapplication:$idprogram:idstudent = $idstudent
application:$idapplication:$idprogram:orden = $orden
application:$idapplication:$idprogram:notified = $notified
application:$idapplication:$idprogram:date = $date
application:$idapplication:$idprogram:approved = $approved
application:$idapplication:$idprogram:amount = $amount

### Entidad Validation
validation:$idvalidation:$idapplication:idcourse = $idcourse

### Entidad Mobility
mobility:$idmobility:idapplication = $idapplication
mobility:$idmobility:startdate = $startdate
mobility:$idmobility:amount = $amount
mobility:$idmobility:date = $date
```

5.3 Transacciones

En Redis las transacciones se utilizan para agrupar un conjunto de comandos en una sola operación atómica [12]. Esto significa que todas las operaciones dentro de una transacción se ejecutan de manera secuencial y en orden, como si fueran una sola operación.
Para comenzar una transacción en Redis, primero se debe usar el comando MULTI, lo que indica que se van a agrupar una serie de comandos en una transacción. Luego, se ejecutan los comandos que se desean incluir en la transacción. Finalmente, se usa el comando EXEC para ejecutar la transacción completa.
Si se produce algún error en cualquiera de los comandos dentro de la transacción, Redis revierte todos los comandos previamente ejecutados y no realiza ninguno de los comandos que quedan pendientes.
Las transacciones en Redis permiten realizar múltiples operaciones como si fueran una sola, lo que garantiza la consistencia de los datos en caso de errores. También es posible ejecutar una transacción de manera atómica, es decir, en su totalidad o no en absoluto, lo que evita situaciones de incoherencia en los datos

6 Ejemplo de Aplicación: Student Mobility

En esta sección se presenta la solución, al problema planteado, aplicada a un caso puntual. En la subsección 6.1 se presenta el caso de estudio y en la subsección 6.2, se detalla la implementación de la solución y su uso.

6.1 Caso de estudio

Para realizar un análisis sobre la solución desarrollada se toma como referencia el caso de estudio de movilidad de estudiantes. [26]
El proceso de negocio "movilidad de estudiantes" es un programa donde estudiantes pueden postularse para estudiar y revalidar materias en universidades del extranjero. Además, como parte del programa se les otorga un monto para que puedan efectuar dichos estudios en el exterior.
Las convocatorias del programa de movilidad se abren cada año en base a un calendario anual definido por la Dirección General de Cooperación y Relaciones Internacionales (GDCIR).
Para cada programa, los estudiantes presentan sus solicitudes, y luego de algunos controles y evaluaciones, se asignan las becas, se firman los contratos y se liberan los pagos correspondientes.

La realidad antes descrita se diseña como un proceso de negocios desarrollado en Flowable, y su modelado se muestra en Figura 6.1.

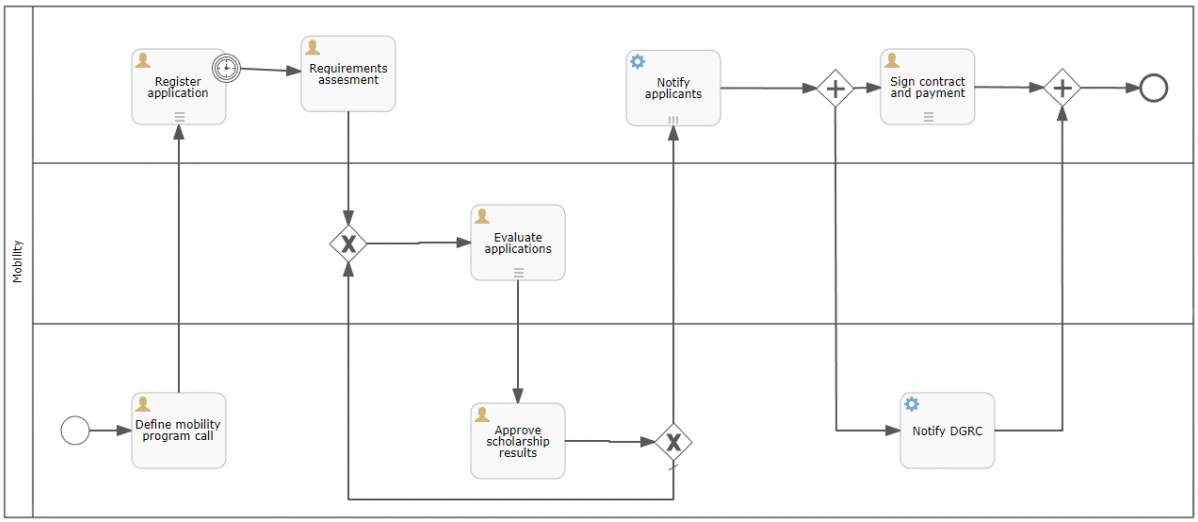


Figura 6.1: Diagrama del proceso movilidad de estudiantes

El proceso comienza cuando la junta escolar define una nueva convocatoria del programa de movilidad. En este momento, un "program" está registrado en la base de datos organizativa.

Dentro de la tarea "Register application", la oficina de registro recibe las solicitudes de los estudiantes (durante 15 días). Para cada uno de ellos, se registra una nueva “application” en la base de datos organizativa en el estado “Initiated” (dentro de la tabla “state”) y se vincula con el correspondiente “program” abierto y "student". Dado que los estudiantes van a realizar cursos en la universidad de destino, pueden seleccionar algunos cursos (de la tabla “course”) de su “career” para validarlos una vez finalizada la movilidad; esta relación se crea dentro de la tabla "validation".

Luego, cada solicitud es descargada y verificada (todas a la vez) por la oficina de registro (tarea "evaluate applications"). Como resultado, las solicitudes se actualizan en la base de datos organizativa cambiando sus estados a "Confirmed" o "Rejected". Las solicitudes rechazadas ya no se consideran dentro del proceso.

Un panel de evaluación (integrado por docentes) realiza el proceso de evaluación (tarea "Evaluate applications") en el que se evalúa y clasifica a los postulantes (ordenados de 1 a n), y se definen potenciales titulares y suplentes en función del número de plazas disponibles para la llamada del programa. Como resultado, las solicitudes se actualizan dentro de la base de datos organizacional, cambiando sus estados a "Holder" o "Substitute", y también agregando el orden asignado dentro de la lista de solicitantes.

Posteriormente la junta escolar evalúa la clasificación de solicitudes (tarea "Approve scholarship results"). Si el informe no es aprobado vuelve al remitente para realizar las modificaciones correspondientes. Una vez que se aprueba el informe, la base de datos de la organización se actualiza para establecer el estado “approved” de las solicitudes del titular y sustituto.

Cada solicitante, también aquellos rechazados durante la evaluación de requisitos, es notificado de los resultados (tarea "Notify applicants"). Esta tarea actualiza el estado “notified” de la solicitud.

Finalmente, se realizan dos tareas en paralelo: los becarios firman sus contratos y reciben su correspondiente pago (tarea "Sign contract and payment"), en la que se actualiza la base de datos organizativa creando una "mobility" para cada solicitud de titular; y los resultados se notifican automáticamente al DGRC (tarea “Notify DGRC”) ya que el dinero excedente se utiliza para futuras llamadas.

La estructura de la base organizacional se compone por datos básicos de la organización y datos de la ejecución del proceso. Las tablas en el modelo relacional teacher, institute, course, state, student y career son parte de los datos básicos de la organización, es decir, no se ingresan como resultado de la ejecución del proceso.

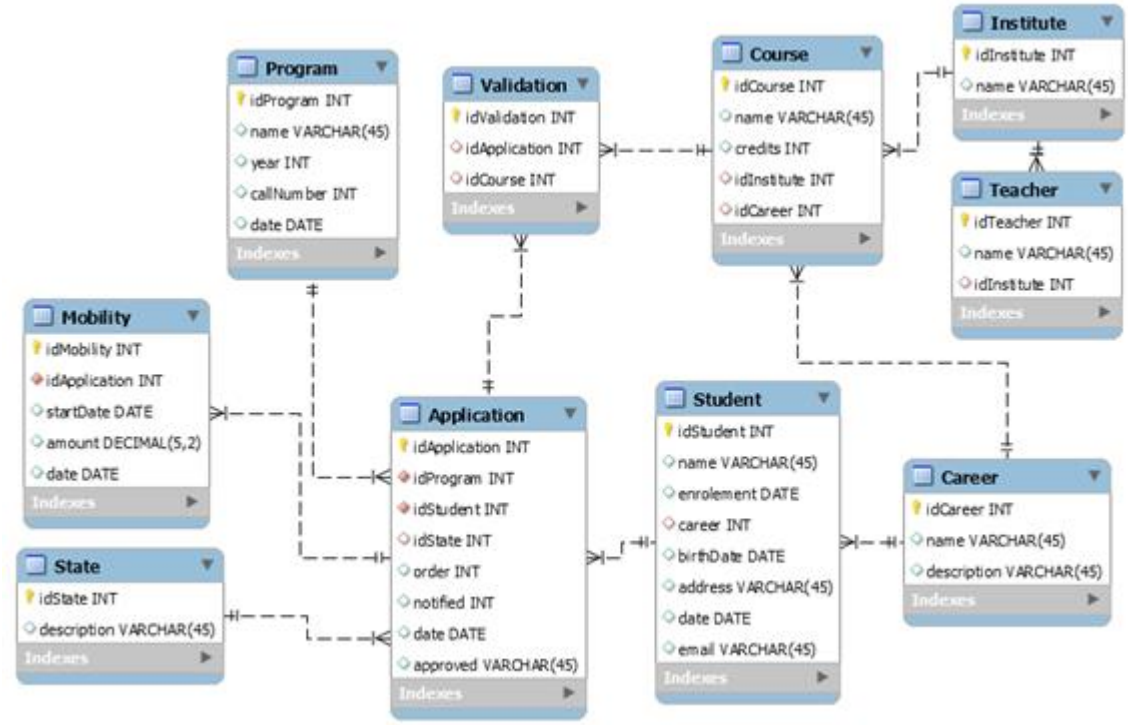


Figura 6.2: Datos organizacionales del proceso movilidad de estudiantes

Hay que destacar que para poder simular la aprobación de solicitudes se incluyó un campo “amount” en “program” y “applications”. La realidad es que estos campos no son definidos por el consejo o los estudiantes, sino que existe para poder simular la realidad de que estudiantes entran dentro del presupuesto debido al monto que van a necesitar para hacer sus cursos. De esta manera se genera la lista de titulares (las primeras *n* solicitudes dentro del presupuesto) y suplentes.

6.2 Implementación del proceso con Flowable, MongoDB y Redis

Para la implementación se desarrolló una aplicación de consola Java. Se utiliza el motor de procesos de Flowable con los datos del motor en MongoDB y los datos de la base organizacional en Redis.

Hay que destacar que no se utiliza Flowable UI para ejecutar el proceso, aplicación proporcionada por el proyecto Flowable, porque la aplicación no admite una configuración para utilizar MongoDB. Es decir, se utiliza otra forma de instanciar el motor de procesos utilizando una librería en fase alpha (experimental). Suponiendo igual que se pueda utilizar, con configuración avanzada o compilando nuevamente Flowable UI, como la librería está en fase experimental pueden ocurrir errores o problemas que no son el foco de esta investigación. Entonces por este motivo se resolvió realizar una aplicación simple de consola en Java para tener control directo sobre el motor de procesos utilizando MongoDB.

Como se explica en el marco teórico, Flowable es un Framework muy flexible y permite utilizarlo de diferentes maneras. Para esta prueba de concepto se incluye el motor de proceso de Flowable como una librería (jar) dentro de una aplicación de ejemplo para correr el proceso. Además del motor de procesos se incluye la librería adicional para la integración de Flowable con MongoDB. También se

incluye librería (console-view) para facilitar el diseño de menús en una aplicación para consolas. Librerías de Slf4j para el manejo del log. Y por último el cliente jedis para poder comunicarse con Redis.

Listing 6.1: Dependencias utilizadas y sus versiones

```
 <dependencies>
    <dependency>
           <groupId>org.flowable</groupId>
           <artifactId>flowable-engine</artifactId>
           <version>6.6.0</version>
    </dependency>
    <dependency>
           <groupId>org.flowable</groupId>
           <artifactId>flowable-process-engine-mongodb</artifactId>
           <version>6.6.0-SNAPSHOT</version>
    </dependency>
    <dependency>
      <groupId>io.bretty</groupId>
      <artifactId>console-view</artifactId>
      <version>3.4</version>
    </dependency>
    <dependency>
           <groupId>org.slf4j</groupId>
           <artifactId>slf4j-api</artifactId>
           <version>1.7.30</version>
    </dependency>
    <dependency>
           <groupId>org.slf4j</groupId>
           <artifactId>slf4j-log4j12</artifactId>
           <version>1.7.30</version>
    </dependency>
    <dependency>
          <groupId>redis.clients</groupId>
          <artifactId>jedis</artifactId>
          <version>4.2.0</version>
    </dependency>
</dependencies>
```

6.2.1 Creación del motor de procesos de Flowable

Lo primero que realiza la aplicación al inicializar es crear el motor de proceso de Flowable. Para esto utiliza MongoDbProcessEngineConfiguration, de la librería flowable-process-engine-mongodb, que nos permite la creación del motor utilizando MongoDB:

