Trabajo Fin de Grado

Automatización de reglas de negocio en la comunicación con sistemas terceros

---

GRADO EN INGENIERÍA INFORMÁTICA

Ricardo Roldán Donaire

Tutor
Alejandro Rey López
Jorge Martínez García-Saavedra

Colmenarejo, Junio 2022

DEDICATORIA

En primer lugar, agradecer a mi tutor Alejandro Rey por aceptar el proyecto propuesto y apoyarme en esta etapa final. A todos mis compañeros de NTTData en especial a Jorge Martínez. Mi familia siempre ha estado apoyándome y confiando en mí, Ricardo y Alex, siempre están cuando se necesita. Y finalmente a Anne, mi apoyo y mi sonrisa todo este año.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ABREVIATURAS

RPA Robotic Process Automation (Automatización Robótica de Procesos)
PTM Préstamos sindicados
TTF Financiación Internacional
SpEL Spring Expression Language
JSON JavaScript Object Notation
Regex Regular expression
BPM Business Process Management
POJO Plain Old Java Object
DAO Data Access Object
JDBC Java DataBase Connectivity
JMX Java Management Extensions
JMS Java Message Service
JCA Java Connector Architecture
REST Representational State Transfer (Transferencia de Estado Representacional)
CIB Corporate & Investment Banking (Banca Corporativa y de Inversión)
CLI Command Line Interface (Interfaz de Línea de Comandos)
API Application Programming Interface (Interfaz de Programación de Aplicaciones)
DTO Data Transfer Object (Objeto de Transferencia de Datos)
UAT User Acceptance Testing (Pruebas de Aceptación del Usuario)

RESUMEN

El objetivo de este documento es recoger de forma concisa el proceso de desarrollo de un sistema automatizado de control de reglas de negocio para productos de préstamos sindicados y financiación internacional.

Se recogerá todo el proceso desde la toma de requisitos hasta la el desarrollo y puesta en marcha del sistema.

De este proyecto cabe destacar el uso de Spring Expression Languaje, con el objetivo de modularizar las lógicas aplicadas sobre determinadas reglas de negocio.

1. INTRODUCCIÓN

1.1. Motivación del trabajo

La transformación digital es un proceso de cambio y adaptación de las empresas al mundo digital, y este es un proceso clave para asegurar nuevas oportunidades de negocio. [1]

Entre los sectores que han invertido en digitalización está el sector financiero, ocupando España el segundo puesto de países con el sector de banca mejor digitalizado del mundo.

Figura 1. Digitalización en sector de Banca. Fuente: [2]

La automatización continúa ganando notoriedad, haciendo que procesos repetitivos y rutinarios sean cada vez más rápidos, cómodos y eficaces. [1]

Este trabajo tiene la motivación de crear un producto de software determinado para una empresa del sector financiero con una perspectiva de puesta en producción. El sistema funcionara con datos de clientes con el objetivo de crear nuevas líneas de financiación corporativa y productos de financiación internacional.

Gracias a los *frameworks* de desarrollo como **Spring Framework** podemos desarrollar **procesos automáticos y procesamientos por lotes** para agilizar tareas y facilitar la toma de decisiones en el sector de la banca.

1.2. Objetivos

El objetivo de este proyecto es diseñar e implementar un *demonio* que permita generar un *array* de datos para distintos servicios de financiación y préstamos a partir de unas reglas de negocio que se aplican en función del producto y el servicio.

- Evaluar y analizar el funcionamiento del sistema
    - Servicios y productos
    - Entradas y salidas de datos
- Diseñar e implementar una solución al problema propuesto
- Desarrollar pruebas y despliegue para pruebas de aceptación con el usuario

Como solución al objetivo se plantea la **modularización de las lógicas** a implementar de forma que se pueda alterar el funcionamiento del servicio sin la necesidad de redesplegar una nueva versión del mismo, lo cual facilita el mantenimiento de los posibles desarrollos evolutivos sobre los productos que alberga.

1.3. Estructura de la memoria

Este apartado permitirá entender el recorrido del trabajo explicando cada una de las secciones que lo componen.

En primer lugar, encontramos el estado del arte (sección 2), este apartado pretende explicar el estado actual de las tecnologías actuales disponibles en el ámbito del **desarrollo de servicios web**, así como las posibilidades que dibujan estas tecnologías.

En el marco regulador se encuentran los normas y legislaciones relativas al desarrollo del proyecto (sección 3).

A continuación, encontramos el capítulo de entorno socioeconómico en el cual se aborda el presupuesto del proyecto, su planificación y el impacto socioeconómico (sección 4).

Siguiendo al capítulo del entorno socioeconómico abordamos las tecnologías empleadas (sección 5), hará hincapié en la lista de recursos empleados para la resolución del problema

planteado, entrando en profundidad en las posibilidades que existen y como orientarlas hacia el objetivo del sistema que se plantea.

A continuación, se exponen todos los elementos necesarios para el diseño del sistema detallando las secciones y campos necesarios para el mismo (sección 6), posteriormente se especificarán los requisitos funcionales y no funcionales del sistema (sección 7).

El siguiente capítulo (sección 8) aborda su implementación y plan de validación, en el cual se detalla el proceso de desarrollo, sus características principales y su configuración.

Dentro de la gestión de proyecto se detalla el plan de validación el cual se abordará el desarrollo de pruebas unitarias, evaluación de resultados y validación de cobertura del código, y pruebas en entornos previos.

Finalmente (sección 9), encontramos las conclusiones analizando todos los capítulos anteriormente mencionados, sintetizando y exponiendo las soluciones a los problemas propuestos antes y durante la ejecución del proyecto y ofreciendo alternativas para futuros desarrollos evolutivos.

1.4. Glosario de términos

A continuación, se definen las palabras y expresiones técnicas referentes al proyecto abordado:

- **Demonio**: Tipo de proceso que se ejecuta en segundo plano, en lugar de ser controlado directamente por el usuario.
- **Frontend**: Parte del desarrollo de programación web encargada de la interfaz de usuario, con la que interactúa el mismo.
- **Backend**: Parte del desarrollo de programación web encargada de la lógica de negocio.
- **Framework**: es un entorno de trabajo, un conjunto de conceptos estandarizado que permite resolver tareas o problemas.
- **RESTful**: aplicaciones basadas en REST (estilo de arquitectura de software para sistemas web)
- **String**: Cadena de caracteres
- **Array**: Estructura de datos que permite almacenar un conjunto de datos homogéneos.
- **Dashboard**: Tablero de monitorización de servicios y entornos de ejecución.
- **Hash**: Función criptográfica que toma de entrada un valor de tamaño aleatorio y devuelve un valor de salida de tamaño fijo.
- **Sprint**: Cada uno de los ciclos iterativos dentro de un proyecto desarrollado bajo la metodología Scrum.

- **Fintech**: Empresas dedicadas a la aplicación de nuevas tecnologías sobre actividades financieras.
- **Grid**: Cuadricula, en el documento se hace referencia al objeto que genera el sistema.

2. ESTADO DEL ARTE

Existen diferentes lenguajes de programación, para el desarrollo de un servicio web.

A la hora de elegir un lenguaje de desarrollo *backend* las empresas han de realizar un análisis exhaustivo de los requisitos, con el objetivo de eficientar el desarrollo, para ello se ha de tener en cuenta no solo el objetivo del proyecto si no las características del mismo y sus posibles **desarrollos evolutivos** [3].

En este ámbito del desarrollo *backend* se destacan los siguientes lenguajes de programación entre la gran cantidad de leguajes que existen.

2.1. Java

Java [4] es uno de los más conocidos actualmente y por ende uno de los más demandados en el mercado. Es un lenguaje orientado a objetos, fue diseñado con el objetivo de ejecutarse en cualquier dispositivo (multiplataforma) [5].

Entre sus principios básicos podemos destacar que es:

- Simple: Su sencillez hace que sea un lenguaje fácil de aprender
- Multihilo: Cada hilo de representa un proceso individual de ejecución.
- Seguro: Su seguridad y estabilidad hace que se utilice en prácticamente cualquier entorno.
- Multiplataforma. El desarrollo único permite que podamos ejecutarlo en distintos sistemas operativos.

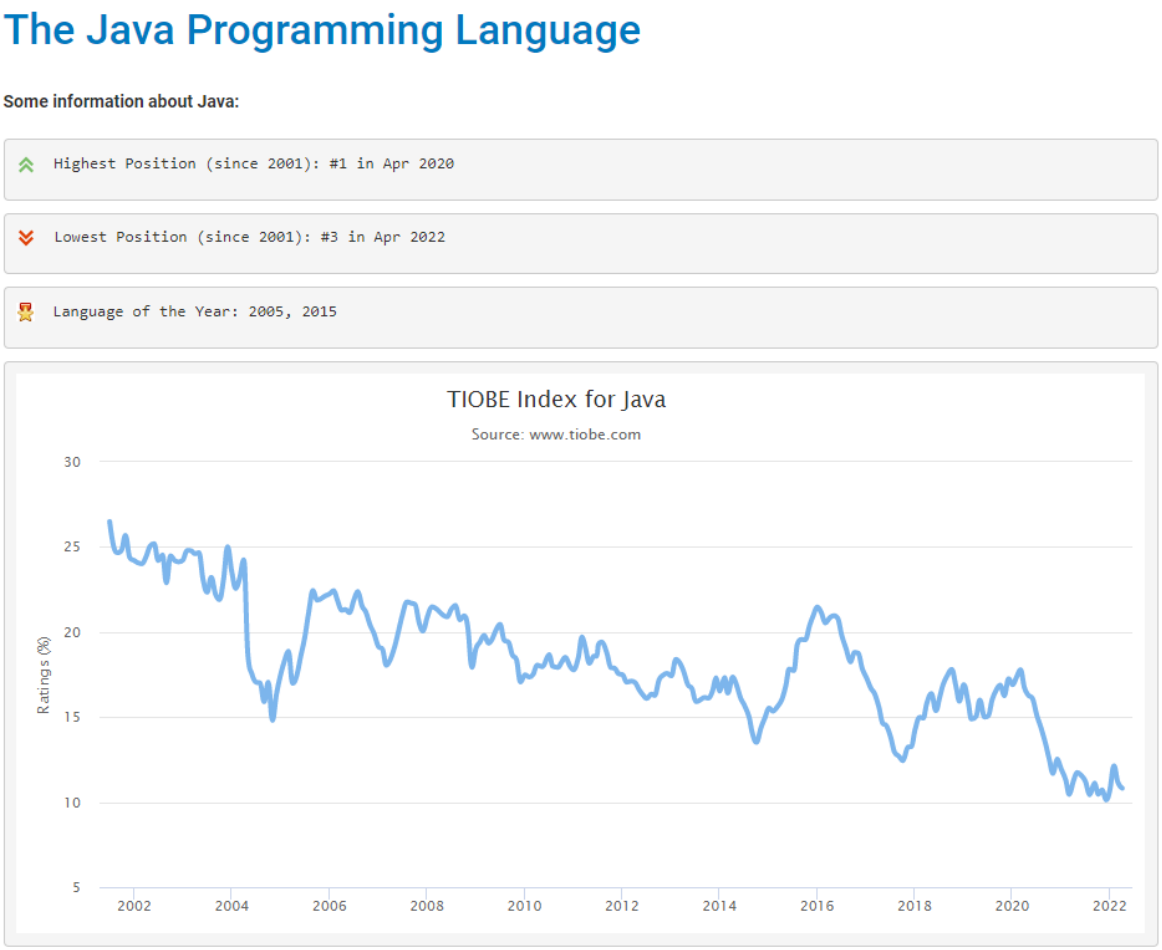


Figura 2. TIOBE Index for Java. Fuente: [6]

En la figura 2 podemos observar como el uso de Java ha caído ligeramente los últimos años, a pesar de ello sigue siendo uno de los leguajes de programación más utilizado debido a que una gran cantidad de aplicaciones web se han desarrollado con este lenguaje, y esto es porque es un lenguaje muy robusto versátil que se puede ejecutar prácticamente en cualquier dispositivo.

2.2. JavaScript

JavaScript [7] es uno de los lenguajes más populares actualmente [8] ya que permite el desarrollo de frontend y backend con la misma sintaxis, reduciendo la carga de trabajo. Infraestructuras de aplicaciones web como Express.JS [9] bajo el marco de aplicación de NodeJS [10] nos permiten el desarrollo de aplicaciones web, diseñando tanto su aplicación frontal como las lógicas de negocio.

