Universidad Internacional de La Rioja

Escuela Superior de Ingeniería y Tecnología

Grado en Ingeniería Informática

El legado de las aplicaciones heredadas

Ubicación del código fuente:

https://github.com/xrodriguezang

Trabajo fin de estudio presentado por:	Rodríguez Angrill, Xavier
Línea de investigación:	Ingeniería en la Web, servicios Web y Seguridad en la Web.
Director/a:	Soltero Domingo, Francisco José
Fecha:	05/07/2021

Resumen

La información es el eje central sobre la cual se diseñan y construyen las aplicaciones en las organizaciones. Normalmente, las aplicaciones están creadas sobre las tecnologías más recientes del mercado. A menudo, transcurren los años y estas tecnologías pasan a estar totalmente obsoletas, donde un mantenimiento ineficiente y una pérdida total de soporte enmarcan un producto en el cual ya no se realizan actualizaciones y apenas se producen mejoras. Aquella información para la cual se crearon sigue siendo estratégica, con lo cual, estas piezas son altamente críticas. Es en este momento cuando estas aplicaciones devienen heredadas o legacy.

Afrontar un proceso de actualización de estas viejas tecnologías es una tarea compleja, a la cual, más tarde o más temprano, deberán hacer frente las organizaciones. Existen diferentes estrategias para planificar un proceso de modernización de una aplicación heredada. Sin embargo, no hay una dirección o una guía efectiva que escriba de las pautas que se deben seguir. Es una tarea compleja a la cual el departamento informático deberá hacer frente y, donde una mala decisión, puede acarrear una solución ineficiente.

Palabras clave: aplicaciones heredadas, protección de sistemas, microservicios, modernización.

Abstract

Information is the central axis on which applications in organizations are designed and built. Typically, applications are created on the latest technologies in the market. Often, the years go by and these technologies become completely obsolete, where inefficient maintenance and a total loss of support form a product which no more updates and improvements are made. The information for they were created remains strategic, as a result, these components are highly critical. At this moment, these applications become legacy.

The mechanisms for upgrading these old technologies is a complex task. Sooner or later, organizations will have to deal with it. There are different strategies for planning a legacy application modernization process. However, there is no effective direction or guide that writes down the guidelines that must be followed. It is a complex task that the IT department will have to face, where a bad decision can lead to an inefficient solution.

Keywords: legacy applications, system protection, microservices, modernization.

Índice de contenidos

Índice de figuras

Índice de tablas

Índice de acrónimos

2FA. Two-factor authentication

API. Application Programming Interface

CD. Continuos Delivery

CGI. Common Gateway Interface

CI. Continuos integration

CMMI. Capability Maturity Model Integration

CRUD. Create Read Update Delete

CSS. Cascading Style Sheets

DAO. Data Access Object

DevOps. Development and Operations

DSL. Domain Specific Language

FaaS. Function as a Software

HATEOAS. Hypermedia as the Engine of Application State

HTML. HyperText Markup Language

HTTPS. Hypertext Transfer Protocol Secure

IDE. Integrated Development Environment

Java EE. Java Platform, Enterprise Edition

JPA. Java Persistence API

JSP. JavaServer Pages

LDAP. Lightweight Directory Access Protocol

PMBOK. Project Management Body of Knowledge

REST. Representational State Transfer

ROI. Return On Investement

SOA. Service-Oriented Architecture

SOAP. Simple Object Access Protol

UI. User Interface

URI. Uniform Resource Identifier

UX. User Experience

XML. eXtensible Markup Language

1. Introducción

Dentro de una empresa coexisten diferentes tecnologías con distintas funcionalidades que pueden interactuar en mayor o menor grado. Por su naturaleza, el software no es creado para vivir eternamente, sino que la capa tecnológica que subyace, con los años, puede quedar totalmente obsoleta. Si a una ineficiente política de mantenimiento se le suma una capa de negocio extremadamente crítica para la organización, estos sistemas acaban finalmente deviniendo reductos intocables, convirtiéndose en aplicaciones heredadas o legacy. Afrontar un proceso de actualización sobre estos sistemas se vuelve una tarea ardua y tediosa, donde resulta complicado escribir unas guías de hoja de ruta para su modernización.

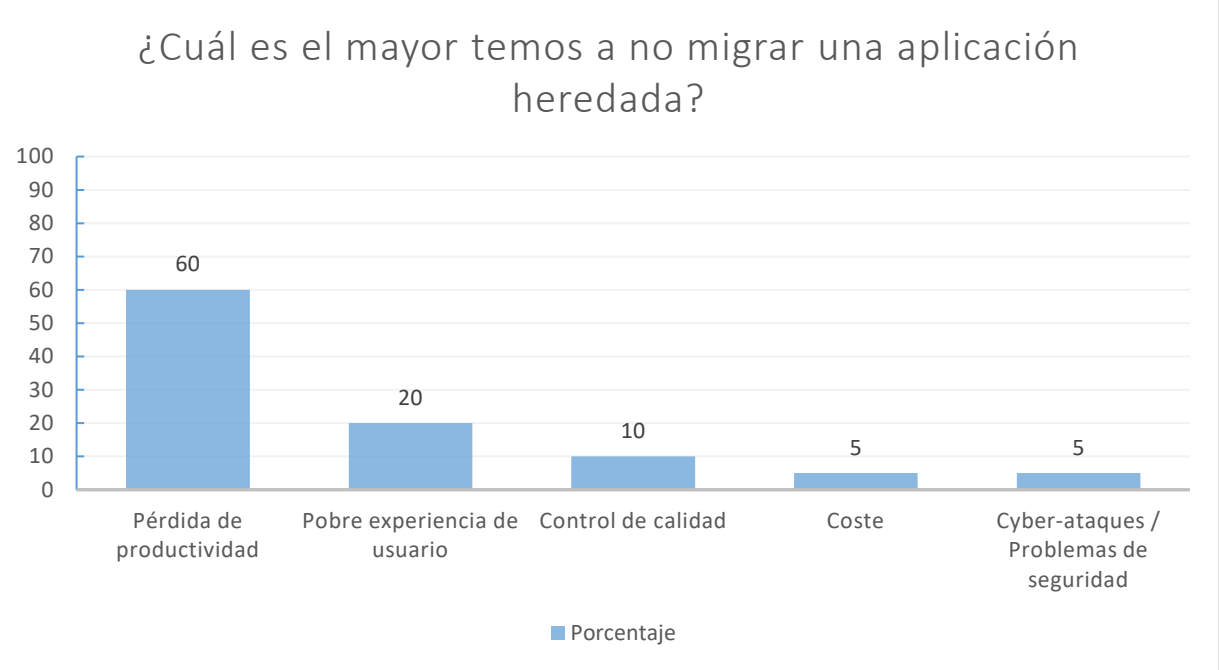

Figura 1. Temores de no migrar una aplicación heredada

Fuente obtenida de (O'Grady, 2020).

No actualizar una aplicación heredada puede tener consecuencias para una organización. Según un estudio realizado por Guidesmiths (O'Grady, 2020), representado en la figura 1, el mayor temor de las empresas a la no actualización de estas aplicaciones está principalmente en una pérdida de productividad. Esto se refiere a qué estos sistemas pueden convertirse en cuellos de botella cuando se mantienen estáticos en un entorno en continua evolución. Como segundo elemento importante, estos sistemas carecen de una experiencia de usuario, al

presentar unos pobres entornos gráficos que en su día fueron aplicados sin ningún tipo de estudio respecto la percepción y satisfacción del usuario.

Actualizar un software no es una tarea sencilla. Hay muchos factores que pueden incrementar significativamente los recursos que se habían planificado inicialmente para su desempeño. En el mismo estudio de Guidesmiths (O'Grady, 2020), como se observa en la figura 2, el factor más importante y sobre el que incide gran parte de los proyectos de actualización está en la brecha del conocimiento.

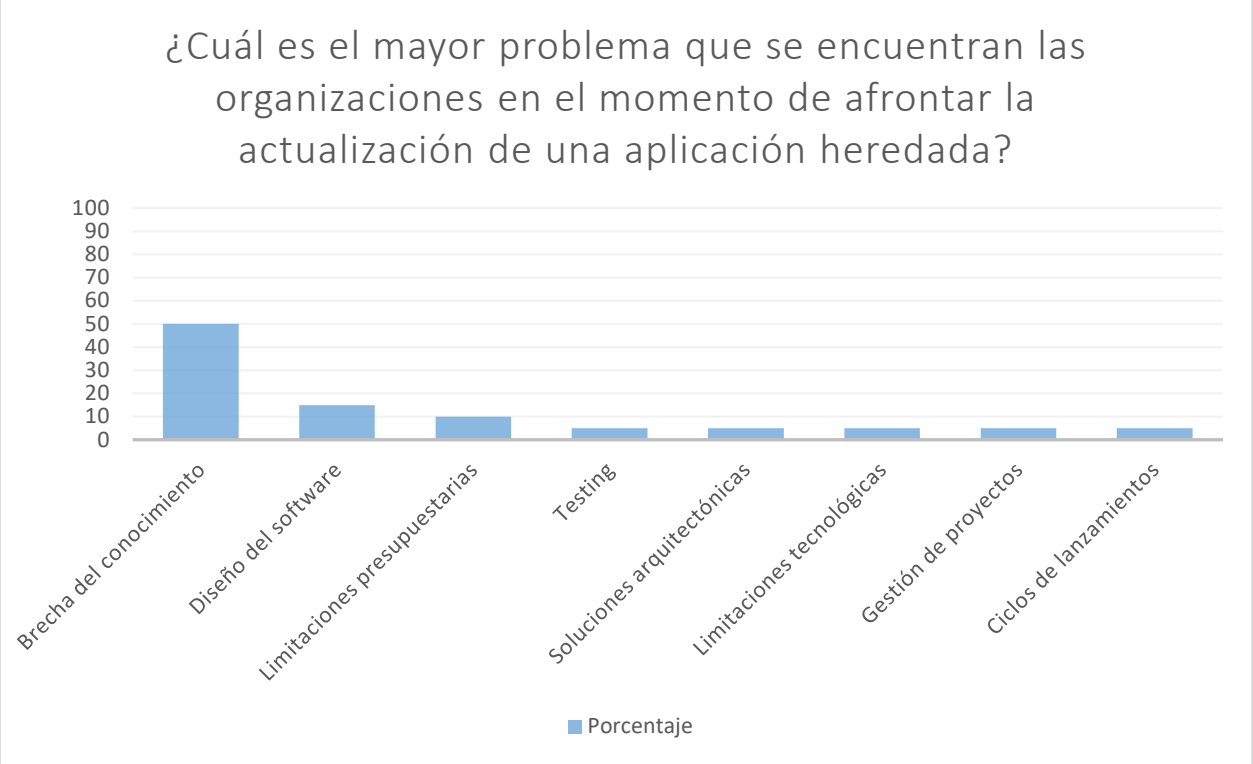

Figura 2. Problemas en el momento de afrontar la actualización del software

Fuente obtenida de (O'Grady, 2020).

El lenguaje o módulo utilizado para el desarrollo de la aplicación puede estar tan obsoleto que incluso puede llegar a ser difícil encontrar profesionales para estas tecnologías. Si a esto se le suma un mal diseño, la empresa se encuentra con un cóctel de código y componentes que puede llegar a ser difícil de digerir.

2. Estado del arte

2.1. Las leyes de Lehman

Una vez se implanta un software en una empresa, éste puede sufrir diferentes tipos de cambios a lo largo de su ciclo de vida. Desde una corrección de un simple bug hasta un complejo sistema de rediseño. En esto se basa la evolución del software y es un proceso que siempre debe estar activo. Así como hay factores que inciden en que el software sufra cambios continuos en su evolución, también hay elementos incidentes que la impiden. Las leyes de Lehman (Lehman, 1979, pág. 108) son una serie de leyes empíricas donde se establece los procedimientos que se deben seguir para que no produzca un desequilibrio en todo proceso de evolución y adaptación del software. En concreto, las leyes que inciden y que explican los motivos por los cuales un software devenga heredado son:

1. Ley de Cambio Continuo.
2. Ley de Complejidad Creciente.
3. Ley de Crecimiento Continuo.
4. Ley de Retroalimentación del Sistema.

Cuando una o varias de estas leyes no se cumple, entonces se produce una degradación del sistema, derivando en un empobrecimiento progresivo de su calidad, en el cual cada vez resulta más complicado afrontar una situación de actualización. Es entonces, cuando las aplicaciones devienen heredadas.

2.2. Las aplicaciones heredadas

Los sistemas heredados se caracterizan por tener un ciclo de vida extremadamente largo, donde raramente se hacen cambios importantes y son piezas muy complejas que ocupan un sitio muy importante en la estrategia de la organización. Estos sistemas son difíciles de mantener, pero representan y tienen un gran valor para la organización (Kangasaho, 2016, pág. 13).

Un sistema heredado se caracteriza:

- Por ser extremadamente crítico para la organización al representar una parte de la lógica de negocio muy importante para la empresa. Cualquier incidencia en su funcionamiento tiene un impacto directo en la organización (Bisbal J., 1999, pág. 103).

- Tienen un coste de mantenimiento alto y muy ocasionalmente son sistemas que pueden sufrir algún cambio significativo. Reemplazar o modificar un simple componente de un sistema heredado tiene un impacto directo sobre los demás componentes (Tuusjärvi, 2021, pág. 10).
- Son componentes cerrados y herméticos, con escasas o muy pobres interfaces.
- Son difíciles de entender y tienen un alto coste de ingeniería inversa.
- Puede resultar complicado encontrar profesionales sobre estas tecnologías.
- No se pueden escalar y son difícilmente integrables con otros sistemas (Bashair A., 2017, pág. 154).

2.3. La gestión de la evolución del sistema

Como se ha dicho anteriormente, la evolución de un sistema puede venir determinada desde un simple cambio en una línea de código, hasta la reimplementación de toda la funcionalidad del sistema. Según la magnitud o la escala del cambio, esta evolución se puede clasificar en tres categorías (Comella-Dorda, 2000, pág. 174):

1. **Mantenimiento**. Este proceso consiste en pequeñas modificaciones incrementales sobre el sistema. Son cambios controlados que se realizan progresivamente en el tiempo y que se encargan de corregir, mejorar o adaptar el software. La comunidad del software considera y admite que el mantenimiento es únicamente una estrategia de microevolución (Rodríguez, 2021, pág. 4),
2. **Modernización**. El sistema es modificado de forma significativa. En este tipo de evolución siempre se conserva una parte importante del sistema existente. Los cambios generalmente consisten en cambios estructurales o mejoras funcionales. Durante la evolución de estos cambios, no es alterada una parte importante de la lógica de negocio.
3. **Reemplazo**. El sistema es sustituido por otro totalmente nuevo. No todas las aplicaciones son candidatas para modernizarse. Una organización puede querer reutilizar la lógica de negocio de la aplicación y los datos del sistema heredado, pero la falta de unas interfaces claras entre ambos elementos provoca que se sea imposible de reutilizar. El coste para evolucionar estas aplicaciones es tan elevado que la aplicación debe acabar siendo reemplazada por un sistema totalmente nuevo. El sistema es reescrito al completo. No es reutilizado ninguna parte del componente original y la aplicación es desarrollada nativamente en su nuevo entorno.

Por tanto, existen estas tres estrategias que en menor a mayor grado permiten la evolución del software y su adaptación a las tecnologías más recientes. Cabe subrayar que el reemplazo es considerado en diferentes trabajos de investigación como una técnica más dentro de la modernización.

2.4. Mantenimiento del software

El mantenimiento es una de las fases más importantes dentro del ciclo de vida del software. Esta fase se distingue por ser la más costosa y las que más se alarga en el tiempo. La ISO 14764 (ISO, 2006) distingue cuatro categorías de mantenimiento:

- El mantenimiento adaptativo se refiere a mantener el software en perfectas condiciones de uso en situaciones cambiantes del entorno.
- El correctivo soluciona los errores detectados.
- El preventivo corrige problemas o errores antes de que se produzcan o detecten.
- El perfectivo se encarga de mejorar la calidad interna del software.

Algunos estudios admiten el mantenimiento como un elemento que forma parte de la evolución nativa del software, modificándolo y moldeándolo a su entorno con pequeños cambios (Rodríguez, 2021, pág. 4). Por tanto, esta fase lleva nativamente un componente de relativa evolución y adaptación al cambio.

Figura 3. Las seis fases del mantenimiento del software

Fuente obtenida de (ISO, 2006).

En la figura 3 podemos observar las seis fases del mantenimiento del software. La retirada del software se produce cuando el software se deja de utilizar o bien éste es reemplazado por otro existente.

Resulta importante clarificar el concepto de migración de la ISO, esta fase consiste en adaptar el software para que éste sea capaz de poder ejecutarse en diferentes entornos (Herrera, 2015, pág. 70). No deja de estar claro el significado y el alcance de la palabra entorno. Siguiendo la lectura realizada sobre diferentes trabajos y estudios sobre este aspecto, se ha considerado que, para este proyecto, el proceso de migración se puede desempeñar, por ejemplo, cuando se realiza un proceso de migración de un sistema operativo, pero se puede producir una migración a resultas de un proceso de modernización.

Dentro de las metodologías o técnicas que se utilizan para actualizar una aplicación heredada, el mantenimiento no es un mecanismo considerado como herramienta para actualizar aplicaciones (Bisbal J., 1999, pág. 104). Ésta es considerada como una fase más del ciclo de vida del software cuya esencia consiste, en parte, en asegurar que la aplicación se adapte progresivamente a su entorno con pequeñas adaptaciones tecnológicas, así como en pequeños cambios durante su ciclo de vida.

2.5. La Modernización de aplicaciones

La modernización del software consiste en reescribir, adaptar o convertir los sistemas heredados a las tecnologías más recientes. Las acciones para actualizar una aplicación heredada pueden ser diversas y depende en gran medida de los recursos que una organización pueda o quiera asumir. La moderación implica cambios significativos en la estructura y funcionalidades de una aplicación. Estos cambios se realizan con el fin de mantener el valor estratégico de las aplicaciones para las empresas (Sánchez, 2013, pág. 2).