```
ProcessEngine processEngine = new MongoDbProcessEngineConfiguration()
        .setServerAddresses(Arrays.asList(new ServerAddress("localhost", 27017)))
        .setDisableIdmEngine(true)
        .setAsyncExecutorActivate(true)
        .buildProcessEngine();
```

Se desactiva (DisableIdmEngine) el componente de administración de identidades (IDM) que desde la versión 6 de Flowable se quitó del módulo de motor de procesos y la lógica se trasladó a módulos separados; además de que no está implementado para MongoDB.
También se activa el ejecutor asincrónico (AsyncExecutorActivate) ya que por defecto Flowable no lo activa. Sin esta configuración habilitada los timers, y tareas asincrónicas, no se ejecutan.
En este punto se crea en MongoDB, si no existe, la base de datos flowable como se explica anteriormente en el uso de la librería flowable-process-engine-mongodb.

Si no ocurren errores (por ejemplo, servicio de MongoDB no levantado) se comienza a dibujar el menú de la aplicación para que el usuario pueda continuar.

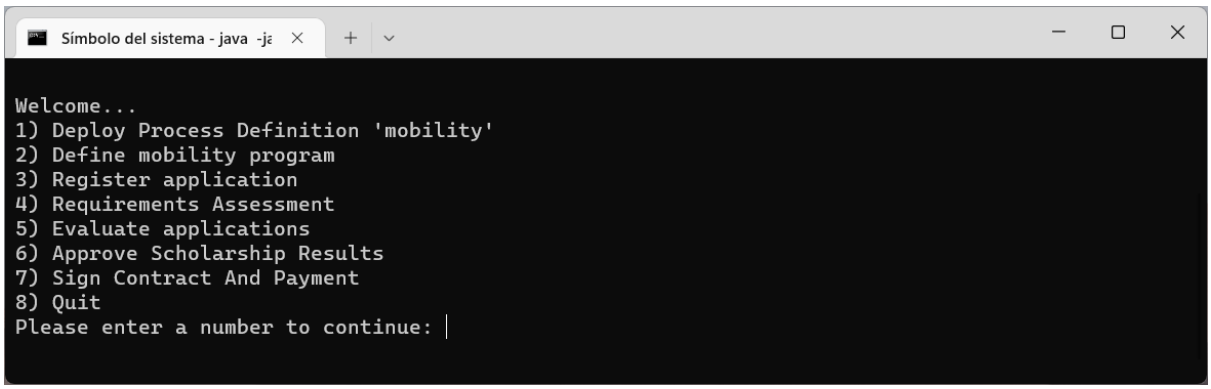


Figura 6.3: Menú de la aplicación

6.2.2 Deploy Process Definition 'mobility'

En la primera opción del menú se realiza el deploy del proceso Student Mobility (si ya no se hizo antes).

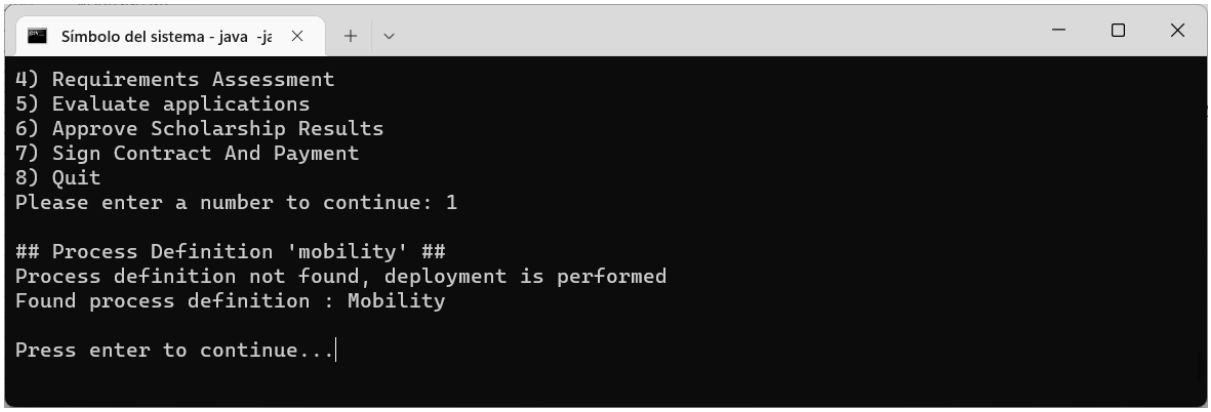


Figura 6.4: Opción 1 del menú: Deploy del proceso

Para realizar el deploy utiliza RepositoryService que se incluye en el motor de Flowable. La definición del proceso se incluye en el directorio resources dentro del JAR.

```
RepositoryService repositoryService = processEngine.getRepositoryService();

ProcessDefinition processDefinition =
repositoryService.createProcessDefinitionQuery().processDefinitionKey("mobility").latestV
ersion().singleResult();

if (processDefinition == null) {
        System.out.println("Process definition not found, deployment is performed");

        Deployment deployment = repositoryService.createDeployment()

        .addClasspathResource("resources/MobilityBPMS_Mobility.bpmn20.xml").deploy();

        processDefinition = repositoryService.createProcessDefinitionQuery()
                        .deploymentId(deployment.getId()).singleResult();
}
```

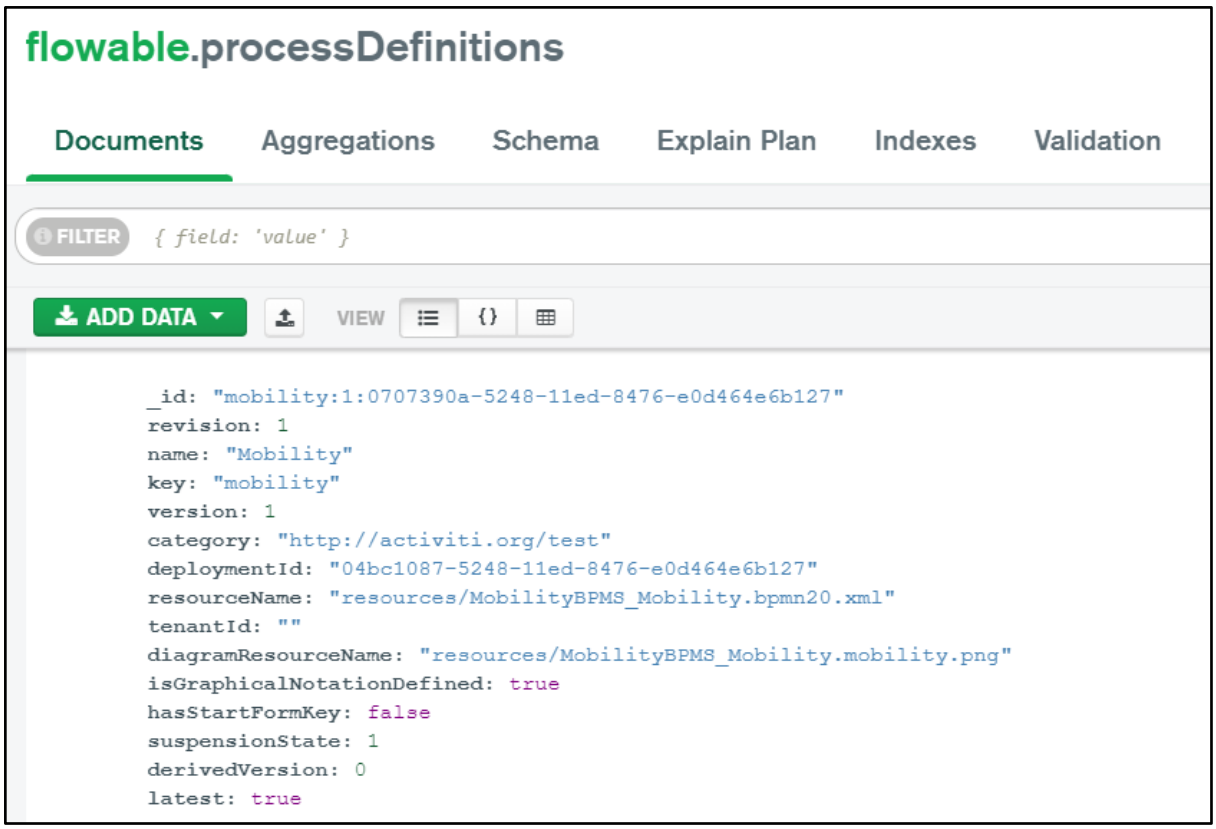


Figura 6.5: Consulta colección processDefinitions en MongoDB Compass

6.2.3 Define mobility program

En este punto del menú se inicia una instancia del proceso “mobility” y se crea la primera tarea “Define mobility program call”. Luego a partir del id de proceso retornado se solicita la tarea. Para esto se utiliza RuntimeService y TaskService de la API del motor de procesos.

```
RuntimeService runtimeService = processEngine.getRuntimeService();

ProcessInstance processInstance = runtimeService.startProcessInstanceByKey("mobility");

this.println("Proceso creado, instanceId: " + processInstance.getProcessInstanceId());

TaskService taskService = processEngine.getTaskService();

List<Task> tasks =
taskService.createTaskQuery().taskCandidateGroup("school_board").taskDefinitionKey("defin
e_mobility").processInstanceId(processInstance.getProcessInstanceId()).list();