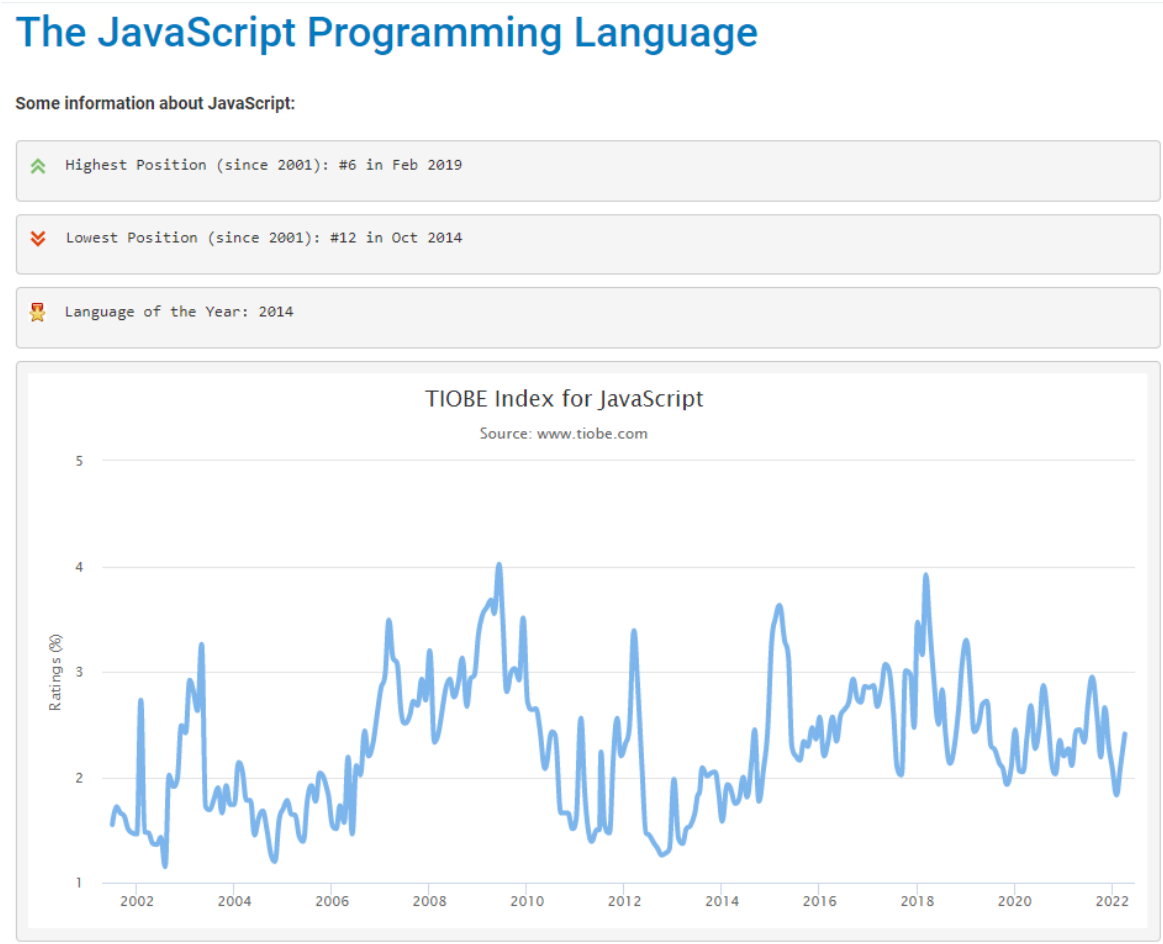


Figura 3. TIOBE Index for JavaScript. Fuente: [11]

2.3. Python

Python [12] se presenta como un lenguaje de programación sencillo de aprender, escribir y entender. Entre sus ventajas podemos destacar la gran variedad de bibliotecas de código que ayudan al desarrollador a evitar redundancia en su código, así como un gran soporte de la comunidad [13].

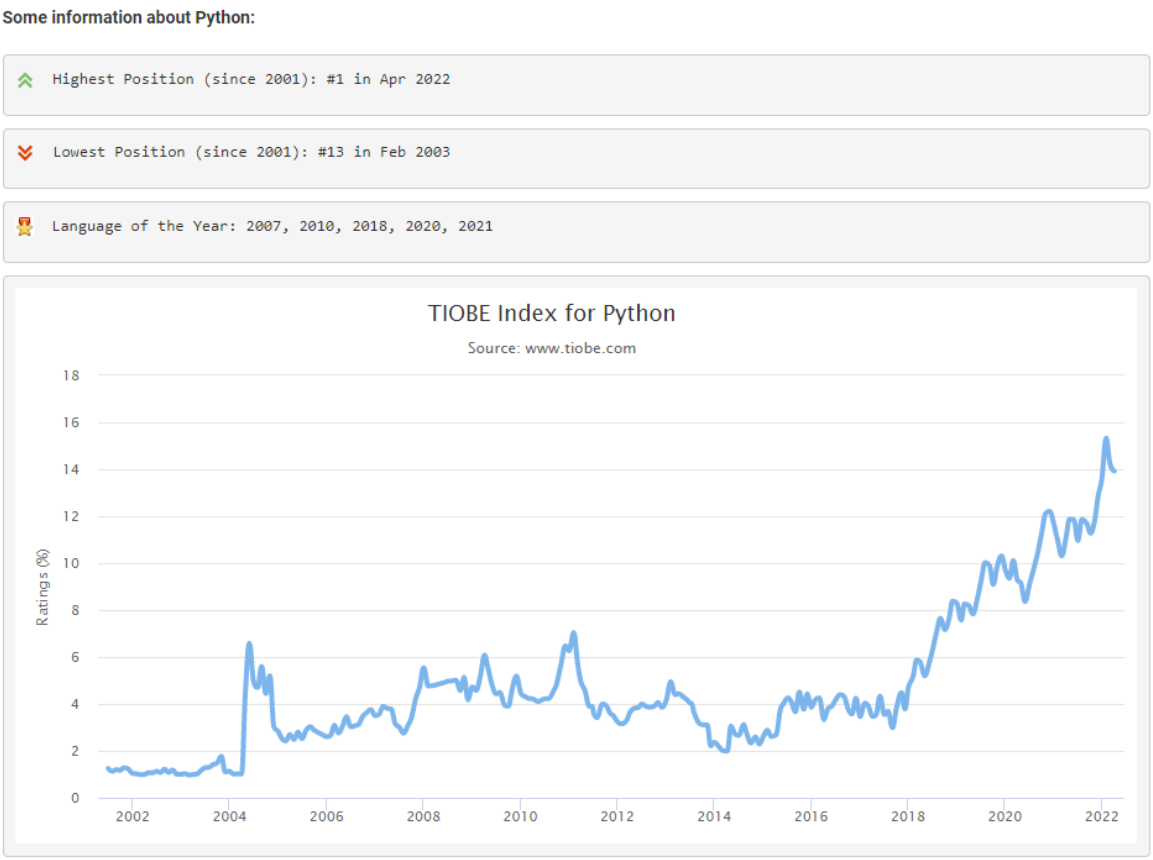


Figura 4. TIOBE Index for Python. Fuente: [14]

Python es uno de los lenguajes más populares actualmente, esto es debido a que se trata de un lenguaje sencillo y de código abierto, aunque sobre el cabe destacar su crecimiento con los nuevos desarrollos de inteligencia artificial y aprendizaje automático.

2.4. PHP

PHP [15] es uno de los lenguajes más extendidos para el desarrollo de aplicaciones del lado del servidor. Cuenta con una basta comunidad de desarrolladores y es simple, lo que permite simplificar el proceso de estructuración de webs [16].

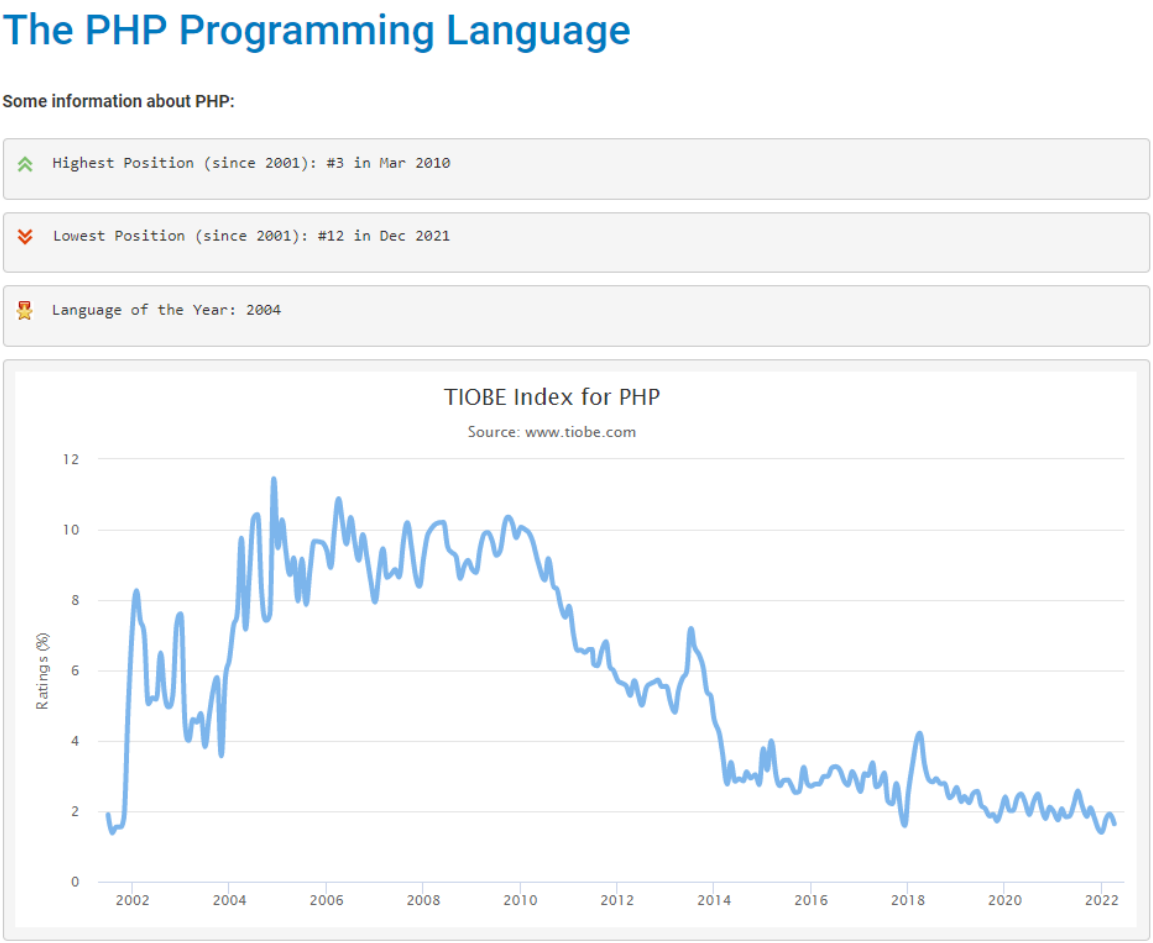


Figura 5. TIOBE Index for PHP. Fuente: [17]

2.5. Metodologías de desarrollo

Las metodologías de desarrollo, son las técnicas y métodos organizativos que permiten organizar el trabajo y desarrollo del producto de la forma más productiva y eficaz.

2.5.1. Waterfall (Cascada)

En la **metodología *waterfall*** la estructura de las etapas es de arriba hacia abajo, ejecutándose **de forma secuencial** hasta el lanzamiento del producto. Al final de cada etapa se realiza una revisión del estado del proyecto. Se comprueba que los requisitos elaborados para esta etapa son cumplidos y de ser así se da paso a la siguiente etapa. [18]

Figura 6. Esquema metodología waterfall. Fuente: [18]

2.5.2. Prototipado

Esta metodología se basa en la creación de un prototipo simple, que ofrece una visión al cliente de cómo será el servicio desarrollado. Si el desarrollo se lleva a cabo se realizan desarrollos evolutivos sobre el prototipo hasta conseguir el producto final deseado por el cliente [19].

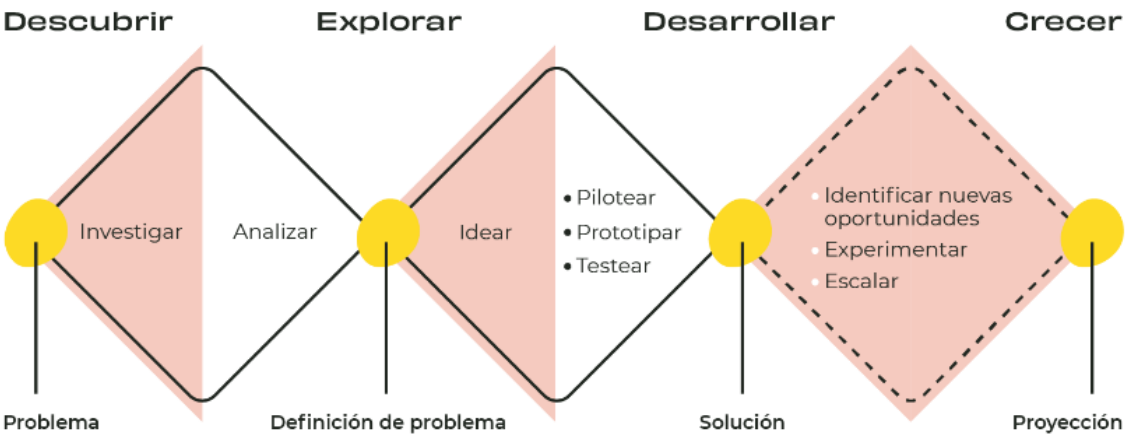


Figura 7. Metodología de prototipado. Fuente: [20]

2.5.3. Metodologías agiles

Las metodologías agiles de desarrollo de software, **son tecnologías incrementales**. Estas se basan en ciclos de desarrollo, esto implica que las funcionalidades previstas pueden variar, aumentarse o reducirse con cada ciclo de desarrollo del producto [21].

Entre las metodologías agiles más comunes podemos encontrar:

- **Kanban**: La metodología Kanban se basa en unos principios básicos:

- Visualizar el flujo de trabajo.
- Establecer metas asequibles.
- Realizar un seguimiento del tiempo.
- Uso de indicadores visuales.

Esto permite realizar un seguimiento de los tiempos de trabajo e identificar los cuellos de botella en el desarrollo del producto [22].

