

UNIVERSIDAD DE ALCALÁ



Escuela Politécnica Superior

**MÁSTER UNIVERSITARIO EN INGENIERÍA DEL
SOFTWARE PARA LA WEB**

Trabajo Fin de Máster

**ESTUDIO DEL FRAMEWORK SPRING, SPRING
BOOT Y MICROSERVICIOS**

Santiago Ramírez Pérez

2020

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

MÁSTER UNIVERSITARIO EN

INGENIERÍA DEL SOFTWARE PARA LA WEB

Trabajo Fin de Máster

“ESTUDIO DEL FRAMEWORK SPRING, SPRING BOOT Y
MICROSERVICIOS”

Autor: Santiago Ramírez Pérez

Director: Salvador Otón

Tribunal:

Presidente:

Vocal 1º:

Vocal 2º:

Calificación:

Fecha: de de

ÍNDICE RESUMIDO

RESUMEN	1
ABSTRACT.....	3
INTRODUCCIÓN.....	5
OBJETIVOS DEL PROYECTO.....	9
ESTADO DEL ARTE.....	13
SPRING BOOT	39
MICROSERVICIOS.....	121
DESARROLLO DEL PROTOTIPO	135
CONCLUSIÓN.....	173
PRESUPUESTO.....	177
BIBLIOGRAFÍA.....	181

ÍNDICE DETALLADO

RESUMEN	1
ABSTRACT	3
INTRODUCCIÓN	5
OBJETIVOS DEL PROYECTO	9
ESTADO DEL ARTE	13
1.1. DESARROLLO WEB	15
1.1.1. <i>Aplicación Web</i>	15
1.1.2. <i>World Wide Web – los Inicios de la web</i>	16
1.1.2.1. Web 1.0	17
1.1.2.2. Web 2.0	18
1.1.2.3. Web 3.0	19
1.2. JAKARTA EE	22
1.3. PROGRAMACIÓN ORIENTADA A ASPECTO	24
1.4. ARQUITECTURA MONOLÍTICA	27
1.5. ARQUITECTURA ORIENTADA A SERVICIOS SOA	29
1.6. SERVICIOS REST	31
1.7. METODOLOGÍA AGILE EN EL DESARROLLO DE SOFTWARE	33
1.8. SPRING FRAMEWORK	35
1.8.1. <i>Proyectos que componen Spring</i>	36
SPRING BOOT	39
2.1. SPRING BOOT STARTERS	44
2.2. SPRING BOOT AUTOCONFIGURATOR	46
2.3. SPRING BOOT CLI	48
2.4. SPRING BOOT ACTUATOR	49
2.5. HERRAMIENTAS PARA LA COMPILACIÓN DEL PROYECTO: MAVEN O GRADLE	50
2.5.1. <i>Apache Maven</i>	50
2.5.2. <i>Gradle</i>	52
2.6. SPRING INITIALIZR	55
2.7. ESTRUCTURA DE UN PROYECTO SPRING BOOT	56
2.8. ORIENTACIÓN A ASPECTO CON SPRING	57
2.9. INVERSIÓN DE CONTROL E INYECCIÓN DE DEPENDENCIA	60
2.9.1. <i>Inversión de control</i>	60
2.9.2. <i>Inyección de dependencia</i>	61
2.9.2.1. Tipos de inyecciones	62
2.10. BEANS EN SPRING BOOT	66
2.10.1. <i>Ciclo de vida de un Bean</i>	66
2.11. USANDO COMMANDLINERUNNER Y APPLICATIONRUNNER CON SPRING BOOT	68
2.12. ANOTACIONES DE ESTEREOTIPOS EN SPRING	70
2.13. PERSISTENCIA DE DATOS CON SPRING BOOT	71

2.13.1.	<i>Spring Data JPA</i>	74
2.13.2.	<i>Patrones de diseños utilizados</i>	77
2.13.2.1.	Patrón Repositorio.....	77
2.13.2.2.	Patrón DAO	78
2.13.3.	<i>Implementando Spring Data JPA</i>	79
2.13.3.1.	Utilizando la base de datos H2.....	79
2.13.3.2.	Utilizando la base de datos MySQL	83
2.13.4.	<i>Auditoria a las entidades con JPA</i>	86
2.14.	APLICACIONES WEB CON SPRING BOOT	90
2.14.1.	<i>Arquitectura MVC en Spring</i>	90
2.14.1.1.	Spring Boot MVC Auto-Configuration	92
2.14.1.2.	Spring Boot Web Starter.....	92
2.14.1.3.	Añadiendo los controladores.....	99
2.14.2.	<i>Servicios REST con Spring Boot</i>	102
2.15.	SPRING DATA REST	105
2.16.	ASEGURANDO LA APLICACIÓN CON SPRING SECURITY	107
2.16.1.	<i>Open Web Application Security Project (OWASP)</i>	109
2.17.	PRINCIPIOS Y PATRONES DE DISEÑO UTILIZADOS POR SPRING	115
2.17.1.	<i>Principios SOLID</i>	115
2.17.1.1.	Principio de responsabilidad única (Single Responsibility Principle).....	115
2.17.1.2.	Open-Close Principle (Principio de Abierto-Cerrado).....	116
2.17.1.3.	Liskov Substitution Principle (Principio de Sustitución de Liskov)	117
2.17.1.4.	Interface segregation (Segregación de Interfaces).....	119
2.17.1.5.	Dependency Inversion (Inversión de dependencia)	120

MICROSERVICIOS..... 121

3.1.	COMPUTACIÓN EN LA NUBE.....	124
3.2.	MICROSERVICIOS	127
3.3.	SPRING CLOUD.....	129
3.3.1.	<i>Spring Cloud Netflix Eureka</i>	130
3.3.2.	<i>Spring Cloud Config</i>	131
3.3.3.	<i>Netflix Zuul</i>	132

DESARROLLO DEL PROTOTIPO 135

4.1.	PROTOTIPO A DESARROLLAR.....	138
4.2.	HISTORIAS DE USUARIO.....	140
4.3.	REQUERIMIENTOS NO FUNCIONALES.....	150
4.4.	DISEÑO TECNOLÓGICO DEL PROTOTIPO	151
4.4.1.	<i>Front-End</i>	151
4.4.2.	<i>Servidores de configuración</i>	152
4.4.3.	<i>Microservicios</i>	152
4.4.3.1.	Autenticación	153
4.4.3.2.	Gestión de eventos.....	153
4.4.3.3.	Facturación.....	156
4.4.3.4.	Reportes	157
4.4.3.5.	Notificaciones.....	158
4.4.4.	<i>Otros componentes usados para el desarrollo de la aplicación</i>	158
4.4.5.	<i>Diseño de la interfaz</i>	160
4.4.6.	<i>Diagramas de Base de Datos</i>	166
4.4.6.1.	Autenticación	167
4.4.6.2.	Gestión de eventos.....	168

4.4.6.3.	Facturación.....	170
4.4.6.4.	Reportes	171
4.4.6.5.	Notificaciones.....	172
CONCLUSIÓN.....		173
PRESUPUESTO.....		177
BIBLIOGRAFÍA.....		181