Modernizar, según las características de la aplicación a actualizar, puede llegar a requerir de un gran esfuerzo por parte de la empresa. No solo por parte de los recursos económicos y de personal que son necesarios para afrontar un proyecto de estas características, sino también por el tiempo que se debe planificar para su realización. En la figura 4 aparecen las cuatro entradas que van a condicionar cualquier estrategia de actualización del software dentro de las organizaciones. También se debe asumir el riesgo que conlleva una u otra elección, pues una migración muy compleja o una actualización pésima del software pueden acarrear una

mala finalización del proyecto. La naturaleza del problema puede variar de una aplicación u otra, incluso de una empresa u otra.

Los presupuestos de las empresas marcan los límites del grado de alteración de una aplicación heredada, donde las soluciones pasan por la reingeniería, abandono o una solución híbrida (Rodríguez, 2021, pág. 1). Éste es seguramente el factor que más va a condicionar cómo se resuelve y el tipo de actualización que se acabe realizando en un sistema legado.

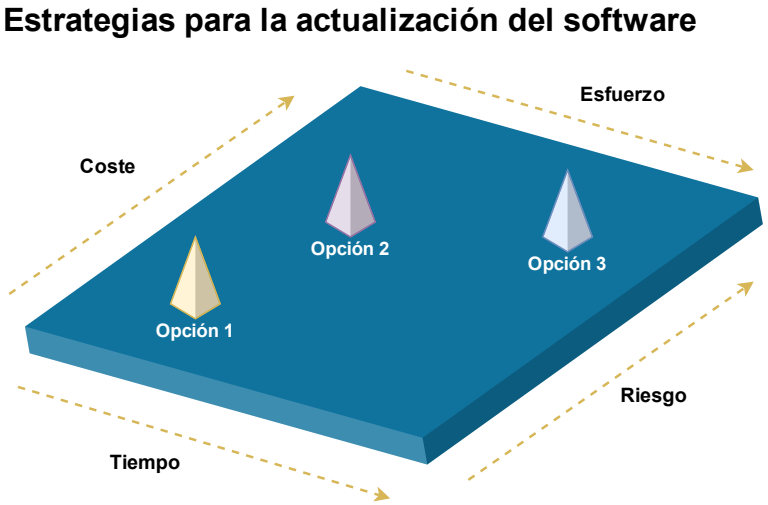

Figura 4. Estrategias para la actualización del software

Fuente obtenida de (Shinde, Sin Fecha).

Otro factor importante es el número de acciones sobre la lógica de negocio de la empresa desempeña el software a actualizar, aquellas aplicaciones que tengan un único proceso de negocio son más fáciles de modernizar y de moldear (Nilsson, 2015, pág. 8).

Las estrategias de la modernización van a estar condicionadas según el nivel de conocimiento que se quiera obtener con el estudio de la aplicación heredada. Esto es significativamente importante, pues va a determinar el grado de análisis y de recursos a destinar para obtener el máximo conocimiento del software. La modernización se categoriza en dos grandes ramas: la modernización de caja blanca (white box modernization) y la modernización de caja negra (black box modernization).

2.5.1. Modernización de caja blanca

La modernización de caja blanca se encarga de realizar un análisis exhaustivo de código y deben ser identificadas y analizadas todas sus interacciones, proporcionando un nivel muy

alto de descripción del sistema (Kangasaho, 2016). Este tipo de modernización no es sencilla puesto que se debe comprender minuciosamente las distintas partes del código. Para la recuperación del diseño, análisis de funcionalidades y diseño arquitectónico se utiliza la ingeniería inversa (Rodríguez, 2021, pág. 4). Obtener un nivel medio o alto de conocimiento puede ser extremadamente costoso según el tipo de la aplicación, puesto que la estructura interna del código puede estar muy degradada y con muchas carencias de estructuración.

Una vez realizado este estudio, el código sufrirá un proceso de restructuración (Comella-Dorda, 2000, pág. 175) aplicando técnicas de modernización. A este proceso también se le suele llamar refactorización (Kangasaho, 2016, pág. 20). Con esto se consigue cambiar totalmente el código interno, pero sin modificar o alterar su comportamiento y funcionalidades. Por tanto, este tipo de análisis puede requerir de un largo proceso de ingeniería inversa que acabará con una reestructuración del código y una reestructuración de la documentación. La aplicación inicial y la final poco tienen que ver, puesto que la aplicación final ha sido implementada partiendo de las nuevas tecnologías.

2.5.2. Modernización de caja negra

La modernización de caja negra no requiere obtener un conocimiento interno del sistema. Malinova (Malinova, 2010, pág. 78) describe este tipo de modernización como el que se encarga de estudiar el comportamiento externo del sistema, analizando sus interfaces y realizando una comprensión exhaustiva de sus entradas y salidas. Este tipo de análisis no requiere de gran esfuerzo y se obtienen resultados más a corto plazo.

Por ejemplo, es muy útil aplicar la modernización de caja negra en un código o en un módulo del cual no se tenga conocimiento, sea extremadamente complejo de entender, no se tenga acceso al código fuente o sea totalmente opaco.

El método de caja negra más utilizado es el de wrapping. (Comella-Dorda, 2000, pág. 176)

2.5.3. Técnicas de modernización

Adoptar una u otra técnica de modernización viene delimitada por dos parámetros:

- La complejidad del sistema a analizar.
- Los recursos que se pueden dedicar.

Bisbal (Bisbal J., 1999, págs. 105-106) define tres tipos de modernización según el grado de afectación y alternación del sistema a modernizar:

1 – Wrapping

Debido a la complejidad y a la enorme cantidad de recursos que puede conllevar un reemplazo, muchas organizaciones se ven forzadas a encontrar soluciones más livianas, sin ser tan intrusivas ni requerir de un análisis exhaustivo del código. Esta técnica consiste en proveer al sistema de una nueva interfaz, que se encontrará por encima de la aplicación heredada y con la cual van a interactuar todos los demás elementos externos. Esta nueva interfaz va a permitir que el sistema pueda ser visible por nuevos componentes que hasta entonces no podían interactuar y le va a proporcionar escalabilidad operacional.

2 – Migración

En sistemas que no se pueda asumir todo un proceso de reemplazo de la aplicación y un wrapping no es una solución. Este proceso consiste en mover la aplicación heredada a una nueva plataforma, manteniendo el modelo de datos y su funcionalidad. La migración es un proceso de reingeniería del software con el fin de ponerlo en funcionamiento en la tecnología más actual (Sánchez O., 2013, págs. 2-3). Esto tiene grandes beneficios a largo plazo ya que permite reducir el mantenimiento, ofrece mayor flexibilidad y el código pasa por un proceso de refactorización que reduce su complejidad.

3 – Reemplazo

El reemplazo de aplicaciones (Redevelopment, también conocido como Big Bang o Cold Turkey) consiste en el rediseño total del sistema sobre una nueva plataforma, incluyendo toda su funcionalidad, en una arquitectura moderna y dentro de un nuevo sistema de base de datos (Bashair A., 2017, pág. 155). Este proceso, según el componente a actualizar, puede requerir de un complejo análisis de ingeniería inversa de todo el sistema con el fin de rediseñar todas sus funcionalidades. Se caracteriza por una reescritura al completo del sistema, sin aprovechar parte alguna de la aplicación origen.

Este tipo de desarrollos pueden llegar a requerir de tal completitud que se debe considerar como un riesgo el hecho que se pueda dar una pérdida de una parte del conocimiento. Con lo cual, puede darse el caso que el nuevo sistema no acabe funcionando tan bien ni con las

mismas funcionalidades como lo hace el viejo sistema (Tuusjärvi, 2021, pág. 15) y, que tampoco, acabe ofreciendo la misma funcionalidad.

Cuando llegan estas soluciones al entorno de producción, puede darse el caso que la aplicación heredada y la nueva aplicación que la sustituya tengan un período de convivencia, con lo cual, se deberán definir políticas para informar y formar con el nuevo producto a los usuarios finales. También, es muy importante establecer fechas de convivencia y disponer planes de contingencia una vez la aplicación heredada sea retirada del catálogo de aplicaciones de la organización. Puede producirse, por ejemplo, que se deba volver a activar la aplicación antes de su retirada definitiva.

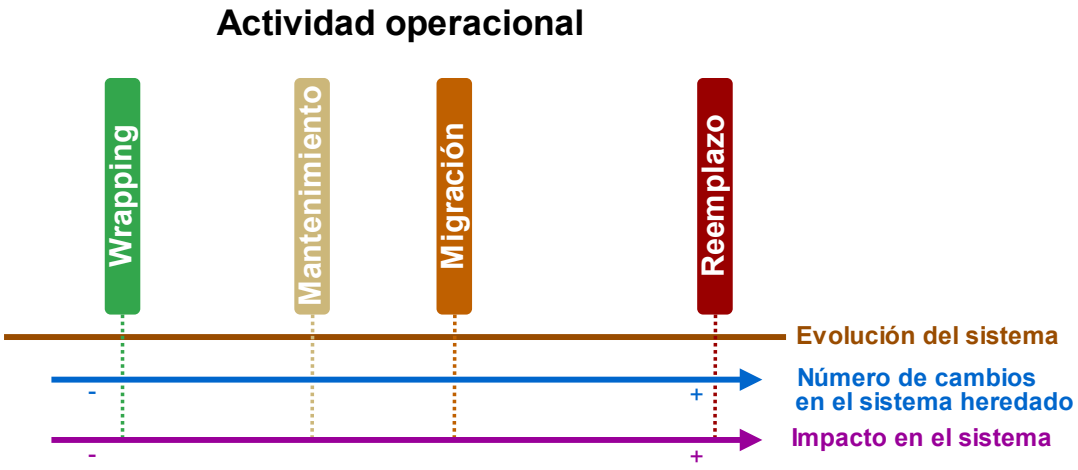

Figura 5. Operaciones para afrontar la actualización de una aplicación heredada

Fuente obtenida de (Bisbal J., 1999, pág. 104).

En la figura 5 se puede observar el esfuerzo que se requiere según el tipo de operación de actualización que se aplique sobre el software. Como se ha dicho con anterioridad, el mantenimiento es añadido para posicionar e informar del impacto que tiene esta fase sobre el software, pero es ningún caso es considerada como una técnica de modernización.

2.6. Wrapping

El wrapping consiste en envolver la aplicación heredada con una interfaz funcional que va a permitir encapsular todas las funcionalidades de la aplicación que van a ser reutilizadas (Malinova, 2010), con el fin de conseguir un sistema con una nueva y moderna interfaz.

En los sistemas heredados se puede aplicar el wrapping de interfaz de usuario, de datos y funcional (Rodríguez, 2021).

1 – Wrapping de interfaz de usuario

Ésta es la parte más visible de la aplicación y consiste en rediseñar la interfaz de usuario con el fin de mejorar la UI y la UX. Esta técnica resulta relativamente sencilla cuando la lógica de negocio y la capa de presentación son elementos que se encuentran totalmente segregados. No es un wrapping intrusivo, pues se utiliza la misma interfaz de comunicación con el back-end. Ésta es una técnica muy utilizada cuando se requiere aplicar un nuevo entorno visual completamente actualizado a la aplicación.

2 – Wrapping de datos

Se redefine una nueva interfaz de acceso a la capa de persistencia. Un caso interesante de wrapping de datos es el propuesto por Pérez-Castillo (Pérez-Castillo, 2012), dónde se expone una solución de este tipo añadiendo una capa de Web Services, la cual es el único punto de acceso y con la que se ofrecerá la información de la base de datos heredada.

3 – Wrapping funcional

Este tipo de wrapping no solo ofrece envolver la capa de persistencia, sino que además le añade la funcionalidad del software legado, accediendo directamente las interfaces públicas y privadas del sistema. Para conseguir este fin, Comella-Dorda (Comella-Dorda, 2000) propone tres tipos de técnicas:

- Exponer la funcionalidad y la base de datos por medio de CGI. Se basa en exponer un endpoint en el cual se publicará toda la lógica de negocio y la capa de persistencia de la aplicación. Este sistema invocará los métodos que se definirán en una capa de middleware.
- Object-Oriented Wrapping (OOW). Se basa en llamadas a métodos remotos sobre tecnologías distribuidas. Un ejemplo sería CORBA.
- Wrapping orientado a componentes. Sería similar a OOW, pero se basa en descomponer la interfaz de la aplicación en pequeñas unidades de lógica de negocio que pasan a ser los componentes. Un ejemplo serían los EJB.

Estas técnicas permiten que los sistemas heredados sean reutilizados en entornos distribuidos. Con la evolución del software y la aparición de la arquitectura orientada a servicios, se obtiene un nuevo mecanismo para implementar la capa wrapping funcional, donde son numerosos los estudios que aportan soluciones y exponen experiencias sobre esta forma de modernizar aplicaciones heredadas.

2.6.1. Propuestas tecnológicas al wrapping

Para aplicar la modernización por wrapping en una aplicación heredada se deben decidir dos piezas esenciales:

- La capa tecnológica que va a envolver la aplicación.
- El tipo de mensajería que va a ofrecer esta capa tecnológica

Todo ello para conseguir un único punto de acceso a las operaciones del sistema heredado. Para dar una respuesta tecnológica existen diversas herramientas, entre las cuales, una de las más utilizadas, es aplicar una solución SOA al sistema.

Arquitectura orientada a servicios

Una arquitectura orientada a servicios o SOA es un paradigma donde un conjunto de funcionalidades es expuesto a través de una interfaz de servicios, para que otras aplicaciones o servicios las puedan consumir. Un servicio es un componente distribuido, reutilizable y autocontenido que se encuentra definido dentro de esta interfaz y que forma parte de un catálogo de servicios. Con esta tecnología se encapsula el código legado detrás de una capa SOA (Salvatierra G., 2013, pág. 841), con lo cual, se permite ofrecer su funcionalidad de una manera modular y granular a través de los servicios web.

Dentro del marco de opciones tecnológicas de SOA wrapping cabe destacar dos aproximaciones muy parejas (Kangasaho, 2016, pág. 35):

- SOAP wrapping
- REST wrapping

Un sistema SOA, además, debe proporcionar transparencia a las aplicaciones clientes. Es decir, éstas no deben conocer cómo y dónde se ejecutan los microservicios. En este tipo de entornos se pueden exponer desde unos pocos servicios a centenares de ellos. Para responder y agilizar el consumo de los microservicios aparece el concepto de catálogo de servicios, desarrollado por un componente que implementa el patrón service registry. Con este patrón se consigue que los servicios sean descubiertos para posteriormente asociar el servicio con la aplicación consumidora (Bean, 2010, págs. 13-14) para que la información se transmita correctamente. El service registry es el único punto de entrada de cualquier petición que consuma de un microservicio, donde se encuentran centralizadas todas las operaciones sobre el sistema legado.

SOAP wrapping

SOAP es un protocolo estándar que define la interoperabilidad entre distintos procesos por medio del intercambio de ficheros XML. Para exponer y publicar las diferentes operaciones que están representadas por servicios se utiliza un fichero WSDL, donde se describe la interfaz pública de los servicios web. En este fichero se define la interfaz de los endpoints, con los cuales se realiza cualquier interacción sobre el sistema. Esta capa SOAP se utiliza para la modernización de caja negra sobre el sistema legado, donde el wrapper es implementado por esta capa de negocio y publicado a partir de servicios web (Canfora, 2006).

REST wrapping

La termino REST fue introducido en la tesis doctoral de Fielding (Fielding, 2000) en el año 2000. Es considerada una técnica de arquitectura del software para sistemas distribuidos. A diferencia de SOAP, REST no representa ningún estándar ni obliga a cumplir estrictas reglas. En su tesis, Fielding (Fielding, 2000) escribió los cuatro principios básicos de la arquitectura:

- Identificación de recursos a partir de la URI. Las URIs con la cuales se opera sobre los servicios REST deben ser intuitivas, dónde debe ser fácil determinar y de una forma muy compresiva el recurso sobre el cual se interactúa y que éste sea fácil de encontrar.
- Una interfaz uniforme que representa una única forma de acceso a los recursos. Los recursos se deben poder acceder de una forma sencilla y con un único sistema de operaciones. Estas operaciones se pueden expresar utilizando los correspondientes verbos HTTP, siendo principalmente “GET”, “POST” “PUT” y “DELETE”.
- Es una comunicación sin estado. En la comunicación entre el cliente y el servidor no se transmite ningún estado. Por tanto, en cada llamada se debe disponer de toda la información para que ambos sistemas puedan operar. Esto es una ventaja significativa, dado que esta ausencia hace que el sistema sea fácilmente escalable.
- Utilizar hipermedios. Todos los mensajes REST deben contener toda la información que se espera de los elementos sobre los cuales se actúa. Además, la interfaz REST permite obtener el mismo objecto con distintas representaciones, como por ejemplo XML y JSON. Los metadatos del recurso son añadidos en las cabeceras HTTP. De este modo, el actor que realiza la petición y el que la emite deben estar de acuerdo con el tipo de representación que se espera.

Al definir la interfaz REST se obtiene un sistema de mensajería que expone y da visibilidad a las diferentes operaciones que se definen sobre los diferentes recursos del sistema.

Una buena implementación REST, además, debe cumplir con los niveles de madurez de modelo de Richardson (Fowler, 2010):

1. Nivel 0. Swamp of POX: Se va a utilizar HTTP para realizar las interacciones remotas.
2. Nivel 1: Recursos: Se identifica los recursos a través de un URI, sin especificar la acción que se realiza sobre el mismo.
3. Nivel 2: Verbos HTTP. La API debe hacer uso de los verbos, utilizándolos correctamente, así como sus respuestas.
4. Nivel 3: Hypermedia controls: es utilizar HATEOAS (Hipertexto como el mecanismo del estado de la aplicación). El que se pretende con este HATEOAS es que, al realizar una petición sobre un recurso, el mismo va a devolver información de como trabajar o manipular el recurso.

En la figura 6 se plasma el nivel de importancia y cumplimiento de los niveles de madurez de Richardson, donde se consigue la gloria del REST cuando se cumplen los 4 niveles.