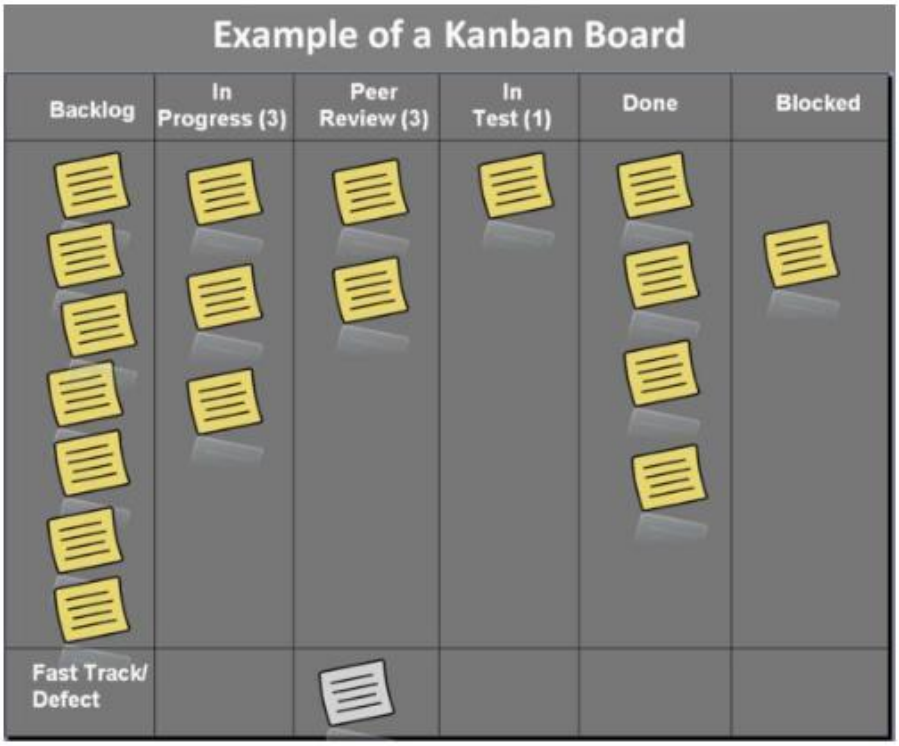


Figura 8. Ejemplo de pizarra Kanban. Fuente: [23]

- **Scrum** [24]: La metodología Scrum aborda proyectos de forma flexible. Consiste en una metodología iterativa basada en Sprints de 1-4 semanas. Al final de cada Sprint se realiza una revisión del Sprint con la finalidad de controlar los cambios y ver el estado del proyecto.

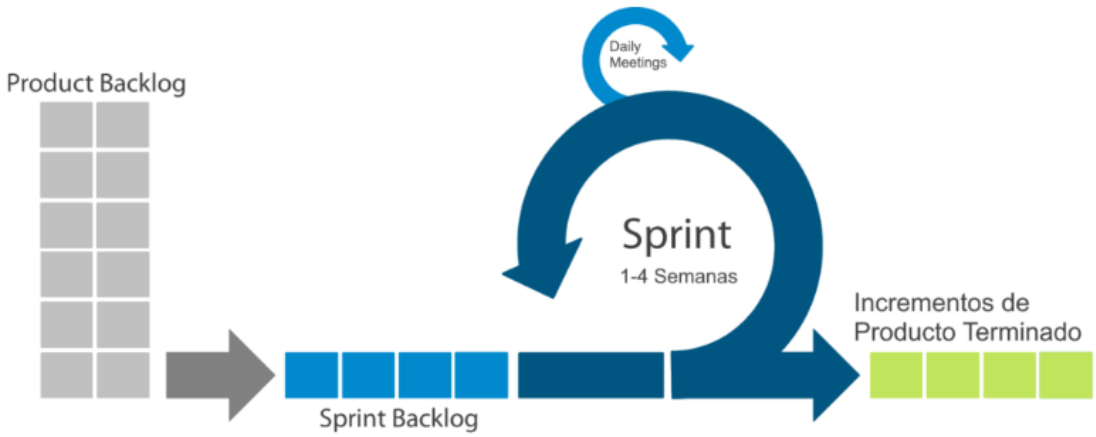


Figura 9. Esquema Metodología Scrum. Fuente: [25]

3. MARCO REGULADOR

Este apartado se exponen las normas y regulaciones actuales que se han de tener en cuenta a para el desarrollo correcto del proyecto

- **Reglamento General de Protección de Datos (RGPD)** [26]**.**
- **Ley Orgánica de Protección de Datos (LOPD)** [27].

Al contener datos de carácter personal, debemos tratar estos datos conforme a la ley vigente y evitar el tráfico ilícito de los mismos.

4. ENTORNO SOCIOECONOMICO

4.1. Planificación

Como metodología de trabajo se ha utilizado **Scrum** una de las metodologías agiles anteriormente comentada (sección 2). Para ello se han realizado *Sprints* o ciclos de 2 semanas.

A continuación, se muestra un Gantt para entender los ciclos de desarrollo del proyecto.

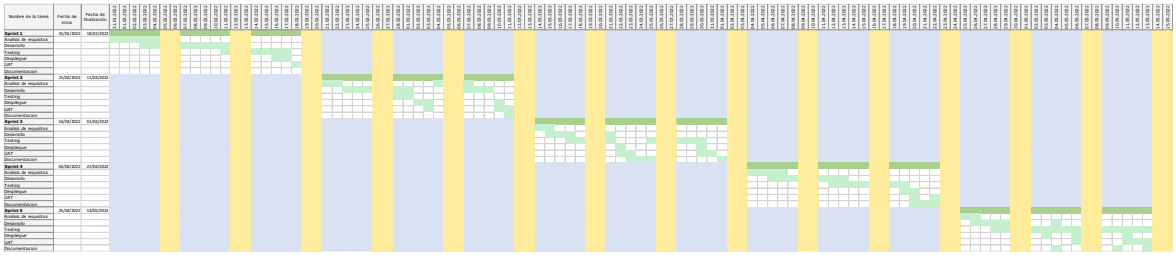


Figura 10. Diagrama Gantt del proyecto.

Nombre de la tarea	Fecha de inicio	Fecha de finalización	31.01.2022	01.02.2022	02.02.2022	03.02.2022	04.02.2022	05.02.2022	06.02.2022	07.02.2022	08.02.2022	09.02.2022	10.02.2022	11.02.2022	12.02.2022	13.02.2022	14.02.2022	15.02.2022	16.02.2022	17.02.2022	18.02.2022	19.02.2022	20.02.2022	21.02.2022	22.02.2022	23.02.2022	24.02.2022	25.02.2022	26.02.2022	27.02.2022	28.02.2022	01.03.2022	02.03.2022	03.03.2022	04.03.2022	05.03.2022	06.03.2022	07.03.2022	08.03.2022	09.03.2022	10.03.2022	11.03.2022
Sprint 1	31/01/2022	18/02/2022																																							
Analisis de requisitos																																									
Desarrollo																																									
Testing																																									
Despliegue																																									
UAT																																									
Documentacion																																									
Sprint 2	21/02/2022	11/03/2022																																								
Analisis de requisitos																																									
Desarrollo																																									
Testing																																									
Despliegue																																									
UAT																																									
Documentacion																																									

Figura 11. Diagrama de Gantt de sprints iniciales.

Tal y como vemos en las figuras 10 y 11 se trata de *Sprints* de 3 semanas (5 días hábiles por semana) que suman un total de 150 días por 5 *Sprints* de desarrollo del proyecto.

4.2. Presupuesto

En este apartado se elabora el presupuesto del proyecto, teniendo en cuenta los gastos asociados al personal necesario para el desarrollo del proyecto. También habrá que tener en cuenta el coste de las distintas herramientas utilizadas para el desarrollo.

Para la realización de este presupuesto se ha hecho una estimación de los costes para el desarrollo único de este servicio.

- **Coste de personal**: Estos costes hacen referencia a los gastos en los salarios de los empleados encargados del proyecto. La tabla muestra los gastos asociados a cada integrante del equipo de trabajo. Se estima el coste en del desarrollo del proyecto.

Rol	Salario bruto anual	Coste/hora	Coste/Dia	Duración del trabajo	Total
Programador	21500 €	11,19 €	55.98 €	150 días	8398,44 €
TOTAL					**8398,44 €**

Tabla 1.Tabla de coste de personal.

- **Costes del software**: Al ofrecer licencias de todos los productos de software necesarios para el proyecto y trabajar bajo la infraestructura del banco, los gastos asociados a software son de 0€

- **Coste del hardware:** Coste asociado a los equipos necesarios para la realización del proyecto.

Equipo	Coste (€)	Uso en el proyecto	Duración del proyecto	Amortización	Coste imputable
Dell Inspiron 15 intel Core i5 4,50 Ghz	749	100%	150 días	5 años	62,41 €
TOTAL					**62,41 €**

Tabla 2. Tabla de coste del hardware.

A continuación, se muestra la tabla de la en la cual se muestra la deducción del precio total para el presupuesto del proyecto.

CONCEPTO	COSTE
Personal	8398,44 €
Software	0 €
Hardware	62,41 €
Total, antes de riesgo	8458,85 €
Riesgo (12%)	1015,06 €
Total, antes de beneficio	9473,91 €
Margen de beneficio (15%)	1421,08 €
Total, sin IVA (21%)	10894,99 €
TOTAL	**13182,95 €**

Tabla 3. Tabla coste del proyecto.

4.3. Impacto socio-económico

El proyecto está enfocado al desarrollo de un proceso de datos fiscales con el objetivo de automatizar ciertas tareas. En este ámbito hay que tener presente el impacto socioeconómico se produce.

El crecimiento de las *fintech* trae nuevas oportunidades de negocio y su crecimiento se ha acelerado a partir de la crisis de la covid-19 [28].

La automatización de procesos en el sector de la banca permite reducir el gasto en personal, evaluando reglas de negocio hasta ahora evaluadas manualmente, esto a su vez abre nuevas líneas de inversión en desarrollo de productos automáticos que permitan consolidar el sector de la banca en el ámbito de las aplicaciones web.

Esta migración del sector financiero hacia una banca más tecnológica abre la posibilidad de investigación en nuevos desarrollos, ya sea con la migración de servicios a procesos más automatizados, como la creación de nuevos servicios.

Por otro lado, tecnologías emergentes como la inteligencia artificial pueden suponer nuevos productos financieros basados, como prestamos inteligentes o inversiones con aprendizaje automático, capaces de analizar los nuevos mercados.

También aparecen nuevas fronteras que abordar en cuanto al desarrollo de estas tecnologías, ya que se hace imprescindible aumentar la accesibilidad de estos servicios a los usuarios más desfavorecidos.
Por este lado la migración hacia una banca digital está dejando de lado a las personas con más problemas de accesibilidad a este tipo de infraestructura digital.

5. TECNOLOGIAS EMPLEADAS

Esta sección aborda la descripción de las tecnologías empleadas en el diseño y desarrollo del proyecto en cuestión.

5.1. Java & Spring framework

Java tiene su origen en 1995, fue creada y comercializada por la compañía Sun Microsystems en 1995. Entre sus principales características destaca por su capacidad de ser ejecutado en cualquier dispositivo y su orientación a objetos [29].

En lo tocante a sus características principales podemos destacar y exponer las siguientes:

- **Simple**. Ofreciendo la funcionalidad, eliminando características confusas de otros lenguajes de programación.
- **Orientado a objetos**. Permite organizar el diseño entorno a entidades u objetos, con unos atributos y métodos pertenecientes a estos.
- **Portable**. Al poder ejecutarse en cualquier tipo de hardware podemos desarrollar software para cualquier tipo de plataforma
- **Recolector de basura**. Este se encarga de eliminar los objetos no utilizados y hacer un uso eficiente de la memoria.
- **Sólido**. Evita la corrupción del código, protegiendo la privacidad de los objetos.
- **Multihilo**. Puede realizar tareas simultaneas dentro del mismo programa, mejorando el rendimiento y la velocidad de ejecución.

En otro orden de cosas tenemos el *framework* Spring, lanzado en 2003 por Rod Johnson, siendo una plataforma de código abierto. Desde entonces se ha convertido en el *framework* más popular en el desarrollo de código de alto rendimiento. Esto es por su finalidad de estandarizar, agilizar, manejar y resolver problemas derivado del desarrollo. [30]

Spring resulta ser un *framework* clave para el desarrollo empresarial al permitir del despliegue en cualquier plataforma gracias a Java y ofreciendo un modelo para la configuración y programación de aplicaciones empresariales.

Esto supone una ventaja para los equipos de desarrollo ya que permite centrarse en la lógica de negocio haciendo el proceso más corto y eficaz, evitando tareas y configuraciones repetitivas que se albergan bajo este *framework*.

Entre las características de Spring, tenemos las siguientes que ofrecen una cantidad considerable de servicios [30]:

- **Tecnologías**: como la inyección de dependencias, eventos, recursos, validación, enlace de datos, conversión de tipo, **SpEL**.
- **Acceso a datos**: soporte DAO, JDBC entre otros
- **Gestión de transacciones**.
- **Integración:** comunicación remota, JMS, JCA, JMX, correo electrónico, tareas, programación, caché.
- **Pruebas (Testing)**: simulacro de objetos, el *framework* TestContext, Spring MVC prueba, WebTestClient.
- **Programación orientada a aspectos (AOP)**: permite la implementación de rutinas transversales.
- MVC (**Modelo Vista Controlador**).
- **Seguridad**.
- **Frameworks web**: Spring WebFlux y Spring MVC.
- **Procesamiento de datos por lotes**.
- **Administración Remota**: a través de este módulo se puede configurar la visibilidad y gestión de los objetos Java para la configuración local o remota vía JMX.
- Es un ***framework* liviano** debido a su implementación **POJO** (Plain Old Java Object), Spring Framework no obliga al programador a heredar ninguna clase ni a implementar ninguna interfaz.