ÍNDICE DE ILUSTRACIONES

ILUSTRACIÓN 1-MODELO CLIENTE-SERVIDOR.....	16
ILUSTRACIÓN 2-TIOBE INDEX	22
ILUSTRACIÓN 3-ESQUEMA ORIENTACIÓN A ASPECTO SPRING	24
ILUSTRACIÓN 4-ARQUITECTURA MONOLÍTICA	27
ILUSTRACIÓN 5-PROYECTO QUE COMPONEN SPRING	36
ILUSTRACIÓN 6-GOOGLE TRENDS DE LOS FRAMEWORK MÁS POPULARES DE JAVA	41
ILUSTRACIÓN 7-COMPARACIÓN ENTRE MAVEN Y GRADLE.....	54
ILUSTRACIÓN 8-SPRING INITIALIZR.....	55
ILUSTRACIÓN 9-ESTRUCTURA PROYECTO SPRING.....	56
ILUSTRACIÓN 10-CICLO DE VIDA DE UN BEAN.....	66
ILUSTRACIÓN 11-RANKING BASES DE DATOS – FUENTE: HTTPS://DB-ENGINES.COM/EN/RANKING_TREND	72
ILUSTRACIÓN 12-ORM EN JAVA	76
ILUSTRACIÓN 13-FORMULARIO LOGIN BASE DE DATOS H2.....	82
ILUSTRACIÓN 14-ESTRUCTURA TABLA BASE DE DATOS H2.....	83
ILUSTRACIÓN 15-ESTRUCTURA BASE DE DATOS.....	84
ILUSTRACIÓN 16-TABLA USUARIO	85
ILUSTRACIÓN 17-RESULTADO SENTENCIA DE CONSULTA A TABLA USUARIO	86
ILUSTRACIÓN 18- RESULTADO DE LA CONSULTA REALIZADA A TABLA USUARIO	89
ILUSTRACIÓN 19-ARQUITECTURA MODELO VISTA CONTROLADOR	90
ILUSTRACIÓN 20-PATRÓN FRONT CONTROLLER EN SPRING MVC	91
ILUSTRACIÓN 21-COMPARACIÓN TECNOLOGÍAS DE LAS VISTAS EN JAVA	96
ILUSTRACIÓN 22-EJEMPLO DE SPRING DATA REST	106
ILUSTRACIÓN 23-VENTANA LOGIN POR DEFECTO SPRING SECURITY	108
ILUSTRACIÓN 24-CONTRASEÑA GENERADA AL EJECUTAR LA APLICACIÓN	109
ILUSTRACIÓN 25-CATEGORIA DE INFRAESTRUCTURA DE LA NUBE	125
ILUSTRACIÓN 26 - INFRAESTRUCTURA TECNOLÓGICA.....	151
ILUSTRACIÓN 27 - VENTANA DE INICIO DE SESIÓN	160
ILUSTRACIÓN 28 - MENÚ ADMINISTRACIÓN	161
ILUSTRACIÓN 29 - DASHBOARD	162
ILUSTRACIÓN 30 - VENTANA LISTADO DE EVENTOS	162
ILUSTRACIÓN 31 - VENTANA LISTADO DE LOCALES	163
ILUSTRACIÓN 32 - VENTANA LISTADO DE ARTÍCULOS	163
ILUSTRACIÓN 33 - VENTANA PRINCIPAL ROL USUARIO	164
ILUSTRACIÓN 34 - VENTANA DETALLE EVENTO	165
ILUSTRACIÓN 35 - VENTANA DETALLE ARTÍCULO PROMOCIONAL	166
ILUSTRACIÓN 36 - DIAGRAMA ENTIDAD-RELACIÓN AUTENTICACIÓN	167
ILUSTRACIÓN 37 - DIAGRAMA ENTIDAD-RELACIÓN EVENTOS	169
ILUSTRACIÓN 38 - DIAGRAMA ENTIDAD-RELACIÓN FACTURACIÓN	170
ILUSTRACIÓN 39 - DIAGRAMA ENTIDAD-RELACIÓN REPORTES.....	171
ILUSTRACIÓN 40 - DIAGRAMA ENTIDAD-RELACIÓN NOTIFICACIÓN	172

RESUMEN

Las tecnologías web avanzan de una manera exorbitante en la actualidad, donde los frameworks juegan un papel fundamental en ese desarrollo gracias al conjunto de librerías, buenas prácticas y herramientas que facilitan la creación de sitios web a nivel empresarial y entre estas se encuentra SpringBoot como una extensión de Spring Framework, el cual se ha convertido desde su creación en uno de los mejores framework para el desarrollo de aplicaciones web profesionales en el lenguaje de programación Java.

Uno de los aspectos más relevante de SpringBoot es la facilidad que ofrece en la creación de microservicios, los cuales son utilizados por grandes empresas como Netflix, Amazon, Google y otras.

En el presente trabajo de fin de master se plantea la investigación, análisis y conocimiento de Spring Boot como Framework de desarrollo de aplicaciones web empresariales, mostrando cómo influye en el mundo de las tecnologías web, sus aspectos más relevantes, sus ventajas, debilidades y características.

Con el objetivo de demostrar las bondades que ofrece Spring Boot, se ha desarrollado una aplicación de gestión de eventos, donde se aplica un modelo de arquitectura basada en microservicios.

Palabras clave:

Aplicación Web, Framework, Spring Boot, Spring Framework, Microservicios.

ABSTRACT

Currently web technologies are moving fast, where frameworks plays a fundamental role thanks to the development of libraries, best practice and tools that facilitate the website creation on an enterprise level as in the case of SpringBoot, which has become one of the most important framework for web development in the Java programming language.

One of the most relevant aspects of Spring Boot is how easy it is to create microservices which are used by big enterprises such as Netflix, Amazon, Google and others.

This Final Master Project looks forward to insight, analysis and knowledge of Spring Boot as an enterprise web development framework, showing how it influences the world of web technologies, their most relevant aspects, their advantages, weaknesses and characteristics.

With the objective of demonstrating the advantages of Spring Boot, an event management application has been developed, where a microservice architecture model is being applied.

Keywords:

Web Application, Framework, Spring Boot, Spring Framework, Microservices.

INTRODUCCIÓN



El desarrollo web ha cambiado mucho desde sus inicios hasta el día de hoy, pasando de desarrollar un simple “Hola Mundo” en un lenguaje de hipertexto como HTML hasta la creación de plataformas encargadas de crear y desplegar sistemas web con un solo clic. Esto se debe a los grandes avances que se han tenido el Internet y la creación de un conjunto de tecnologías web que han facilitado este desarrollo.

Dentro de ese conjunto de tecnologías existen los Framework web, los cuales son fundamentales para el desarrollo de aplicaciones empresariales seguras y escalables en la actualidad. Estos intervienen en los diferentes ciclos de un sistema y proveen a los equipos de desarrollo de las herramientas necesarias para la construcción de aplicaciones que satisfaga las necesidades establecidas por el negocio.

En el presente trabajo se aborda el estudio de Spring Boot y Spring Framework como tecnologías web para el desarrollo de aplicaciones empresariales, mostrando en forma de análisis sus beneficios, sus características y componentes que lo han convertido en el Framework más utilizado en el lenguaje de programación Java.