Task task = tasks.get(0);
```

Al momento en que se crea la tarea se ejecuta el controlador LoadTeachers que se encarga de obtener los Teachers de la base KV de Redis y dejarlos en la variable de proceso teacher_list.

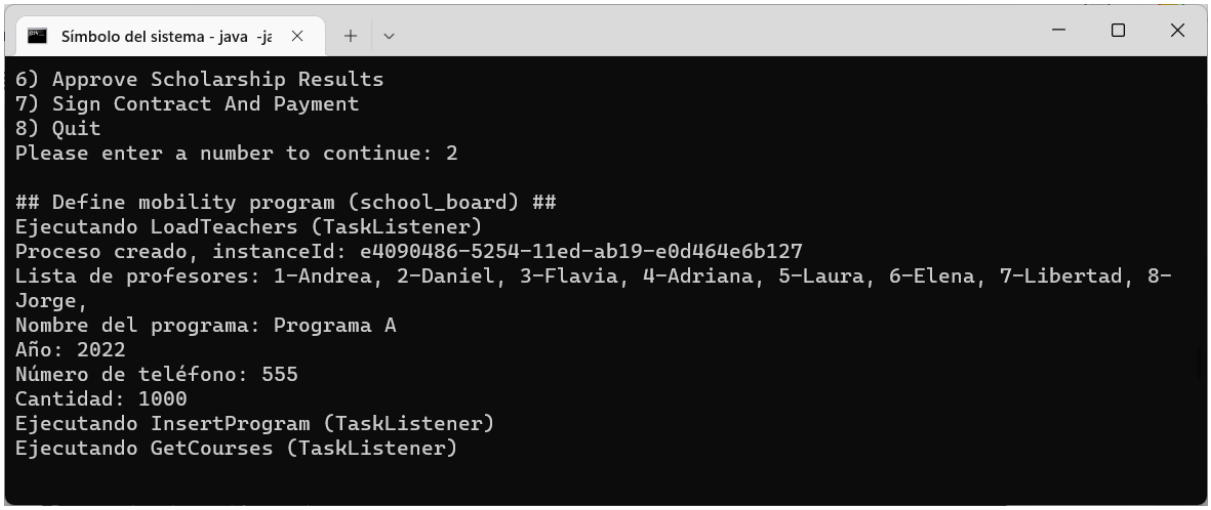


Figura 6.6: Ejecución opción 2 del menú

En la aplicación se visualiza la lista de Teachers y se solicita al usuario ingresar la información del Program que es almacenada en variables del proceso.
Luego de completada la tarea se ejecutan los siguientes listeners:

- InsertProgram. Es el encargado de crear, a partir de las variables, el Program en la base KV de Redis.
- GetCourses. Se ejecuta al crear la siguiente tarea. Obtiene los Courses de la base KV y los almacena en la variable courses_list para ser utilizado más adelante en el proceso.

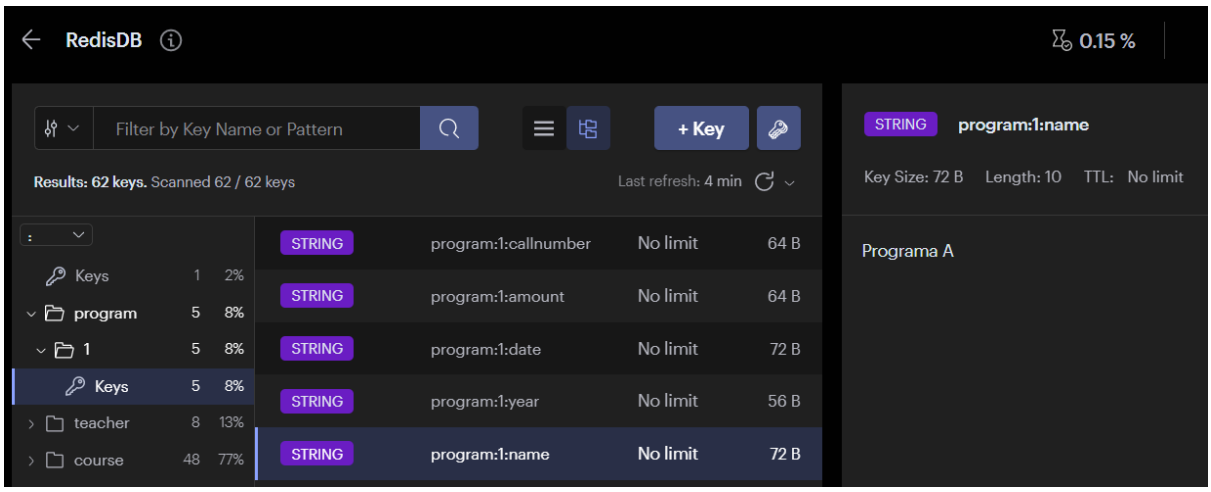


Figura 6.7: Program ingresado en la base KV de Redis

La clave program_seq mantiene el contador de program como se muestra en la siguiente figura.

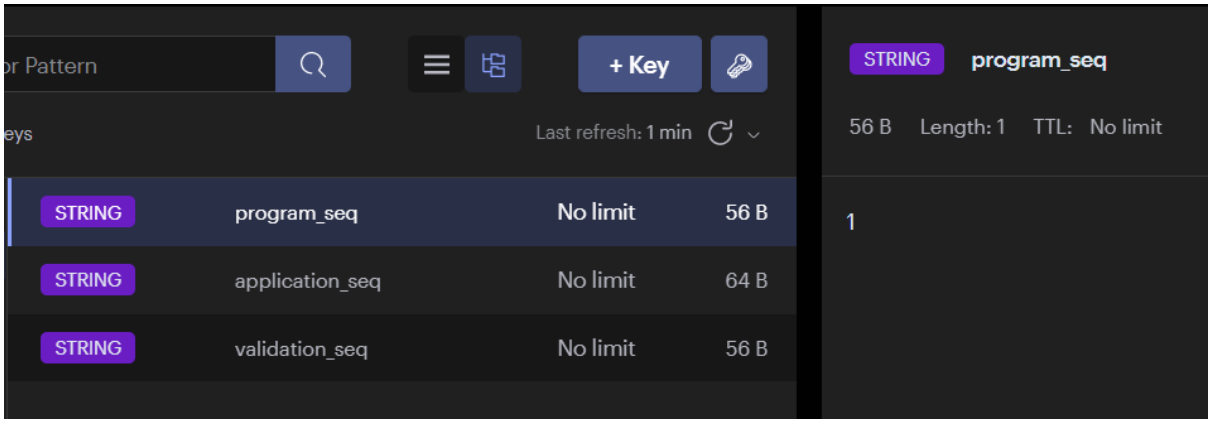


Figura 6.8: Secuencia program_seq

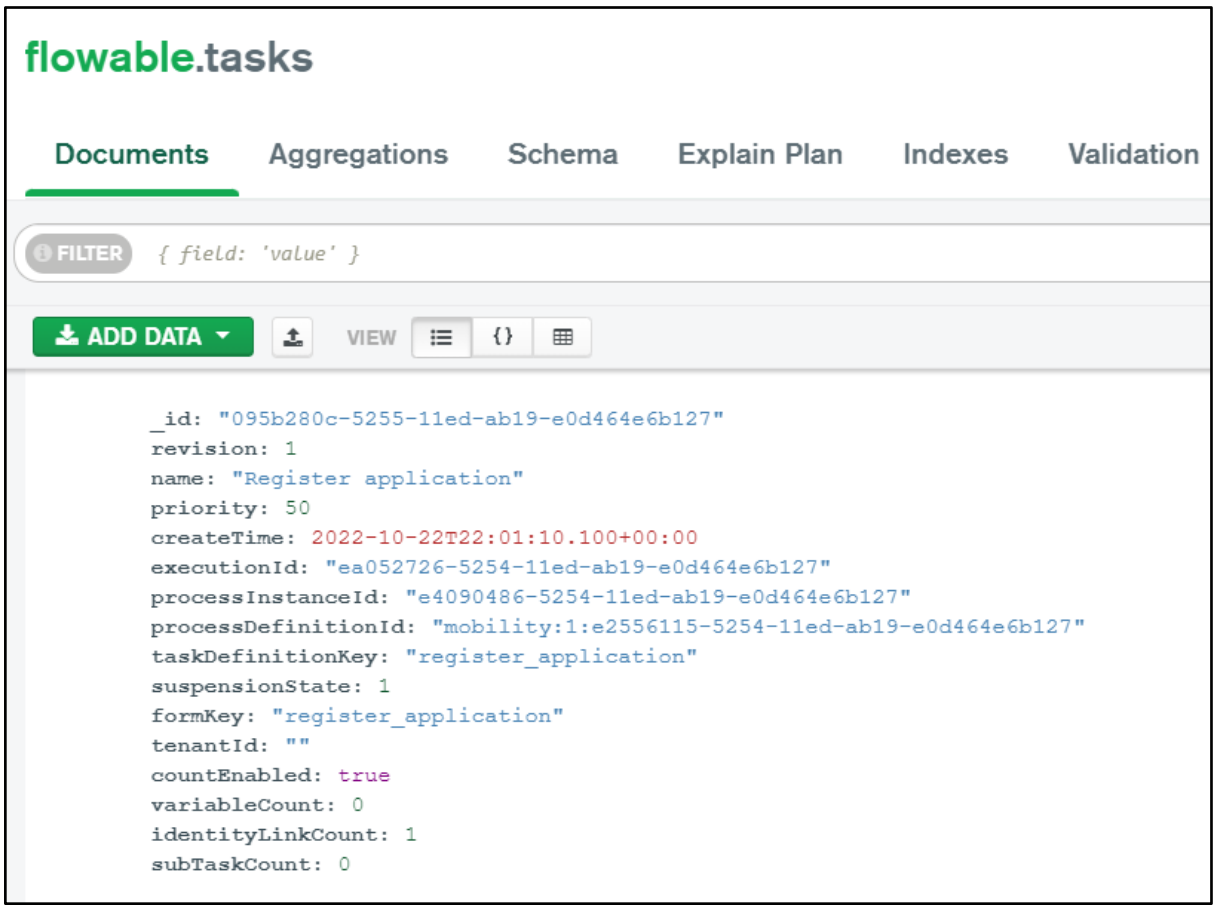


Figura 6.9: Colección Tasks de MongoDB

6.2.4 Register application

Al ingresar en este punto del menú se le presentan al usuario todas las tareas “Register application” que existan para todos los procesos iniciados.

Cuando el usuario selecciona la tarea con la que quiere trabajar tiene la posibilidad de registrar una nueva application. Es una tarea en loop con un timer de finalización. Mientras estemos en el tiempo estipulado vamos a obtener siempre la misma tarea para poder registrar una nueva application.

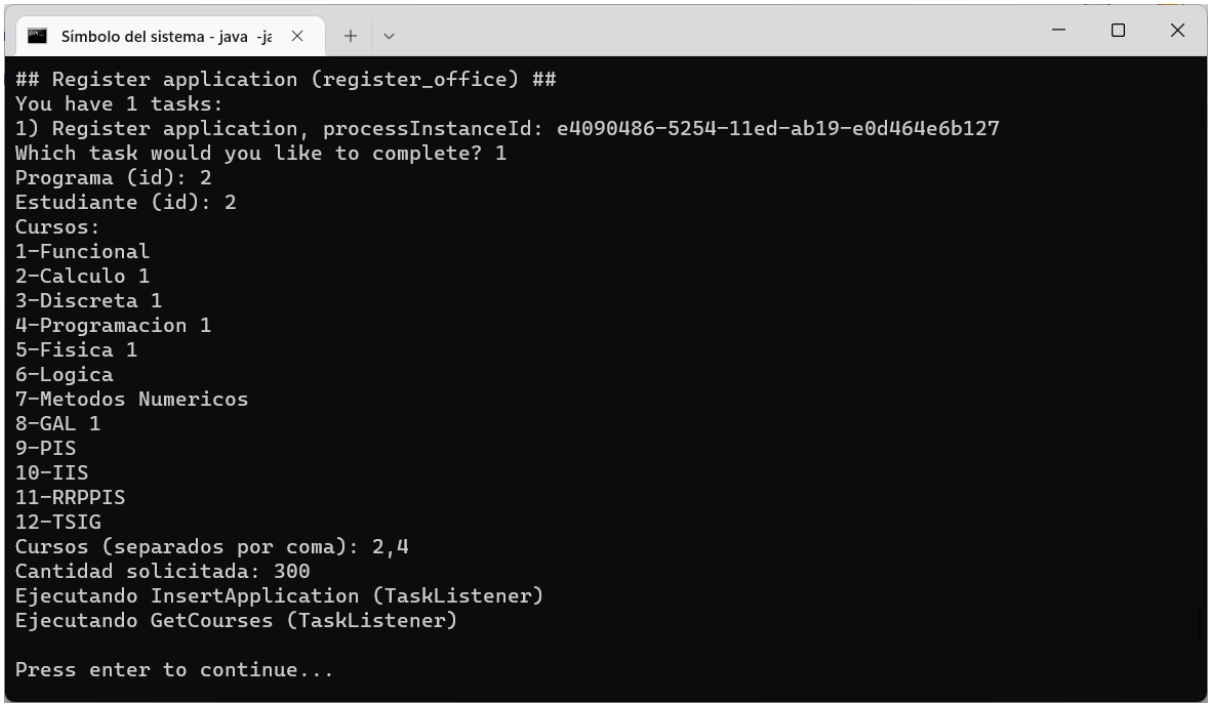


Figura 6.10: Ejecución opción 3 del menú

El usuario tiene que ingresar el id de estudiante y la lista (separadas por coma) de cursos relacionados. También ingresa la cantidad solicitada. Toda esta información queda registrada como variables de proceso.
Luego de completada la tarea se ejecutan los siguientes listeners:

- InsertApplication. Encargado de crear, a partir de las variables, la Application en la base KV de Redis. Además, crea una Validation por cada curso que el usuario seleccionó.
- GetCourses. Al volver a crear otra tarea “Register application” se vuelve a ejecutar el listener.

En la siguiente Figura 6.11 se pueden visualizar las applications ingresadas en este ejemplo. Se filtran todas las applications del programa 1.

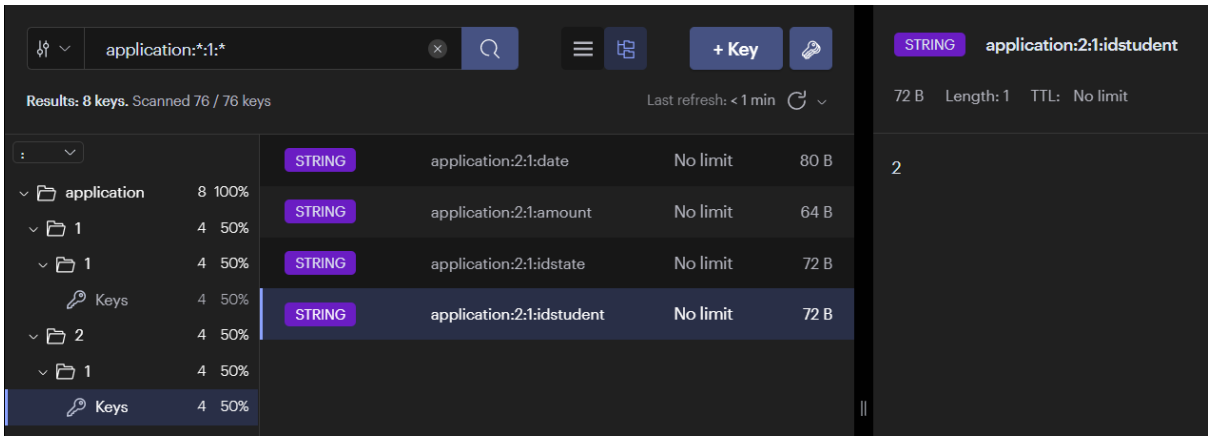


Figura 6.11: Applications del program con id 1

En la Figura 6.12 se muestra la tarea que se encuentra actualmente en ejecución.

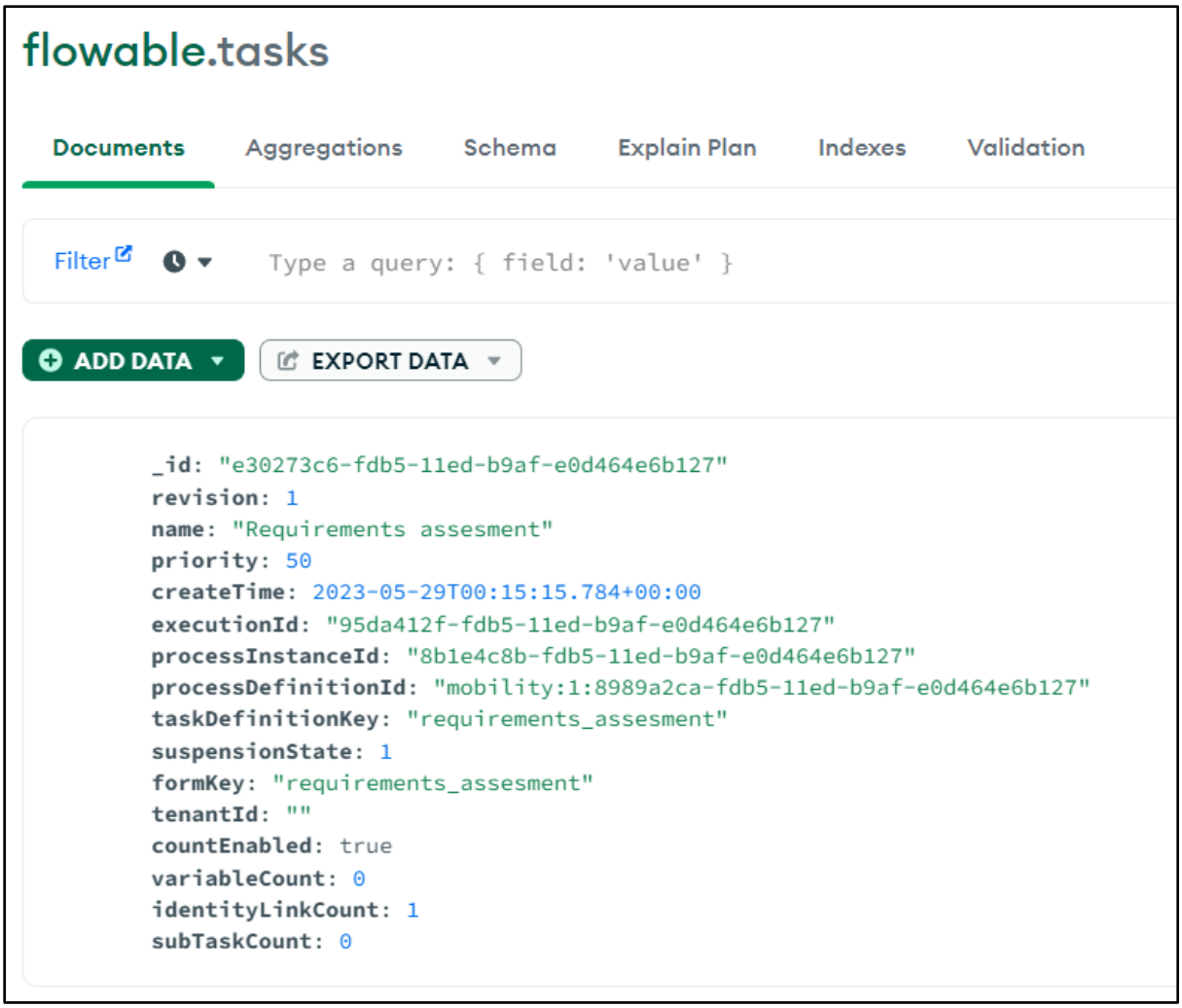


Figura 6.12: Actualización de la colección task

En la siguiente imagen se presenta colección con las variables definidas hasta el momento. Se hace uso de la visualización en tablas que tiene MongoDB Compass.

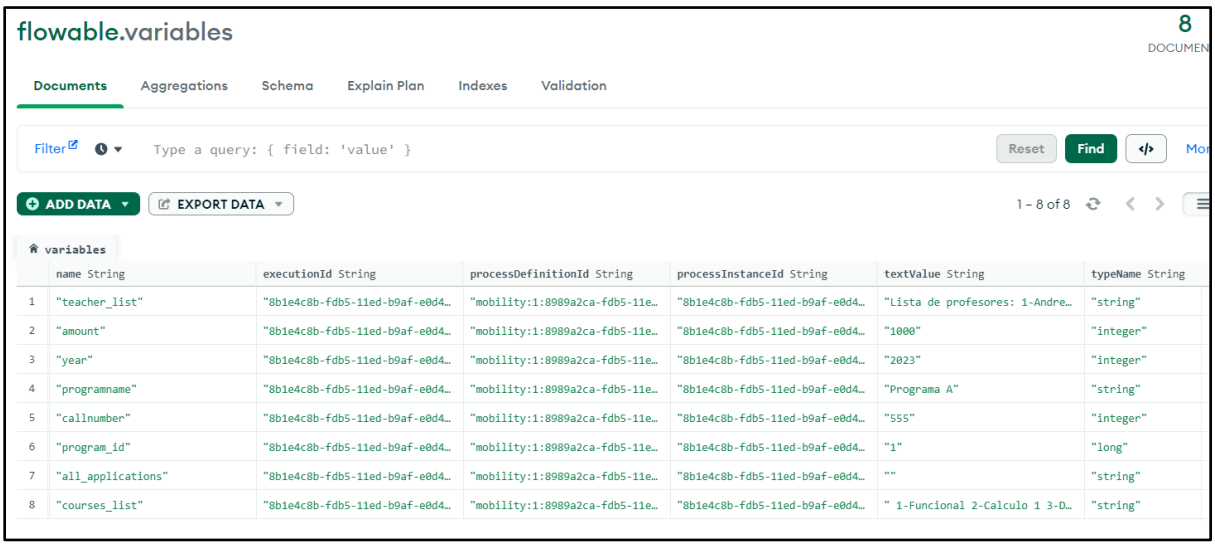


Figura 6.13: Vista en tabla de la colección variables

6.2.5 Requirements Assessment