5.2. Spring Expression Lenguage (SpEL)

Spring Expression Lenguage(SpEL) es un lenguaje de expresión que admite consultas y manipulación de un gráfico de objetos en tiempo de ejecución. SpEL ofrece la funcionalidad de creación de plantillas de cadenas [31].
Existen varios operadores distintos en **SpEL** [32]:

- Aritméticos
- Relacionales
- Lógicos
- Condicionales
- Expresiones regulares (regex)

Estas operaciones con SpEL nos permiten ejecutar lógica de manera modular, ya que podemos inyectar estas expresiones a través de ficheros de configuración. Otra opción es inyectar esta lógica desde un servicio externo, lo cual permite modularizar estas lógicas de negocio sin necesidad de actualizar el software ya implementado.

5.3. Mockito

Java Mockito [33] es uno de los frameworks más utilizados en java para el desarrollo de test unitarios.

Un Mock es un objeto Java que nos permite simular un objeto desarrollado, permitiendo que los test de puedan ejecutar sin dependencias, de forma aislada.

Esto nos permite realizar depuración de código y elaborar informes de cobertura del código, para así evaluar de manera porcentual la cantidad de código que ha sido comprobado mediante test unitarios [34].

5.4. NOVA

Nova es una plataforma de desarrollo, despliegue y administración de CIB (*Corporate and Investment Banking*) y aplicaciones de solución al cliente perteneciente el a BBVA.

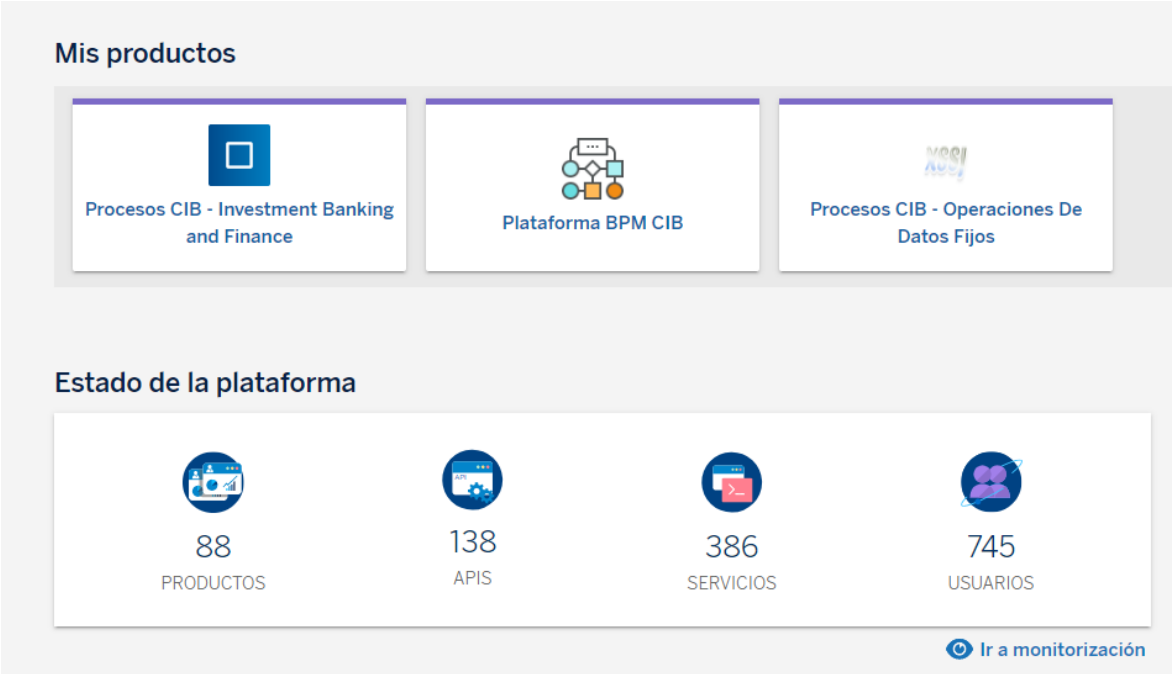


Figura 12. Dashboard NOVA pantalla inicial.

Desde NOVA se monitorizan los productos, servicios y APIs. Desde cada proceso podremos ver los entornos operativos, así como los servicios desplegados en cada entorno. También existen herramientas para la realización de planes de despliegue y ejecución de servicios en cualquiera de los entornos.

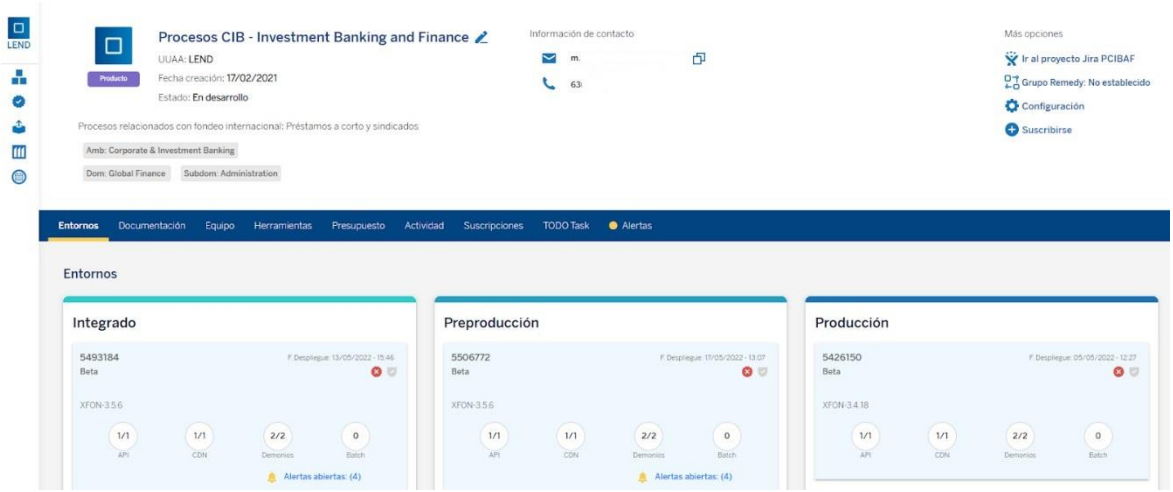


Figura 13. Dashboard NOVA procesos CIB.

6. DISEÑO DEL SISTEMA

En esta sección, se detalla el diseño del sistema, sus componentes y el funcionamiento del mismo.

El sistema está compuesto por dos procesos para el tratamiento de productos de Préstamos Sindicados (PTM) y Financiación Internacional (TTF). Para estos productos el usuario realizara distintas operaciones y en función de estas el demonio tomara una acción distinta. Estas operaciones son las siguientes:

- Alta de nuevo PTM
- Evento de PTM
    - Rollover Standard
    - Rollover Increase
    - Rollover Decrease
    - Cancelación Total
- Alta de nuevo TTF

Los procesos de *rollover* y cancelación total se aplican siempre sobre operaciones ya dadas de *alta*/*rollover* con anterioridad.

Para la extracción de requisitos tendremos en cuenta una muestra de campos sobre los que se aplicara cierta lógica a modo ilustrativo ya que este cuenta con más de 100 campos sobre cada operación.

6.1. Diagrama de flujo

El diagrama de flujo mostrará el funcionamiento del demonio a implementar.

Este consta de dos procesos en segundo plano, el primero aplica las lógicas necesarias para cada operación y el segundo es un proceso que se ejecuta cada 5 minutos comprobando que las lógicas no han sido actualizadas.

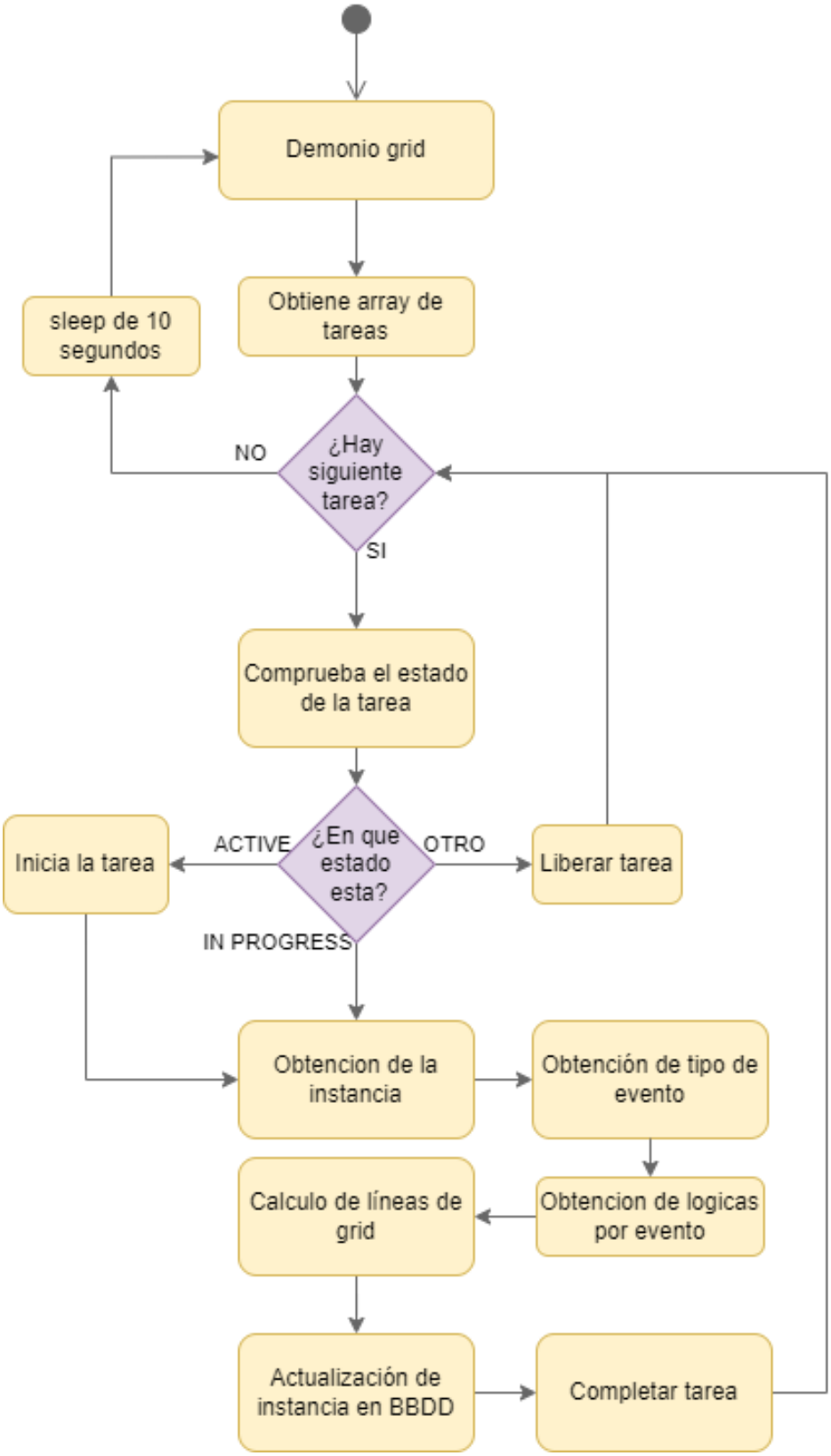


Figura 14. Diagrama demonio grid.

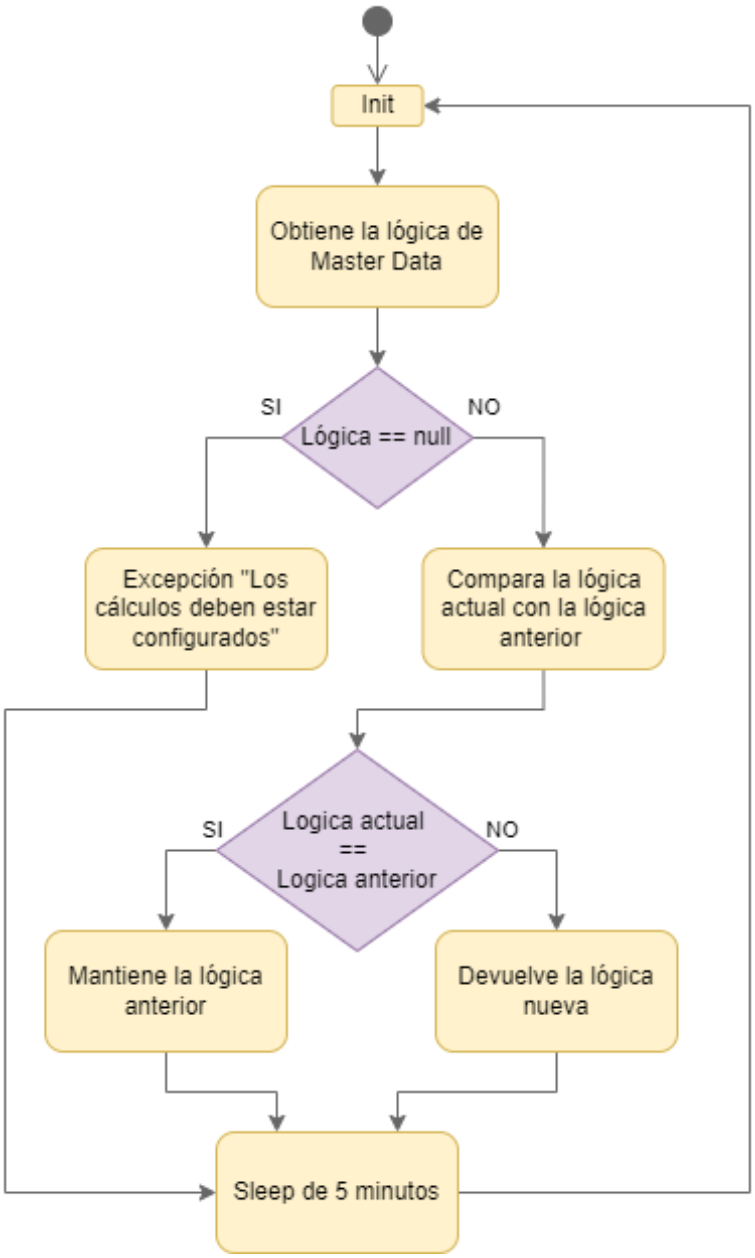


Figura 15. Carga de lógicas.