También se realizará un análisis sobre los distintos modelos de arquitectura utilizados para el desarrollo de aplicaciones web, que van desde una arquitectura monolítica donde los componentes se encuentran acoplados en un único bloque de manera eficiente pero a la vez difícil de actualizar y mantener; hasta llegar a una arquitectura basada en microservicios, que plantea la separación de los componentes en pequeños servicios web independientes y capaces de funcionar por sí mismo.

Por último se plantea la realización de un prototipo utilizando como base de desarrollo el Framework Spring Boot y sus componentes con la finalidad de aplicar los conocimientos obtenidos sobre el desarrollo de aplicaciones web a lo largo del máster y mostrar las bondades del Spring.

OBJETIVOS DEL PROYECTO



Los objetivos de este trabajo son:

1. Realizar un análisis y evaluación de Spring Boot como Framework de desarrollo de aplicaciones web.
2. Aplicar los conocimientos adquiridos en las diferentes asignaturas a lo largo del máster.
3. Desarrollar una aplicación web de gestión de eventos en forma de prototipo utilizando las herramientas, módulos y librerías que provee Spring Boot e implementando un modelo de arquitectura basada en microservicios.

ESTADO DEL ARTE



1.1. Desarrollo web.

En la actualidad el desarrollo web es definido como cualquier proceso realizado para crear, modificar y desplegar un sitio web o aplicación que se ejecute a través de internet o en una red privada.

Para hablar del desarrollo web es fundamental definir que es una aplicación web y sus componentes, al igual que mencionar los sucesos importantes que sirvieron como base para poder crear sistemas web con el Framework Spring y SpringBoot.

1.1.1. Aplicación Web

Una aplicación web es un programa informático donde se procesan funciones específicas solicitadas por usuarios o sistema a través de un navegador web u otra herramienta. Estas son capaces de ejecutar las tareas necesarias para dar repuestas al usuario por el mismo medio que fue solicitada.

En la actualidad existen muchos tipos de aplicaciones web que van desde simples sistemas utilizados para convertir de euros a dólares hasta plataformas más complejas como un sistema de gestión hospitales o una red social. Lo cierto es que para utilizar estas aplicaciones es necesario realizar la petición a través de una red, sea esta una red interna y restringida o una pública como lo es internet.

La mayoría de las aplicaciones web creadas requieren de un servidor web donde son alojadas y ejecutadas; este se encarga de administrar las peticiones realizadas por los distintos clientes, procesar las tareas asociadas a dicha petición, en muchos casos realizar el almacenamiento de los datos y retornar una respuesta. Hoy en día este esquema es lo que llamamos cliente-servidor que conlleva una petición-respuesta como se muestra en la siguiente ilustración.

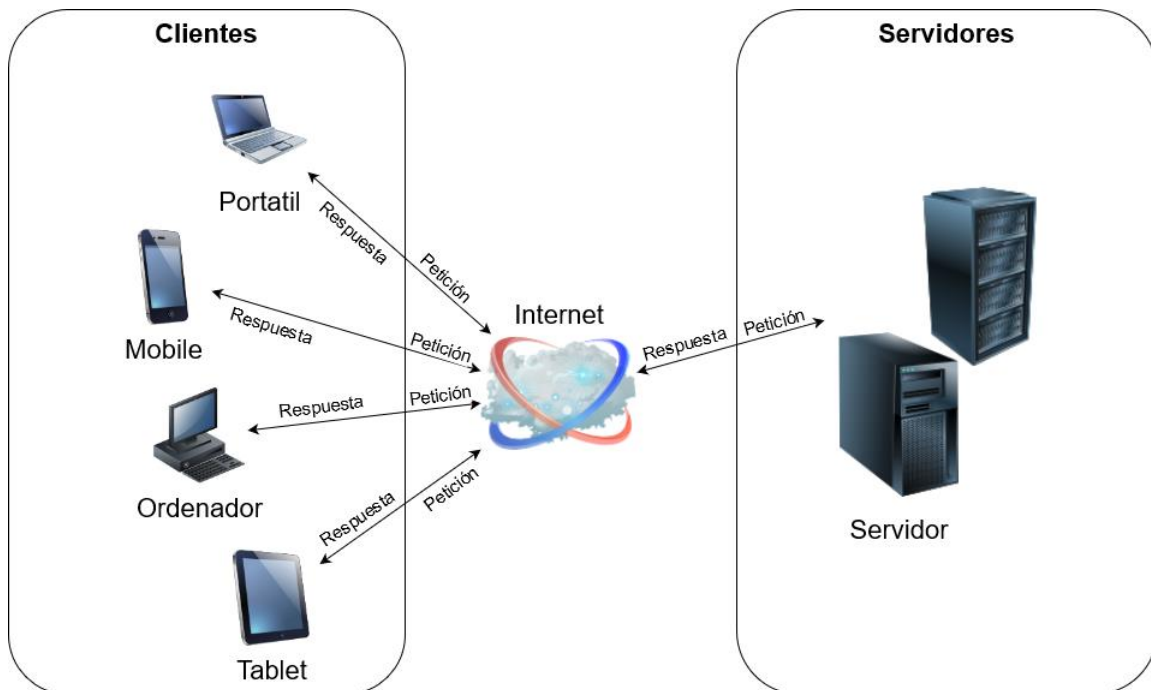


Ilustración 1-Modelo Cliente-Servidor

1.1.2. World Wide Web – los Inicios de la web

La World Wide Web es un sistema interconectado de páginas web públicas accesibles a través de Internet, esta es comúnmente llamada WWW, W3, o la Web. (Mozilla, 2020).

Luego de que los científicos inventarán las primeras computadoras y la manera de establecer una comunicación entre ellas a través de la red Internet, se empezaron a producir nuevos avances que han hecho posible que en la actualidad existan un sin número de aplicaciones web por todo el mundo capaces de solucionar todo tipo de problemas y satisfacer las necesidades de muchos negocios y clientes.

El primer intento para la creación de la web fue realizado en 1989 con la propuesta planteada por Tim Berners-Lee, quien trabaja como ingeniero de software en la organización europea para la investigación nuclear, mejor como CERN, por sus siglas en inglés. Según lo comentado por Berners-Lee en la entrevista publicada en World Wide Web Consortium (W3C), afirma que:

“En aquellos días, le parecía frustrante que, había información diferente en diferentes computadoras, pero tenía que iniciar sesión en diferentes computadoras para obtenerla. Además, a veces tenía que aprender un programa diferente en cada computadora. Entonces descubrir cómo funcionaban las cosas era realmente difícil. A menudo era más fácil ir y preguntarle a la gente cuándo estaban tomando café.” (www.W3C.org)



Con esa necesidad elaboró la propuesta que consistía en unificar el internet con la tecnología de hipertexto para compartir información, la cual luego de varias revisiones y con ayuda de su compañero Robert Cailliau, fue aceptada por CERN a finales de 1990 con el nombre de “WWW” y para principios de 1991 se lanzó el navegador web llamado “WorldWideWeb” junto con la descripción del lenguaje HTML y poco más tarde el primer servidor Web llamado httpd que permitía operar por encima de la red de Internet.

Dicho sistema de hipertexto se compuso de los siguientes elementos (Mozilla, 2020):

- HTML: Lenguaje de marcado de hipertexto. El formato de publicación para la Web, incluida la capacidad de formatear documentos y vincular a otros documentos y recursos.
- URI: Identificador uniforme de recursos. Un tipo de "dirección" que es única para cada recurso en la Web.
- HTTP: Protocolo de transferencia de hipertexto. Permite la recuperación de recursos vinculados de toda la Web.