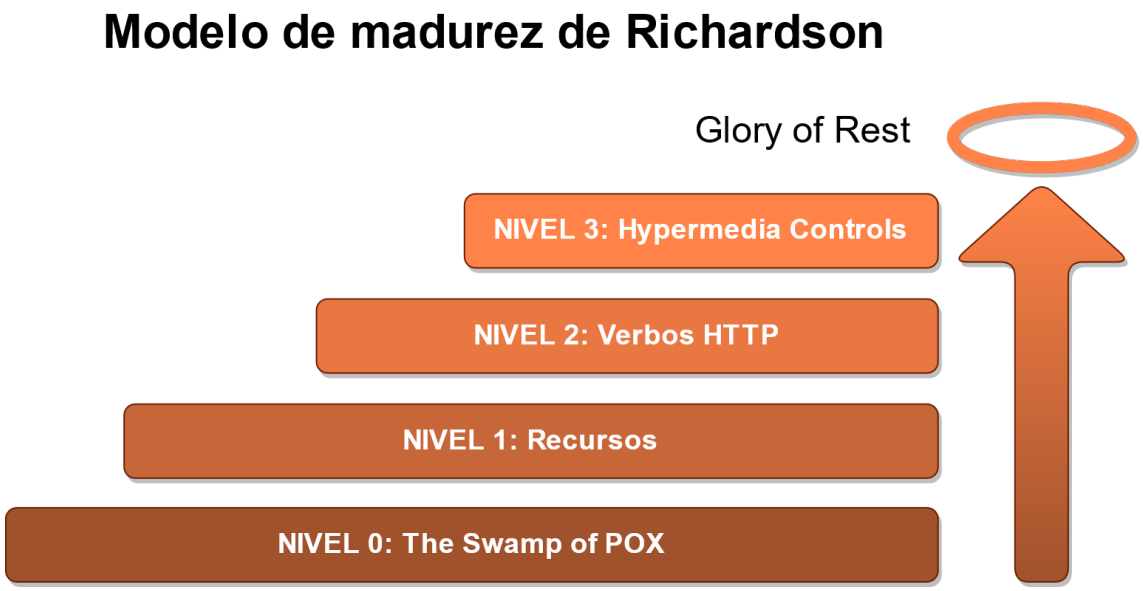

Figura 6. El Modelo de madurez de Richardson

Fuente obtenida de (Fowler, 2010).

Aplicar un proceso de modernización con una REST Service Layer produce la identificación de los recursos que proveerá la aplicación heredada, con la cual se permite la interacción entre

el cliente y el servidor, estableciendo una relación contractual de los recursos del sistema (Echeverria, 2015, pág. 2380).

2.6.2. La arquitectura de microservicios

A media que el software evoluciona, incrementa su complejidad y son necesarias nuevas soluciones con una alta concurrencia, ejecutándose en diferentes sistemas de hardware y que permitan una sincronización y coordinación entre elementos (Dragoni, 2016, pág. 2). Es entonces cuando gana sentido el concepto de una arquitectura totalmente segregada de pequeños servicios, donde se quiere conseguir un completo ecosistema de pequeñas acciones. Esta tecnología es un modelo de arquitectura de sistemas distribuidos donde todos los módulos funcionales están implementados con microservicios. Los microservicios son piezas esenciales y responden a una cierta funcionalidad, donde contienen acciones muy granulares y sin dependencia entre otros microservicios. El uso de este tipo de arquitecturas es recomendado en situaciones en las cuales se realicen modernizaciones de aplicaciones que requieran de flexibilidad, escalabilidad y disponibilidad (Tuusjärvi, 2021, pág. 22). Responden, en gran parte, a las necesidades de los requisitos no funcionales de las aplicaciones.

Dentro de la definición y diseño de una modernización por wrapping, un punto muy importante es la decisión tecnológica de los elementos que se van a utilizar para envolver el sistema heredado. Este elemento puede ir completamente integrado dentro de la tecnología de la misma aplicación o bien puede definirse en una capa tecnológica aparte de la aplicación heredada. Los microservicios resultan unos componentes esenciales para el enfoque de procesos de modernización de aplicaciones puesto que permiten exponer de una forma granular y segregada las distintas operaciones que se realizan sobre el sistema, con una alta capacidad de evolución y con un enfoque multiplataforma (Knoche, 2018, pág. 45). Estos elementos permiten extraerse de aplicaciones monolíticas, donde el uso de un servicio o la orquestación de diferentes servicios va a permitir exponer cualquier funcionalidad de la aplicación heredada.

Con ellos se consigue centralizar y envolver toda la lógica funcional de la capa subyacente de la aplicación heredada.

2.7. Serverless

Una de las primeras referencias sobre este modelo de computación fueros escritas por Ken Fromm, donde apareció la idea de liberar a los desarrolladores con la tediosa carga de tener que definir para cada aplicación su entorno de ejecución. No implica que los servidores dejen de ser un actor en el desarrollo e implantación de aplicaciones, sino que su instalación no es necesaria para el manejo de aplicaciones (Fromn, 2012). Con esta tecnología se permite el desarrollo directo de aplicaciones, pues el servidor se encuentra autocontenido y éste es plenamente configurable.

La programación serverless, también conocida como FaaS, es solución muy eficiente para la implantación en entornos con diferentes proveedores en la nube, con el fin de reducir costos operativos mediante la optimización y la gestión eficiente de los recursos, permitiendo un ecosistema serverless que fomente la implementación de servicios adicionales en la nube (Castro, 2017, pág. 2658). Es el proveedor el que se encarga de ofrecer todos los recursos donde serán contenidas las aplicaciones, así como de los recursos del sistema para que se ejecuten de una forma eficiente. También de escalar o liberar recursos de las imágenes de las aplicaciones cuando se requiere. Por tanto, extrae de la gestión de la capa inferior que conforma la infraestructura donde residirán los componentes.

2.8. Gradle

Maven y Gradle son dos herramientas para el ensamblaje de proyectos Java. Gradle utiliza Groovy basado en un DSL, con el cual se obtiene un conjunto de elementos que permiten definir de una manera muy ágil y declarativa los proyectos, empaquetando e incluso incluyendo procesos de automatización para la publicación en contenedores. Además, esta solución de desarrollo permite los building incrementales, donde solo aquellos componentes que son modificados se llegan a compilar y publicar (Varanasi, 2015, pág. 2). Otra pieza clave es el Gradle Wrapper (gradlew), un fichero en batch que permite la CI de aplicaciones, en el cual los servidores autocontenidos se arrancan automáticamente sin configuración adicional.

2.9. La familia Spring Framework

Spring Framework es un entorno de trabajo para el desarrollo de aplicaciones de código abierto para la plataforma Java. Esta metodología de trabajo fue inicialmente escrita por Rod Johson en el año 2000, para dar una perspectiva más sencilla al desarrollo de aplicaciones con

tecnología Java. Nace como respuesta a la necesidad de simplificar la visión de los componentes y sus respectivas configuraciones, dando consistencia y agilidad a las tecnologías que se estaban implantando. Éste es un marco de referencia en el mundo de la programación y desarrollo que se postula como un elemento básico para una visión DevOps.

Spring Boot

Spring Boot es un mecanismo que permite configurar de una forma realmente sencilla las aplicaciones basadas en Spring. El principal objetivo de esta tecnología es reducir el tiempo del desarrollador para aplicar distintas tecnologías dentro de un proyecto Spring-Java y, que éstas, se configuren de una forma ligera y ágil. Esto se consigue gracias a los *starters*. Estas piezas o componentes permiten inyectar dentro de una aplicación Spring toda una infraestructura tecnológica que se puede ejecutar, sin realizar importaciones de librerías, crear directorios, código java y complejas configuraciones de los componentes (Antonov, 2018, págs. 8-9). Spring Boot se encarga de reducir la complejidad que presenta la instalación y configuración de los diferentes módulos que se utilizan en un proyecto, aportando un potente mecanismo de autoconfiguración (Siva Prasad Reddy, 2017, pág. 1).

Spring Cloud Config Server

Spring Cloud Config es un sistema que implementa el patrón External Configuration Store, con el cual se consigue que las aplicaciones no tengan la configuración autocontenida, sino que sus propiedades son servidas por un sistema externo.

El patrón define que (K., 2010):

1. Las aplicaciones deben poder recuperar de estos sistemas cualquier tipo de configuración. Como puede ser la conexión de base de datos o los parámetros de protección de la aplicación.
2. Debe proveer de los distintos entornos de las aplicaciones como pueden ser el de prueba, desarrollo, staging o producción, sin necesidad de modificar o recompilar ningún elemento.
3. Una misma configuración debe ser expuesta a distintas instancias de un mismo servicio.
4. Las propiedades pueden ser modificadas en caliente.

5. Las propiedades solo deben ser accesibles y protegidas para aquellos usuarios autorizados.
6. Los cambios deben estar correctamente monitorizados.

Spring Cloud Server cumple con estos puntos, a la vez que permite que estas propiedades estén externalizadas a la misma aplicación, siendo servidas por este componente y recuperadas de un repositorio remoto como pueden ser Git o Subversion. Este servidor de configuraciones expone las propiedades a través de una Api-Rest, en que los distintos clientes se conectan para recuperar sus configuraciones y en sus respectivos entornos (Long, 2017, pág. 83).

Spring Cloud Netflix Eureka Server

Eureka Server es un Service Discovery de microservicios. Los microservicios se registran a Netflix Eureka Server implementando pequeñas funcionalidades de lógica de negocio de la organización, con el fin que sean consultados de una forma totalmente transparente por las aplicaciones o microservicios consumidores. Este sistema también tiene la capacidad de aportar una disponibilidad muy alta por sus respectivas aplicaciones clientes o bien otros servicios (Raman. R. CSP, 2018, pág. 155).

Cuando un microservicio arranca, éste se comunica con el servidor Eureka Server para notificar si está disponible, su estado, donde se encuentra ubicado, además de un conjunto de metadatos con los cuales se podrá consultar por diferentes parámetros del microservicio. En la configuración por defecto, en intervalos de 30 segundos cada uno de los microservicios se comunica con el servidor con el fin de informar de su estado. A este tipo de notificación se le llama heartbearts. Una cosa importante que se debe tener en cuenta es que, hasta que un cliente, el servidor y la instancia del servicio no tienen los mismos metadatos, el servicio de estará activo para el componente consumidor. Esto se produce a los tres heartberts. Del mismo modo, si el servidor Eureka no ha recibido los tres heartberts esperados de un microservicio, éste será eliminado del service registry.

Así es como Eureka Server mantiene un registro vivo, actualizado y plenamente funcional de todo el ecosistema de microservicios que conforman el sistema (Larsson, 2019, págs. 251-259).

Spring Data

Spring Data es un proyecto cuya función es la de facilitar el acceso a la capa de persistencia. Las bases de datos que soporta pueden ser relacionales o bien de tipo NoSQL. El objetivo de esta tecnología es reducir el esfuerzo que implica la definición y acceso a una base de datos, proporcionando una interfaz de comunicación que agiliza de una forma significativa el acceso a la capa de persistencia (Brisbin, 2013, pág. 13).

Con Spring Data se da un paso más a las ya implementadas interfaces Hibernate o JPA, con una centralización y unificación de acceso al back-end, proporcionando:

1. Funcionalidades de CRUD.
2. Operaciones DAOs que se pueden definir como interfaz y son auto implementadas.

Una característica que ha dado significativa popularidad a esta tecnología es la implementación de acceso a las clases de dominio a través de las propiedades que forman los distintos objetos persistentes. Este proceso se basa en la creación de consultas SQL a partir de los nombres de los métodos (Gierke O., 2021), sin necesidad de implementar código.

Spring Security

Spring Security es un módulo Java / Java EE del proyecto Spring que permite configurar y customizar la autenticación, autorización y otros elementos de seguridad sobre proyectos Spring (Alex B., s.f.). Con este control se permite, por un lado, discriminar los usuarios que van a poder entrar en la aplicación y, por otro, la autoridad que se le va a proporcionar y que va a delimitar las operaciones que se van a poder realizar sobre la aplicación. Esta herramienta permite parametrizar y configurar de una manera relativamente sencilla y ágil el sistema de acceso a las aplicaciones.

La terminología que ayuda a entender cómo funciona Spring Security (J., 2018, págs. 29-30):

- **Principal**. Un usuario, dispositivo, aplicación que quiere interactuar con la aplicación.
- **Authentication**. Proceso que valida si el principal es realmente quien dice ser.
- **Credenciales**. Usuario y contraseña introducidos por el principal.
- **Authorization**: Una vez el principal se ha autenticado, se validan las acciones que puede realizar sobre la aplicación.

- **GrantedAuthority**: Objeto Spring que le limitaran los permisos al principal. Por ejemplo, un role.
- **SecurityContext**: Objecto Spring Security dónde está toda la información del principal.

La autenticación de un usuario empieza con su primer acceso a la parte protegida de la aplicación. Spring Security redireccionará al usuario a la pantalla de introducción de credenciales. Una vez éstas son validadas, se devolverán los atributos del usuario.

Este componente es plenamente integrable con multitud de servidores de autenticación como pueden ser CAS Server y Keycloak. Su uso aporta una significativa ventaja puesto que se encarga de implementar toda mensajería y comunicación del protocolo en el cual se integra.

2.10.Thymeleaf

Thymeleaf es un motor de plantillas Java server-side para el procesamiento y creación de ficheros HTML, XML, JavaScript, CSS y texto. Este framework es extremadamente extensible y puede ser utilizado tanto en entornos web como en entornos no web. Dispone de módulos para Spring Framework con los cuales se obtiene un desarrollo muchas más intuitivo y ágil si se compara con otras tecnologías más tradicionales como pueden ser las JSP.

Thymeleaf tiene una gran aceptación por la comunidad de desarrolladores web, pues todo aquello que es dinámico dentro de una plantilla puede volver a un valor predeterminado en caso de que se habrá sin que un servidor se esté ejecutando (Kunjumohamed, 2016, pág. 672).

2.11.Keycloak

Keycloak es un producto software de código abierto de gestión de identidades y accesos (Identity and Access Management) focalizado a aplicaciones y servicios modernos. El sistema ofrece un servicio de inicio de sesión único y unificado, Single-Sign On, con el cual se consigue que un usuario pueda acceder a varias aplicaciones con una sola instancia de identificación.

Este sistema de autenticación y autorización dispone de una consola centralizada de gestión, en la cual se definen aplicaciones, usuarios y roles dentro de un realm (Thorgersen, 2021, pág. 11). Un realm que puede ser entendido como un tenant, este elemento es individual e independiente de otros realms y, una aplicación, solo podrá acceder por el realm en el cual se ha definido. Keycloak, además, es extensible, permite integraciones con protocoles estándares, como puede ser Oauth 2.0 y sincronización con LDAP y Active Directory.

Keycloak es un proyecto de Wildfly comunity que se encuentra bajo la tutela de Red Hat.

2.12.Scrum

Scrum es una metodología ágil de gestión y desarrollo de proyectos que tiene un enfoque iterativo e incremental. Este marco de trabajo fue creado por Jeff Sutherland junto con su equipo de desarrollo en la década de los 90. Es un método iterativo puesto que divide el proyecto en ciclos temporales llamados Sprints y, es incremental, ya que en cada iteración se produce una evolución del producto añadiéndole nuevas capacidades, a la vez que se mejoran o completan los requisitos u objetivos del sistema.

En Scrum existen tres roles principales (Sutherland, 2010, pág. 14):

1. El Product Owner. Es responsable de garantizar el máximo ROI. Este role determina prioridades y representa a la empresa o los usuarios finales.
2. El equipo de trabajo. Es el grupo que va a desarrollar el software.
3. El Scrum Master. Asegura que el equipo de trabajo se coordine y el producto se desarrolle correctamente. No es en ningún caso un director del equipo de trabajo, sino que lo sirve y lo protege de posibles interferencias.

Los principales elementos en los cuales se compone Scrum son (Sutherland, 2010, pág. 10):

- Product Backlog: Lista de tareas priorizadas que tienen valor para el negocio o el cliente.

- Sprint: Ciclos temporales de Spring que pueden ser de 1 a 4 semanas.

- Spring Planning: El Product Owner y el equipo de desarrollo revisan el Product Backlog y deciden cuales tareas van a ser programadas para un nuevo Sprint.

- Daily Scrum: Cada día el equipo de Scrum se va a reunir no más de 15 minutos donde se van a responder a las preguntas: ¿Qué hiciste ayer? ¿Qué vas a hacer hoy? ¿Hay algo que impida el desempeño de tu trabajo?

- Restrospective: Cuando un Spring termina, el equipo analiza cómo ha ido el Sprint y se inspecciona y analiza lo que se ha hecho.

3. Objetivos

La empresa FuxPin S.L., dedicada a trabajos de carpintería y pintura del hogar, quiere actualizar parte del ecosistema de sus aplicaciones. En un primer paso, para empezar el

proceso de viabilidad de estas actualizaciones, se pide proceder con la renovación de la aplicación de gestión de ventas. Esta aplicación ha sido mantenida por distintas empresas de servicios, las cuales han realizado modificaciones fuertemente customizadas. Durante las entregas del producto, la empresa no ha tenido implantada una política de control y de calidad de los cambios, con lo cual, el software presenta una significativa complejidad de compresión. La aplicación, además, está implementa sobre una solución tecnológica que se ha quedado sin soporte. Sin embargo, a nivel funcional, el software desempeña una función altamente crítica para la empresa y de su lógica de negocio. Por tanto, esta solución ha acabado siendo una aplicación heredada.

Para este proyecto se pide que parte de las funcionalidades de la aplicación heredada sea de acceso externo para su consumo dentro de la empresa. Este acceso consistirá en proveer de dos acciones para:

- Inyectar los roles del sistema heredado durante el proceso de autenticación y autorización del usuario.
- Proveer de un listado de clientes de la aplicación heredada.