6.2. Detalle de secciones y campos

Esta sección detalla las secciones y campos que consumirá el servicio de generación de *grid* para su posterior tratamiento ya sea para el producto PTM (alta y eventos) o TTF.
Se mostrarán las secciones más representativas que componen el modelo de datos dado su gran tamaño.

6.1.1. PTM (Préstamos Sindicados)

El sistema consume un objeto JSON que contiene las siguientes secciones

newPTM		loanReference
		loanDetails
		rolloverDetails
		system
		interestDetails
		mgmtInfo
		sttlementDetails
		funding
		midasNYData
		amigaAsiaData
		amigaNYData
eventPTM	**newEventPtm**	interestDetails
		rolloverDetails
		funding
		systems
		midasNYData
		settlementDetails
		amigaAsiaData
		decreaseData
		amigaNYData
		rolloverData
		increaseData
		cancelData

Tabla 4. Secciones PTM.

A continuacion se muestran a modo de ejemplo dos instancias consumidas por el servicio demonio:

```
"businessData": {
    "taskTypeId": "altaPtm",
    "instanceId": "1633619574053a2554d14ee96",
    "iniciador": "XE82089",
    "xfonData": {
        "instanceId": "1633619574053a2554d14ee96",
        "iniciador": "XE82089",
        "businessId": "PTM-ALTA-1633619574053",
        "operationType": "alta",
        "processType": "PTM",
        "status": "IN_PROGRESS",
        "ptmProcess": {
            "newPtm": {
                "rolloverDetails": {"dayNum": ""...},
                "funding": {"referenceIndex": "LIBOR"...},
                "loanDetails": {"amount": 1500.78...},
                "settlementDetails": {"acPayment": "01"...},
                "amigaAsiaData": {},
                "mgmtInfo": {"bookCode": ""...},
                "loanReference": {"loanSubtype": "ER"...},
                "supervisorComments": "",
                "interestDetails": {"margin": 1.5...},
                "systems": {"opInClan": "YES"...},
                "midasNYData": {"loanType": ""...},
                "amigaNYData": {},
                "validatorComments": ""
            },
            "ptmID": ""
        }
    },
    "businessId": "PTM-ALTA-1633619574053",
    "action": "accept"
}
```


Figura 16. JSON alta PTM.

Figura 17. Evento PTM.

Remarcando que los objetos de Midas y Amiga solo contendrán atributos en los casos en los que la operación provenga de una geografía perteneciente a Asia o Nueva York

Dentro del objeto "**xfonData**" podemos encontrar los campos "**operationType"** y "**processType"** que nos indican el tipo de producto (PTM o TTF) y la operación a realizar (alta o eventos). Con estos datos el demonio generara las líneas de grid pertinentes para cada instancia

Una vez realizado el tratamiento de datos por parte del demonio la instancia contendrá un objeto grid tipo array conteniendo todos los datos necesarios para la consumición del servicio.

6.1.2. TTF (Financiación Internacional)

El sistema consume un objeto JSON proveniente de base de datos con las siguientes secciones dentro del proceso ttfProcess:

newTTF	midasNYDataTtf
	formulaTtf
	customerPaymentDetailsTtf
	economicInformationTtf
	transactionTypeDetailsTtf
	primaTtf
	additionalInformationTtf
	amigaNYdataTtf
	rolloverTtf
	mt103Ttf
	rateTtf
	mt202Ttf
	economicInformationTotalTtf
	systemsTtf

Tabla 5. Secciones TTF.

A continuación, se muestra una figura que ejemplifica el modelo de datos de las instancias de TTF:

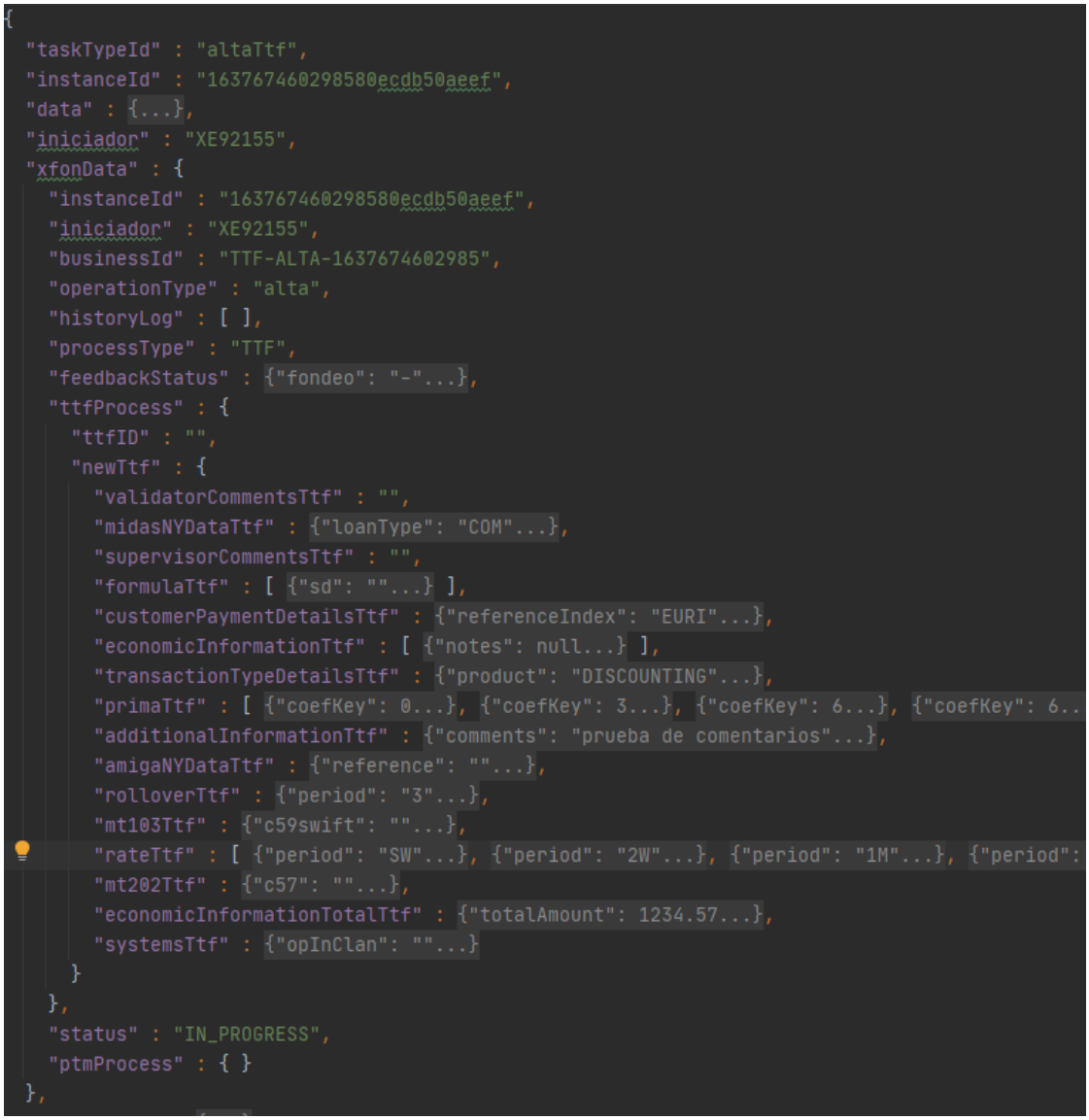


Figura 18. Ejemplo instancia TTF.

6.3. Líneas de Grid

En este apartado se expondrá el funcionamiento del demonio para los distintos procesos y productos que consumirá, como los atributos que consume y sus lógicas propias.

6.3.1 Alta PTM

El demonio consume los datos requeridos para el préstamo, su validación por parte del supervisor, con estos datos generara un array de atributos necesarios para ser consumidos por servicios posteriores y generación de hojas de financiación.

Pare el proceso de **alta** el demonio debe generar la información necesaria con los datos obtenidos de la instancia de alta para que el siguiente servicio pueda generar la las hojas de financiación referentes a este producto.

En el caso del **alta de PTM el grid generara una única línea** con más de 100 atributos por lo que se mostraran en la siguiente tabla las **lógicas más complejas y/o interesantes** a nivel de proyecto de modo que ilustren de forma significativa el funcionamiento del proceso.

Atributo GRID	Ruta del atributo/s que consume	Lógica BPM - ALTA	Ejemplo entrada	Ejemplo salida
room	#loanReference.entity	Elimina prefijos	058 - MILAN	“MILAN”
branchId	#loanReference.branch	Extrae solo el número	209 - NEW YORK	“209”
principalAmount	#loanDetails.amount	Si amount es distinto de null se guarda con dos decimales. Si amount es null se guarda como un String vacío	15.1	“15.10”
maturityDate	#loanReference.operationType #rolloverDetails.rolloverDate #loanDetails.MaturityDate	Si Operation Type = RCF y Rollover Date distinto de null, devuelve Rollover Date con formato ("dd/MM/yyyy") si no Si MaturityDate es distinto de null, devuelve MaturityDate con formato ("dd/MM/yyyy") sino devuelve un String vacío	1654623432	“07/06/2022”
prima	#funding.primaCorporativa #funding.extrabasisPrima	Si primaCorporativa es distinto de null, entonces se suma primaCorporativa + primaExtrabasis	0.1 0.04	“0.14”

tipoTotal	#funding.fundingRate	Si Funding Rate es distinto de null --> Si Funding Rate = -0.2 --> -0.20000 Si Funding Rate = 0 --> "0.00001" sino setear el valor de Funding Rate sino ""	1.02	“1.12000”

Figura 19. Lógicas PTM.

A continuación, se muestra un ejemplo del objeto grid obtenido tras el tratamiento de la instancia:

```
"grid" : [ {
  "bookCode" : "",
  "daysToRollover" : "",
  "accionEjecutarStar" : "ALTA NUEVA",
  "contractType" : "C",
  "facilityMaturityDate" : "2022/06/01",
  "eligibility" : "",
  "productFundingType" : "DRD",
  "acReceipt" : "01",
  "settlementAmountRCF" : "1500.78",
  "importeAmiga" : "",
  "scf" : "",
  "gtf" : "",
  "interestBasis" : "1",
  "eventTypeOperationType" : "Alta_RCF",
  "sendBefore12" : "N",
  "rolloverDateFunding" : "2020/10/22",
  "instruccionAmiga" : "",
  "branchId" : "701",
  "beneficiaryForfaitingSupplier" : "",
  "period" : "1M",
  "prima" : "0.14",
  "tipoTotal" : "1.12000",
  "fechaValorAmigaNYData" : "",
  "customerName" : "CUSTOMER NUM.103408",
  "autorizador" : "PRESTAMOS SINDICADOS",
  "externalReference" : "",
  "baseRate" : "0.98",
  "loanTypeBic" : "",
  "indFacilityExRate" : "D",
  "specialVigilance" : "NO",
  "operationType" : "RCF",
  "subType" : "RCF",
  "rpsc" : "false",
  "tipoMensaje" : "",
```


Figura 20. Ejemplo generación grid ALTA PTM.

6.3.2 Alta TTF

TTF es la abreviatura del producto de financiación internacional, del cual abordaremos a continuación el tratamiento de las lógicas y campos que generara el demonio sobre las instancias de alta.

El tratamiento a realizar una vez incluido y validados los datos de la operación, JBPM debe generar la información necesaria para que el robot pueda ejecutar los pasos en MIDAS o AMIGA, además de que JBPM genere las hojas de FONDEO. Actualmente

BPM genera las líneas del grid con los campos requeridos y los muestra en pantalla en una tabla plana.