Luego de inventar la Web, Berners-Lee en conjunto con un grupo de compañeros fundó el Consorcio de la World Wide Web (W3C World Wide Web Consortium) en 1994. Este consorcio está compuesto por los principales grupos de interés de la Web con el objetivo de estandarizar y establecer las directrices para garantizar que la Web funcione para todos y que evolucione de manera responsable.

Para finales de la década de los 90 ya existían miles de sitios web por todo el mundo, demostrando el gran impacto y acogida de esta tecnología.

1.1.2.1. Web 1.0

La Web 1.0 es el término utilizado para denominar la primera implementación de la WWW y los sitios web creados desde 1989 hasta mediados de la década del 2000.

Estos sitios web se caracterizaban por ser estáticos y de solo lectura, en donde toda la información era centralizada y la función de los usuarios se limitaba a visualizar el contenido sin poder realizar interacción o intercambio de datos con la web.

Entre las ventajas que ofrecía la Web 1.0 para los interesados están:

- Permitía tener una presencia en internet.



- Utilizaban un lenguaje de marcado de hipertexto sencillo.
- Compartir información para todo el mundo dentro de la red de internet.

Es bueno mencionar que, una vez colocado el contenido en los sitios, realizar la actualización de dicha información resultaba una tarea muy compleja, por lo que muchos sitios web permanecían desactualizados. Otra de las limitaciones que aparecieron durante la Web 1.0 era la poca seguridad de los sitios y la pobre arquitectura.

Durante el periodo que comprende la Web 1.0 se crearon la mayoría de los lenguajes de programación y herramientas que utilizamos en la actualidad para facilitar el desarrollo de sitios web. Entre estos lenguajes y herramientas están:

- Java (1991-1996) y Java EE (2001)
- Javascript (1995)
- CSS (1996)
- PHP (1995)
- ActionScript (1997)
- Macromedia Flash (1996)
- Python (1991)
- Django (2005)
- PHP-Nuke (2000)

Todas estas herramientas condujeron a que la web se desarrollará de forma exponencial, pasando de tener pocos sitios web a una WWW que poseía millones.

1.1.2.2. Web 2.0

Con los avances realizados en los servidores y las herramientas de creación de sitios durante la Web 1.0, nace la Web 2.0 haciendo referencia a todos los sitios web capaces de establecer una interacción directa con los usuarios a través del intercambio de información de manera que pasa de ser un simple contenedor de contenido que solo era creado por los gestores del sitio a ser un espacio bidireccional donde quien accedía al sitio podía colocar contenidos que otros visualizaban.

El termino Web 2.0 se puede apreciar por primera vez en el artículo llamado *Fragmented Future* escrito por Darcy DiNucci y publicado en la revista *Print magazine* en abril de 1999, pero no es hasta el 2004 cuando Dale Dougherty, co-fundador de la editora O'Reilly lo popularizó.



La Web 2.0 es la revolución empresarial en la industria de la computación causada por el paso a Internet como plataforma, y un intento de comprender las reglas para el éxito en esa nueva plataforma. La principal de esas reglas es esta: cree aplicaciones que aprovechen los efectos de la red para mejorar a medida que más personas las usen. (Tim O'Reilly, 2006).

Durante esta etapa la Web creció de manera exponencial hasta el punto de que en 2006 existían más de ochenta millones de sitios web, de los cuales muchos de ellos eran interactivos, dinámicos y se podía compartir todo tipo de información con facilidad. Con el tiempo se le fue acuñado el término de Web social por la enorme cantidad de foros, blogs, wiki y redes sociales que fueron desarrolladas.

Más que un avance en la tecnología, la Web 2.0 se convirtió en un fenómeno social, en donde un gran número de personas que tenían acceso a internet pasaba la mayor parte del día en las distintas comunidades, foros y redes sociales, compartiendo información y estableciendo el futuro de las comunicaciones entre personas.

Durante esa etapa los lenguajes de programación y herramientas para desarrollo web incluían cada vez más funcionalidades para hacer de los sitios lugares interactivos y dinámicos, entre las plataformas web que marcaron historia en la Web 2.0 están:

- Facebook: Red social.
- YouTube: Plataforma web utilizada para compartir videos.
- Wikipedia: Enciclopedia libre como plataforma de artículos.
- LinkedIn: Comunidad social orientada a empresas.
- MSN Spaces: Plataforma para blogs que ofrecía Microsoft.
- Google Maps: Plataforma de Google para mostrar mapas de todo el planeta.
- Flickr: Plataforma para almacenar fotos y videos.

Sin embargo, es bueno mencionar que los lenguajes seguían siendo limitados y con APIs muy sencillas. Los más usados para el desarrollo web eran un HTML básico, PHP, ASP, los applets de Java, JavaScript con muchas limitaciones por los navegadores o Adobe Flash si se deseaba aportar contenido multimedia.

1.1.2.3. Web 3.0

La Web 3.0 nace entre finales de la década del 2000 y principios de la década pasada con el nombre de Web semántica porque supera las limitaciones existentes en la Web 2.0 sobre el acceso,



organización y gestión de los datos, haciéndola una Web más inteligente para dar respuesta a las inquietudes de los usuarios de una manera rápida y sencilla.

World Wide Web Consortium definió la web semántica como:

La Web Semántica es una Web extendida, dotada de mayor significado en la que cualquier usuario en Internet podrá encontrar respuestas a sus preguntas de forma más rápida y sencilla gracias a una información mejor definida. Al dotar a la Web de más significado y, por lo tanto, de más semántica, se pueden obtener soluciones a problemas habituales en la búsqueda de información gracias a la utilización de una infraestructura común, mediante la cual, es posible compartir, procesar y transferir información de forma sencilla. Esta Web extendida y basada en el significado, se apoya en lenguajes universales que resuelven los problemas ocasionados por una Web carente de semántica en la que, en ocasiones, el acceso a la información se convierte en una tarea difícil y frustrante. (W3C, 2009)

El objetivo principal de la Web Semántica es impulsar la evolución de la Web actual al permitir que los usuarios encuentren, compartan y combinen información más fácilmente. La Web Semántica, como se imaginó originalmente, es un sistema que permite a las máquinas "comprender" y responder a complejas solicitudes humanas basadas en su significado. Tal "comprensión" requiere que las fuentes de información relevantes estén estructuradas semánticamente.

Durante esta etapa se logró lo siguiente:

- Que las aplicaciones se interconectarán entre sí de manera sencilla.
- La creación de una arquitectura orientada a la web.
- Se incluyeron metadatos y una estructuración en capas para segmentar los contenidos.
- Gestión de grandes volúmenes de datos tanto en base de datos como en la nube.
- Ejecución y visualización desde cualquier dispositivo que tenga acceso a internet.
- Personalización y flexibilidad.
- Automatización, integración y reutilización de aplicaciones de maneras más eficientes.
- Implementación de un Modelo de negocio Software as a Service (SaaS) en muchas empresas.
- Creación masiva de plataforma de código abierto.