3.1. Estructura del proyecto

El proyecto de final de grado se organiza en los siguientes capítulos:

- **Capítulo 1**. Se explica el contexto actual de las aplicaciones heredadas en las organizaciones y cuáles son las principales inquietudes que explican los motivos del por qué éstas sigan siendo piezas importantes en las organizaciones y se mantienen estáticas en un mundo tecnológico que es muy dinámico.
- **Capítulo 2**. Estado del arte. Se ha realizado una lectura de distintos trabajos, artículos, libros y documentos web con el fin de comprender y focalizar cómo se debe afrontar este tipo de actualizaciones. El estudio ha servido para comprender a nivel teórico el tipo de problemática que representan estos elementos para las organizaciones y cuáles son los mecanismos que se deben aplicar y seguir según las singularidades de cada aplicación. Con esto se ha conseguido converger a la solución que se presenta para el caso de uso del presente proyecto.

- **Capítulo 3**. Objetivos. Se presenta brevemente el caso de uso y se expone la planificación que ha de permitir que se presente la solución con éxito. Finalmente se expone el ecosistema de aplicaciones que se van a utilizar para poder resolverlo.
- **Capítulo 4**. Contribución del trabajo. En este apartado se expone cómo se ha afrontado el caso de uso y de qué manera se ha decidido resolver. Ésta es la parte más técnica, en la cual se consigue llevar a la práctica todo lo aprendido en el apartado de estado del arte. Gracias a la orquestación de las diferentes tecnologías presentadas, éstas son utilizadas y emplazadas con el fin de conseguir un entorno plenamente en ejecución y listo para probar.
- **Capítulo 5**. Se exponen los resultados obtenidos de las pruebas realizadas sobre las diferentes partes tecnológicas que conforman el proyecto. También se muestra el resultado de la valoración del proyecto por parte de dos usuarios expertos.
- **Capítulo 6**. Conclusiones. En este apartado se reflexiona sobre el trabajo realizado y se exponen una serie de valoraciones respecto a cómo afrontar un proceso de modernización.
- **Capítulo 7**. Futura evolución del software. Se proponen mejoras y aquellos puntos que se deben evolucionar del proyecto con el fin que se optimicen las funcionalidades para las cuales se ha diseñado. Estas mejoras inciden también en la reducción de los costes de mantenimiento.

3.2. Objetivo general

El objetivo general del proyecto consiste en diseñar una solución, plenamente funcional, que afronte la modernización de una aplicación utilizando la estrategia de caja negra sobre una aplicación heredada. Para ello se van a instalar y desarrollar un pequeño ecosistema de herramientas, así como toda su interacción, con el fin de dar respuesta a cualquier operación que sea requerida por el nuevo sistema de aplicaciones.

3.3. Objetivos específicos

- Familiarizarse con la gestión de aplicaciones con Keycloak, una solución de gestión de acceso e identidad.

- Estudio de parte del ecosistema de herramientas Spring Framework en el desarrollo de aplicaciones. Con estas tecnologías se van a desarrollar la nueva aplicación de ventas y los microservicios que permitirán el consumo de datos de la aplicación heredada.
- Adquirir conocimientos de Spring Security. Cómo y cuándo se utiliza para la integración de aplicaciones con servidores de identidad y de qué manera se resuelve la customización de un proveedor de autenticación.
- Aplicar una solución con Spring Cloud Config Server. Este sistema permite externalizar la configuración de las aplicaciones. Las propiedades ya no estarán autocontenidas dentro de un fichero en la misma aplicación, sino que serán provistas remotamente por este servidor. Con esto se consigue que se adapten a un entorno plenamente distribuido y escalable, en el cual, según los parámetros de configuración del nuevo entorno, el servidor Cloud Config proveerá de unas u otras propiedades.
- Conocer cómo se utiliza la solución Eureka Netflix Server. Entender el funcionamiento y poner en práctica de toda una arquitectura de microservicios. Comprender cómo se registran y cómo se realiza el consumo por parte de las aplicaciones clientes. También, cómo se escalan y de qué manera encaja el servidor de configuración Cloud Config Server con este componente.

3.4. Metodología

En el mundo del desarrollo web existen distintas metodologías que se pueden utilizar para el desarrollo del software; desde metodologías más tradicionales como pueden ser PMBOK o CMMI, hasta metodologías ágiles como pueden ser Scrum o Kanban.

Se pueden adoptar metodologías tradicionales cuando se tiene un conocimiento muy extenso y acotado del caso de uso, éste no será propenso a sufrir variaciones significativas y donde los requisitos iniciales del proyecto difícilmente van a cambiar. Estas tecnologías tienen un enfoque predictivo, a partir del cual se traza una línea secuencial de desarrollo del trabajo y con una única dirección. Una metodología ágil acepta cambios y no se fundamenta con unos requisitos que poco deben variar. El cambio forma parte de la evolución nativa en el desarrollo del software. Si bien versiones más nuevas de las tecnologías tradicionales empiezan a aceptar el cambio como parte del desarrollo del software, aún no extraen al desarrollador de la carga en la gestión que implica la actualización de la documentación en el desarrollo del software.

Para el caso de estudio de este proyecto, al ser tecnologías que en la gran mayoría no se conoce su funcionamiento o aplicación, se opta por la tecnología ágil Scrum.

- Para el desempeño del presente proyecto se han definido las distintas tareas del proyecto en el Product Backlog, las cuales van definiendo las historias que se incorporan en los distintos sprint del proyecto. Una ventaja significativa de un enfoque ágil del proyecto es la capacidad de proporcionar en el final de cada Sprint de un entregable funcional. Esto permite aflorar posibles problemas en la realización del proyecto, pero también permite redefinir pequeños acondicionamientos para los demás sprint. El proyecto se divide en dos tipos de sprint:
  - Sprints que se utilizan para testear y aprender de nuevas tecnologías.
  - Sprints que se utilizan para el engranaje y ensamblaje de todos los elementos que se van testeando, así como la realización del documento del proyecto.

Para el desempeño del proyecto final de grado se han definido sprint de dos semanas, excepto el último Sprint, que apenas llega a 3. Finalmente, van a ser 8 sprint, los cuales, a nivel de resumen, tienen como principales tareas:

Tabla 1. Planificación en Sprints del proyecto

Release planning	***Principales tareas de cada* Sprint**
Sprint 1 **15 -28 marzo** **40 horas**	• Instalación del sistema Raspbian en el microordenador Raspberry Pi. • Keycloak: instalación en la Raspberry, con una base de datos Postgresql. • Testeo de Keycloak. Aprendizaje del funcionamiento de la gestión de usuarios, roles y aplicaciones dentro de un realm.
Sprint 2 **29 marzo – 11 abril** **40 horas**	• Integrar una primera aplicación con Spring Security delegando la autenticación de usuarios a Keycloak. • Comprender como funciona Spring Security y dónde puede encajar el consumo de microservicios para inyectar los roles de la aplicación heredada dentro de este framework.

Sprint 3 **12 – 25 abril** **40 horas**	• Aplicar una solución Spring Cloud Config en la aplicación creada en el anterior sprint. Instalar, comprender y probar cómo se pueden proveer las configuraciones de diferentes aplicaciones o servicios externos, dentro de los distintos entornos que pueden llegar a tener las aplicaciones. El repositorio escogido para guardar estas configuraciones es GitHub. • Empezar con el documento del proyecto, focalizándose en el apartado del estado del arte.
Sprint 4 **26 abril – 9 mayo** **40 horas**	• Preparar el borrador para entregar el documento y empezar con el estudio del funcionamiento de una arquitectura de microservicios. • La herramienta escogida para unas primeras pruebas es Eureka Netflix Server. Empezar una prueba funcional de este software.
Sprint 5 **10 – 23 mayo** **40 horas**	• Continuar con las pruebas funcionales del Eureka Netflix Server e instalar este software cómo un servicio en la Raspberry. • Externalizar las configuraciones del microservicio y de Eureka Netflix Server con Spring Cloud Config Server. • Crear el microservicio que consultará de los roles de la aplicación heredada. Éste se va a registrar a Eureka Server, con su configuración en el Cloud Config Server. • Empezar las pruebas de cómo se configuran y arrancan aplicaciones serverless Java.
Sprint 6 **24 mayo – 6 junio** **40 horas**	• Empezar a integrar todos estos componentes para disponer de unas primeras funcionalidades del caso de uso en la Rapsberry Pi. • Continuar con el documento del proyecto.
Sprint 7 **7 – 20 junio** **40 horas**	• Instalar la aplicación principal de ventas en la Raspberry Pi, con Spring Security + Spring Cloud Config Server + Cliente de los servicios de Eureka Netflix Client. • Últimas pruebas del código + documentación del código.

	• Continuar con el documento del proyecto.
Sprint 8 **21 junio – 8 julio** **50 horas**	• Realizar el microservicio que recupere el listado de clientes de la aplicación heredada. • Dejar el entorno de la Raspberry listo para pruebas externas. • Testeo de todos los componentes empleados. • Revisión del código del GitHub para mejorar y documentar los diferentes elementos del proyecto. Pequeñas refactorizaciones de código. • Finalizar el documento del proyecto.

Los puntos de historia calculados una vez realizado el primer Sprint han permitido replantear y recalcular los demás sprints, al tener una visión más veraz de la velocidad del trabajo. Con ellos se consigue limitar mejor aquello que se puede estudiar y aquello que se puede entregar en cada Sprint.

3.5. Herramientas empleadas

Se describen las principales herramientas software y hardware utilizadas para resolver el enfoque práctico del caso de uso del proyecto:

- **Raspberry Pi 4**. Forma parte de la familia de ordenadores de una sola placa desarrollados por Rapsberry PI Foundation, donde el diseño se centra en el tamaño reducido de sus componentes. En este microordenador serán instalados todos los sistemas de software utilizados e implementados para el proyecto.
- **Raspbian GNU / Linux 10 – buster**. Raspberry Pi Os basada en una distribución derivada de Debian optimizada para el hardware de Raspberry Pi.
- **PostgreSQL**. Sistema de gestión de bases de datos relacional y orientado a objetos, de código abierto y multiplataforma.
- **Wildfly 13.0.3**. Servidor de aplicaciones de código abierto basado en Java EE. Es multiplataforma al ser ejecutado sobre la máquina virtual java.

- **Keycloak 12.0.4 – released**. Es una solución de acceso e identidad de código abierto. Es también un sistema Single-Sign On, con lo cual, con una sola vez que se introduzcan las credenciales, se permite el acceso a múltiples aplicaciones.
- **Gradle.** Es un sistema de automatización de código.
- **Spring Boot y parte del ecosistema Spring Framework.** Es un sistema que permite la configuración inicial y preparación de las aplicaciones java de una manera muy simple y ágil. A través de sus starters se van añadiendo configuraciones predeterminadas que añaden funcionalidades que solo requieren de configuración.
- **Conocer Spring Cloud.** De qué manera su pueden escalar las aplicaciones y qué mecanismos agilizan una replicación de aplicaciones. Concretamente se estudian las tecnologías de Spring Cloud Config Server y Eureka Netflix Server.
- **Intellij.** IDE utilizado para el estudio y desarrollo de los diferentes elementos que conforman el proyecto.
- **Scrum:** metodología ágil para el desempeño del proyecto.

4. Contribución del trabajo

4.1. Introducción

La empresa FuxPin S.L., dedicada a trabajos de carpintería y pintura del hogar, quiere actualizar parte del ecosistema de sus aplicaciones. En primer paso, para empezar el proceso de viabilidad de estas actualizaciones, se pide proceder con la renovación de la aplicación de gestión de ventas. Esta aplicación ha sido mantenida por distintas empresas tecnológicas las cuales han realizado modificaciones fuertemente customizadas. Durante las entregas del producto, la empresa no ha tenido implantada una política de control y de calidad de los cambios, con lo cual, el software presenta una significativa complejidad de compresión. Esta aplicación, además, está implementada sobre una solución tecnológica que se ha quedado sin soporte. Sin embargo, a nivel funcional, el software desempeña una función altamente crítica para la empresa, siendo clave dentro de la línea estratégica de la lógica de negocio de la organización.

Por estos motivos, esta aplicación ha acabado siendo una aplicación heredada. En este componente raramente se hacen cambios y solo se realizan aquellas modificaciones que son imprescindibles y aseguran el correcto funcionamiento de la aplicación.

4.2. Contexto del problema

Después de una serie de estudios y valoraciones, FuxPin S.L. ha decido no asumir la inversión de recursos económicos y de personal necesarios para el desempeño de una migración completa del sistema o para la implementación de una nueva aplicación con la misma lógica de negocio que la aplicación original. Por estos motivos, se le va a proporcionar una nueva aplicación, pero con funcionalidades que, por su complejidad, se deben reutilizar de la aplicación heredada. Por tanto, se pretende proveer a la aplicación heredada de los mecanismos que le permitan emitir cualquier operación que ahora solo es visible en las interfaces internas de la aplicación. Con todo esto, se va a conseguir una reducción significativa del coste de desarrollo de la nueva aplicación.

Como primer paso para el estudio de este nuevo sistema, la nueva aplicación, a diferencia de la aplicación heredada, debe externalizar la autenticación y autorización de los usuarios, utilizando un nuevo servidor de gestión de acceso e identidad. Este sistema debe acabar adoptándose de forma gradual por todas las demás aplicaciones de la empresa. Se requiere también que se opte por un sistema Single-Sign On. Esto va a permitir que, con una sola autenticación, se tenga acceso al resto de aplicaciones.

A su vez, para permitir el acceso a las diferentes acciones de la nueva aplicación, los roles deben ser gestionados por el mismo sistema de gestión de identidad. Sin embargo, hay roles que solo estarán definidos en la aplicación heredada. Será en el momento de autenticarse en la nueva aplicación cuando estos roles heredados deberán ser inyectados a los nuevos roles definidos en el nuevo sistema.

Un usuario que acceda a la nueva aplicación, por tanto, podrá recibir los roles de estas dos fuentes. Sin embargo, los roles que se obtienen de la aplicación heredada deben servir para permitir realizar ciertas operaciones o acciones que se encuentren mapeadas en la capa de servicios que va a envolver la aplicación heredada. Es decir, los roles de la aplicación heredada serán utilizados para poder mostrar en la nueva aplicación las operaciones que se aún van a ser vigentes y operativas de la aplicación heredada. Las demás operativas se añadirán en posteriores desarrollos del proyecto, con el fin de ampliar el wrapping funcional de la aplicación heredada.

Otro punto que se va a reutilizar de la aplicación heredada es el listado de clientes que este sistema tiene en su base de datos. Con esto se debe poder listar esta información dentro de la nueva aplicación.

4.3. Concreción del caso de uso

Para el desarrollo del proyecto se va a realizar el caso de uso de la autorización y autenticación de la aplicación de ventas. Concretamente se iniciará el proyecto con la descentralización del sistema de autenticación de usuarios, donde actualmente estos permisos se encuentran integrados y gestionados por la misma aplicación de ventas heredada.

Se quiere así segregar el control del acceso de usuarios para delegarlo al sistema de identidades Keycloak. Será el departamento de administración el que, a través de un usuario con role gestor en Keykloak, añadirá los usuarios junto con sus roles dentro de una aplicación definida en un realm de Keycloak. En la nueva aplicación de ventas puede haber diferentes tipos de perfiles: particulares, representantes de empresas, representantes de entidades, etc., cada una con unas acciones singulares y específicas en el momento de acceder a la aplicación.

Este caso de uso es el paso inicial que requiere la empresa para valorar y plantear la posterior adaptación de este sistema de autenticación al resto de aplicaciones, donde, hoy en día, cada una tiene su propio control de acceso y permisos de usuarios.

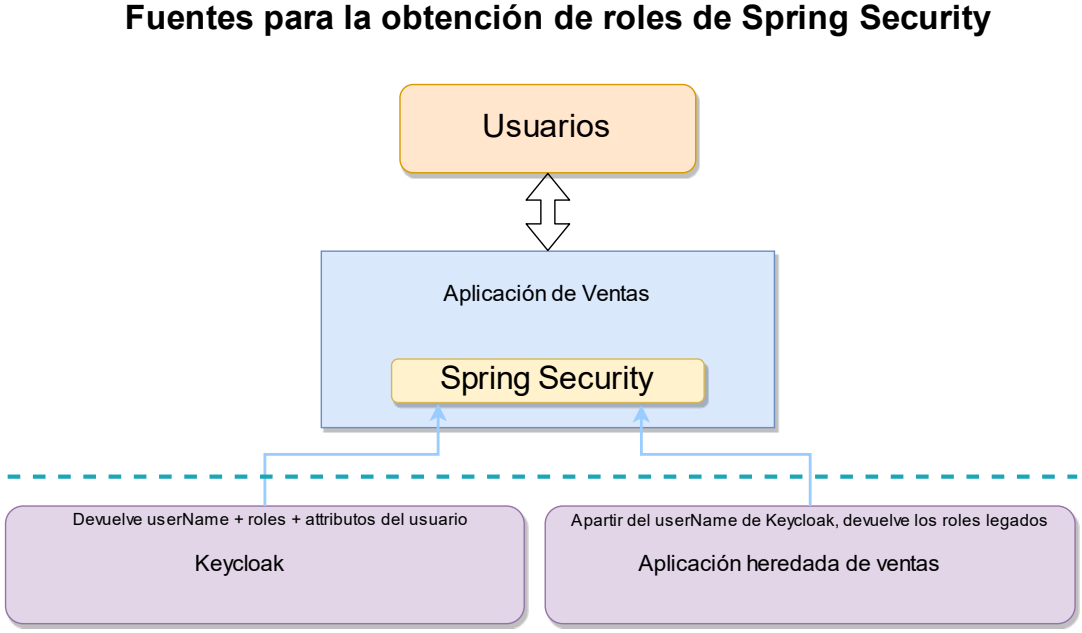

Figura 7. Proceso de recuperación de roles de Spring Security

Fuente de elaboración propia.

Sin embargo, como requerimiento añadido, se delegará también la autorización a la actual aplicación heredada, con el fin de inyectar posibles roles que contenga dicha aplicación y que no han sido definidos dentro *Keykloak*. En la figura 7 se describe gráficamente el proceso de recuperación de roles de ambos sistemas. Estas dos fuentes van a determinar todos los permisos sobre las diferentes acciones del sistema legado.