Luego de ejecutado el timer de la tarea anterior el proceso continúa en la tarea Requirements Assessment. Desde este punto del menú el usuario puede, luego de seleccionar la tarea con que desea trabajar, visualizar la lista de solicitudes y aprobar por id la que corresponda.
La lista de aplicaciones fue cargada previamente en la variable all_applications por el listener ListApplications al momento de asignar la tarea.

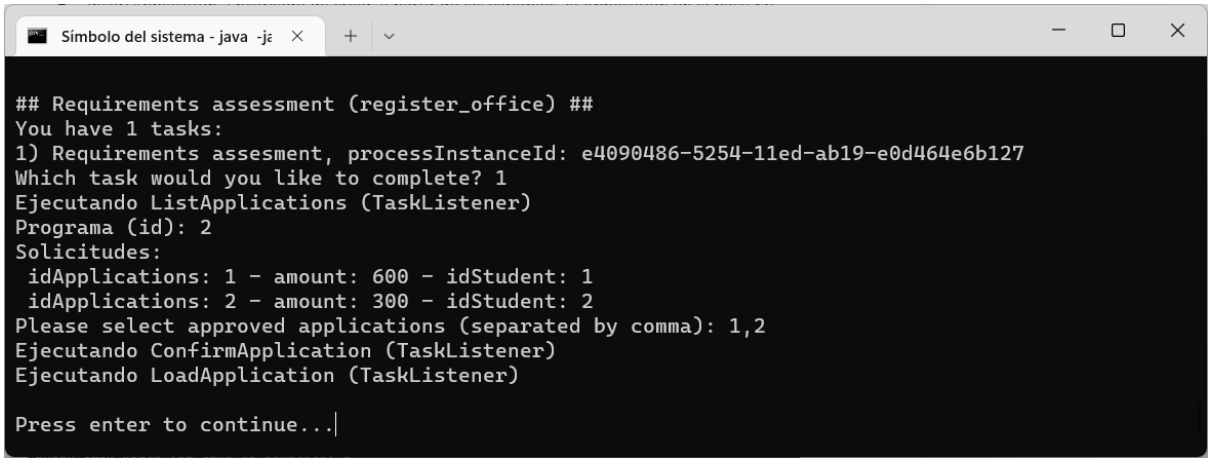


Figura 6.14: Ejecución opción 4 del menú

Luego de completada la tarea se ejecuta el siguiente listeners ConfirmApplication, encargado de actualizar el estado de cada solicitud según corresponda: aprobada o rechazada.
También completa la variable apps_list con todas las solicitudes aprobadas. Variable utilizada por la siguiente tarea, multi-instancia, para evaluar cada solicitud.

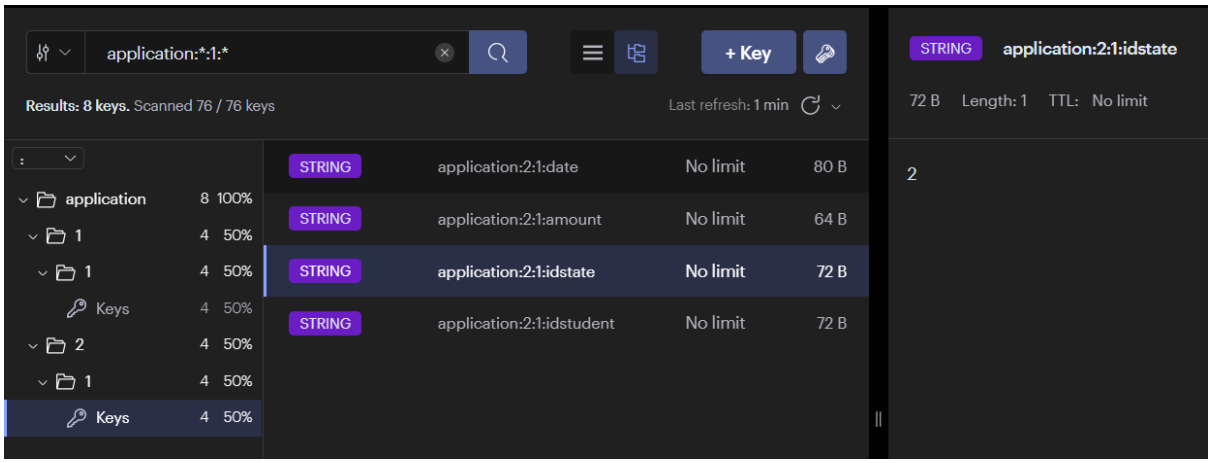


Figura 6.15: Application 2 con estado confirmado (2)

En la Figura 6.16 se muestra la variable app_list. Hay que destacar que el valor de la variable se incluye directamente en esta colección y no como referencia a la colección byteArrays. Esto es uno de los ajustes que se hicieron en la librearía (ver Anexo B) ya que no funcionaba la referencia.

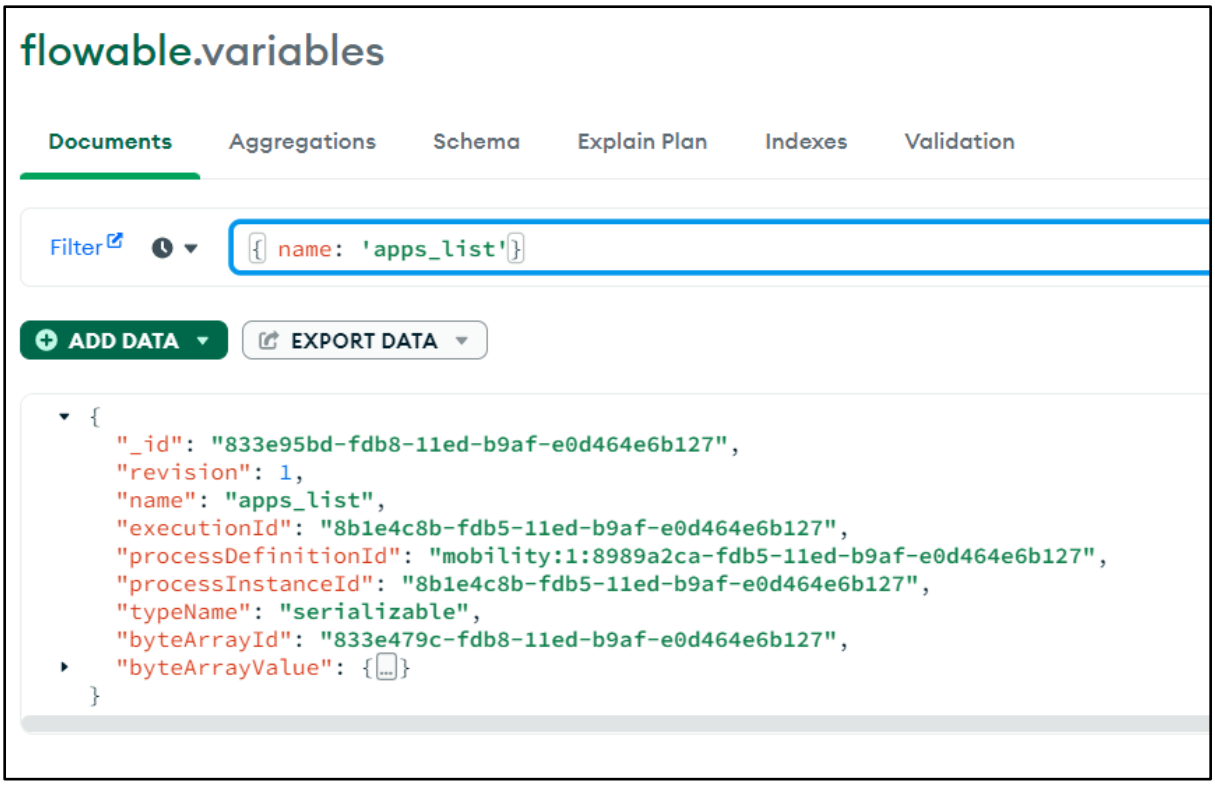


Figura 6.16: app_list en colección variables

6.2.6 Evaluate applications

Desde este menú el usuario puede acceder a la tarea multi-instancia para evaluar cada solicitud aprobada. Al crear cada instancia se ejecuta el listener LoadApplication encargado de cargar más información para cada solicitud. Luego para cada solicitud el usuario debe indicar el orden.

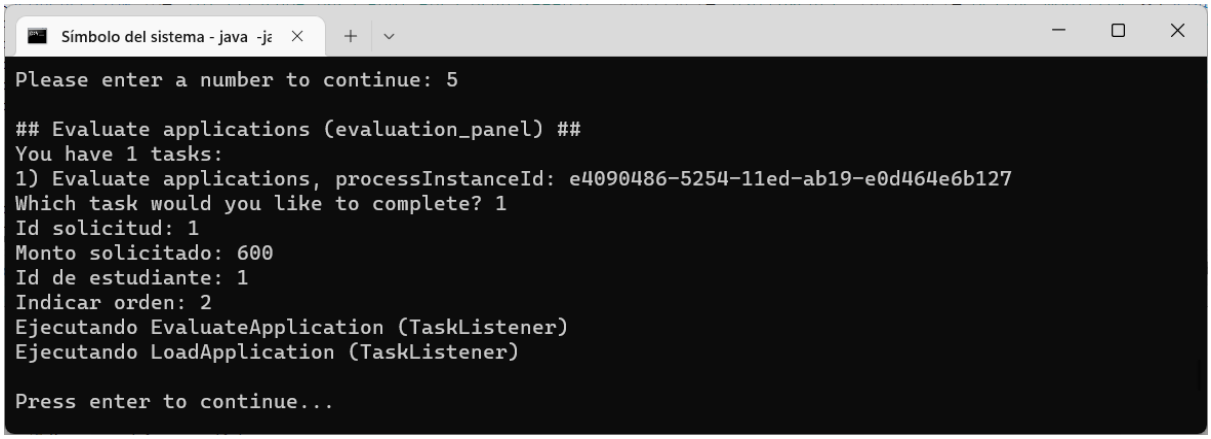


Figura 6.17: Ejecución opción 5 del menú (solicitud 1)

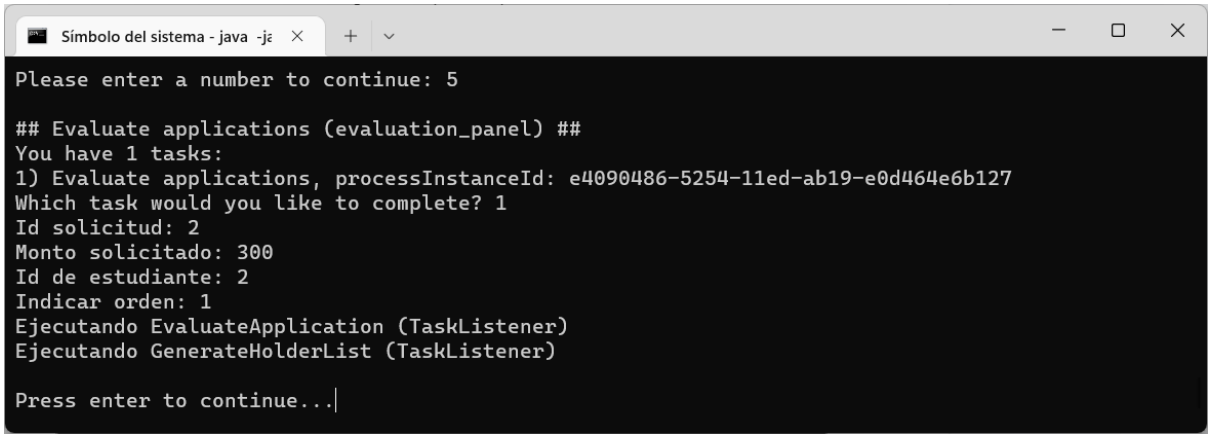


Figura 6.18: Ejecución opción 5 del menú (solicitud 2)

Completa cada solicitud se ejecuta el listener EvaluateApplication encargado de persistir, en la base Redis, el orden indicado por el usuario.
Luego de completadas todas las solicitudes el proceso continúa en la siguiente tarea Approve Scholarship Results.

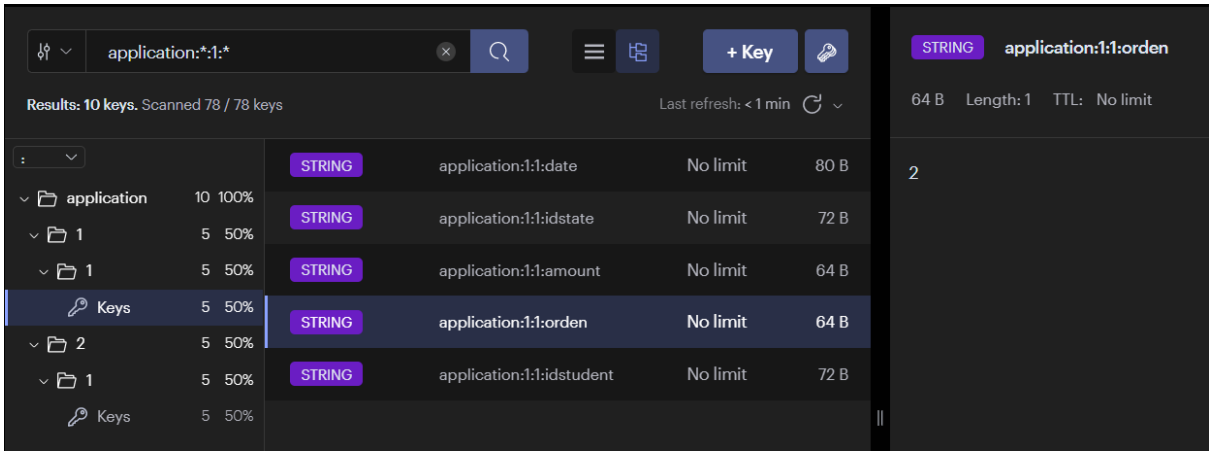


Figura 6.19: Se agrega el campo orden en la application

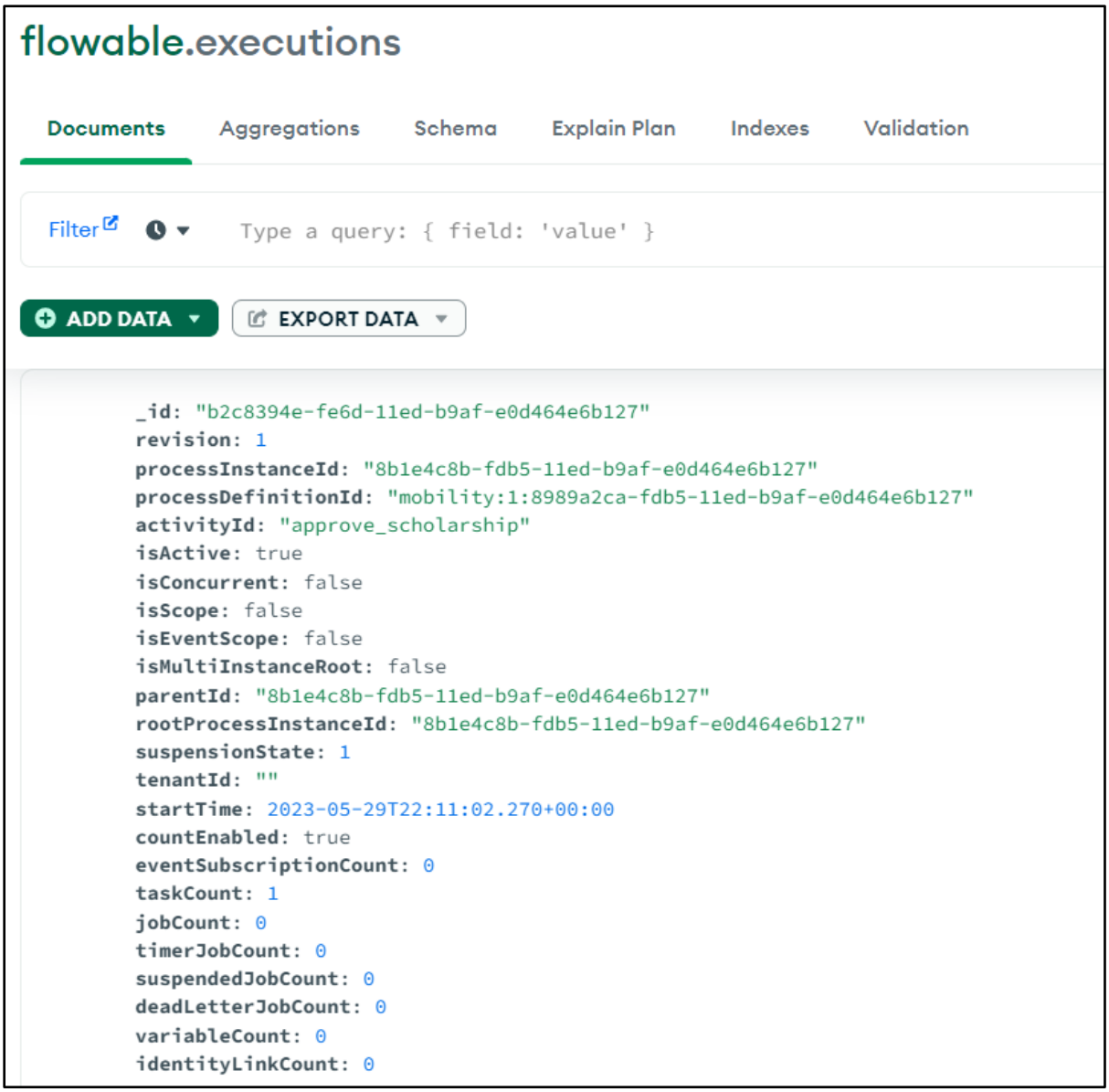


Figura 6.20: colección executions con el estado actual de la instancia del proceso

6.2.7 Approve Scholarship Results

Desde este punto del menú se accede a la tarea en donde el usuario puede aprobar el orden definido previamente. En el listener GenerateHolderList, ejecutado al momento de crear la tarea, se forma una lista de titulares y suplentes según el monto disponible.

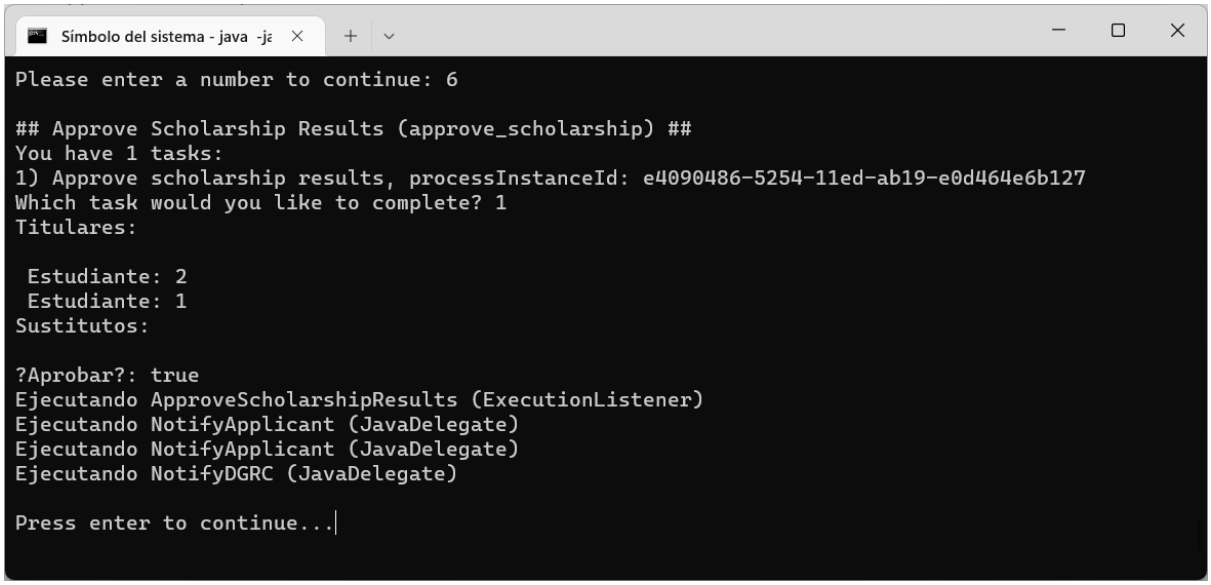


Figura 6.21: Ejecución opción 6 del menú

Si la solicitud fue aprobada continua el flujo hacia la tarea (de servicio) en donde se notifica a cada estudiante. También en la flecha que va hacía Notify Applicants se ejecuta listener que persiste approved en true.