A nivel de desarrollo web surgieron muchos Frameworks de desarrollo y los existentes incluyeron muchas herramientas de gran utilidad para la creación de sitios web.

Existen varios artículos de diversos autores como el caso de Fernando Almeida y su publicación llamada "Concept and Dimensions of Web 4.0", que determinan que la Web 3.0 finaliza en el año



2019 y a partir de ese momento inicia la Web 4.0, definiéndola como una nueva generación de plataformas ligadas a la industria 4.0 que emplean nuevos modelos de comunicación entre máquinas, asistentes virtuales e interacción más compleja y personalizada con los usuarios.

Sin embargo, en la actualidad hay muy poca información oficial sobre las características de la Web 4.0, la dimensión tecnológica que abarca y el periodo de tiempo en que inicia.

Gracias a los avances en la Web, la tendencia actual del desarrollo web y los lenguajes de programación es la creación de plataformas flexibles, con excelente arquitecturas e infraestructuras y capaces de ofrecer una respuesta inmediata a las solicitudes que se realizan.



1.2. Jakarta EE

Jakarta EE es el nombre actual de lo que muchos todavía llaman Java Enterprise Edition (Java EE), la cual es una tecnología y herramienta que proporciona a los desarrolladores un conjunto de API que ayudan a acortar el tiempo de desarrollo, reducir la complejidad y mejorar el rendimiento de las aplicaciones que son creadas usando el lenguaje de programación Java para ambientes empresariales.

El lenguaje de programación Java es un lenguaje orientado a objetos de alto nivel que tiene una sintaxis y un estilo particular. Una plataforma Java es un entorno particular en el que se ejecutan aplicaciones de lenguaje de programación Java. (Oracle, 2010)

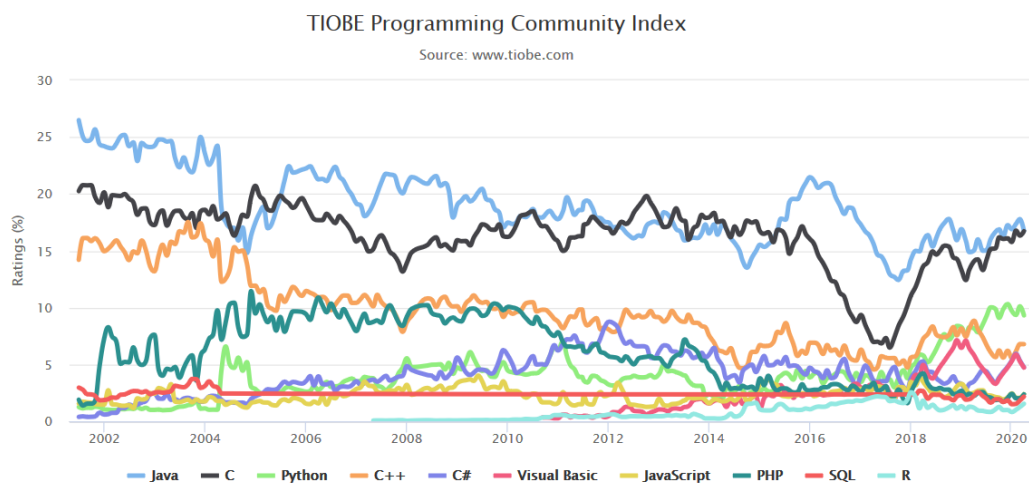


Ilustración 2-TIOBE Index

Java lleva muchos años siendo el lenguaje de programación más utilizado en todo el mundo gracias a las siguientes características:

- Independencia de plataforma: cualquier programa creado en Java podrá ser ejecutado desde sistemas operativos distinto gracias a la Máquina Virtual de Java.
- Simplicidad: Posee una cantidad enorme de funciones que permite que se pueda realizar cualquier tarea de una manera simple y enfocándose solo en las características necesarias, lo que hace que sea uno de los lenguajes fácil de aprender.
- Orientación a objetos: Fue diseñado para que podamos definir las estructuras de una manera parecida a lo que existe en el mundo real.
- Seguridad: Es un lenguaje que implementa capas de accesos a sus componentes y la memoria que lo hacen muy seguro y estable.
- Multihilos: Permite que varios procesos se ejecuten de manera simultánea.



Para finales del año 1999 se lanzó la primera versión de Java 2 Platform, Enterprise Edition (J2EE) como una “Colección de especificaciones y directivas de programación para facilitar el desarrollo de aplicaciones de servidor distribuidas multi-nivel, alineada fuertemente con Internet” (Martínez et al., 2009).

Esta versión contenía paquetes opcionales para mensajes, generación dinámica de páginas Web o programas de email en Java, que luego se convirtieron en un estándar representado por un conjunto de APIs y directivas, soportadas por un servidor de aplicación para desarrollar y ejecutar aplicaciones en red de gran escala, multicapa, escalables, fiables y seguras.

Versión	Fecha de lanzamiento
J2EE 1.2	Diciembre de 1999
J2EE 1.3	Septiembre del 2001
J2EE 1.4	Noviembre 2003
Java EE 5	Mayo 2006
Java EE 6	Diciembre 2009
Java EE 7	Abril 2013
Java EE 8	Agosto 2017
Jakarta EE	Febrero 2018

Tabla con las versiones lanzadas de Jakarta EE (fuente: Oracle)

Java EE recibe el nombre de Jakarta EE en el año 2018 gracias a que Oracle decide donar las especificaciones, desarrollo y gestión a la Eclipse Foundation, pero sin ceder el registro de la marca, por lo cual la fundación se vio obligada a cambiar el nombre y realiza una encuesta, en la que sale como ganador “Jakarta EE”. Java EE versión 8 tenía unos meses de haber sido lanzada cuando paso a llamarse Jakarta EE 8.

Jakarta EE es considerado como un estándar para la Web. Proporcionando técnicas para implementar muchos aspectos de una aplicación empresarial, como manejar solicitudes web, acceder a la base de datos, conectarse a otros sistemas empresariales e implementar servicios web.