Por tanto, los roles pueden venir determinados por dos fuentes (representados en la figura 7):

- Los roles definidos dentro de Keycloak.
- Los roles definidos en la aplicación heredada.

Para conseguir este funcionamiento, una vez Spring Security recupere los datos de Keyckoak, a partir del código de usuario que este sistema devuelve, este campo será enviado a un microservicio implementado para esta finalidad, para que éste devuelva los roles del sistema legado. Una vez se vuelva a la aplicación, toda esta comunicación será transparente al usuario y le permitirá interactuar en el nuevo sistema a partir de los roles obtenidos de estas dos fuentes.

Una vez realizada esta parte, en una segunda acción sobre el sistema legado, el wrapping funcional deberá de proveer del listado de clientes de la aplicación heredada. Es decir, deberá aportar de una llamada que permita poder consultar por el listado de antiguos clientes para que estos sean presentados en la nueva aplicación.

Hay tres grandes ejes que se deben abordar para el correcto desarrollo del caso de uso:

- Conocer el funcionamiento de Keycloak. Cómo se define una aplicación, los usuarios, sus respectivos roles y, todo ello, dentro de un *realm*.
- Conocer cómo se integra Spring Security con Keycloak para la autenticación del usuario. Estudiar cómo se delega la autorización a través de los roles. Definidos, por un lado, en el mismo Keycloak y, por otro, en la aplicación heredada.
- Aplicar una solución de *wrapping* funcional. Ésta consistirá en la implementación de la capa de microservicios que va a envolver cualquier operación sobre la aplicación heredada. Se debe proporcionar de:

- Con el fin de implementar un primer microservicio que devolverá los roles de la aplicación heredada a Spring Security para que estos terminen inyectados en la nueva aplicación.
- Realizar un microservicio que permita recuperar el listado de clientes de la aplicación heredada.

4.4. Conociendo Keycloak

Una vez se accede a la consola de administración de Keycloak, se procede a dar de alta a una aplicación teniendo presente las siguientes terminologías:

- Un cliente es, para el caso de uso, la nueva aplicación de ventas.
- Un realm está constituido por un conjunto de clientes. Un usuario que accede a una aplicación lo hará a través de un realm. Por tanto, un usuario no podrá acceder a las aplicaciones de los otros realms.
- Usuario: usuario que accederá a la aplicación de ventas.
- Role: Delimitará los permisos dentro de la aplicación de ventas. Estos roles son mapeados a través de GrantedAuthority en Spring Security.

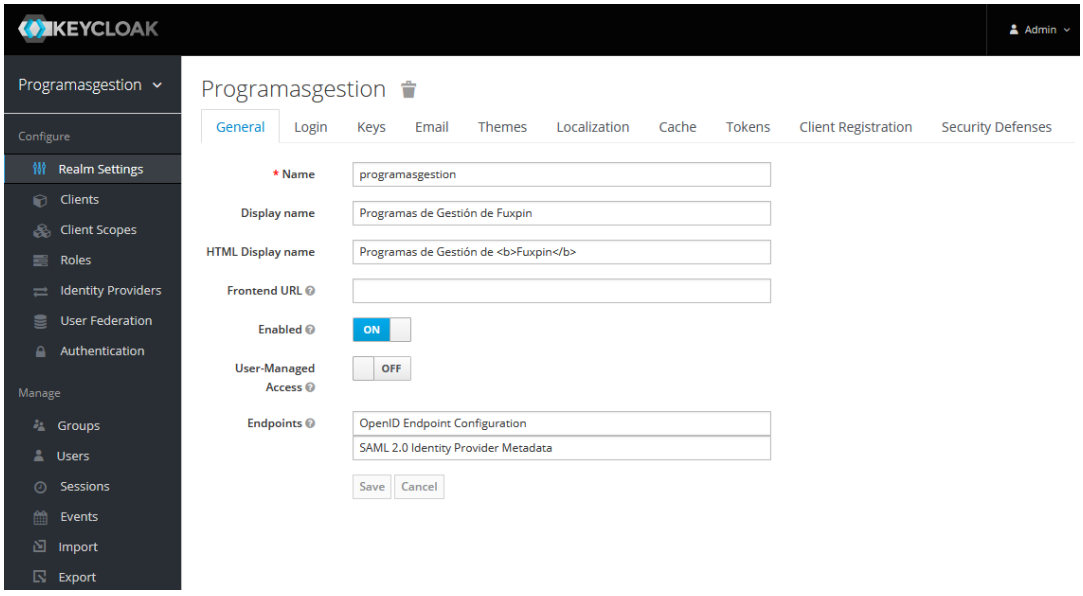

Figura 8. Edición de un realm en Keycloak

Fuente de elaboración propia.

Antes de empezar con el alta de aplicaciones dentro de este sistema de autenticación y autorización, se debe dar de alta el realm donde estarán definidas las aplicaciones de FuxPin S.L.

A partir de la creación del realm, el siguiente paso es definir las aplicaciones que los van a conformar. Tal y como se puede observar en la figura 8, Keycloak tiene una consola de administración muy intuitiva que permite definir de una forma muy ágil los diferentes elementos que van a permitir a los usuarios el acceso a las aplicaciones.

4.5. Flujo de autenticación y autorización de usuarios

El proceso de autenticación se inicia en el momento que un usuario accede a la parte protegida de la aplicación, es en este momento cuando el flujo de autenticación va a redireccionar al usuario a la pantalla inicial de Keycloak, en la cual, se le van a pedir por sus credenciales. Una vez éstas son validadas, se devolverá al usuario a la nueva aplicación de ventas.

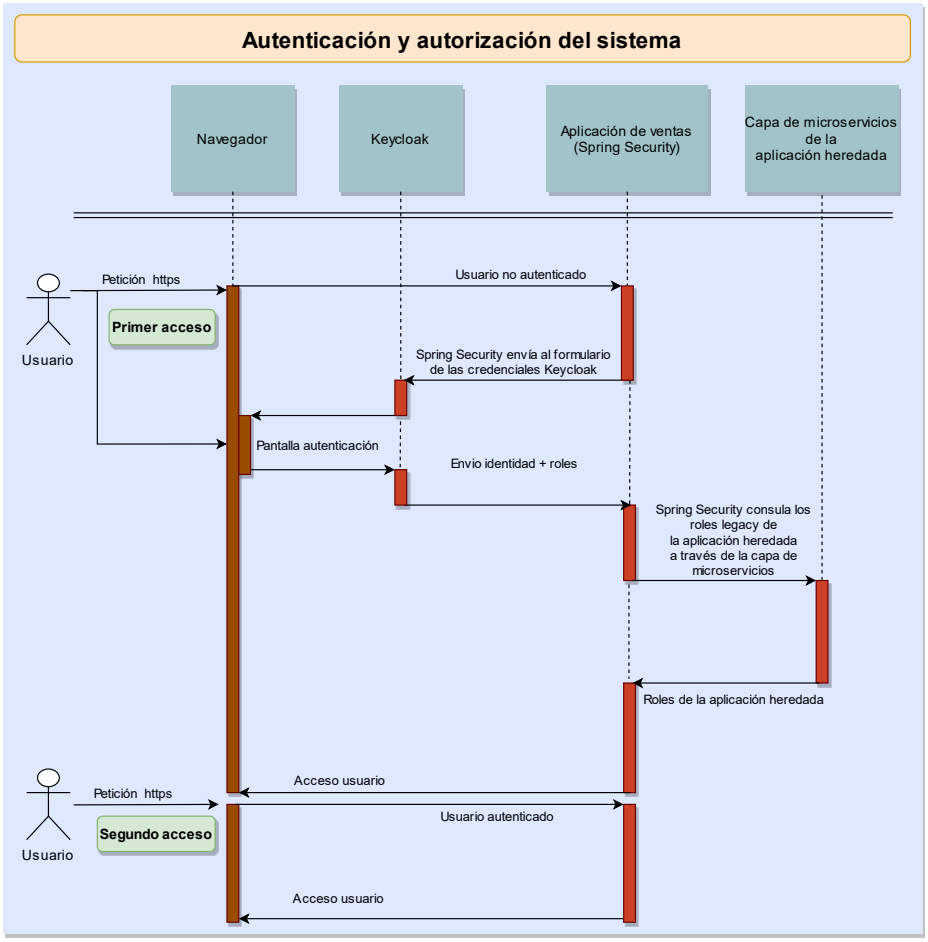

Figura 9. Flujo de autenticación y autorización del usuario

Fuente de elaboración propia.

Como se ha comentado en anteriores puntos, antes de redireccionar a la nueva aplicación se deberá añadir, en la integración de Spring Security con Keycloak, la consulta al microservicio que va a recuperar los roles de la aplicación legada para que finalmente estos sean inyectados en el sistema de autorización. Este proceso de autenticación y autorización se puede observar gráficamente en la figura 9.

Los roles son los que van a determinar las operaciones que puede realizar el usuario en la nueva aplicación.

4.6. Modernización de caja negra y la capa de microservicios

Para afrontar una solución efectiva de cómo debe la aplicación heredada publicar sus funcionalidades, se ha realizado un estudio de modernización de caja negra de la aplicación heredada, con el cual, se han analizado y detectado las entradas y salidas de las actuales interfaces del sistema heredado. Con esto consigue el sistema de operaciones que conforman la capa de wrapping de la aplicación heredada. De este modo se pretende encapsular y envolver todas las funcionalidades para que éstas sean reutilizadas, con el fin de conseguir un sistema con una nueva y moderna interfaz.

Para este caso de uso se va a implantar una capa de wrapping funcional, donde las acciones principales de la lógica de negocio serán:

- Detectadas: se analizan y detectan las acciones que son requeridas para la nueva aplicación y que seguirán desempañándose en la aplicación heredada.
- Adaptadas: Se adaptan al nuevo sistema tecnológico.
- Externalizadas: Se hacen visibles a la capa funcional.
- Publicadas: el wrapping funcional las ofrece a todos los sistemas externos que requieran de su consulta.

También se ha realizado un estudio de la base de datos con el fin de analizar posibles acciones que puedan ser necesarias para la nueva aplicación, donde la información es consultada directamente y expuesta también a la nueva aplicación a través del wrapping funcional.

Una vez realizado el estudio sobre el sistema legado, el siguiente paso ha consistido en diseñar la arquitectura sobre la cual se obtendrán todas estas funcionalidades. La modernización por wrapping se va a realizar con una capa de mensajería REST. Esta capa será el punto central

donde se proveerá de cualquier operación sobre el sistema heredado. Esta mensajería REST, a su vez, se obtendrá implantado una arquitectura de microservicios.

En la figura 10 se puede observar la arquitectura propuesta para el wrapping funcional. Tal y como se puede observar, se ha seleccionado la solución Spring Eureka Netflix Server para sistema central y de enjambre de los microservicios.

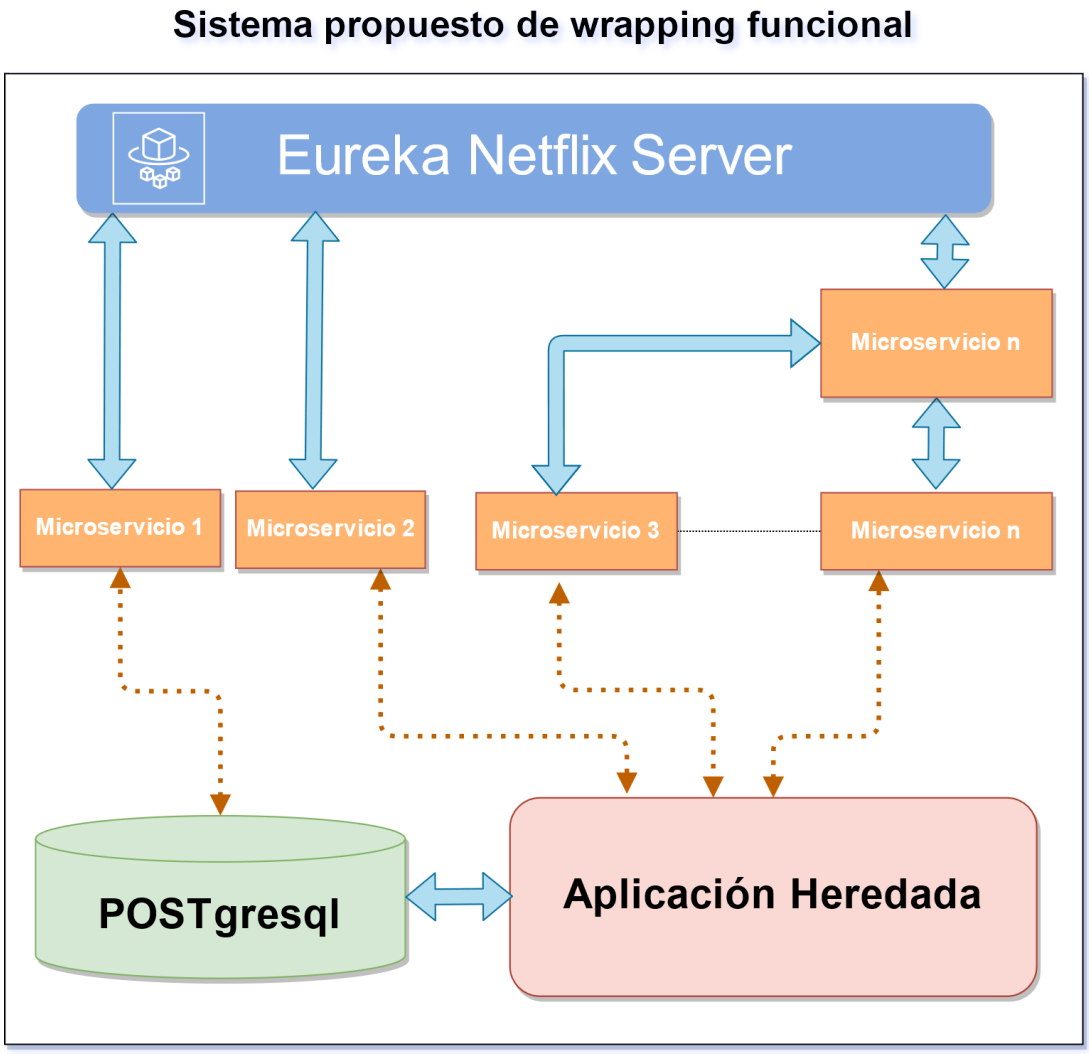

Figura 10. Solución propuesta de wrapping funcional

Fuente de elaboración propia.

4.7. Requisitos del sistema

Durante la fase de toma de requisitos de proyecto, se han definido el conjunto de requisitos funcionales y no funcionales que el nuevo sistema debe cumplir. Estos requisitos deben asegurar un mínimo de calidad del producto entregado y que responda realmente a aquellas funcionalidades para las cuales se ha diseñado.

4.7.1. Requisitos funcionales

En las siguientes tablas se muestran los requisitos funcionales que determinan la implementación del caso de uso.

Tabla 2. Requisitos funcionales de Keycloak

Keycloak	***Requisitos***
Gestión de acceso e Identidades	• Debe aportar la configuración de una aplicación desde el mismo sistema de gestión de acceso e identidad. • Gestión de usuarios, roles y realms. • Posibilidad de activar una autenticación 2FA. • El sistema debe ser Single-Sign On y Single-Sign Out. • Soporte OpenID Connect.

Tabla 3. Requisitos funcionales la aplicación de ventas

Aplicación de ventas	***Requisitos***
Seguridad	• Protegida con Spring Security y capacidad de inyectar los roles de dos fuentes distintas: Keycloak y la aplicación legacy • El sistema debe aportar la seguridad a la aplicación y solo permitir acceder a aquellas aplicaciones que estén dentro del mismo realm. • Un usuario no debe realizar, según su role, una acción o acceder a una página donde no tiene permisos.
Datos de los usuarios	• Toda la información de los usuarios deberá recuperarse de keycloak.
Acciones de la aplicación	• Debe aportar una pantalla para consultar la información recuperada de los usuarios de keycloack.

	• La página de la gestión de clientes se podrá visualizar por todos los usuarios. • La página de consulta de clientes del sistema legado solo se podrá visualizar por los usuarios con rol admin. • En la página de la gestión de ventas, debe aportar un gráfico de evolución de productos vendidos y datos de almacenes. Estos se deben visualizar por todos los usuarios. • En la página de la gestión de ventas, los datos de los productos solo se deben visualizar a los usuarios con role admin.

Tabla 4. Requisitos funcionales del sistema de microservicios

Sistema de microservicios	***Requisitos***
Funcionalidad	• Debe aportar un microservicio que permita recuperar los roles de la aplicación heredada. • Debe aportar un microservicio que recupere los clientes de la aplicación heredada.

4.7.2. Requisitos no funcionales

En las siguientes tablas se muestran los requisitos no funcionales que se deben cumplir con el fin de aportar calidad al software.

Tabla 5. Requisitos no funcionales de Keycloak

keycloak	***Requisitos***
Calidad	• Se debe proporcionar un manual a los gestores de usuarios y roles de keycloak.

Tabla 6. Requisitos no funcionales del sistema de microservicios

Sistema de microservicios	***Requisitos***
Rendimiento	• El sistema debe tener una disponibilidad muy alta. • Los servicios deben ser escalables. • El proceso de replicar un microservicio debe ser inmediato.
Funcionalidad	• Los microservicios deben ser totalmente transparentes a las aplicaciones clientes u otros microservicios. • El consumo de las interfaces entre consumidor – productor debe ser simple y con una interfaz de integración muy simple.
Calidad de código	• El código del microservicio tiene que estar bien documentado. • El código debe estar bien estructurado, primando que los servicios siempre vengan representados por una interfaz. • En el MarkDown del componente se debe explicar las herramientas utilizadas y como se interactúa con él, además de aquella información que se considere importante. • Utilización del plugin SonarQube Analyzer de Intellij.
Seguridad	• Todos los microservicios que se registren al servidor Eureka Netflix Server deberán hacerlo después de una autenticación básica con Spring Security.