A continuación, al igual que en el proceso de alta PTM se muestra una tabla con los atributos más ilustrativos del del funcionamiento de lógicas para el servicio:

Atributo GRID	Ruta del atributo	Lógica BPM - ALTA	Ejemplo entrada	Ejemplo salida
loanSubtype	#loanSubtype	Extrae las dos últimas letras del atributo	“15 - LR”	“LR”
interestBasis	#customerPaymentDetails.midasBaseCode	Campo midasBaseCode si existe, si no “”	“”	“”
floor	#transactionTypeDetails.floor0	Si el campo floor0 existe y es true entonces YES, de lo contrario NO	false	“NO”
amigaId	#customerPaymentDetails.ssiAmiga	Si existe el campo ssiAmiga coger el valor numérico anterior al guión	12 – Test	“12”

Tabla 6. Lógicas TTF

A continuación, se muestra ejemplo del objeto grid tras la realización del alta TTF:

```
"grid" : [ {
  "loanSubtype" : "LR",
  "autoSettlements" : "N",
  "bookCode" : "",
  "rolloverDayNumber" : "3",
  "accionEjecutarStar" : "ROLL-OVER",
  "businessId" : "TTF-ALTA-1637674602985",
  "acReceipt" : "01",
  "eventTypeOperaitonType" : "Alta_TTF",
  "loanTypebic" : "COM",
  "importeAmiga" : "1234.57000",
  "maturityDate" : "2021/12/30",
  "gtf" : "",
  "interestBasis" : "2",
  "sendBefore12" : "N",
  "valueDateAmiga" : "1970/01/15",
  "branchId" : "058",
  "period" : "1M",
  "campo70" : "prueba campo c70",
  "customerNumberFacility" : "191775",
  "methodReceipt" : "01",
  "prima" : "0.0100000",
  "tipoTotal" : "1234.57000",
  "methodPayment" : "01",
  "campo72" : "prueba campo c72",
  "customerName" : "ENI SPA",
  "autorizador" : "HUB TRADE FINANCE",
  "externalReference" : "SCFundefined",
  "indFacilityExRate" : "",
  "specialVigilance" : "NO",
```


Figura 21. Ejemplo generación de grid alta TTF.

6.3.3 Eventos PTM

El proceso de eventos consiste en generar una modificación, fusión, aplicación o reducción de las operaciones ya existentes, pudiendo ser dichas modificaciones sobre fechas, importes, margen o cancelación sean sobre instancias dadas de altas o instancias que ya tengan un evento ejecutado.

Dependiendo de la tipología de eventos dividimos en:

- **Rollover Standard**: Se refiere al proceso de prórroga de la fecha de vencimiento del préstamo.
- **Rollover Increase**: Aumento del importe acordado en la operación y por lo tanto genera una nueva fecha de Rollover.
- **Rollover Decrease**: Reducción del importe acordado en la operación.

- **Cancelación Total**: Adelanta la fecha de finalización (campo *maturityDate*) de la operación.

Estas operaciones se realizarán únicamente para el servicio PTM.

Como en el caso de alta PTM cada uno de estos eventos tendrá sus propias lógicas de negocio, y en función del tipo de operación se tendrá en cuenta el numero y tipo de líneas de grid que el demonio generará. Esto se especifica en detalle en el apartado de análisis y requisitos.

```
ptm:
  rolloverIncreaseCalculations: >
    {
       "PRIME":{
          "historyLineCalculations":null,
          "subType":"RENOVACION_INCREASE_PRIME",
          "newLineCalculations":[
             {
                "fieldName":"increaseAmount",
                "expression":"#utils.notEmpty(#businessData.xfonData.ptmProcess.eventP
             },
             {
                "fieldName":"businessId",
                "expression":"#businessData.businessId"
             },
             {
                "fieldName":"ourReference",
                "expression":"#utils.notEmpty(#businessData.xfonData.ptmProcess.eventP
             },
             {
                "fieldName":"subType",
                "expression":"#subType"
             },
```

Figura 22. Lógicas RolloverIncrease PRIME.

7. ANALISIS Y REQUISITOS

Este apartado analiza el funcionamiento del sistema y extrae los requisitos que se han de cumplir para el correcto funcionamiento del mismo.

- **Funcionales**

  - Requisito Funcional 1

Nombre	Array grid
Prioridad	Alta
Importancia	Alta
Descripción	El demonio generara sobre cada instancia un objeto ‘grid’ tipo array con los datos necesarios para cada instancia.

Tabla 7. Requisito Funcional 1

  - Requisito Funcional 2

Nombre	Tipado de atributos
Prioridad	Alta
Importancia	Alta
Descripción	Cada atributo del grid será del tipo String, independientemente de la entrada de las lógicas, y en caso de ser un valor nulo se devolverá como un String vacío.

Tabla 8. Requisito Funcional 2

  - Requisito Funcional 3

Nombre	Alta PTM
Prioridad	Alta
Importancia	Alta
Descripción	El demonio genera una línea con lógicas propias para el servicio de alta PTM.

Tabla 9. Requisito Funcional 3

  - Requisito Funcional 4

Nombre	Rollover Standard PTM
Prioridad	Alta
Importancia	Alta
Descripción	Para el servicio rollover standard, el demonio generará un array de strings.

Tabla 10. Requisito Funcional 4

- Requisito Funcional 5

Nombre	Rollover Increase PTM
Prioridad	Alta
Importancia	Alta
Descripción	Para el servicio rollover Increase, existen tres posibilidades según el tipo de operación: • TL El demonio generara dos líneas de grid una con los datos históricos de la operación y otra con los datos relativos a la operación 'increase' • RCF Únicamente genera una línea de grid con los datos relativos a la operación 'increase' • PRIME Únicamente generara una línea de grid con los datos relativos a la operación 'increase'

Tabla 11.Requisito Funcional 5

- Requisito Funcional 6

Nombre	Rollover Decrease PTM
Prioridad	Alta
Importancia	Alta
Descripción	Para el servicio rollover decrease existen 3 tipos de operación según el tipo de operación: • TL El demonio genera dos líneas, una con los datos históricos de la operación y otra con los datos relativos a la operación 'decrease' • RCF Únicamente se genera una línea de grid con los datos relativos a la operación • PRIME Únicamente se genera una línea de grid con los datos relativos a la operación

Tabla 12. Requisito Funcional 6

- Requisito Funcional 7

Nombre	Cancelación Total PTM
Prioridad	Alta
Importancia	Alta
Descripción	Para el servicio de cancelación total se genera una línea de grid con los datos relativos a la cancelación del servicio PTM

Tabla 13. Requisito Funcional 7

- Requisito Funcional 8

Nombre	Alta TTF
Prioridad	Alta
Importancia	Alta
Descripción	Para el servicio de alta TTF se genera una línea de grid con los datos relativos a la operación.

Tabla 14. Requisito Funcional 8

- **No funcionales**

  - Requisito No Funcional 1

Nombre	SleepDaemon
Prioridad	Alta
Importancia	Alta
Descripción	El servicio demonio se ejecutará cada 10 segundos

Tabla 15. Requisito No Funcional 1

  - Requisito No Funcional 2

Nombre	Actualización Lógicas
Prioridad	Alta
Importancia	Alta
Descripción	Las lógicas se actualizarán cada 5 minutos

Tabla 16. Requisito No Funcional 2

  - Requisito No Funcional 3

Nombre	Actualización Lógicas
Prioridad	Alta
Importancia	Alta
Descripción	Las lógicas se actualizarán cada 5 minutos

Tabla 17. Requisito No Funcional 3

8. GESTION DE PROYECTO

8.1. Implementación

Este capítulo abarca todo lo relacionado con la implementación y el desarrollo del sistema, así como su configuración previa.

8.1.1. Java

Para comenzar el desarrollo del demonio se dispone de la herramienta NovaCLI. Se trata de un CLI (Command Line Interface) el cual permite generar en nuestro puesto local el código inicial para empezar el desarrollo del servicio demonio.

Figura 23 Generación servicio con NovaCLI.

El servicio demonio que se va a implementar consta de los pasos definidos y diseñados anteriormente (véase Figura 33).

El primer paso inyectar las dependencias de los servicios que consume el demonio en cuestión gracias a la anotación @Autowired. Para este desarrollo principalmente necesitamos los servicios de que nos permiten obtener tareas de los procesos PTM y TTF y su posterior persistencia en la base de datos.

```
@Autowired
private IServiceXfonservices serviceXfonservices;
```

Figura 24. Inyección de dependencias Spring.

Una vez inyectadas las dependencias, se realiza una configuración sobre las mismas.

```java
@Override
public String completeTask(final String requestTaskComplete, final String taskId, final String nsId) throws XFONException
{
    LOG.debug("[ServiceXfonservices] -> [completeTask]");

    LOG.debug("[ServiceXfonservices] -> [completeTask]: novaMetadata  [{}]", this.novaMetadata.toString());

    // Do the call to endpoint via RestHandler
    return  ClientUtils.manageException(()->this.restHandlerXfonservices.completeTask(this.novaRequestConfig, this.novaMetada
}



@Override
public String searchTasksPrioritized(final String nsId, final String q, final Boolean isFullSearch, final Integer pageSize, f
{
    LOG.debug("[ServiceXfonservices] -> [searchTasksPrioritized]");

    LOG.debug("[ServiceXfonservices] -> [searchTasksPrioritized]: novaMetadata  [{}]", this.novaMetadata.toString());

    // Do the call to endpoint via RestHandler
    return  ClientUtils.manageException(()->this.restHandlerXfonservices.searchTasksPrioritized(this.novaRequestConfig, this.
}
```

Figura 25. Configuración de servicio.

Una vez tenemos todos los servicios necesario inyectados se desarrolla el funcionamiento del servicio demonio. Para este servicio una anotación importante seria *@Scheduled (fixedDelayString = "10000")* que nos permite asignar el tiempo de espera entre una ejecución y la siguiente del proceso, en este caso cada 10 segundos como se contempla en los requerimientos del servicio. Este proceso demonio seguirá el flujo propuesto en el diagrama mostrado en el diseño del sistema.

Es necesario crear los *DTOs* necesarios para manejar las reglas de negocio por parte del demonio.

```java
@Getter
@Setter
public class CalculatedFieldDTO {
    private String fieldName;
    private String value;
}
```

Figura 26. DTO campos del grid.

En este caso el *DTO* es sencillo ya que los campos que rellenara el *grid* solo tendrán un nombre de campo y un valor, que se calcula en tiempo de ejecución con *SpEL*.

Otro de los métodos importantes que se comenta es el de generación del *grid*, que como se explica en el diagrama de flujo, obtiene las lógicas en función del tipo de evento y subType de la instancia

```java
@Override
public JSONArray generateGrid(GenericObjectDB instance) throws XFONException {
    JSONObject businessData = instance.getBusinessData();
    EventCalculations eventCals= getEventCalculations(businessData);
    String subType=eventCals.getSubType();
    List<Map<String, Object>> historyLines=null;
    List<List<Map<String,Object>>> previousGrids=(List<List<Map<String, Object>>>) jsonUtils.getJsonAttribute(businessData,
    if(previousGrids!=null&&!previousGrids.isEmpty()) {
        historyLines = previousGrids.stream().flatMap(List::stream).sorted(this::compareByDealDate).collect(Collectors.toList());
    }
    return doGenerateGrid(businessData, eventCals, subType, historyLines);
}
```

Figura 27. Método generateGrid

Todo el desarrollo incluye diferentes ficheros de configuración entre ellos los ficheros application.yml y application-LOCAL.yml con la configuración necesaria para los distintos entornos de ejecución, estos no se mostrarán debido a que contienen datos sensibles de negocio.

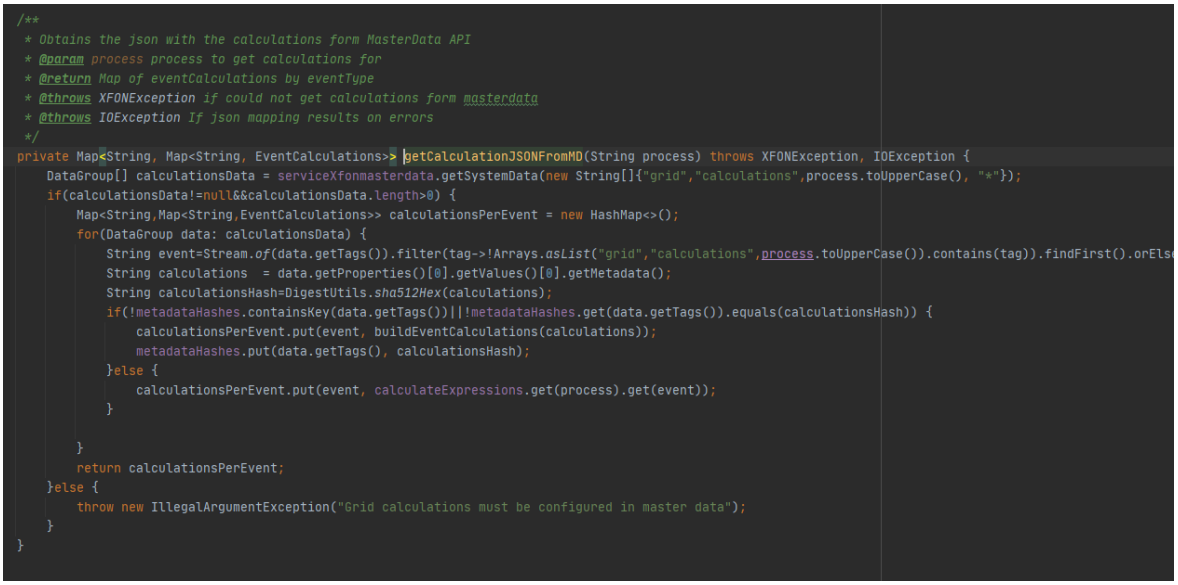
Adicionalmente existe un fichero con extensión .yml para cada proceso soportado por el demonio de grid, estos ficheros contendrán las lógicas implementadas que serán útiles para el desarrollo de los test las pruebas en el entorno local.