1.3. Programación Orientada a Aspecto

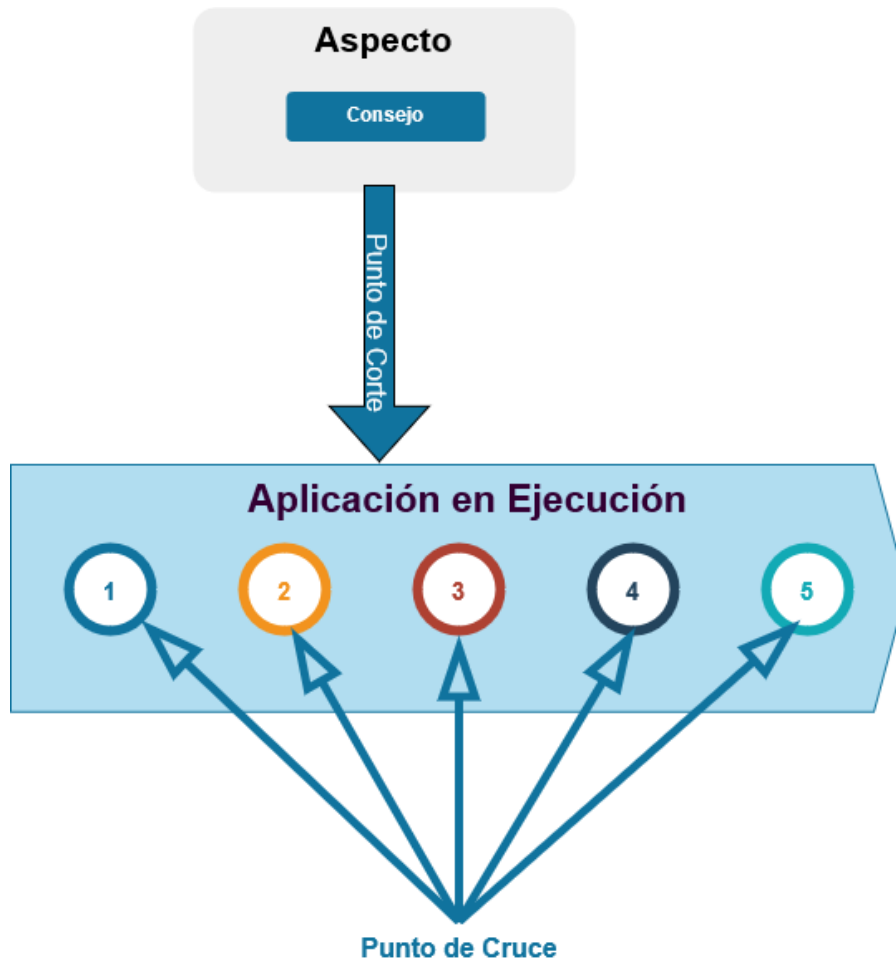


Ilustración 3-Esquema orientación a aspecto Spring

La programación orientada a aspecto nace como un paradigma utilizado por muchos lenguajes de programación que busca una modularización de las preocupaciones que componen un sistema a través de la separación de responsabilidades por medio de la gestión de las funcionalidades que son comunes entre los diferentes módulos de una aplicación.

Estas funcionalidades comunes que se repiten en muchos puntos de la aplicación reciben el nombre de funcionalidades transversales al sistema porque no forman parte de las características principales pero son necesarias para el funcionamiento del sistema.

Una preocupación es un requisito o consideración específica que debe abordarse para satisfacer el objetivo general del sistema. Un sistema de software es la realización de un conjunto de preocupaciones (Laddad, 2003).



Un aspecto es una unidad modular que se dispersa por la estructura de otras unidades funcionales. Los aspectos existen tanto en la etapa de diseño como en la de implementación. Un aspecto de diseño es una unidad modular del diseño que se entremezcla en la estructura de otras partes del diseño. Un aspecto de programa o de código es una unidad modular del programa que aparece en otras unidades modulares del programa (Kiczales G. et al., 1997).

Este paradigma se enfoca en modelar un comportamiento común, de manera que se pueda extraer esas funciones transversales y repetitivas para ejecutarla en aquellos puntos de la aplicación donde sea necesario sin estar mezclado con el código al que se aplica.

Uno de los puntos clave para el desarrollo de la programación orientada a aspecto ha sido la dificultad que presenta el paradigma de programación orientada a objeto para abordar las preocupaciones transversales en aplicaciones grandes o complejas, ya que mientras una aplicación va creciendo en funcionalidad, aparecen muchas clases que tiene la misma responsabilidad no esencial, provocando que parte de código o métodos se repitan en dichas clases.

El objetivo de ambos paradigma es modularizar de una manera eficiente el sistema y es importante destacar que la programación orientada a aspecto complementa la programación orientada a objetos, funcionando por encima de esta, en lugar de competir con ella. AOP viene a solucionar los problemas que presenta la POO cuando se trata de implementar una lógica transversal en gran escala.

La programación orientada a aspecto y la programación orientada a objetos se pueden usar juntas para escribir aplicaciones potentes, porque ambas proporcionan diferentes formas de estructurar el código. La programación orientada a objetos se centra en hacer de todo un objeto, mientras que la programación orientada a aspecto presenta el aspecto, que es un tipo especial de objeto que inyecta y envuelve su comportamiento para complementar el comportamiento de otros objetos. (Cosmina, 2020)

Este paradigma se compone por los siguientes conceptos:

- Aspectos (aspect): son elementos transversales o funcionalidades que se repiten durante todo el sistema y que deben ser implementadas de forma separada. En términos de desarrollo representa la sección de código que se separó del resto del programa.



- Punto de corte (pointcut): es donde se determina en que sección del programa se debe añadir el aspecto a través de distintas formas: Expresiones regulares, nombres o incluso dinámicamente en tiempo de ejecución según el valor de ciertos parámetros.
- Consejo (advice): es el código o algoritmo que se ejecutará dentro del aspecto de forma automática.

Entre sus ventajas tenemos:

- Evita la dispersión de código.
- Permite construir implementaciones más comprensibles, adaptables y reutilizables.
- Simplifica el código y mejora su mantenimiento.
- Reduce el acoplamiento entre funciones.
- Vuelve más limpio el código fuente.
- Se puede combinar con cualquier otro paradigma de programación, como lo es la programación orientada a objetos.

A pesar de todas las bondades que ofrece este paradigma posee una desventaja, sufre de un anti-patrón de diseño “acciones a distancia”, que provoca la interacción no prevista de componentes muy distantes de un sistema (Wikipedia, 2020). Esto hace difícil entender el funcionamiento de la aplicación a nivel de llamadas a métodos.

Los ejemplos más comunes donde se aplica este paradigma dentro de una aplicación empresarial son los módulos de: la seguridad, la producción de logs, el cache y las transacciones.

En la actualidad muchas de las aplicaciones empresariales están compuestas por una gran cantidad de módulos principales que contienen el desarrollo de las funcionalidades que caracterizan la aplicación, pero a la vez, poseen muchas funcionalidades que son comunes dentro de dichos módulos.

1.4. Arquitectura monolítica

Hasta hace unos años, casi todas las aplicaciones empresariales que se desarrollaban eran basadas en una arquitecta monolítica, donde los sistemas se estructuraban de una manera en que todos los aspectos funcionales estuvieran acoplados y sujetos en un mismo sistema. Esto desencadenaba que existan muchas capas entrelazadas dentro del sistema, lo cual resulta difícil a la hora de tratar de implementar múltiples tecnologías en una capa.

Una aplicación monolítica es una aplicación de software con un alto nivel de complejidad que ejecuta un grupo completo de tareas para implementar un caso de uso completo. (Jecan, 2017).