Tabla 7. Requisitos no funcionales del servidor de configuraciones

Spring Cloud Config Server	***Requisitos***
Seguridad	• El sistema debe aportar la seguridad a la aplicación y solo permitir acceder a aquellas aplicaciones que conocen del

	usuario y contraseña de la autenticación básica proporcionada por Spring Security.
Rendimiento	• El sistema debe tener una disponibilidad muy alta. • El sistema debe ser fácilmente escalable.

4.8. Diseño de la propuesta

Para el desempeño de este proyecto se procederá con el desarrollo de cada uno de los siguientes componentes que conforman el proyecto:

- Se procederá con la instalación de Keycloak, para su posterior configuración y estudio de funcionamiento. Es este sistema se darán los roles a los usuarios que van a delimitar las nuevas acciones que se realicen en la nueva aplicación de ventas.

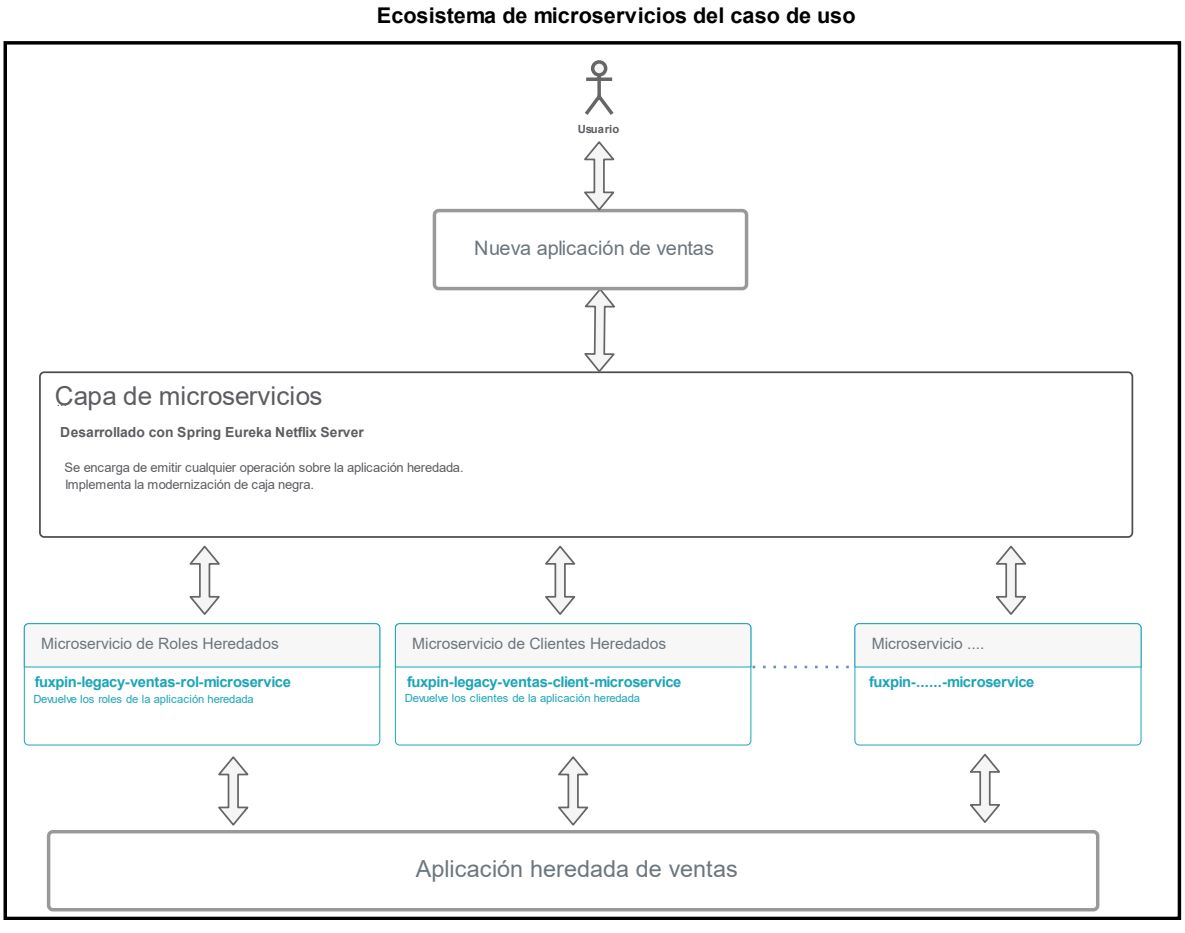

Figura 11. Diseño del sistema de microservicios para el caso de uso

Fuente de elaboración propia.

- Desarrollo de una nueva aplicación de ventas con Spring Boot. Esta aplicación será implementada con toda la nueva lógica de negocio, donde, además, tendrá una parte de acciones que utilizará a través del wrapping funcional de la aplicación heredada.

- Desarrollo de una capa de microservicios. Ésta será la emisora de la parte funcional de la aplicación heredada. La aplicación heredada solo será accesible a través de este sistema de wrapping. Como se puede observar en la figura 11, éste será el único punto de acceso a la lógica de negocio del antiguo sistema.

- Desarrollo de un microservicio que se integrará con la capa de microservicios y ofrecerá los roles heredados a los usuarios que accedan a la nueva aplicación de ventas.

- Desarrollo de un microservicio que devolverá el listado de clientes de la aplicación heredada para que estos sean listados en la nueva aplicación.

La realización de ambos microservicios asentará las bases, para posteriores desarrollos, de la adaptación progresiva de parte de la funcionalidad de la aplicación heredada.

4.8.1. La nueva aplicación de ventas

Tabla 8. Descripción de la nueva aplicació de ventas

Componente	**fuxpin-aplicacion-ventas**
Descripción	Nueva aplicación de ventas que permitirá le gestión de clientes, clientes legados, materiales, almacenes y ventas.
Ubicación del código fuente	https://github.com/xrodriguezang/fuxpin-aplicacion-ventas

Como se ha dicho en anteriores puntos, la funcionalidad de esta aplicación es la de proveer a los usuarios de una nueva aplicación web para la gestión de ventas. A su vez, pequeñas partes de su lógica de negocio quedan delegadas a la aplicación heredada. A través del wrapping funcional se obtienen los roles heredados así como de las funcionalidades heredadas que se van a utilizar en la nueva aplicación.

Como elementos tecnológicos más relevantes para el caso de uso:

- Protegida con Spring Security. Este framework se integra con Keycloak y la capa de microservicios, ofreciendo total transparencia en el sistema de autenticación y autorización a la nueva aplicación.
- Obtiene su configuración de un servidor Spring Cloud Config Server.
- Será cliente de un servidor de microservicios implementado e integrado con Spring Eureka Netflix Server.

En la figura 12 se representa el diseño arquitectónico de los diferentes componentes que se van a emplear para el desarrollo de la aplicación. Gracias a la integración de todas estas pequeñas soluciones tecnológicas, se obtiene una aplicación plenamente ejecutable.

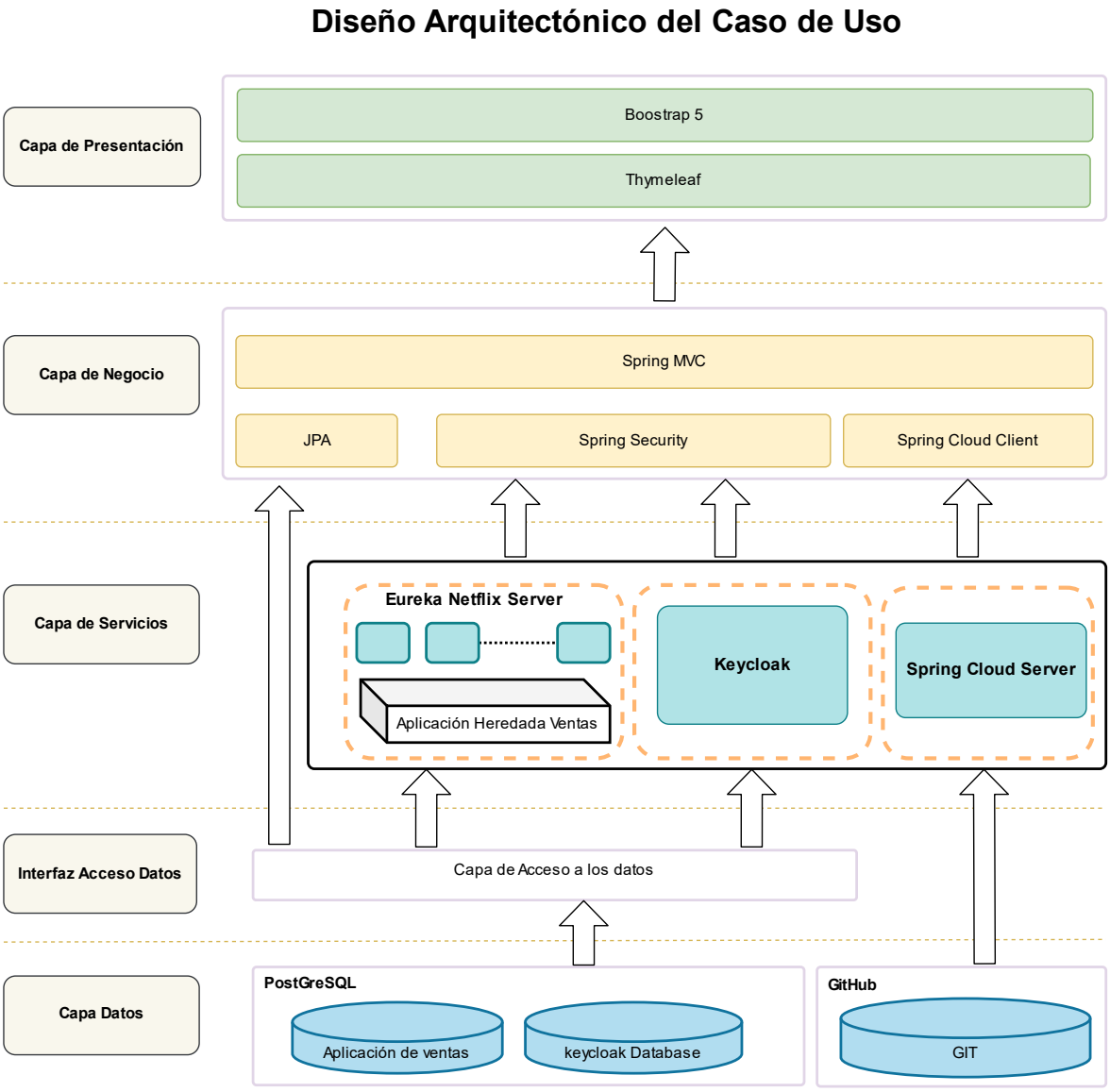

Figura 12. Diseño arquitectónico del caso de uso de la aplicación de ventas

Fuente de elaboración propia.

La integración con el servidor de identidades no consiste en una única integración con el servidor de autenticación, sino que también se encuentra integrado con un microservicio con el fin de inyectar los roles que no han sido dados de alta en el nuevo sistema.

Inicialmente, para este caso de uso, esta aplicación tiene definidos dos roles:

- **admin**. Este role viene determinado por la aplicación heredada. Con este role se va a determinar el acceso y la ejecución de las demás acciones heredas que se van a desarrollar en el futuro y que van a formar parte de la capa de wrapping.
- **user**. Este role se configura dentro de Keycloak para esta aplicación. Este rol determina también el acceso y ejecución de ciertas acciones en la nueva aplicación.

En la siguiente tabla se muestran las acciones se han realizado para el caso de uso del proyecto:

Tabla 9. Acciones de los roles de la nueva aplicación de ventas

Role	***Acciones***
admin	• Accede a su perfil para poder visualizar sus datos personales. • Acceso a la página de gestión de los clientes. • Acceso a la página del listado de clientes del sistema legado. Solo este role tiene dispone de esta visualización. • Visualización, dentro de la gestión de ventas, del gráfico que muestra la evolución de ventas y datos de los almacenes. • Visualización, dentro de la gestión de ventas, del listado de productos junto con el almacén o almacenes donde se encuentran almacenados. Solo este role puede visualizar esta información dentro de esta página.
user	• Accede a su perfil para poder visualizar sus datos personales. • Acceso a la página de gestión de los clientes. • Visualiza, dentro de la gestión de ventas, el gráfico de evolución de ventas y datos de los almacenes.

Mediante el uso de Spring Security se protege el back-end de posibles accesos a acciones con roles sin permiso. A su vez, en el front-end, la utilización de Thymeleaf - Spring Security permite discriminar para que se acabe mostrando cierta información en las páginas según el role del usuario.

Para la capa de persistencia:

1. Utilización de PostgreSQL. Éste será el soporte de base de datos y en el cual está contenida toda la información del modelo de datos.
2. Utilización de Spring Data – JPA. Agiliza la interacción con la base de datos PostgreSQL.

Para la capa de presentación:

1. Thymeleaf. Gracias a este framework se recupera toda información del back-end para poderse mostrar en pantalla.
2. Boostrap 5. Framework que aporta sencillez y facilidad para la presentación de los datos en pantalla.

4.8.2. Servidor de configuración Spring Cloud Config Server

Tabla 10. Descripción de la solución Spring Cloud Config para las configuraciones

Componente	**fuxpin-cloud-config-server / fuxpin-properties**
Descripción	Sistema que se encarga de proveer de todas las configuraciones a todas las aplicaciones que conforman el ecosistema Fuxpin S.L.
Ubicación del código fuente	Servidor: https://github.com/xrodriguezang/fuxpin-cloud-config-server Propiedades: https://github.com/xrodriguezang/fuxpin-properties

Esta tecnología es una pieza clave para dotar a todos los componentes del proyecto de escalabilidad de un modo muy ágil. Con esto se consigue un requisito en el cual la empresa FuxPin S.L. ha realizado especial hincapié: una disponibilidad muy alta del sistema. Esto se consigue con dotar al software de escalabilidad horizontal.

El Servidor de configuración fuxpin-cloud-config-server es la única aplicación del nuevo ecosistema de aplicaciones que tendrá su configuración autocontenida. Todas las demás aplicaciones van a recuperar sus configuraciones de este servidor. Esto se consigue con el paso variables de entorno o de sistema cuando las aplicaciones empiezan su ejecución. Las variables se encuentran configuradas por defecto en las distintas máquinas que forman el ecosistema de aplicaciones de FuxPin S.L.

Estas configuraciones, sin embargo, no están contenidas dentro de este mismo servidor, sino que estarán localizadas en un GitHub remoto. El funcionamiento de este mecanismo consiste en los siguientes puntos:

- El framework permite una gran variedad de opciones para poder recuperar los ficheros de configuración sobre un repositorio GitHub. Para este caso de uso, se han utilizado dos ramas: develop y main. Las propiedades de la rama develop son las utilizadas durante el proceso de desarrollo. Mientras que, cuando una aplicación se ejecuta en producción, las propiedades que facilita este servidor a las aplicaciones clientes son de la rama main (Spring Cloud, s.f.).
- Cuando el servidor de configuración arranca o se le fuerza una carga de contexto, éste realiza un pull del GitHub dentro de un directorio local de la máquina en la cual reside. La localización de dicho directorio se configura como una propiedad del servidor de configuración.
- Cada vez que una aplicación o microservicio arranque, este preguntará al servidor Cloud Server sobre su configuración. Es en este momento cuando el servidor de configuración vuelve a realizar un nuevo pull y así poder disponer de la última configuración subida de la aplicación.
- Cuando se produzca una posible pérdida de comunicación con GitHub, el servidor va a proveer de las últimas propiedades recuperadas del repositorio remoto. Por tanto, no se van a dejar las aplicaciones sin funcionar cuando éstas arranquen y falle la conexión con Github.
- Para dar protección a la comunicación entre las aplicaciones consumidoras y este servidor, se le ha añadido el framework Spring Security. Solo aquellas aplicaciones que conozcan de las credenciales definidas en este sistema podrán recuperar sus configuraciones.

- Al formar parte de la familia Spring Cloud, este servidor es fácilmente escalable. Con lo cual, en producción se podrá replicar una nueva instancia de este componente.
- Toda la conexión se realiza con protocolo https.

Con este sistema se van a proveer de todas las propiedades de las aplicaciones que se van a desarrollar dentro de la empresa FuxPin S.L. y estén diseñadas con Spring Framework. En la figura 13 se pueden observar los principales elementos que definen el servidor Cloud y todos aquellos componentes externos con este sistema.

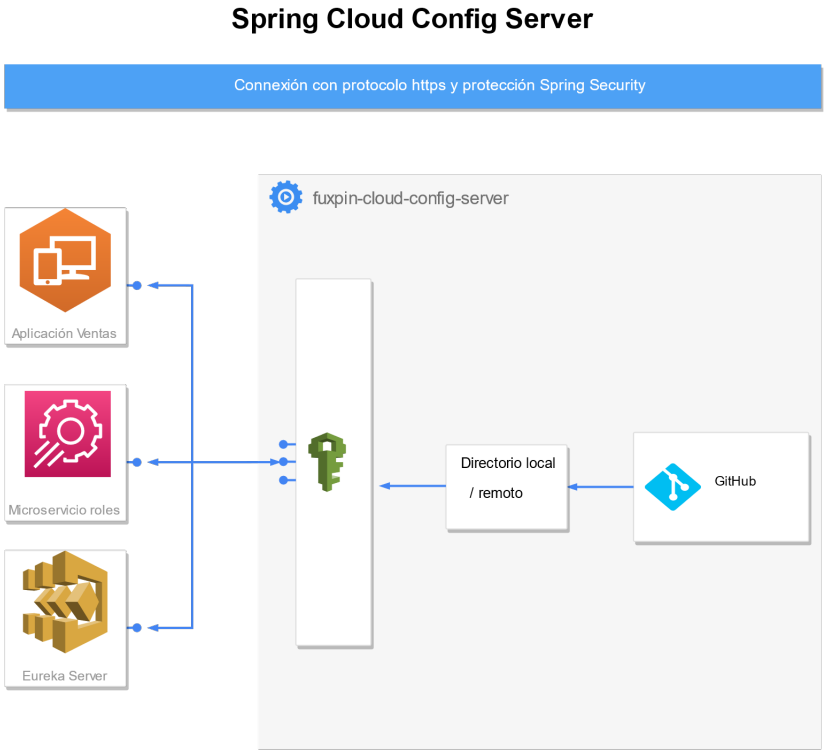

Figura 13. Sistema Spring Cloud Config Server

Fuente de elaboración propia.