En las siguientes Figura 6.22 y Figura 6.23 se visualiza el cambio de estados en una de las applications aprobadas y notificadas.

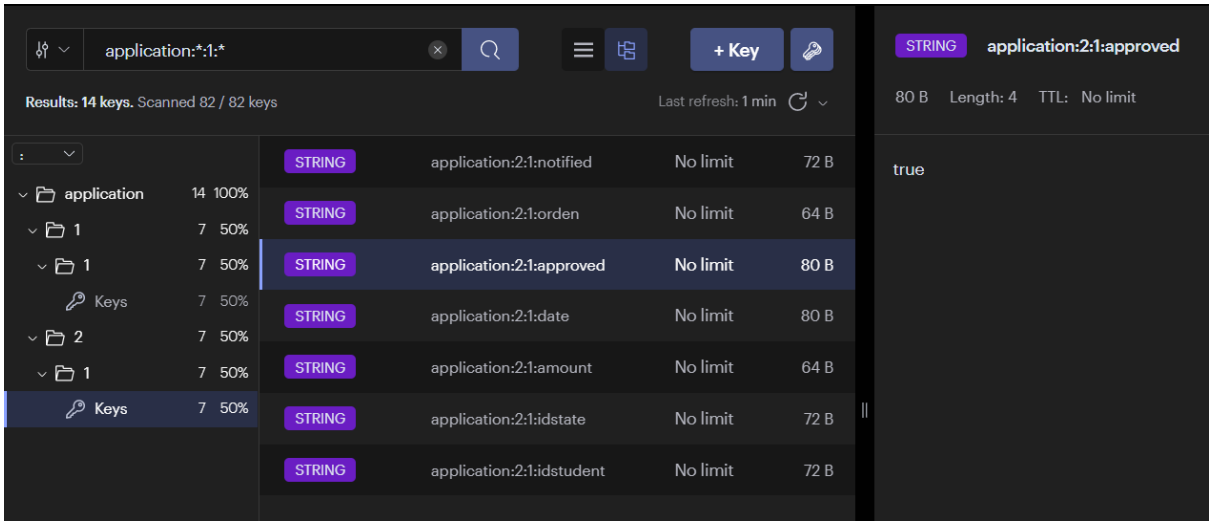


Figura 6.22: se marca como aprobada la application 2

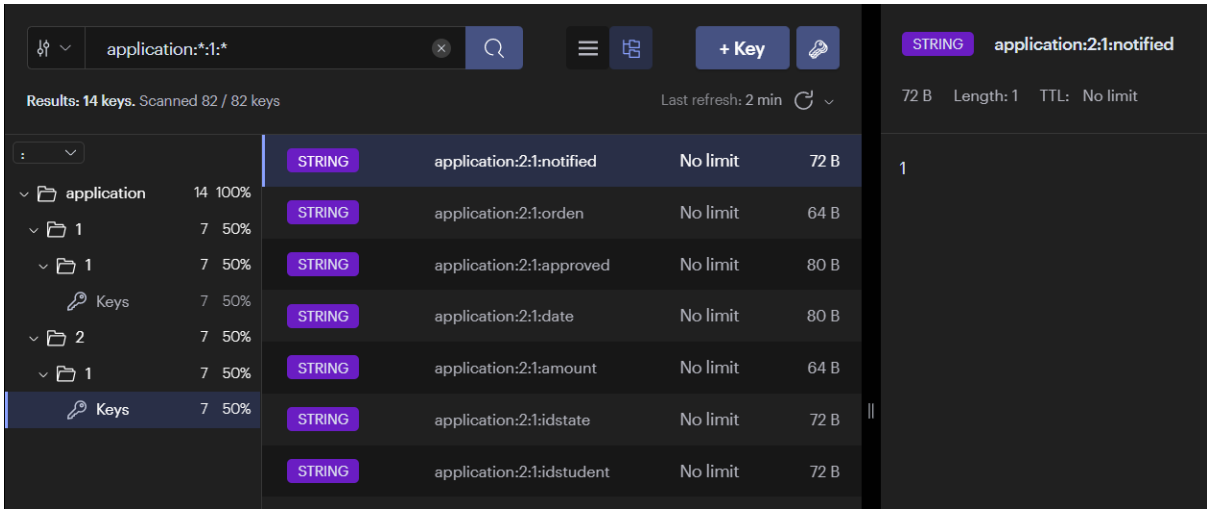
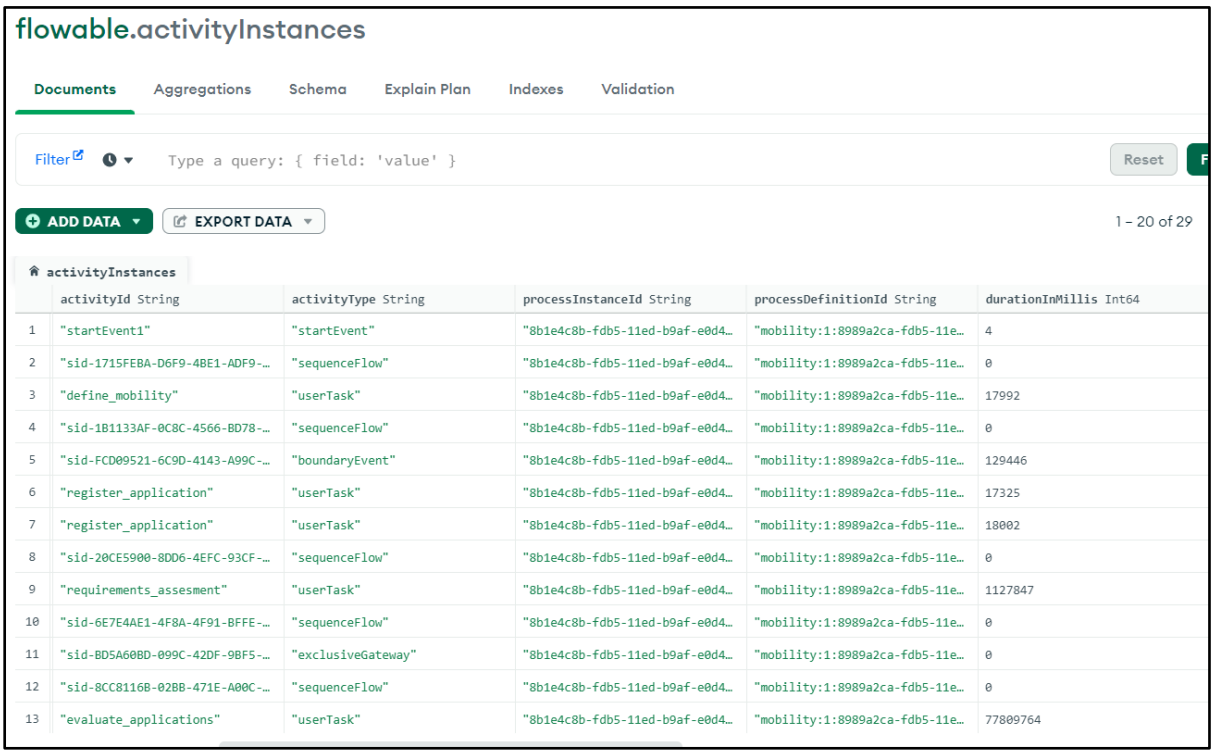


Figura 6.23: se marca como notificada la application 2

En la Figura 6.24 se presenta más información del proceso, colección activityInstances, en donde se muestra una vista parcial del estado actual de cada actividad ocurrida en el proceso.



	activityId String	activityType String	processInstanceId String	processDefinitionId String	durationInMillis Int64
1	"startEvent1"	"startEvent"	"8b1e4c8b-fdb5-11ed-b9af-e0d4…	"mobility:1:8989a2ca-fdb5-11e…	4
2	"sid-1715FEBA-D6F9-4BE1-ADF9-…	"sequenceFlow"	"8b1e4c8b-fdb5-11ed-b9af-e0d4…	"mobility:1:8989a2ca-fdb5-11e…	0
3	"define_mobility"	"userTask"	"8b1e4c8b-fdb5-11ed-b9af-e0d4…	"mobility:1:8989a2ca-fdb5-11e…	17992
4	"sid-1B1133AF-0C8C-4566-BD78-…	"sequenceFlow"	"8b1e4c8b-fdb5-11ed-b9af-e0d4…	"mobility:1:8989a2ca-fdb5-11e…	0
5	"sid-FCD09521-6C9D-4143-A99C-…	"boundaryEvent"	"8b1e4c8b-fdb5-11ed-b9af-e0d4…	"mobility:1:8989a2ca-fdb5-11e…	129446
6	"register_application"	"userTask"	"8b1e4c8b-fdb5-11ed-b9af-e0d4…	"mobility:1:8989a2ca-fdb5-11e…	17325
7	"register_application"	"userTask"	"8b1e4c8b-fdb5-11ed-b9af-e0d4…	"mobility:1:8989a2ca-fdb5-11e…	18002
8	"sid-20CE5900-8DD6-4EFC-93CF-…	"sequenceFlow"	"8b1e4c8b-fdb5-11ed-b9af-e0d4…	"mobility:1:8989a2ca-fdb5-11e…	0
9	"requirements_assesment"	"userTask"	"8b1e4c8b-fdb5-11ed-b9af-e0d4…	"mobility:1:8989a2ca-fdb5-11e…	1127847
10	"sid-6E7E4AE1-4F8A-4F91-BFFE-…	"sequenceFlow"	"8b1e4c8b-fdb5-11ed-b9af-e0d4…	"mobility:1:8989a2ca-fdb5-11e…	0
11	"sid-BD5A60BD-099C-42DF-9BF5-…	"exclusiveGateway"	"8b1e4c8b-fdb5-11ed-b9af-e0d4…	"mobility:1:8989a2ca-fdb5-11e…	0
12	"sid-8CC8116B-02BB-471E-A00C-…	"sequenceFlow"	"8b1e4c8b-fdb5-11ed-b9af-e0d4…	"mobility:1:8989a2ca-fdb5-11e…	0
13	"evaluate_applications"	"userTask"	"8b1e4c8b-fdb5-11ed-b9af-e0d4…	"mobility:1:8989a2ca-fdb5-11e…	77809764

Figura 6.24: colección activityInstances

6.2.8 Sign Contract And Payment

Para cada solicitud aprobada, desde este punto del menú, el usuario puede completar la tarea Sign Contract And Payment.

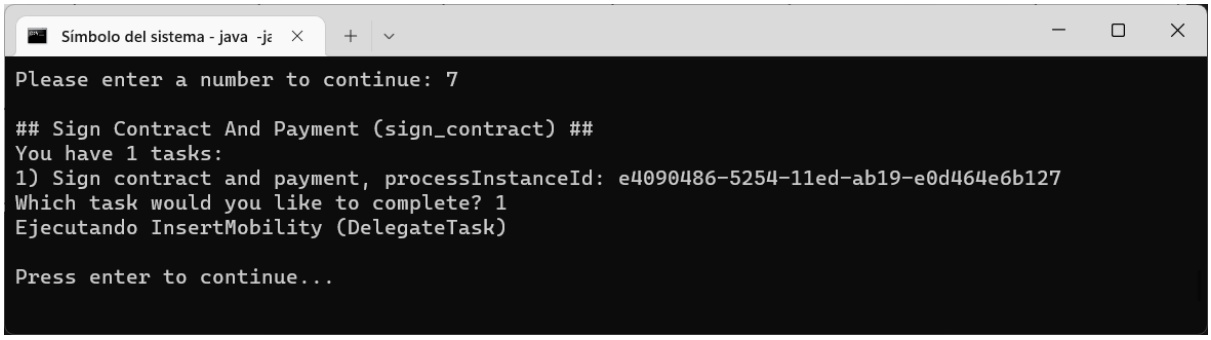


Figura 6.25: Ejecución opción 7 del menú (solicitud 1)

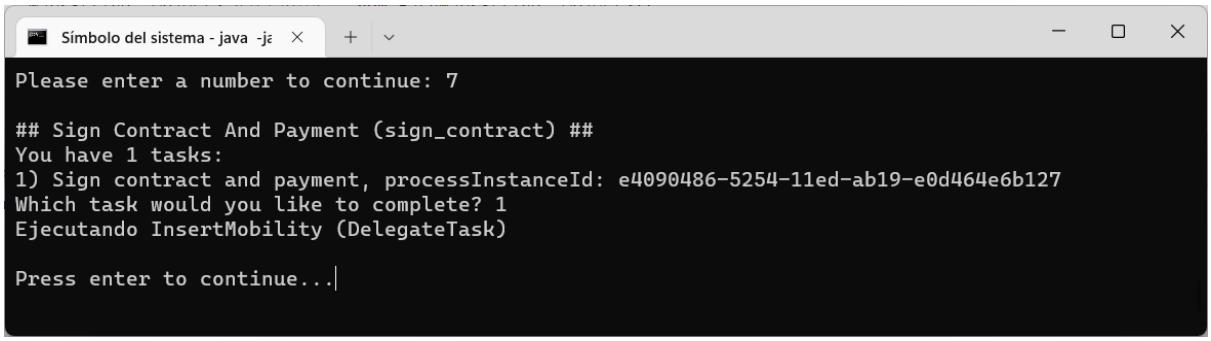


Figura 6.26: Ejecución opción 7 del menú (solicitud 2)

Luego de ejecutada cada tarea se persiste en la base organizacional una nueva mobility. Para el caso presentado en el ejemplo se pueden visualizar en Redis las siguientes claves.

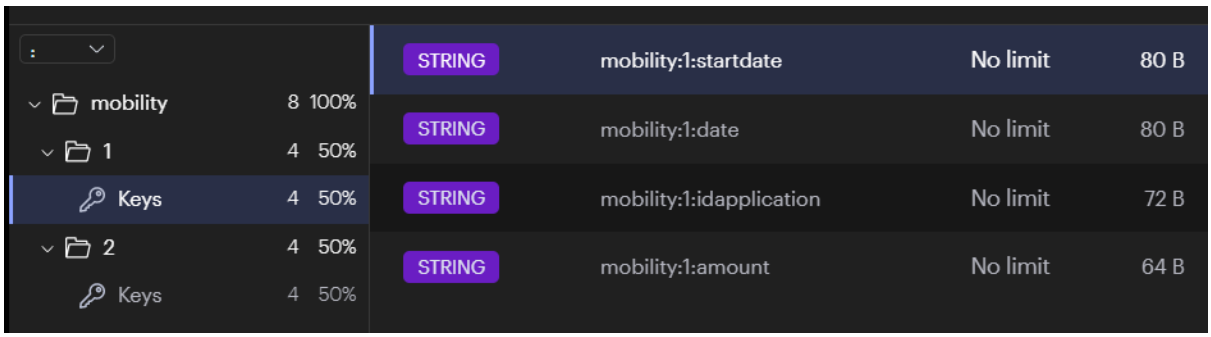


Figura 6.27: Mobility (1) en Redis KV

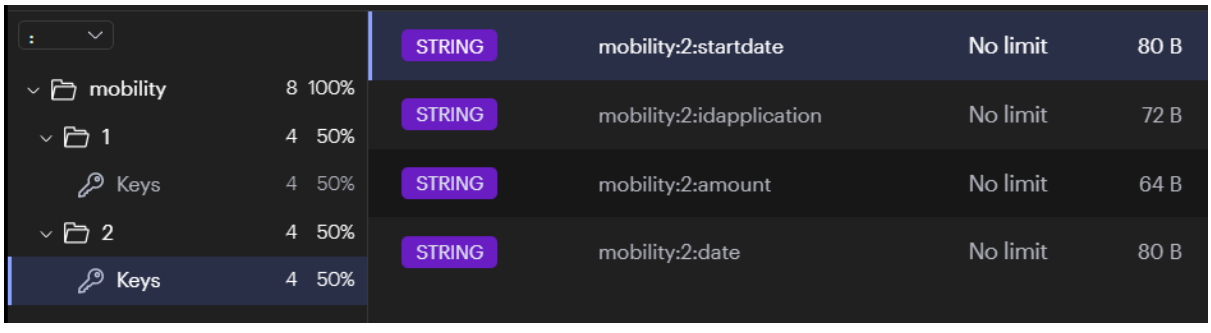


Figura 6.28: Mobility (2) en Redis KV

Como la herramienta visual RedisInsight-v2 no tiene una forma para visualizar más de una clave a la vez, se utiliza en la Figura 6.29 un script para mostrar los valores de cada entrada.