Por otro lado, existe un proceso demonio que consume las lógicas cada 5 minutos. Este es relativamente más sencillo ya que solo tiene que comprobar cada 5 minutos si estas han cambiado, y de ser así actualizar estas expresiones para que el servicio principal (demonio de grid) realice esta tarea con las lógicas actualizadas

```java
/**
 * Initialization Method
 * @throws ParseException If calculations JSONs are not valid
 */
@Scheduled(fixedDelay=300000)
@PostConstruct
public void init() throws ParseException {
    try {
        ptmBranchesTab = buildBranchesMap( serviceXfonmasterdata.getSystemData(new String[]{"ptm","branches"}));
        ttfBranchesTab = buildBranchesMap( serviceXfonmasterdata.getSystemData(new String[]{"ttf","branches"}));
        ptmAccountsTab = buildAccountMap(serviceXfonmasterdata.getSystemData(new String[]{"ptm","cuentaCoAmiga"}));
        ttfAccountsTab = new HashMap<>();
        calculateExpressions= new HashMap<>();
        calculateExpressions.put("PTM",getCalculationJSONFromMD( process: "PTM"));
        calculateExpressions.put("TTF",getCalculationJSONFromMD( process: "TTF"));
    } catch (XFONException | IOException  e) {
        throw new BeanInitializationException("Error parsing calculations JSON", e);
    }
}
```

Figura 28. Demonio para la obtención de lógicas.

A la hora de comparar las lógicas actuales que almacena el grid, con las posibles actualizaciones realiza una función *hash* criptográfica sobre ambas para comparar esta salida en lugar de comparar una a una todas las lógicas, si el hash generado es distinto implica que hay un cambio de lógicas por lo que se realizara la actualización.



```
/**
 * Obtains the json with the calculations form MasterData API
 * @param process process to get calculations for
 * @return Map of eventCalculations by eventType
 * @throws XFONException if could not get calculations form masterdata
 * @throws IOException If json mapping results on errors
 */
private Map<String, Map<String, EventCalculations>> getCalculationJSONFromMD(String process) throws XFONException, IOException {
    DataGroup[] calculationsData = serviceXfonmasterdata.getSystemData(new String[]{"grid","calculations",process.toUpperCase(), "*"});
    if(calculationsData!=null&&calculationsData.length>0) {
        Map<String,Map<String,EventCalculations>> calculationsPerEvent = new HashMap<>();
        for(DataGroup data: calculationsData) {
            String event=Stream.of(data.getTags()).filter(tag->!Arrays.asList("grid","calculations",process.toUpperCase()).contains(tag)).findFirst().orEls
            String calculations  = data.getProperties()[0].getValues()[0].getMetadata();
            String calculationsHash=DigestUtils.sha512Hex(calculations);
            if(!metadataHashes.containsKey(data.getTags())||!metadataHashes.get(data.getTags()).equals(calculationsHash)) {
                calculationsPerEvent.put(event, buildEventCalculations(calculations));
                metadataHashes.put(data.getTags(), calculationsHash);
            }else {
                calculationsPerEvent.put(event, calculateExpressions.get(process).get(event));
            }

        }
        return calculationsPerEvent;
    }else {
        throw new IllegalArgumentException("Grid calculations must be configured in master data");
    }
}
```

Figura 29. Obtención de lógicas.

8.1.2. Spring Expression Languaje (SpEL)

La implementación de las lógicas del demonio corre a cargo de SpEL como se mencionó anteriormente. Para ambos productos (PTM y TTF) se han desarrollado unas lógicas especificas en función del tipo de evento (alta, increase, decrease o cancelación total), para ello el demonio almacena en memoria las expresiones.

Una vez ha extraído la instancia sobre la que va a generar el objeto grid consultará la sección determinada por el campo ‘eventType’ y procederá a generar sus lógicas.

```
{
   "fieldName": "clientNameProducto",
   "expression": "''"
},
{
   "fieldName": "unidad",
   "expression": "'PTM'"
},
{
   "fieldName": "subType",
   "expression": "#utils.notEmpty(#businessData.xfonData.ptmProcess.newPtm.loanReference,'operationType')?#businessData.xfonData.ptmProcess.newPtm.loanReference.operationT
},
{
   "fieldName": "subUnidad",
   "expression": "#utils.notEmpty(#businessData.xfonData.ptmProcess.newPtm.loanReference,'subunidad')?#businessData.xfonData.ptmProcess.newPtm.loanReference.subunidad:''"
},
```

Figura 30. Lógicas SpEL.

Estas lógicas son un array de objetos con un campo y una expresión (véase Figura 26) que Java ejecuta en el mismo contexto. Por lo que este campo se autocompleta siguiendo unas reglas o expresiones preconfiguradas.

Como ejemplo de lógica para explicar el funcionamiento de SpEL tomaremos el campo “interestBasis”

Figura 31. Ejemplo SpEL.

La estructura es la del DTO (véase Figura 26), compuesta por "fieldName" y "expression". La primera parte de esta expresión es *#utils.notEmpty* que es un método Java que permite evaluar un atributo en concreto para comprobar si este existe o no. Por otro lado, la estructura de esta expresión es equivalente a una condición *if then*. Sin embargo, la nomenclatura es distinta:

```
[Condicion] ? [condicin cumplida] : [condicion no cumplida]
```

Figura 32. Esquema de lógicas.

En este caso comprueba que el atributo "midasCode" exista, de ser así lo registra como valor de la expresión y de no ser así registra un String vacío (´´).

Figura 33.Ejemplo 2 lógica de facilityMaturityDate

El ejemplo de la figura (véase Figura 33) es más completo. En primer lugar se observa que consume los atributos relativos al mapa *businessData* sobre el que se aplica una comparación, siguiendo a este mapa esta la inyección del método Java *utils.notEmpy* que comprueba si el objeto esta vacío y además se observa que SpEL permite ejecutar código java directamente. En este caso observamos como creamos un nuevo objeto *SimpleDateFormat* y se realiza un formato de fecha sobre el atributo *maturityDateFunding.*

8.2. PLAN DE VALIDACIÓN

El objetivo del plan de validación consiste en la realización de test unitarios, con el objetivo de comprobar el correcto funcionamiento de cada uno de los componentes del demonio implementado.

Otro objetivo importante en el plan de validación es la complejidad del sistema evitando el uso de ciclos redundantes en el código o eliminando código muerto.

8.2.1. Test unitarios

En primer lugar, se resuelven los test unitarios en el entorno local, con Mockito framework.

Mockito permite realizar aserciones sobre los distintos métodos desarrollados en el sistema. Para ello inyecta "servicios mockeados" que simulan el funcionamiento del servicio real.

```
@Test
public void scheduledActiveTest() throws XFONException, CommonPersistException404, ParseException, CommonPersistException400, CommonPersistException500,
    when(serviceXfonservices.searchTasksPrioritized(any(), any(), any(), any(), any(), any(), any(), any(), any())).
            thenReturn("[{\"id\":\"taskId\",\"businessId\":\"testBusinesId\",\"status\":{\"id\": \"ACTIVE\"}}]");
    when(persistenceServices.getInstanceByBusinessId(any())).thenReturn(Utils.buildResponseEntity( processType: "PTM",  operationType: "rolloverIncrease"));
    //when(iGridGenerationService.generateGrid(any())).thenReturn();
    scheduledTask.gridDaemon();
    verify(serviceXfonservices,times( wantedNumberOfInvocations: 1)).completeTask(any(), any(), any());
    verify(persistenceServices,times( wantedNumberOfInvocations: 1)).updateInstanceById(any(),any());
}

@Test(expected=XFONException.class)
public void shedduledGenerateGridError() throws CommonPersistException404, XFONException, ParseException, StartTaskException404, SearchTasksPrioritizedExc
    when(serviceXfonservices.searchTasksPrioritized(any(), any(), any(), any(), any(), any(), any(), any(), any())).
            thenReturn("[{\"id\":\"taskId\",\"businessId\":\"testBusinesId\",\"status\":{\"id\": \"ACTIVE\"}}]");
    when(persistenceServices.getInstanceByBusinessId(any())).thenReturn(Utils.buildResponseEntity( processType: "PTM",  operationType: "rolloverIncrease"));
    when(iGridGenerationService.generateGrid(any())).thenThrow(new XFONException());
    when(serviceApi.releaseTask(any(),any())).thenReturn(null);
    scheduledTask.gridDaemon();

}
```

Figura 34. Ejemplo test demonio de grid.

Podemos observar en la figura (Figura 34) como podemos devolver un *String* determinado con "*when(metodo).thenReturn(salida esperada)*" . Destacar también el hecho de que los test unitarios no abarcan únicamente los casos de uso correcto, también hay que abordar las excepciones y verificar un correcto comportamiento en estos casos.

8.2.2. Resultados y comparativas de los test unitarios

A continuación, se evalúan los resultados obtenidos por los test unitarios teniendo en cuenta tanto las entradas como las salidas de datos.

La ejecución de los test devuelve el estado de los mismos en función de su ejecución.

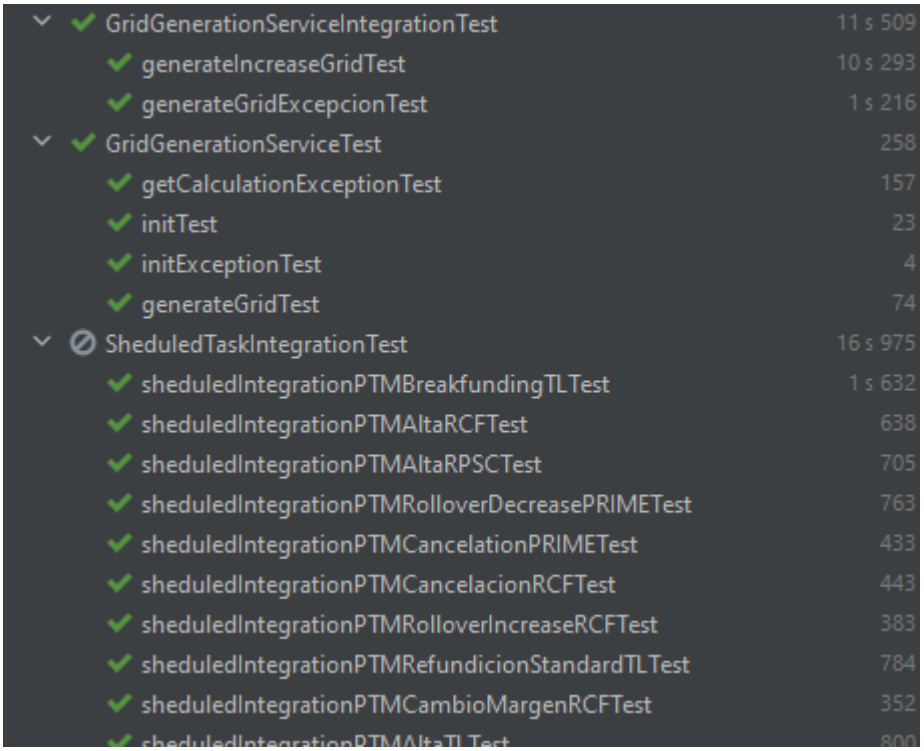


Figura 35. Resultados de los test unitarios.

La ejecución de los test también permite analizar el porcentaje de líneas de código que están evaluadas en estos. Esto garantiza la cantidad de código que ha sido probado, aumentando la calidad del mismo y garantizando unos estándares mínimos de buenas prácticas de cara al cliente.

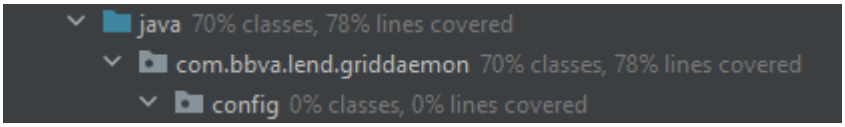


Figura 36. Porcentaje de cobertura del sistema.

8.2.3. Cobertura Global (SonarQube)

Una vez desarrollados los test unitarios y comprobando que el desarrollo cumple un nivel de cobertura aceptable por el cliente (en torno al 70%), evaluaremos la calidad del código con la herramienta ***sonarQube*** el cual garantiza que se cumplan unos estándares mínimos en seguridad y buenas prácticas.

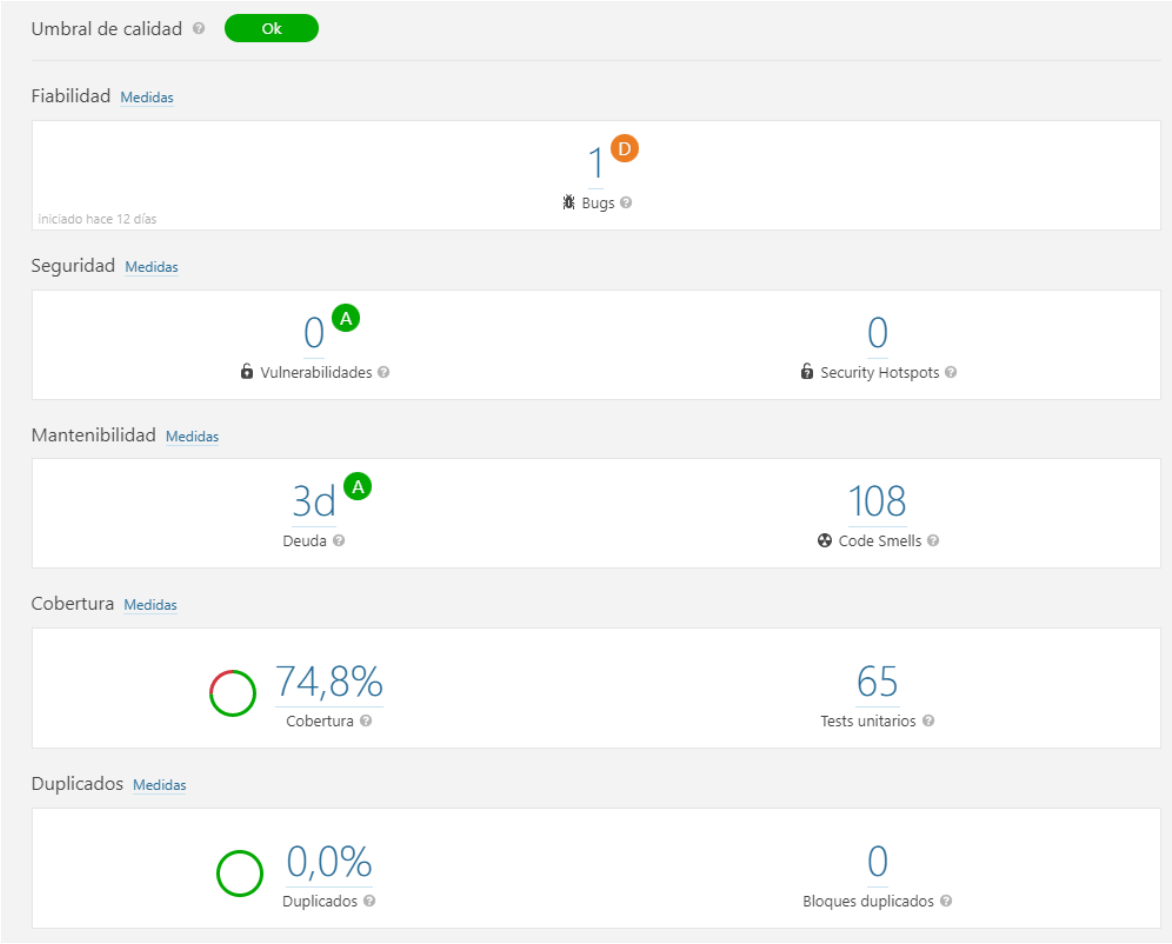


Figura 37. Dashboard de SonarQube.