Otro punto importante de este sistema es el mecanismo que sigue el servidor para poder proveer de las propiedades a las aplicaciones clientes. El repositorio GitHub contiene la estructura de directorios:

```
│ fuxpin-eureka-server.yml
│ fuxpin-role-legacy-microservice.yml
│ ventas.yml
│
├──localhost
│       fuxpin-eureka-server-localhost.yml
```

```
│           fuxpin-role-legacy-microservice-localhost.yml
│           ventas-localhost.yml
│
└─────production
            fuxpin-eureka-server-production.yml
            fuxpin-role-legacy-microservice-production.yml
            ventas-production.yml
```

Donde:

1. En la raíz se encuentra la configuración por defecto de las aplicaciones y comunes a todos los entornos.
2. En el directorio localhost estarán las propiedades exclusivas del entorno de desarrollo.
3. En el directorio production estarán las propiedades exclusivas del entorno de producción.

La comunicación entre una aplicación cliente y Spring Cloud Config se produce a partir de los siguientes parámetros:

1. Servidor donde se recuperan los datos. Estos pueden ser:

Tabla 11. Entorno de desarrollo y producción de fuxpin-config-server

Entorno desarrollo	**Entorno producción**
https://xrodrig.dnsnet.info:8446/fuxpin-config-server/	**https://pi.intranet.cat:8446/fuxpin-config-server/**
Caracterización: **Recupera los ficheros dentro del directorio raíz + /localhost**	**Caracterización:** **Recupera los ficheros dentro del directorio raíz + /production**

2. Nombre de la aplicación: por ejemplo, sería fuxpin-legacy-ventas-rol-microservice.
3. Rama donde recuperar la configuración: develop o main.

Con estos tres parámetros, el sistema cliente-servidor implementa las llamadas REST contractuales que van a permitir a la aplicación cliente arrancar correctamente, siendo una configuración prácticamente transparente a las aplicaciones.

4.8.3. La capa de microservicios de la aplicación heredada

Tabla 11. Descripción de la solución de microservicios Eureka Netflix Server

Componente	**fuxpin-eureka-server**
Descripción	Service Registry de la capa de microservicios. En este sistema se registran los microservicios que forman parte del wrapping funcional de la aplicación heredada. También se comunican con el servidor las aplicaciones clientes con el fin de consumir de dichos microservicios.
Ubicación del código fuente	https://github.com/xrodriguezang/fuxpin-eureka-server

Como herramienta de service discovery de microservicios se va a utilizar Eureka Netflix Server. Cada vez que un microservicio entre en ejecución, éste se va a registrar al servidor de registro de Eureka Server para que sea consumido por los demás microservicios u aplicaciones clientes. Este servidor va a recuperar sus propiedades gracias al servidor de configuraciones fuxpin-cloud-config-server.

Se puede observar en la figura 14 los principales componentes tecnológicos que caracterizan el servidor de microservicios y que ofrecen una total transparencia entre la comunicación entre los clientes y los microservicios.

Los microservicios que se registren en Eureka Netflix Server deben cumplir las siguientes singularidades:

- La integración de un microservicio o una aplicación cliente con Eureka Server es mediante autenticación básica - Spring Security.
- Los servicios deben ser escalables. Es decir, un mismo servicio se puede replicar en diferentes imágenes de un contenedor e incluso en la misma máquina, sin afectar el correcto funcionamiento de las demás instancias del mismo microservicio. Una vez se registran las n-instancias del mismo microservicio en Eureka Server, este servidor se encargará de balancear cualquier petición cliente sobre el ecosistema de microservicios.

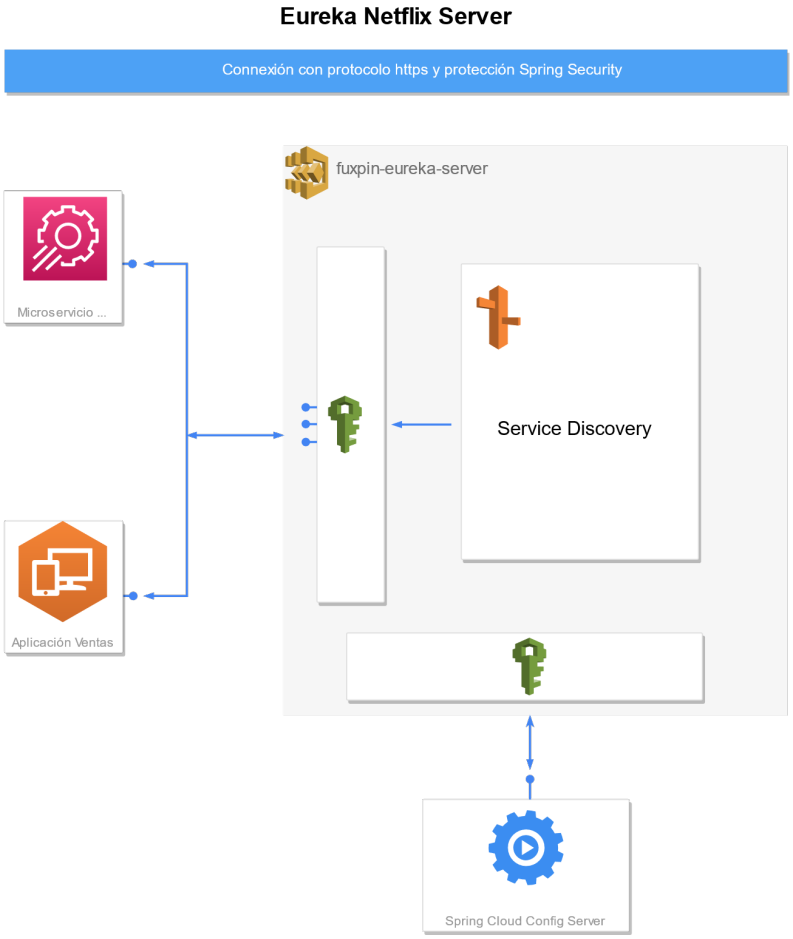

Figura 14. Descripción gráfica del servidor Eureka Netflix Server

Fuente de elaboración propia.

- En este sistema se estarán ejecutando todos los microservicios que forman parte emisora del wrapping funcional de la aplicación heredada. Por tanto, la aplicación legada solo será accesible a través de esta composición de microservicios.
- Los microservicios serán serverless. Es decir, con la entrega de un ejecutable se podrá ejecutar en cualquier máquina que disponga de una máquina virtual java de versión superior o igual a 11.

4.8.4. Microservicio de recuperación de role de la aplicación heredada.

Tabla 12. Datos del microservicio que devuelve los roles de la aplicación heredada

Componente	**fuxpin-legacy-ventas-rol-microservice**
Descripción	Microservicio que devuelve el role de la aplicación heredada.

Ubicación del código fuente	https://github.com/xrodriguezang/fuxpin-legacy-ventas-rol-microservice

Este componente tiene las propiedades:

- La configuración es proveída por el servidor fuxpin-cloud-config-server. Toda comunicación entre ambos sistemas se realiza con Spring Security y protocolo https.
- Se registra a fuxpin-eureka-server mediante Spring Security y protocolo https.
- Interactúa con la base de datos de la aplicación heredada con el fin de devolver los roles heredados.
- Dispone también de un método de monitorización, en el cual se informará del estado de la base de datos o del mismo sistema. Esta parte queda lista para ser consumida más adelante en posteriores desarrollos para un posible estudio de disponibilidades.
- La interfaz de contrato entre este microservicio y las aplicaciones clientes es muy sencilla y no requiere de implementación de código. Esto se obtiene gracias al uso de Spring Cloud Netflix Feign Client.
- Toda comunicación con el microservicio se produce mediante API-Rest.

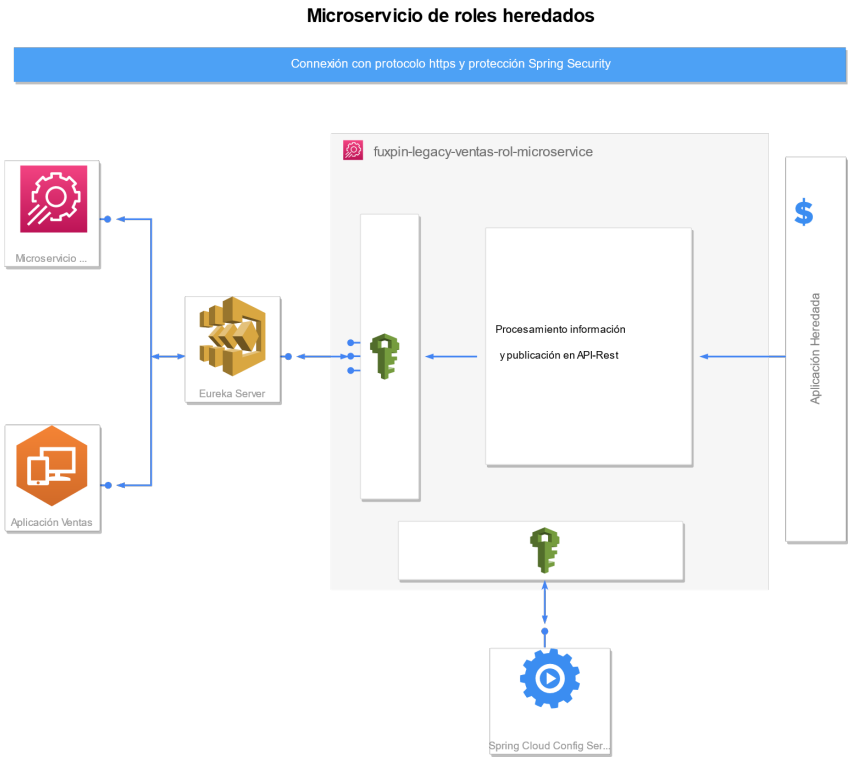

Figura 15. Descripción gráfica del microservicio de roles heredados

Fuente de elaboración propia.

- El microservicio recupera los roles de la aplicación a partir del código de usuario que se devuelve al usuario de Keycloak. Es Spring Security el que lo va a recuperar para posteriormente, durante el proceso de autorización, enviar al microservicio para que devuelva los roles heredados del usuario.

Se puede observar en la figura 15 de los principales componentes que caracterizan el microservicio de roles heredados. Este será el punto de recuperación de los roles de la aplicación heredada.

4.8.5. Microservicio de recuperación de los clientes de la aplicación heredada.

Tabla 13. Datos del microservicio que devuelve los clientes de la aplicación heredada

Componente	**fuxpin-legacy-ventas-client-microservice**
Descripción	Microservicio que devuelve el listado de clientes de la aplicación heredada.
Ubicación del código fuente	https://github.com/xrodriguezang/fuxpin-legacy-ventas-client-microservice

Este componente tiene las mismas propiedades que el microservicio descrito en el punto anterior, destacando:

- La configuración es proveída por el servidor fuxpin-cloud-config-server y se registra a fuxpin-eureka-server mediante Spring Security y protocolo https.
- Fácilmente escalable.
- Interactúa con la base de datos de la aplicación heredada con el fin de devolver el listado de clientes del sistema legado.

Ambos microservicios forman una primera parte de estudio y desarrollo en la cual se ha podido comprobar y evaluar la relativa facilidad en que se convierten pequeñas funcionalidades en sistemas plenamente escalables.

5. Resultados y evaluación

5.1. Resultados

El presente proyecto ha tenido por finalidad la comprensión de las tecnologías que entran en juego para un proceso de modernización de un producto heredado. Para este caso de uso ha consistido, en primer punto, en cómo aplicar una modernización de caja negra y, en segundo punto, cómo y de qué manera se deben orquestar las distintas tecnologías que se han empleado para la resolución del proyecto.

Después de testear Spring Security, este sistema ha proporcionado:

1. Añadir y configurar la protección a las aplicaciones de una forma casi inmediata.
2. Delegar la identificación de usuarios a sistemas de autenticación y autorización, como pueden ser Keycloak, Cas Server, LDAP, etc. Esto se consigue utilizando elementos proporcionados en forma de librerías y starters, donde, si no se realizan cambios importantes, definiendo pequeños ficheros de configuración la integración es inmediata.

Después de testear Spring Cloud Config Server, este sistema ha proporcionado:

1. Agiliza que las aplicaciones pasen a ser replicadas. Con pocos parámetros de entorno, los componentes que constituyen el ecosistema FuxPin S.L son autoconfigurados. Esto es una gran ventaja, pues con un fichero contenedor de la aplicación, el software puede ser ejecutado en cualquier entorno de hardware y en el momento de su arranque, según las propiedades de dicho entorno, recuperará de las propiedades customizadas. Por tanto, ésta es una opción ideal para aquellas soluciones donde se busca un alto rendimiento y una gran disponibilidad de las aplicaciones. También es importante subrayar que, con este software, se proporciona al sistema legado de una tecnología que le permite adaptarse al entorno Cloud.
2. Las aplicaciones pueden recuperar en “caliente” su propia configuración, sin necesidad de subir una nueva versión de la aplicación. Solo con un rearranque de la aplicación o bien a través del actuator de Spring Boot. Esto sí es realmente importante, pues con subir una nueva versión del fichero de configuración de la aplicación al Github, éste pasa a ser directamente recuperado por la aplicación, inicializándose con los nuevos parámetros.

3. Permite segregar ficheros según entornos. Es decir, el fichero de desarrollo, preproducción, staging y producción pueden estar separados en distintos directorios y disponer de ficheros comunes para aquellas propiedades que sean las mismas para los diferentes entornos.
4. Un pequeño punto en contra de este sistema es que, al tener el sistema de ficheros externalizados de las mismas aplicaciones, se pierde el autocompletar que ofrecen los IDE de los ficheros con extensión yml o properties, con lo cual, se deben conocer de antemano las propiedades de cada starter de Spring Boot.

Después de utilizar el Eureka Netflix Server, este sistema ha proporcionado:

1. Agilidad y sorprendente facilidad con la cual se pueden añadir y quitar microservicios. El sistema aporta sencillez tanto para registrar los microservicios como para aquellas aplicaciones que son clientes de estos.
2. Después de realizar replicaciones de un mismo microservicio, el sistema de balanceo de microservicios es gestionado por el mismo servidor, con lo cual, se evita tener que definir un sistema de Load Balancer con su completa gestión de nodos. Solo con arrancar una nueva instancia de un microservicio existente, éste se registra y ya puede ser utilizado por las aplicaciones clientes, encargándose Eureka del balanceo de las peticiones.
3. El sistema no requiere implementar código entre un microservicio y sus respectivos clientes para establecer la comunicación entre ambos elementos. Herramientas como el Spring Cloud Netflix Feign Client se encargan de todo este trabajo. Con definir una interfaz comuna entre el productor – consumidor, el sistema está listo que los clientes puedan utilizar los métodos de dicha interfaz. Toda la mensajería es transparente a los clientes y el mismo sistema se encarga de implementar las mensajerías REST con las cuales se realiza toda la transferencia de datos.

Si bien las soluciones Cloud Config y Eureka Server llegan configuradas de serie para aportar escalabilidad al software, el punto más remarcable de ambas soluciones es la extraordinaria sencillez con la que se pueden desarrollar aplicaciones y que éstas acaben funcionando a corto plazo de tiempo, adaptadas a un mundo tecnológico en el cual cada vez se exige más la programación en la nube.

5.2. Evaluación

Para evaluar el presente proyecto se ha pedido a dos usuarios expertos que realicen valoraciones sobre la solución implementada del proyecto. Con anterioridad, se les ha realizado una breve explicación de la tecnología utilizada para después presentar la aplicación final en funcionamiento. También se les ha proporcionado el enlace al Github donde se encuentran todos los componentes implementados para el desempeño del proyecto.

La evaluación ha consistido en responder un test con 9 preguntas. Éstas se debían valorar con una puntuación de 0 a 9, siendo 0 el valor la puntuación más baja y 9 la más alta. Las personas a las que se les ha pedido la valoración han sido:

Usuaria Marina Batet – jefa de sección de Acceso e Interoperabilidad de la Diputación de Tarragona.

Usuario Ferran Rodríguez – jefe de proyecto de Datos y Análisis de la Diputación de Tarragona.

Los resultados obtenidos se describen en la siguiente tabla:

Tabla 14. Evaluación del proyecto realizada por usuarios expertos

Pregunta	***Valoración Marina***	***Valoración Ferran***
Sobre el sistema de microservicios. ¿El proceso de replicación de un mismo microservicio es una tarea ágil y que se aplica de una forma inmediata dentro de la misma máquina o en una nueva imagen virtual de la máquina?	8	8
¿Con la solución que aporta Eureka Netflix Server se mejora el balanceo de nodos de un sistema de microservicios?	8	8
¿El proceso por el cual una aplicación cliente se integra con Eureka Netflix Server para poder recuperar los datos de un microservicio, es un sistema transparente?	9	9

¿El nuevo sistema ha mejorado significativamente la disponibilidad con respecto al sistema legado?	7	7
Sobre el proceso de configurar las propiedades de las aplicaciones remotamente a partir de un servidor Spring Cloud Config Server. ¿Este sistema favorece los procesos de integración, entrega y despliegue continuo de las aplicaciones?	7	9
¿El código fuente presenta comprensión de lectura y distingue perfectamente los diferentes componentes que conforman un proyecto?	8	8
¿El código fuente está perfectamente comentado y disponen todos los proyectos de un Markdown en el cual se definen los componentes principales que conforman el software?	8	8
¿Keycloak ofrece un sistema intuitivo, completo y que satisface todas las necesidades que se requieren para el acceso a la nueva aplicación de ventas?	8	7
Sobre la parte tecnológica escogida para la resolución del proyecto. ¿Crees que puede ser útil ampliar conocimientos de alguna tecnología aplicada para la resolución del proyecto y que ésta es candidata para su uso en proyectos personales o corporativos?	8	9

Con estas encuestas se ha obtenido una valoración objetiva sobre la resolución del proyecto y permite visualizar una opinión externa de las herramientas utilizadas.