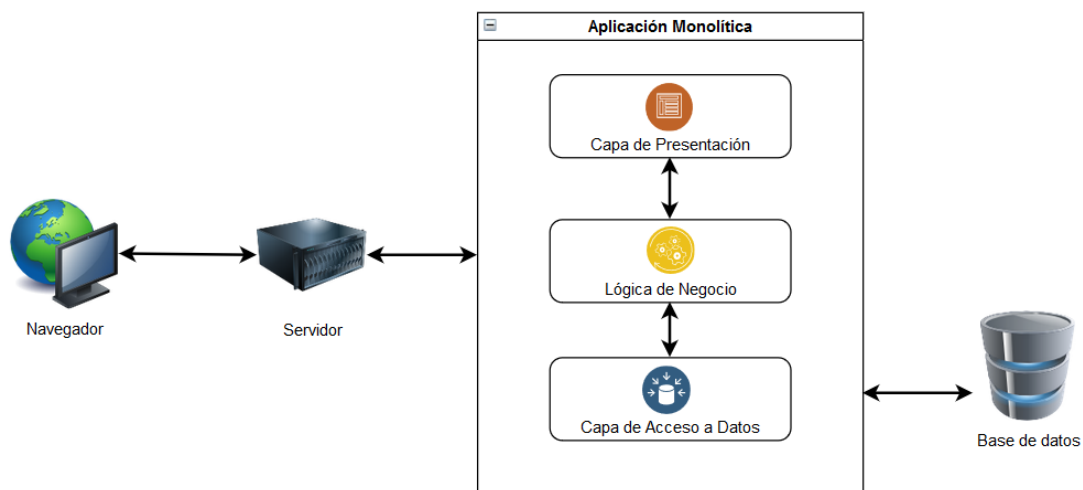


Ilustración 4-Arquitectura Monolítica

Como se puede apreciar en la ilustración anterior, esta arquitectura ofrece las siguientes ventajas:

- Facilidad en el desarrollo: Al desarrollarse como un todo y no se modulariza implica una complejidad menor con relación a otra arquitectura.
- Implementación sencilla: En la mayoría de los casos se puede lanzar todo el sistema con solo colocar el empaquetado en un servidor de aplicaciones.
- Escalabilidad simple: Con solo añadir más servidores de aplicaciones y de base de datos con un balanceador de carga a nivel de infraestructura se puede escalar todo el sistema.
- Eficiencia: El entorno en el que se construyen este tipo de soluciones está muy bien definido y ofrece poco margen a los fallos. (Viewnext, 2018).



Todas estas ventajas se aplican perfectamente para desarrollar aplicaciones sencillas, en donde el crecimiento con el paso del tiempo es reducido. Sin embargo, a medida que los sistemas crecen en cantidad de funcionalidades y desarrollos, esta arquitectura presenta una serie de inconvenientes:

- Entornos muy rígidos y difícilmente actualizables debido a la gran cantidad de código base y acoplamiento existente.
- En muchas ocasiones es difícil identificar y solucionar problemas específicos debido a que es necesario entender todo el bloque de código para realizar cualquier cambio, lo que conlleva a que un cambio sencillo requiera un gran esfuerzo.
- Por cada cambio es necesario desplegar toda la aplicación por completo provocando una interrupción de los servicios.
- Sobrecarga del contenedor de la aplicación dando como resultado un mayor tiempo de inicio para dicha aplicación.
- Al tratarse de un único ambiente, es imposible trabajar en varios ambientes al mismo tiempo.
- Alta curva de aprendizaje para el equipo que desarrolla debido a que es necesario conocer todos los módulos que componen el sistema al detalle para realizar un desarrollo eficiente.
- Dificultad para escalar la aplicación, ya que una arquitectura monolítica solo puede escalar en una dimensión, es decir, como un todo, sin posibilidad de escalar sus componentes de manera independiente de acuerdo con lo que necesite.
- Poca flexibilidad para implementar nuevas y diferentes tecnologías.

Estas y otras desventajas han provocado que en muchas ocasiones las aplicaciones monolíticas no se actualicen, dando lugar a que los desarrolladores se vean obligados a aplicar parches con mucha frecuencia para tratar de cumplir con las necesidades del negocio, convirtiendo los sistemas en inestables y carentes de una arquitectura bien definida.

Es bueno mencionar que en la actualidad pese a las dificultades que acarrea, la arquitectura monolítica sigue siendo la más empleada para construir los sistemas debido a la resistencia al cambio de muchos equipos de desarrollo. Sin embargo, cada día toma más auge la arquitectura de microservicios.



1.5. Arquitectura orientada a servicios SOA

La arquitectura orientada a servicios (SOA por sus siglas en inglés) es un enfoque de diseño en el que múltiples servicios colaboran para proporcionar un conjunto final de capacidades. Un servicio aquí generalmente significa un proceso de sistema operativo completamente separado (Newman, 2015).

Esta arquitectura nace como solución a los problemas que presentan la arquitectura monolítica, basando su enfoque en la reutilización del software a través de servicios bien definidos, en los cuales, varias aplicaciones puedan utilizar de forma transparente e independiente dichos servicios sin la necesidad de realizar un desarrollo dentro del sistema monolítico.

La Arquitectura Orientada a Servicios (SOA) surgió de una combinación de los conceptos de la programación orientada a objeto y la componentización de aplicaciones. En SOA, una aplicación se divide en varias partes conocidas como servicios. Un servicio proporciona una funcionalidad accesible para otros servicios a través de varios protocolos, como el Protocolo simple de acceso a objetos (SOAP). (Dragoni, et al., 2016)

La implementación de la arquitectura orientada a servicios para el desarrollo de sistemas durante la Web 2.0 dio lugar al modelo de negocio “Software como un Servicio” (SaaS en inglés), el cual es un modelo de software que se basa en el pago por uso, donde los clientes solo pagan por el uso del software alojado en un servidor remoto.

En este modelo el proveedor gestiona todo lo relacionado con la aplicación, incluida la seguridad, la disponibilidad, el almacenamiento y el rendimiento. Esto libera a su departamento de tecnología de la información (TI) de la gestión compleja de software y hardware, lo que les permite concentrarse en tareas más importantes. (Salesforce, 2015).

En la actualidad SaaS hace referencia a la computación en la nube. Aprovechando todas las herramientas que brinda la red para ofrecer las siguientes ventajas:

- Pago solo por lo que se usa.
- Alta disponibilidad.
- Escalabilidad.
- Tolerancia a fallos.
- Mayor seguridad.
- Actualización continua.



Sin embargo, a pesar de todas las bondades que ofrece este modelo de arquitectura, existían algunas limitaciones como son que los protocolos de comunicación existentes entre servicios diferían entre sí y a la vez no existía un esquema detallado sobre qué servicio dividir.

Esto inconvenientes dieron paso a un nuevo tipo de modelo arquitectónico, el modelo basado en microservicios.



1.6. Servicios REST

Transferencia de estado representacional, comúnmente conocido como REST por su acrónimo en inglés es un estilo arquitectónico de software para diseñar sistemas distribuidos que forma parte de la arquitectura cliente servidor, definiendo una manera de comunicación y el intercambio de datos entre componentes y sistemas web.

REST fue introducido y definido en 2000 por Roy Fielding en su disertación doctoral. REST es una alternativa ligera a mecanismos como RPC (Llamada a procedimiento remoto) y Servicios web (SOAP, WSDL, etc.). (Cosmina, 2020).

REST es un estándar para el intercambio de datos a través de la Web en aras de la interoperabilidad entre sistemas informáticos. Los servicios web que se ajustan al estilo arquitectónico REST se denominan servicios web RESTful o simplemente servicio REST, los cuales permiten a los sistemas solicitantes acceder y manipular los datos utilizando un conjunto uniforme y predefinido de operaciones (Relan, 2019).

Los servicios REST son actualmente la forma más utilizada para que las aplicaciones se comuniquen entre sí y dentro de un sistema, son definidos como recursos web que poseen un identificador (URI) para ser accedido. Esto es posible gracias a que estos servicios utilizan HTTP y HTTPS como protocolos de comunicación por defecto.