8.2.4. Despliegue en entornos previos

Una vez validado el sistema se lanza, el dashboard de Nova permite visualizar los *logs* del servicio para capturar posibles errores en el funcionamiento del proceso. En entornos previos se harán pruebas por parte del equipo de desarrollo y con el cliente para verificar el correcto funcionamiento del sistema.

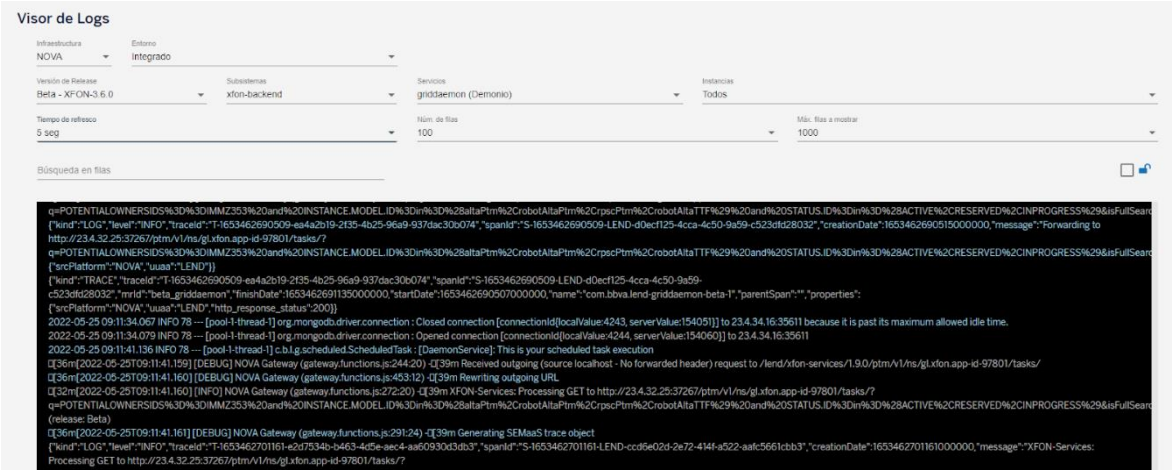


Figura 38.Visor de logs de NOVA.

9. CONCLUSIONES Y TRABAJOS FUTUROS

9.1. Conclusiones

Tal y como se describe en los objetivos, el Trabajo Fin de Grado consiste en evaluar y analizar un sistema de automatización de reglas de negocio, posteriormente diseñar e implementar una solución a este problema y finalmente realizar un plan de validación y pruebas. En este aspecto podemos destacar que se han cumplido todos los objetivos, todo ello queda documentado en el mismo TFG.

A nivel de conocimientos aprendidos se puede destacar ciertos aspectos relacionados con el conocimiento de grado, por ejemplo, desarrollo web o distintos métodos organizativos.

En cuanto a la organización cabe destacar el uso de Scrum como metodología ágil. Este proceso iterativo ha sido fácil de implementar gracias al conocimiento de las distintas Técnicas Agiles de Desarrollo de Software (TADS)

El conocimiento adquirido en la carrera en especial Java aplicado al desarrollo web como en Tecnologías Informáticas para la Web también ha sido de gran ayuda para asentar conocimientos sobre el desarrollo de aplicaciones y servicios web como el desarrollado en este TFG.

9.2. Trabajos futuros

En cuanto a las líneas futuras sobre este desarrollo, existen infinidad de posibilidades de escalar este tipo de productos.

Es interesante la escala del servicio a nuevas líneas de negocio, al estar desarrollado con expresiones regulares ejecutadas en tiempo de ejecución, hace que su escalabilidad aumente y su mantenimiento sea más ágil y sencillo, ya que podemos añadir nuevas líneas de negocio y este servicio podría ser actualizable con pocos cambios.

Por otro lado, es interesante el funcionamiento de SpEL en otro tipo de servicios, ya que podemos establecer diferentes configuraciones de salida utilizando distintos marcadores de servicio sobre las mismas instancias.

REFERENCIAS

[1] A. Escribano, «TelefonicaTech,» 1 Marzo 2022. [En línea]. Available: https://empresas.blogthinkbig.com/evolucion-definitiva-hacia-digitalizacion-industria/. [Último acceso: 10 Junio 2022].

[2] J. M. Bentué, «Universitat Oberta de Catalunya,» 21 Abril 2021. [En línea]. Available: https://blogs.uoc.edu/economia-empresa/es/el-reto-de-la-digitalizacion-para-la-banca-post-covid/. [Último acceso: 1 Marzo 2022].

[3] M. Presta, «back4app,» 2022. [En línea]. Available: https://blog.back4app.com/es/los-diez-mejores-lenguajes-de-backend/. [Último acceso: 12 Abril 2022].

[4] «Java,» [En línea]. Available: https://www.java.com/es/. [Último acceso: 20 Abril 2022].

[5] A. Robledano, «OpenWebinars,» 12 Agosto 2019. [En línea]. Available: https://openwebinars.net/blog/que-es-java/. [Último acceso: 15 Mayo 2022].

[6] TIOBE, «TIOBE java,» 13 Marzo 2022. [En línea]. Available: https://www.tiobe.com/tiobe-index/java/. [Último acceso: 13 Marzo 2022].

[7] «developer.mozilla.org,» [En línea]. Available: https://developer.mozilla.org/es/docs/Web/JavaScript. [Último acceso: 20 Abril 2022].

[8] G. González, «Genbeta,» 30 Abril 2021. [En línea]. Available: https://www.genbeta.com/desarrollo/javascript-tiene-comunidad-casi-14-millones-programadores-mitad-todos-desarrolladores-mundo#:~:text=Estiman%20que%20para%20el%20primer,utilizan%20en%20todo%20el%20mundo.. [Último acceso: Mayo 14 2022].

[9] E. Geek, «ifgeekthen,» 9 Marzo 2021. [En línea]. Available: https://ifgeekthen.nttdata.com/es/que-es-expressjs-y-primeros-pasos. [Último acceso: 11 Mayo 2022].

[10] J. Lucas, «openWebinars,» 4 Septiembre 2019. [En línea]. Available: https://openwebinars.net/blog/que-es-nodejs/. [Último acceso: 11 Mayo 2022].

[11] TIOBE, «TIOBE javascript,» 13 Marzo 2022. [En línea]. Available: https://www.tiobe.com/tiobe-index/javascript/. [Último acceso: 13 Marzo 2022].

[12] «Python,» [En línea]. Available: https://www.python.org/. [Último acceso: 20 Abril 2022].

[13] R. KeepCoding, «keepCoding,» 21 Enero 2022. [En línea]. Available: https://keepcoding.io/blog/desarrollo-web-con-python/. [Último acceso: 11 Mayo 2022].

[14] TIOBE, «TIOBE python,» 13 Marzo 2022. [En línea]. Available: https://www.tiobe.com/tiobe-index/python. [Último acceso: 13 Marzo 2022].

[15] «PHP,» [En línea]. Available: https://www.php.net/. [Último acceso: 20 Abril 2022].

[16] I. d. Souza, «Rockcontent,» 9 Marzo 2020. [En línea]. Available:

] https://rockcontent.com/es/blog/php/. [Último acceso: 14 Abril 2022].

[17 TIOBE, «TIOBE php,» 13 Marzo 2022. [En línea]. Available:] https://www.tiobe.com/tiobe-index/php/. [Último acceso: 13 Marzo 2022].

[18 «TicPortal,» 22 Junio 2017. [En línea]. Available: https://www.ticportal.es/glosario-] tic/waterfall-metodologia-desarrollo-secuencial. [Último acceso: 14 06 2022].

[19 Felipe, «Hostingplus,» 6 Julio 2021. [En línea]. Available:] https://www.hostingplus.com.es/blog/modelo-de-prototipos-que-es-y-cuales-son-sus-etapas/. [Último acceso: 12 Junio 2022].

[20 Denise Sanchez, Carmen Gerea, «Freed,» 15 Marzo 2021. [En línea]. Available:] https://freed.tools/blogs/ux-cx/prototipo. [Último acceso: 14 Junio 2022].

[21 S. universidades, «Santander Becas,» 21 Diciembre 2021. [En línea]. Available:] https://www.becas-santander.com/es/blog/metodologias-desarrollo-software.html. [Último acceso: 14 Junio 2022].

[22 J. Martins, «asana,» 11 Noviembre 2020. [En línea]. Available:] https://asana.com/es/resources/what-is-kanban. [Último acceso: 14 Junio 2022].

[23 L. A. Garcia, «Inesem,» 29 Octubre 2018. [En línea]. Available:] https://www.inesem.es/revistadigital/gestion-empresarial/kanban-el-metodo-para-desarrollar-proyectos-de-exito/. [Último acceso: 14 Junio 2022].

[24 «Scrum,» [En línea]. Available: https://www.scrum.org/. [Último acceso: 16 Abril] 2022].

[25 J. S. Hurtado, «Iebschool,» 3 Diciembre 2021. [En línea]. Available:] https://www.iebschool.com/blog/metodologia-scrum-agile-scrum/. [Último acceso: 14 Junio 2022].

[26 «BOE,» 27 Abril 2016. [En línea]. Available:] https://www.boe.es/doue/2016/119/L00001-00088.pdf. [Último acceso: 9 Mayo 2022].

[27 «BOE,» 5 Diciembre 2018. [En línea]. Available:] https://www.boe.es/buscar/pdf/2018/BOE-A-2018-16673-consolidado.pdf. [Último acceso: 9 Mayo 2022].

[28 A. Z. Fernandez, «La Vanguardia,» 6 Mayo 2021. [En línea]. Available:] https://www.lavanguardia.com/economia/20210506/7429996/crecimiento-sector-fintech-despega-necesita-profesionales-cualificados-master-eada-brl.html. [Último acceso: 14 Junio 2022].

[29 Redactor Rock Content, «¿Qué es Java? Conoce las particularidades de este lenguaje] de programación,» 5 Junio 2019. [En línea]. Available: https://rockcontent.com/es/blog/que-es-java/. [Último acceso: 15 Marzo 2022].

[30 Y. Muradas, «OpenWebinars,» 5 Junio 2018. [En línea]. Available:] https://openwebinars.net/blog/conoce-que-es-spring-framework-y-por-que-usarlo/. [Último acceso: 11 Abril 2022].

[31 R. Johnson, J. Hoeller, K. Donald, C. Sampaleanu, R. Harrop, A. Arendsen, T.] Risberg, D. Davison, D. Kopylenko, M. Pollack, T. Templier, E. Vervaet, P. Tung, B.

Hale, A. Colyer, J. Lewis, C. Leau, M. Fisher, S. Brannen, R. Laddad, A. Poutsma, C. Beams, T. Abedrabbo, A. Clement, D. Syer y O. Gierke, «Spring Framework,» [En línea]. Available: https://docs.spring.io/spring-framework/docs/3.0.x/reference/index.html. [Último acceso: 15 Abril 2022].

[32] Baeldung, «Baeldung,» 28 Marzo 2022. [En línea]. Available: https://www.baeldung.com/spring-expre. [Último acceso: 3 Junio 2022].

[33] «mockito,» [En línea]. Available: https://site.mockito.org/. [Último acceso: 25 Mayo 2022].

[34] C. Á. Caules, «Arquitectura Java,» 30 Noviembre 2015. [En línea]. Available: https://www.arquitecturajava.com/java-mockito-y-los-mock-objects/. [Último acceso: 11 Abril 2022].

[35] G. Gonzalez, «Genbeta,» 30 Abril 2021. [En línea]. Available: https://www.genbeta.com/desarrollo/javascript-tiene-comunidad-casi-14-millones-programadores-mitad-todos-desarrolladores-mundo#:~:text=Estiman%20que%20para%20el%20primer,utilizan%20en%20todo%20el%20mundo.. [Último acceso: 11 Mayo 2022].