6. Conclusiones

En el mundo tecnológico es muy común encontrar piezas dentro de las empresas que acaben siendo aplicaciones heredadas. Este proyecto se ha realizado con el fin de conocer de los mecanismos y de las herramientas que proceden en el momento de afrontar un proyecto de actualización de estos sistemas.

Hay muchas alternativas que se pueden aplicar con el fin de corregir estas situaciones que, cada vez más, se van acomplejando dentro de una organización. Estudiar en qué situación se encuentra una aplicación heredada y cuáles son los mejores mecanismos para abordar una posible renovación de su tecnología no es una tarea fácil. Su complejidad no reside en aplicar una u otra solución tecnológica, sino en trazar correctamente un camino a seguir, con unas decisiones tecnológicas que realmente se optimicen con los recursos que se aporten. Es en este momento cuando entran en juego las diferentes técnicas de modernización explicadas en el proyecto.

Es muy común, dentro de las organizaciones, que aparezca cierta incertidumbre sobre cómo afrontar estas situaciones, pues en la naturaleza del software, éste debería estar vivo y en continua evolución y, excepcionalmente, acabar siendo heredado. Sin embargo, es una situación que se repite muy a menudo en las organizaciones. Con el tiempo, esta tecnología va quedando más obsoleta y, una respuesta cada vez más tardía incrementa la magnitud del cambio y de los recursos dedicados para este desempeño, derivando en una total inanición del producto. Con el fin de evitar que estas actualizaciones sean realmente dramáticas, aparecen las distintas técnicas de modernización. Éstas dan respaldo a les diferentes respuestas que se puedan dar, limitadas en gran parte por los recursos que una organización pueda o quiera aportar.

No todas las aplicaciones pueden seguir la misma metodología de modernización. Un sistema que es totalmente opaco o en el cual no se tenga ninguna interfaz funcional bien localizada no se le podrá aplicar un wrapping funcional. La única opción será una modernización de caja blanca, en la cual, se deberá conocer al completo el funcionamiento del sistema legado o bien partir de un sistema en el cual se desconoce toda la lógica de negocio. En cambio, si se pueden de localizar las interfaces funcionales de componente, éste se podrá actualizar con una modernización de caja negra y con ello, se podrá ignorar cómo realmente está implementado internamente. También se puede dar el caso que sea totalmente factible aplicar una modernización de caja negra pero la organización decide aplicar una modernización de caja blanca con el fin de reemplazar al completo el sistema legado. Con se ha dicho anteriormente, esto depende y se responde con la cantidad de recursos que se dediquen a estos proyectos.

También cabe subrayar que una aplicación heredada no aparece casualmente en las organizaciones. Éstas son causa de un mal mantenimiento y de no saber afrontar con celeridad una situación que cada vez se vuelve más insostenible dentro de las organizaciones.

7. Trabajos futuros

El trabajo del final de grado ha consistido en aplicar una capa envolvente a aplicación heredada, con el fin de extraer cualquier operación sobre el sistema legado y así poder ofertar su funcionalidad a través de una nueva interfaz de microservicios. Cualquier operación pasará a delegarse a esta capa de wrapping.

Para posteriores evoluciones del sistema se proponen ciertas acciones que pretenden afianzar el producto a un mundo tecnológico donde cada vez se requiere más disponibilidad, así como también la reducción de los costes de mantenimiento.

Se proponen las mejoras:

1 – Aplicar Dockers

Aplicar una solución de Dockers permitiría al sistema que la gestión del entorno de ejecución pase a un mundo virtual, en el cual, una vez realizada una imagen, ésta sería replicada dentro del sistema contenedor tantas veces como sea necesaria la escalabilidad horizontal. En este sistema deben acabar derivando todos los componentes implementados en el proyecto, así como todas las futuras aplicaciones que se vayan creando.

2 – Adoptar Jenkins

Dentro del movimiento DevOps priman los acrónimos CD y CI. Los desarrolladores no solo desempeñan un papel de generadores de código, sino que pasan a tener un peso muy importante en todas las fases y acciones de integración y entrega del producto. Se realiza una reconversión de este rol para que empiece a asumir tareas de operaciones, orientándolo también a un perfil de sistemas. Jenkins es una herramienta que permitirá responder a esta filosofía, en la cual se permite definir los procesos para conseguir una integración y una entrega continua del producto, a la vez que se dispone de utilizar una gran variedad de herramientas gracias a su sistema de plugins.

3 – Aplicar una solución Datadog / Elastic Search – Kibana para el monitoreo de aplicaciones cloud.

Monitorizar los logs con el fin de explotar la información y extraer métricas que permitan entender el comportamiento de los usuarios, así como detectar posibles errores que no se reporten. Permite el seguimiento y evaluación de incidencias.

4 – Añadir un completo set de pruebas de integración, unitarias y funcionales. Añadir también pruebas de Selenium.

5 – Aplicar elementos analizadores de código como pueden ser SonarCloud Code Analyzer y SonarQube. Con estos sistemas de validación del código se pretende dar calidad al código y que éste gane en comprensión.

REFERENCIAS BIBLIOGRÁFICAS

Alex B., T. L. (s.f.). *Spring Security Reference*. Obtenido de Spring Security Reference: https://docs.spring.io/spring-security/site/docs/current/reference/html5/

altexsoft. (2017). *Legacy System Modernization: How to Transform the Enterprise for Digital Future*. Obtenido de https://www.altexsoft.com/media/2017/01/Legacy-Software-Modernization.pdf

Antonov, A. (2018). *Spring Boot 2.0 Cookbook: Configure, test, extend, deploy, and monitor your Spring boot application both outside and inside the cloud.* Birmingham: Packt.

Bashair A., S. K. (2017). *Systematic Review of Legacy System Migration*. Obtenido de Systematic Review of Legacy System Migration: https://ieeexplore.ieee.org/document/8253057

Bean, J. (2010). *SOA and Web Services Interface Design: Principles, Techniques, and Standards.* Burlington, USA: Elsevier Inc.

Bennett, K. R. (05 de 2000). *Software Maintenance and Evolution: a Roadmap*. Obtenido de Software Maintenance and Evolution: a Roadmap: https://www.researchgate.net/publication/221555773_Software_Maintenance_and_Evolution_a_Roadmap

Bisbal J., L. D. (1999). *Legacy Information Systems: Issues and Directions*. Obtenido de https://www.semanticscholar.org/paper/Legacy-Information-Systems%3A-Issues-and-Directions-Bisbal-Lawless/446b85482609887bbb7e28d4ecfb0294914dbc15

Brisbin, J. G. (2013). *Spring Data.* O'Reilly.

Canfora, G. F. (10 de 2006). *Migrating interactive legacy systems to Web services*. Obtenido de Migrating interactive legacy systems to Web services: https://ieeexplore.ieee.org/document/1602355

Castro, P. I. (2017). *Serverless Programming (Function as a Service).* Obtenido de Serverless Programming (Function as a Service): https://www.researchgate.net/publication/318477545_Serverless_Programming_Function_as_a_Service

Comella-Dorda, S. W. (2000). *A Survey of Legacy System Modernization Approaches*. Obtenido de A Survey of Legacy System Modernization Approaches: https://www.researchgate.net/publication/235126722_A_Survey_of_Legacy_System_Modernization_Approaches

Dragoni, N. G.-L. (6 de 2016). *Microservices: yesterday, today, and tomorrow*. Obtenido de Microservices: yesterday, today, and tomorrow: https://www.researchgate.net/publication/305881421_Microservices_yesterday_today_and_tomorrow

Echeverria, R. R. (07 de 2015). *Legacy Web Application Modernization by Generating a REST Service Layer*. Obtenido de Legacy Web Application Modernization by Generating a REST Service Layer: https://ieeexplore.ieee.org/document/7273801

Fielding, R. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Obtenido de http://roy.gbiv.com/pubs/dissertation/top.htm

Fowler, M. (18 de 03 de 2010). *Richardson Maturity Model*. Obtenido de https://martinfowler.com/articles/richardsonMaturityModel.html

Fromn, K. (15 de 10 de 2012). *Why The Future Of Software And Apps Is Serverless*. Obtenido de Why The Future Of Software And Apps Is Serverless: https://readwrite.com/2012/10/15/why-the-future-of-software-and-apps-is-serverless/

Gierke O., D. T. (14 de 05 de 2021). *Spring Data JPA - Reference Documentation*. Obtenido de https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#reference

Harris, T. (18 de 6 de 2020). *DreamFactorty*. Obtenido de https://blog.dreamfactory.com/legacy-system-migration-strategies/

Herrera, V. (2015). *DESARROLLO DE UN PLAN DE GESTIÓN DE MANTENIMIENTO DE SOFTWARE PARA EL DEPARTAMENTO DE SISTEMAS DE LA UNIVERSIDAD POLITÉCNICA SALESIANA BASADO EN LA NORMA ISO/IEC 14764:2006.* Obtenido de DESARROLLO DE UN PLAN DE GESTIÓN DE MANTENIMIENTO DE SOFTWARE PARA EL DEPARTAMENTO DE SISTEMAS DE LA UNIVERSIDAD POLITÉCNICA SALESIANA BASADO EN LA NORMA ISO/IEC 14764:2006: https://dspace.ups.edu.ec/bitstream/123456789/8936/1/UPS-CT005189.pdf

ISO. (09 de 2006). ISO/IEC 14764:2006 - Software Engineering — Software Life Cycle Processes — Maintenance. Obtenido de iso: https://www.iso.org/standard/39064.html

J., T. (2018). *Spring Security 5 for Reactive Applications.* Birmingham: Packt.

K., B. (2010). *Microservices Architecture.* Obtenido de Microservices Architecture: https://badia-kharroubi.gitbooks.io/microservices-architecture/content/patterns/configuration-patterns/externalized-configuration-store-pattern.html

Kangasaho, M. (2016). *Legacy application modernization with REST wrapping.* Obtenido de https://ethesis.helsinki.fi/repository/bitstream/handle/10138.1/5745/thesis_mikko_kangasaho.pdf?sequence=1

Knoche, H. H. (2018). *Using Microservices for Legacy Software Modernization*. Obtenido de https://www.researchgate.net/publication/324961770_Using_Microservices_for_Legacy_Software_Modernization

Kunjumohamed, S. S. (2016). *Spring MVC: Designing Real-World Web Applications.* Birmingham: Packt.

Larsson, M. (2019). *Hands-On Microservices with Spring Boot and Spring Cloud.* Birmingham: packt.

Lehman, M. (1979). Laws of software evolution revisited. Obtenido de https://www.sciencedirect.com/science/article/abs/pii/0164121279900220?via%3Dihub

Long, J. B. (2017). *Cloud Native Java.* O'Reilly.

Malinova, A. (2010). *Approaches and techniques for legacy software modernization*. Obtenido de https://www.researchgate.net/publication/267181092_Approaches_and_techniques_for_legacy_software_modernization

Migration vs Modernisation, which is the right strategy? (22 de octubre de 2020). Obtenido de Migration vs Modernisation, which is the right strategy?: https://www.uktech.news/migration-vs-modernisation-which-is-the-right-strategy

O'Grady, G. (2020). *GuideSmiths.* Obtenido de GuideSmiths: https://www.guidesmiths.com/blog/difficult-issues-considering-a-legacy-application

Pérez-Castillo, R. G.-R. (28 de 03 de 2012). *Software modernization by recovering Web services from legacy databases*. Obtenido de https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.1554

Raman. R. CSP, D. L. (2018). *Building RESTful Web Services with Spring 5.* Birmingham: Packt Publishing.

Rodríguez, A. M. (2021). *Gestión de la evolución del software. El eterno problema de los legacy systems.* Obtenido de https://www.researchgate.net/publication/267376263_Gestion_de_la_evolucion_del_software_El_eterno_problema_de_los_legacy_systems

S., N. (0 de 09 de 2015). *Application modernization:Approaches, problems and evaluation*. Obtenido de http://www8.cs.umu.se/education/examina/Rapporter/SimonNilsson.pdf

Salvatierra G., M. C. (3 de 2013). *Legacy System Migration Approaches*. Obtenido de Legacy System Migration Approaches: https://www.researchgate.net/publication/260325013_Legacy_System_Migration_Approaches

Sánchez O., B. F. (09 de 2013). *Una valoración de Modernización de Software Dirigida por Modelos*. Obtenido de Una valoración de Modernización de Software Dirigida por Modelos: https://www.researchgate.net/publication/270275091_Una_valoracion_de_Modernizacion_de_Software_Dirigida_por_Modelos

Sánchez, Ó. B. (2013). *Una valoración de Modernización de Software Dirigida por Modelos*. Obtenido de Una valoración de Modernización de Software Dirigida por Modelos: https://www.researchgate.net/publication/270275091_Una_valoracion_de_Modernizacion_de_Software_Dirigida_por_Modelos

Shinde, K. (Sin Fecha). *Modernization The New Mantra For Legacy Migration*. Obtenido de https://www.bitwiseglobal.com/blogs/modernization-the-new-mantra-for-legacy-migration/

Siva Prasad Reddy, K. (2017). *Beginning Spring Boot 2 - Applicactions and Microservices with the Spring Framework.* Hyderabad, India: Apress.

Spring Cloud Config. (s.f.). Obtenido de Spring Cloud Config: https://spring.io/projects/spring-cloud-config

Spring Cloud. (s.f.). *Spring Cloud Config*. Obtenido de Spring Cloud Config: https://docs.spring.io/spring-cloud-config/docs/current/reference/html/#_environment_repository

Sutherland, J. (2010). *Jeff Sutherland's Scrum Handbook.* Obtenido de Jeff Sutherland's Scrum Handbook: https://www.researchgate.net/publication/301685699_Jeff_Sutherland%27s_Scrum_Handbook

Thorgersen, S. I. (2021). *Keycloak - Identity and Access Management for Modern Applications.* Packt.

Tuusjärvi, K. (2021). *Modernization of legacy systems and migrations to microservice architecture*. Obtenido de Modernization of legacy systems and migrations to microservice architecture: http://urn.fi/URN:NBN:fi-fe202103016182

Varanasi, B. B. (2015). *Introducing Gradle.* Apress.

Glosario

2FA. Autenticación de doble factor. Es una autenticación de multifactor que no permite el acceso a las aplicaciones solo con las credenciales de los usuarios, sino que le añade una capa adicional con una información que solo el usuario conoce.

Branch. Es una rama de Github. Por defecto este repositorio crea la rama de main.

CAS Server. Central Authentication Service. Es un protocolo que implementa un único acceso de sesión para la web, conocido también como Single Sign On. Es un servidor de autenticación de open source de Apereo Foundation.

Endpoint. Un endpoint es la url donde responde y se está ejecutando un servicio web.

Escalabilidad horizontal. Consiste en tener varias instancias del software ejecutandose para aportar la funcionalidad completa del sistema. En este tipo de modelo, esta red de servidores se llama Cluster. Con este tipo de escalabilidad se consigue potenciar el rendimiento del sistema.

Load Balancer. Un sistema Load Balancer se encarga de enrutar el tráfico entrante entre varios destinatarios. Este sistema ejerce también un control de estados preguntando a las diferentes máquinas destinatarias sobre su estado con el fin de dirigir el tráfico a determinados nodos o máquinas que se encuentran activas.

MarkDown. Markdown es un lenguaje de marcado ligero que facilita la escritura de HTML, siendo muy utilizado en la documentación de proyectos.

Middleware. Capa intermedia o middleware se sitúa en el medio de la comunicación e intercambio de datos de los componentes de un sistema distribuido.

NoSQL. Not Only SQL. Este tipo de base de datos difiere del modelo relacional. Los datos almacenados no requieren de estructuras fijas ni de relaciones entre sus entidades.

Oauth 2.0. Este protocolo que se utiliza para delegar la autorización a diferentes proveedores externos, como puede ser Facebook, Github, Google, etc.

OpenID Connect. OpenID Connect (OIDC) es un sistema de identificación digital descentralizado en el cual el usuario se indentifica en una URL y sus credenciales son verificadas y validadas el servidor que implementa este protocolo, para así obtener información básica del perfil del usuario. Este estándar se desarrolla a partir del protocolo Oauth 2.0.

Pull. Esta acción de GitHub consiste en recuperar del repositorio remoto los ficheros para guardarlos dentro de un directorio local de la misma máquina.

Realm. Un realm es un conjunto de usuarios, los roles que los representan y las aplicaciones a las cuales tienen acceso. Solo aquellos usuarios que pertenezcan a un realm tendrán acceso Single-Sign On a las aplicaciones definidas dentro del realm.

REST Service Layer. Rest Service Layer es un modelo concreto de Arquitectura Service Layer, donde el canal de comunicación entre las diferentes entidades se realiza mediante Rest.

ROI. Return on Investement es una métrica que indica el beneficio que se obtiene a partir de las inversiones realizadas.

Service Discovery. Service Discovery es la detección automática de servicios que se encuentran dentro de una red distribuida.

Single-Sign On. Inicio de sesión único. Con este sistema se consigue que con una sola identificación, el usuario tenga acceso a varias aplicaciones.

Single-Sign Out. Es un proceso de autenticación en el cual se habilita que que un usuario pueda realizar un logout de un conjunto de aplicaciones de forma simultánea.

Staging. Dentro de los entornos de desarrollo, el entorno de staging es identico al entorno de producción. Su finalidad es que las aplicaciones se puedan ser probadas y testeadas en un entorno software y hardware que no debe diferir del entorno de explotación.

Starter. Spring Boot Starter. Se encarga de importar librerías, añadir ficheros de la aplicación, ficheros de configuración e incluso pequeñas implementaciones de código Java a los proyectos Spring Boot.

Wrapping. En programación en termino wrapper se utiliza para encapsular acciones o funcionalidades que forman parte de los componentes software.