```
alvaro@Asus:~$ ./redis_print.sh 'mobility:*'
mobility:2:amount => 600
mobility:1:amount => 300
mobility:2:date => 29/05/2023
mobility:1:idapplication => 2
mobility:1:date => 29/05/2023
mobility:2:idapplication => 1
mobility:1:startdate => 28/05/2023
mobility:2:startdate => 28/05/2023
```

Figura 6.29: valores de Mobility 1 y 2

Existía un solo proceso en ejecución. Al consultar nuevamente en MongoDB se visualizan las colecciones task, variables, executions vacías. Se puede visualizar en la colección historicProcessInstances el histórico de la ejecución.

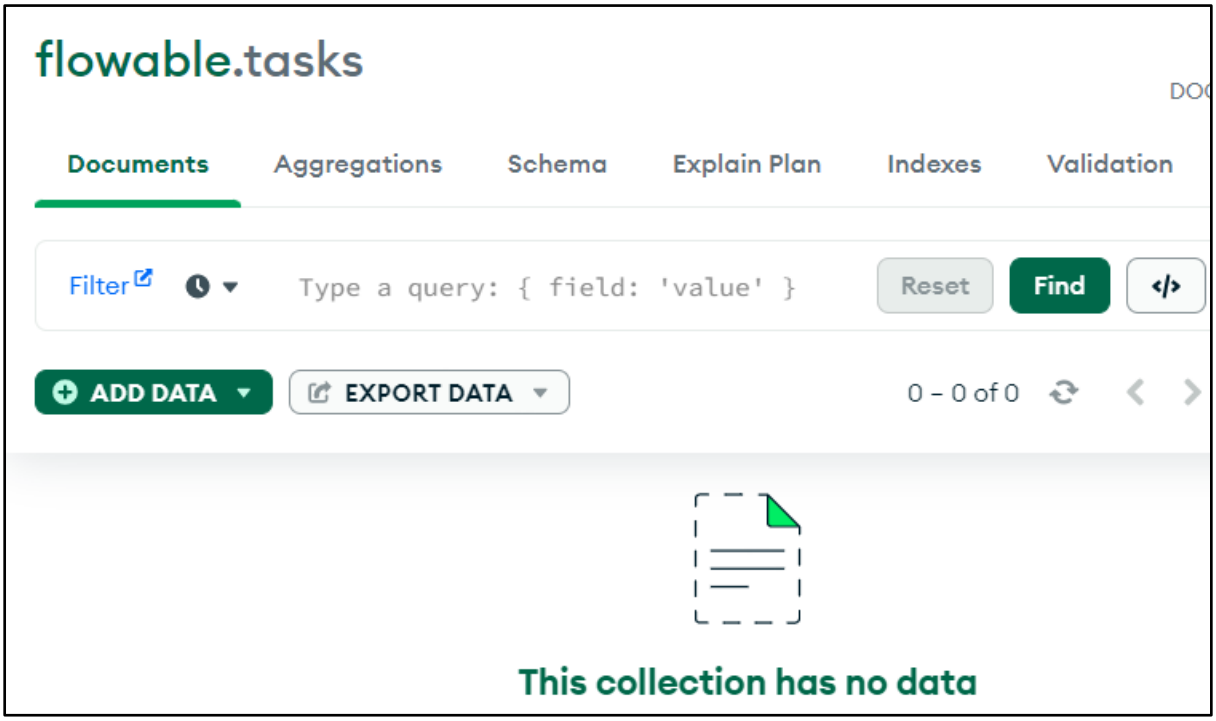


Figura 6.30: colección tasks vacía

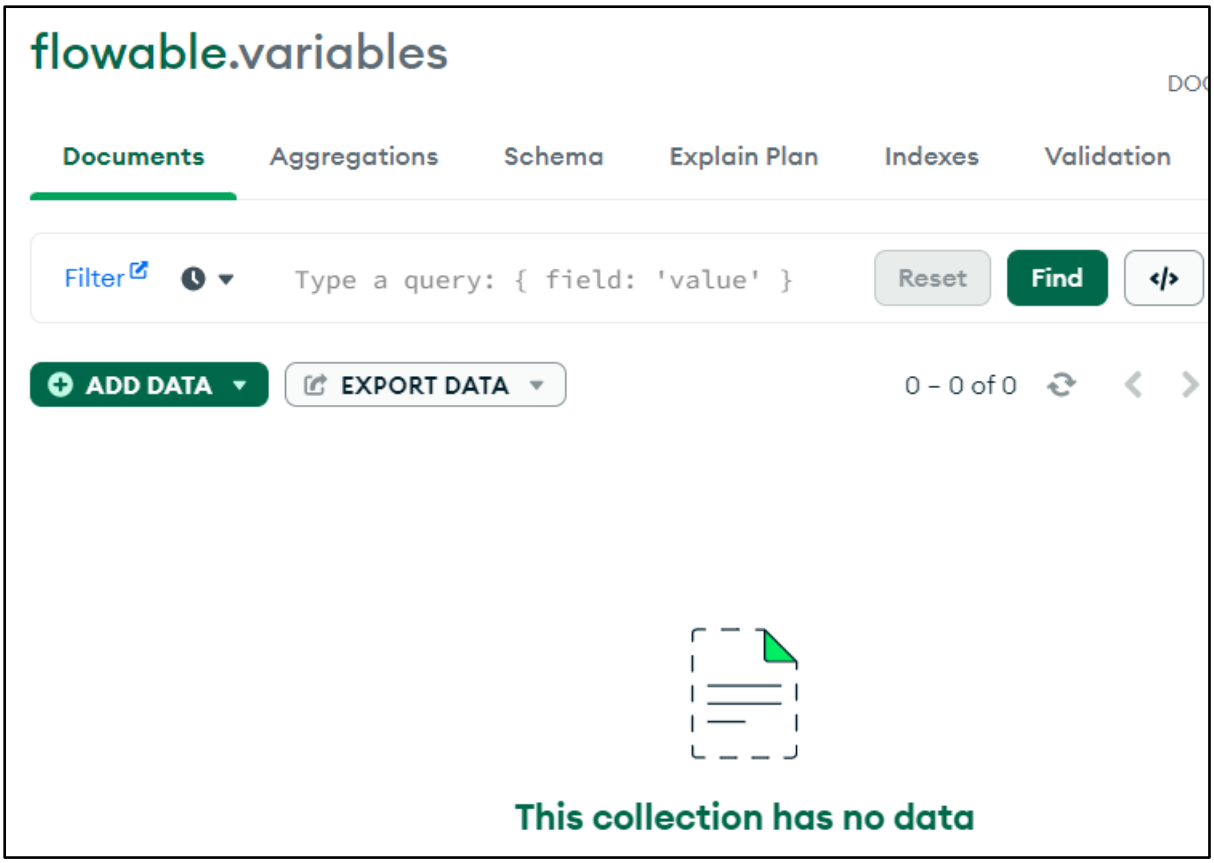


Figura 6.31: colección variables vacía

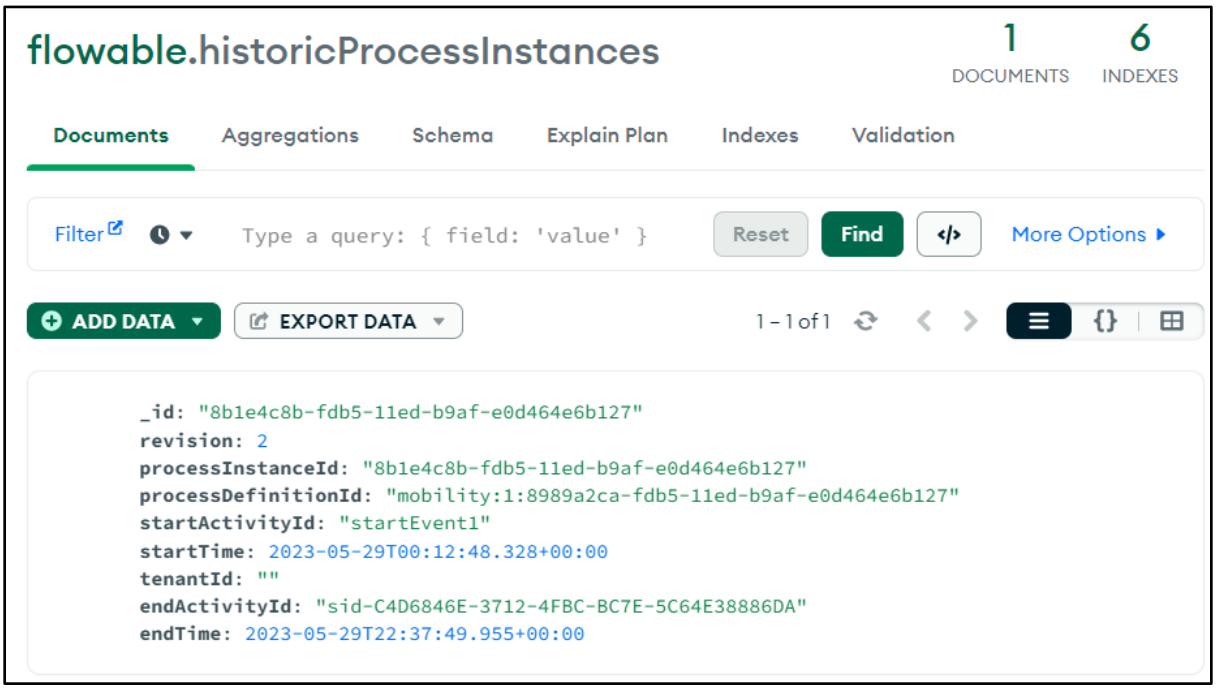


Figura 6.32: colección historicProcessInstances

7 Conclusiones y Trabajo futuro

El objetivo general de este proyecto era investigar la integración de datos de procesos y organizacionales de bases de datos NoSQL utilizando plataformas BPMS y escenarios posibles. Se cumplió con el objetivo general de este proyecto, presentando distintos escenarios de esta integración y presentando una posible solución para uno de ellos. Finalmente, se verificó el correcto funcionamiento de la solución implementada aplicándola a un caso de estudio.

Para la integración de los datos del motor de procesos con bases NoSQL se presentó una solución de un BPMS con una base de datos NoSQL del tipo documento. Se eligió Flowable con MongoDB. Como conclusión se pude indicar que esta integración se podría realizar con otros tipos de bases NoSQL que soporten cierto nivel de transacciones sobre los objetos que administra. Para este proyecto fue fundamental el soporte de MongoDB con transacciones ACID a nivel de documentos. Sin este soporte es imposible garantizar la integridad y consistencia de los datos del motor de procesos.
Otro requerimiento es lograr un mapeo entre el modelo relacional y el base NoSQL elegido. Fue esencial para este proyecto la característica de Flowable que permite incluir otra capa de persistencia, en donde se mapea el modelo relacional al de documentos NoSQL.

Se presentó también una solución para la integración de datos organizacionales de un proceso BPMN con bases de datos NoSQL, utilizando Redis como base NoSQL clave valor. Como conclusión el desafío está en mapear el modelo relacional al tipo clave valor. Hay que destacar que para realizar un buen mapeo se requieren, antes de comenzar, todas las transacciones que puede realizar la aplicación. Esto es fundamental para el modelado. Cualquier nueva consulta que se necesite en la aplicación puede requerir modificar alguna de estas decisiones, duplicando o estructurando la información de otra manera.
También concluir que no hay mayores inconvenientes con el BPMS para la integración de bases NoSQL para datos organizacionales. Cualquier BPMS moderno puede invocar a servicio externo que realice esta tarea.
Entiendo que el mayor inconveniente puede estar en tener 2 transacciones diferentes entre los datos del proceso y los datos organizacionales. Pero también puede ocurrir si se utilizan 2 bases de datos relacionales distintas. En ambos casos se puede requerir por ejemplo un mecanismo de compensación adecuado.

Para mostrar el uso de la solución se implementó un caso de estudio del proceso Student Mobility. Se desarrollo en Flowable utilizando MongoDB para los datos del proceso y Redis KV para los datos organizacionales. Para esto se realizó una aplicación Java de consola instanciando el motor de negocio de Flowable, como trabajo a futuro se podría integrar a la plataforma web de Flowable para poder utilizar procesos desde ahí.

Como conclusión, se puede decir que se cumplió con todos los objetivos pertinentes al Proyecto. Se confirma la posibilidad de utilizar bases de datos NoSQL para los datos del motor de procesos del BPMS respetando ciertas características el motor de procesos (permitir otra capa de persistencia) y la base NoSQL elegida (transacciones, por ejemplo).

8 Referencias

[1] M. Weske, Business Process Management - Concepts, Languages, Architectures, Springer, 2019.

[2] G. Harrison, Next Generation Databases NoSQL, NewSQL, and Big Data.

[3] Object Management Group (OMG), *OMG Business Process Model and Notaton (BPMN 2.0),* p. https://www.omg.org/spec/BPMN/2.0/.

[4] "Flowable," [Online]. Available: https://www.flowable.com. [Accessed 2022].

[5] "Activiti," [Online]. Available: https://www.activiti.org/. [Accessed 2022].

[6] "Camunda," [Online]. Available: https://camunda.com/. [Accessed 2022].

[7] "Bonita," [Online]. Available: https://es.bonitasoft.com/. [Accessed 2022].

[8] "Bizagi," [Online]. Available: https://www.bizagi.com/es. [Accessed 2022].

[9] A. Nayak, A. Poriya and D. Poojary, "Type of NOSQL Databases and its Comparison with Relational Databases".

[10] E. A. Brewer, *Towards robust distributed systems. (Invited Talk),* 2000.

[11] T. Khasawneh, M. AL-Sahlee and A. Safia, "SQL, NewSQL, and NOSQL Databases:," 2020.

[12] "Redis Docs," [Online]. Available: https://redis.io/docs/. [Accessed 2022].

[13] "Amazon DynamoDB," [Online]. Available: https://docs.aws.amazon.com/dynamodb/. [Accessed 2022].

[14] "Riak," [Online]. Available: https://riak.com/products/riak-kv/. [Accessed 2022].

[15] "Cassandra," [Online]. Available: https://cassandra.apache.org/doc/latest/. [Accessed 2022].

[16] "Apache HBase," [Online]. Available: https://hbase.apache.org/. [Accessed 2022].

[17] "MongoDB Manual," [Online]. Available: https://www.mongodb.com/docs/manual/. [Accessed 2022].

[18] "Apache CouchDB," [Online]. Available: https://docs.couchdb.org/en/stable/. [Accessed 2022].

[19] "Neo4j," [Online]. Available: https://neo4j.com/docs/. [Accessed 2022].

[20] "JanusGraph," [Online]. Available: https://docs.janusgraph.org/. [Accessed 2022].

[21] A. Delgado, *Presentación de Propuesta de Actividad Integradora / Tesis.*

[22] "Running Flowable on MongoDB," 08 2018. [Online]. Available: https://blog.flowable.org/2018/08/13/running-flowable-on-mongodb/. [Accessed 2022].

[23] "BSON," [Online]. Available: https://bsonspec.org/. [Accessed 2022].

[24] "Flowable-MongoDB on Github," [Online]. Available: https://github.com/flowable/flowable-mongodb. [Accessed 2022].

[25] "MongoDB Java Driver on Github," [Online]. Available: https://mongodb.github.io/mongo-java-driver/3.12/. [Accessed 2022].

[26] A. Delgado and D. Calegari, *Towards a unified vision of business process and organizational.*

[27] "WSL," [Online]. Available: https://docs.microsoft.com/en-us/windows/wsl/. [Accessed 2022].

Anexo A: Compilar y ejecutar aplicación

Herramientas necesarias

- Java v8
- Maven v3.8.6
- MongoDB v4.4.17 y MongoDB Compass v1.25
- Redis v7.0.5 y RedisInsight v2.8

Instalación e inicialización de MongoDB

Se instaló la versión 4.4 de MongoDB Community Server en Windows. Una vez instalado queda como un servicio disponible en el sistema. Para iniciarlo simplemente manejarlo como cualquier otro servicio de Windows.
También se instaló la versión 1.25 de MongoDB Compass que brinda una interfaz gráfica (GUI) para trabajar con MongoDB.

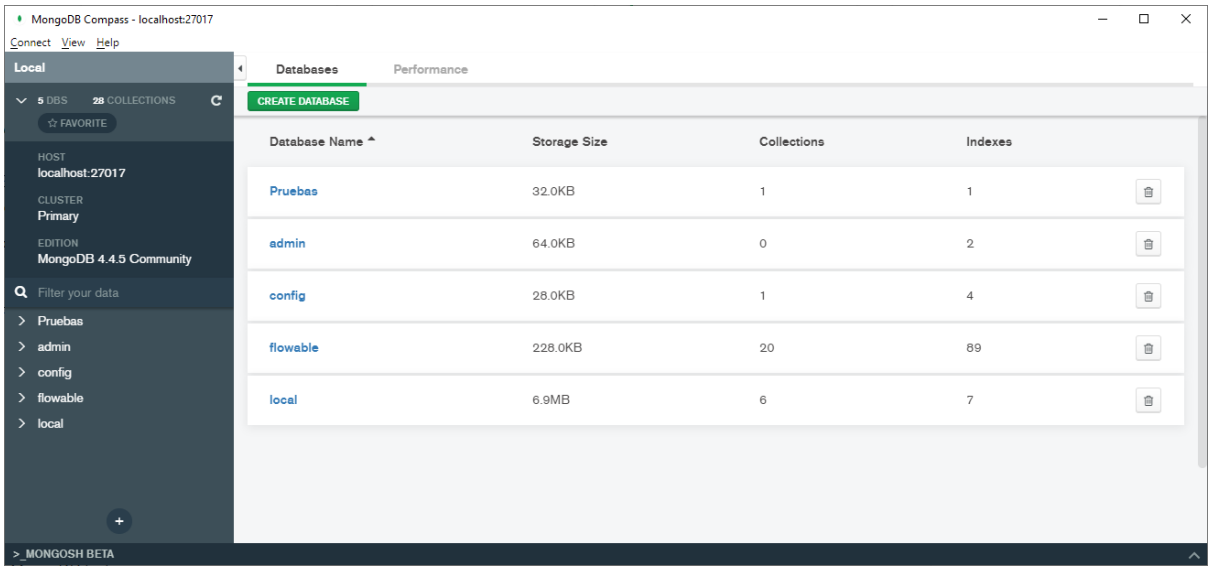


Figura A.1: Pantalla de ejemplo de MongoDB Compass

Replicate set

Para utilizar transacciones es necesario utilizar MongoDB como replicate set y no un servidor standalone (independiente). Para esta prueba se habilitó en un único nodo, para ello incluir en mongod.cfg:

```
replication:
  replSetName: "rs0"
```

Luego ejecutar por única vez en el cliente mongo.exe:

```
rs.initiate()
```

Instalación e inicialización de Redis

Redis oficialmente no soporta Windows. De todas formas, hace uso de la herramienta WSL (Windows Subsystem for Linux) [27] de Windows que permite instalar y correr distribuciones de Linux en Windows.

Una vez completada la guía de instalación del sitio oficial de Redis se obtiene un Ubuntu con el servidor de Redis instalado.

Una vez iniciado se puede acceder utilizando el puerto 6379 como cualquier otro servicio que esté corriendo en Windows.

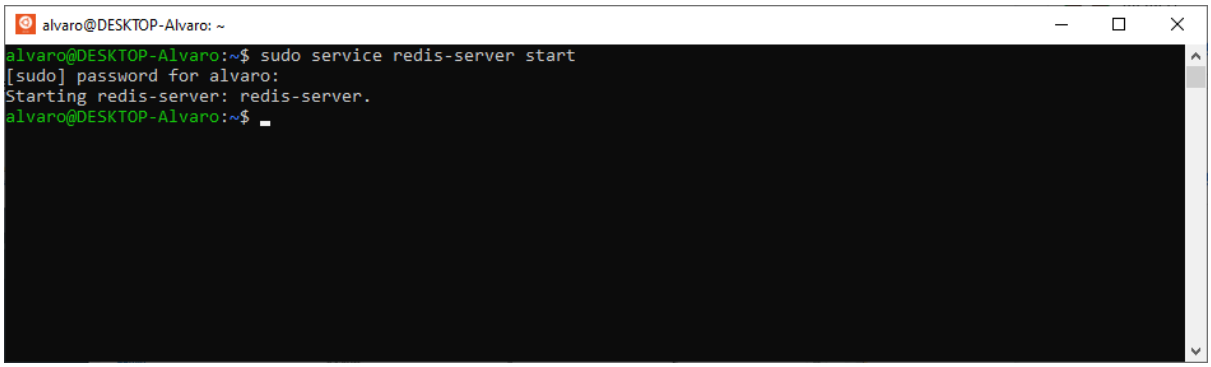


Figura A.2: Iniciando Redis desde una consola de Ubuntu

También se instaló el cliente con interfaz gráfica RedisInsight-v2 (versión 2.8)

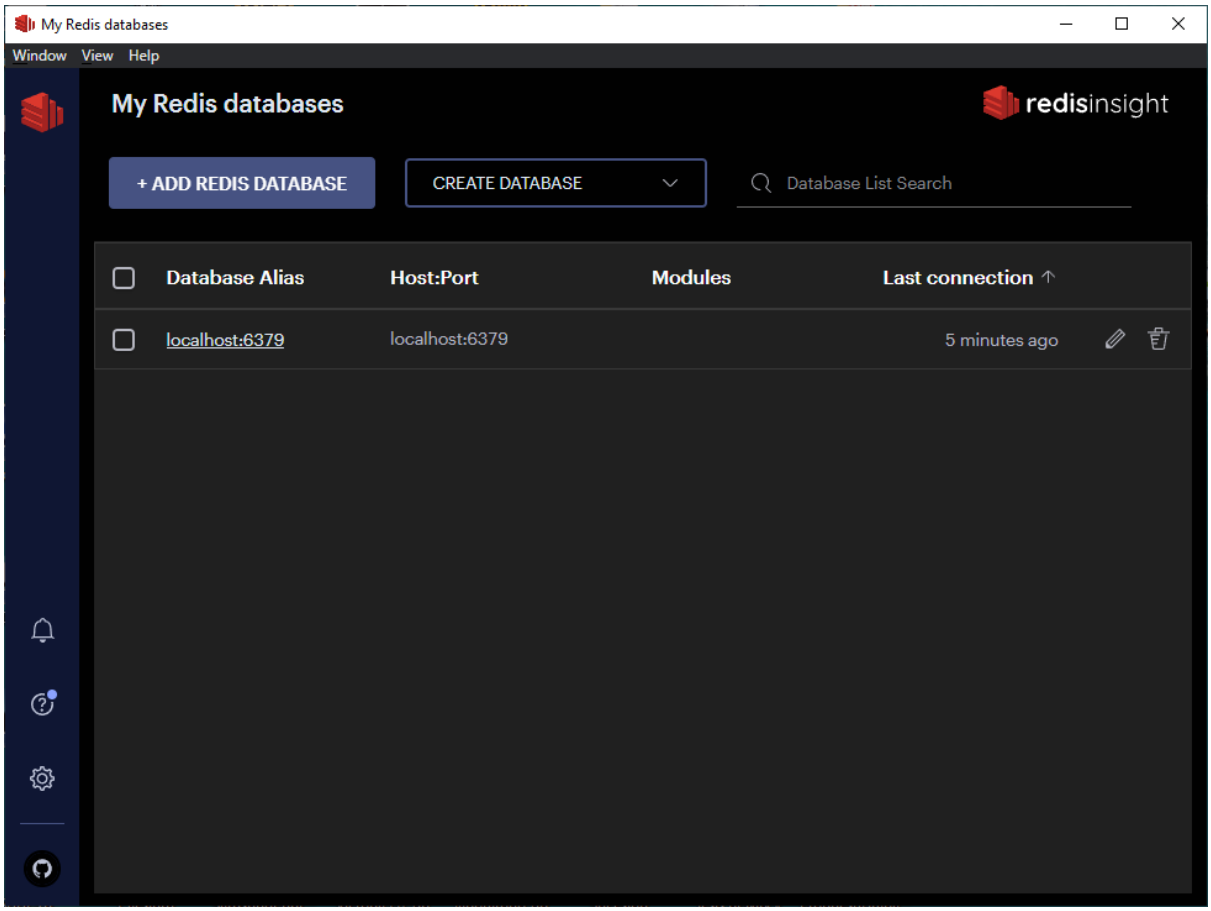


Figura A.3: Pantalla de ejemplo de RedisInsight

Script

Es necesario incluir algunos datos básicos (cursos, profesores, institutos, carreras) en la base de datos Redis. Para esto se incluye el script studentsmobility-redis.txt. La forma más fácil de ejecutarlo es utilizar RedisInsight desde el tab Workbench como se muestra en la siguiente figura.

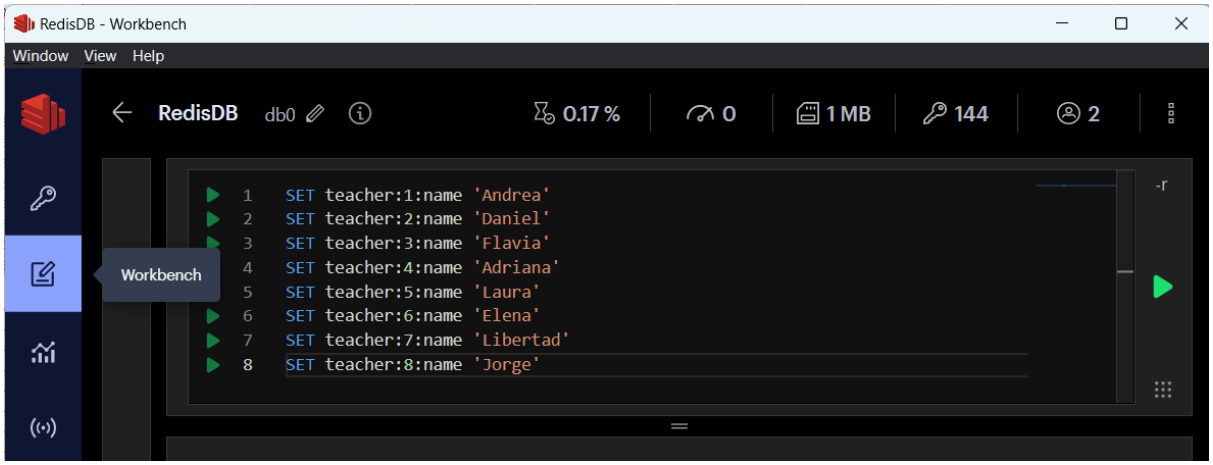


Figura A.4: Pestaña workbeanch para la carga del script

Descargar y compilar

Para obtener el proyecto descargar desde el repositorio:

```
https://gitlab.fing.edu.uy/alvaro.vallve/studentsmobility/
```

Es necesario ajustas la configuración de las bases de datos si no se utilizan los valores por defecto. La configuración se encuentra en el archivo resources/application.properties:

```
mongodb.url=localhost
mongodb.port=27017

redis.url=localhost
redis.port=6379
```

Para compilar y ejecutar el programa:

```
$ mvn clean install
$ mvn exec:java -Dexec.mainClass="principal.StudentsMobilityMain"
```

Anexo B: Ajustes librería MongoDB

Búsqueda de task por key

La librería no incluía la búsqueda de tareas por su clave. Se agrego el siguiente filtro en MongoDbTaskDataManager.createFilter:

```
if (taskQuery.getKey() != null) {
  andFilters.add(Filters.eq("taskDefinitionKey", taskQuery.getKey()));
}
```

Borrado de variables

Al intentar borrar variables por id de tarea la librería retornaba UnsupportedOperationException indicando que el método no estaba implementado. Se implementó de la siguiente manera:

```
protected List<VariableInstanceEntity> findVariableInstancesByTaskId(String taskId) {
  return getMongoDbSession().find(COLLECTION_VARIABLES, Filters.eq("taskId", taskId),
taskId,
       VariableInstanceEntityImpl.class, variableInstanceByTaskIdMatcher, true);
}

@Override
public void deleteVariablesByTaskId(String taskId) {
  List<VariableInstanceEntity> variables = findVariableInstancesByTaskId(taskId);
  if (variables != null) {
    for (VariableInstanceEntity variableInstanceEntity : variables) {
      getMongoDbSession().delete(COLLECTION_VARIABLES, variableInstanceEntity);
    }
  }
}
```

Tiempo de bloqueo de la instancia de proceso

Luego de finalizado el tiempo del timer finalizaba la aplicación con un UnsupportedOperationException al intentar actualizar la colección executions. Se implemento el siguiente método:

```
@Override
public void updateProcessInstanceLockTime(String processInstanceId, Date lockDate, String
lockOwner, Date expirationTime) {
  BasicDBObject updateObject = new BasicDBObject();
  updateObject.append("lockTime", lockDate);

  Bson filter = Filters.and(Filters.eq("_id", processInstanceId),
Filters.or(Filters.eq("lockTime", null), Filters.lt("lockTime", expirationTime)));
  UpdateResult updateResult =
getMongoDbSession().updateImmediately(COLLECTION_EXECUTIONS, filter, updateObject);
  if (updateResult.getModifiedCount() != 1) {
    throw new FlowableOptimisticLockingException("Could not lock process instance");
  }
}
```

Variables referenciadas

El cambio más significativo de la librería es da con un problema a la hora de utilizar variables referenciadas, es decir, variables que no son del tipo básico (string, integer, etc.). En el caso que la variable asignada sea un array de bytes, la misma se almacena en la colección byteArrays y se referencia desde la colección variables. Este comportamiento no funciona, el valor de la variable se pierde al obtenerla.

Si intentó solucionarlo de la forma correcta, es decir almacenando el valor en la colección byteArrays y referenciarlo desde la colección variables, pero no fue posible. La solución que se encontró fue almacenar el valor de la variable directamente en la colección variables.
En la siguiente imagen se pude ver el estado de la variable apps_list, utilizada en la aplicación de ejemplo, con el valor almacenado junto en la colección variables. En este caso el valor es un array de long.

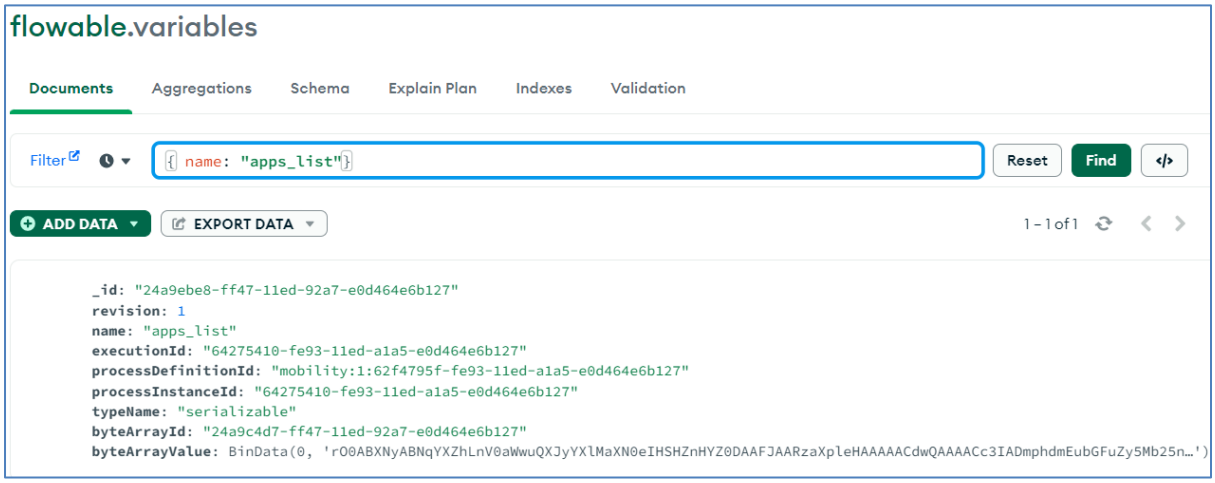


Figura B.1: Variable de tipo byteArray

Los cambios por este motivo son bastantes para incluir en el documento, para ver los cambios consultar el fork de la librería en: https://github.com/alvarovallve/flowable-mongodb