Al utilizar HTTP como protocolo de comunicación los servicios REST no poseen estado, lo que aumenta la escalabilidad ya que el servidor no tiene que mantener, actualizar o comunicar el estado de la sesión.

Entre el conjunto de operaciones que se pueden realizar en los servicios REST están:

Operación	Descripción
GET	Se utiliza para obtener los recursos solicitados.
POST	Se utiliza para crear recursos con los datos enviados en el cuerpo de la llamada.
PUT	Se utiliza para actualizar los datos.
DELETE	Se utiliza para eliminar un recurso.



Estas operaciones pertenecen a una serie predefinida de métodos de petición que posee HTTP, capaces de transmitir distintos tipos de datos, siendo el más utilizado el formato de datos JSON gracias a su sencilla estructura de llave y valor.

Existe un término asociado a los servicios REST llamado API REST, el cual es un componente que sirve para el intercambio y procesamiento de datos entre sistemas o módulos a través del protocolo de comunicación HTTP.



1.7. Metodología ágiles en el desarrollo de software

Las metodologías ágiles tal y como su nombre lo indica son métodos para manejar proyectos, actualmente se puede emplear lo ágil en casi todos los ámbitos de proyectos, pero la misma nació con los proyectos de desarrollo de software.

A diferencia de la metodología en cascada en la que todos los procesos se realizan de una manera secuencial, la metodología ágil divide grandes proyectos en partes pequeñas que sean manejables llamadas iteraciones (Sprint). Al final de cada iteración la cual se produce dentro de un intervalo de tiempo que generalmente son dos semanas en la que se realiza una entrega de valor, lo desarrollado debe ser capaz de ser funcional para recibir retroalimentación de los interesados o usuarios finales.

Con la implementación de estas metodologías, el negocio, es decir, los interesados están más involucrados en todo el proceso de levantamientos de requerimientos, pruebas y en el mismo desarrollo lo que produce unas entregas de mayor calidad y en un rango de tiempo mucho menor, reduciendo el impacto de costos y tiempo.

Estas metodologías han generado una buena aceptación en el ámbito del software lo que ha permitido que empresas robustas hayan dejado de lado las metodologías tradicionales para aplicar estos nuevos marcos de trabajo que les ha permitido inclusive, terminar proyectos que tenían estancados, el ahorro de costos, la constante motivación y satisfacción del cliente.

Estas metodologías se basan en 12 principios que prescriben prácticamente todas sus variantes. Son los siguientes:

- La satisfacción del cliente proviene de la entrega rápida de software en funcionamiento.
- El cambio es bienvenido independientemente de la etapa de desarrollo.
- Entrega de software funcional con frecuencia.
- Los responsables de negocio y los desarrolladores deben trabajar de la mano todos los días.
- Desarrollar los proyectos con equipos motivados, otorgando las herramientas necesarias y confiando en ellos para hacer el trabajo.
- El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
- El software funcional es la medida número uno de éxito.
- Esforzarse por el desarrollo sostenible. Todos los miembros del equipo deben poder mantener el ritmo de desarrollo constante e indefinidamente.
- La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.



- La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado y eliminando el trabajo innecesario, es esencial.
- Los equipos autoorganizados generan los mejores requisitos, arquitecturas y diseños.
- A intervalos regulares, haga que el equipo reflexione para determinar cómo mejorar.

Siguiendo estos principios, estas metodologías plantean el uso de historias de usuarios (User Stories) como técnica de creación o levantamiento de requerimientos del sistema y el conjunto de historias de usuarios forman gran parte de la documentación del proyecto.

Estas historias se plasman como una opinión del cliente sobre lo que debe hacer la aplicación, con el objetivo de comunicar cómo interactuará un usuario con el sistema y documentar los resultados de dicha interacción, es decir, se describe lo que el usuario será capaz de hacer cuando se complete la historia de usuario.

Existe otra técnica para la documentación de requisitos llamada casos de uso, los cuales pertenecen al conjunto de metodologías que componen proceso unificado racional, mayormente conocido como RUP por sus siglas en inglés y al igual que las historias de usuarios, esta técnica se centra en el actor.

Sin embargo, en muchas ocasiones, al crear los requisitos con los casos de uso se genera una excesiva documentación debido a su enfoque centrado a la documentación, que conforme avanza el proyecto y surgen nuevos cambios, dicha documentación no se actualiza y se vuelve obsoleta. Esto ha provocado que cada día esta técnica sea reemplazada por un enfoque ágil con las historias de usuarios.

Las historias de usuarios se componen de las siguientes partes:

- El título: este debe ser sencillo, conciso y abordar de manera general la acción que se desea realizar dentro de la aplicación.
- La narrativa: En esta se define y documenta brevemente la interacción que tendrá el usuario con la historia.

Criterios de aceptación: Estos son requisitos de aceptación con los que debe cumplir la historia una vez desarrollada la acción o funcionalidad.



1.8. Spring Framework

Spring es un Framework para el desarrollo de aplicaciones empresariales en Java que posee una gran cantidad de herramientas y utilidades para agilizar el proceso de desarrollo y despliegue de aplicaciones web. Este es de código abierto y posee una de las mejores comunidades existentes en ese ámbito, en donde los foros están siempre activos y el desarrollo de las nuevas funcionalidades es rápido y con una excelente calidad.

El principal atractivo que Spring siempre ha tenido es el uso de la inyección de dependencia y el contenedor de inversión de control; haciendo frente al conocido modelo EJB. La inyección de dependencia es un patrón de diseño orientado a objetos introducido al mercado como parte de la creación del Framework que plantea que los objetos de una clase sean suministrados a esta de forma automática y cuando sean necesarios, eliminando la dependencia en la creación e implementación de dichos objetos en las clases.

Sin dejar de lado la inyección de dependencia como su atractivo principal, existen otras características que hacen de Spring el Framework el más utilizado sobre el lenguaje Java desde hace unos años. Entre estas características están:

- Sencillo y excelente manejo de las transacciones de bases de datos.
- Integración con otros Frameworks Java, como es JPA, Hibernate ORM, Struts, JSF.
- Framework web MVC para crear aplicaciones web.
- Soporte para el almacenamiento de datos NoSQL, procesamiento en lote y big data.

Las aplicaciones desarrolladas que utilizan Spring siguen el principio de diseño inversión de control a través de un contenedor que mantiene un contexto de aplicación y crea e instancia los componentes solo cuando es necesario y los administra de forma adecuada. Estos componentes conviven de manera eficiente dentro del contexto para crear una aplicación estable.

Otra de las bondades que caracterizan a Spring es la excelente documentación y el conjunto de pruebas que acompañan a cada versión que sale en el mercado. Esto es posible gracias al uso de la metodología de desarrollo guiado por pruebas (TDD) que sigue el equipo de desarrollo, en donde realizan las pruebas de todos los componentes del Framework.

Spring se basa en la filosofía “convención sobre configuración”, reduciendo al mínimo el número de pasos que un desarrollador debe dar en la configuración inicial del proyecto antes de ponerse a trabajar en la parte compleja, centrando sus esfuerzos en lo importante. La idea es clara: no perder el tiempo y dinero en hacer las mismas cosas una y otra vez.

1.8.1. Proyectos que componen Spring

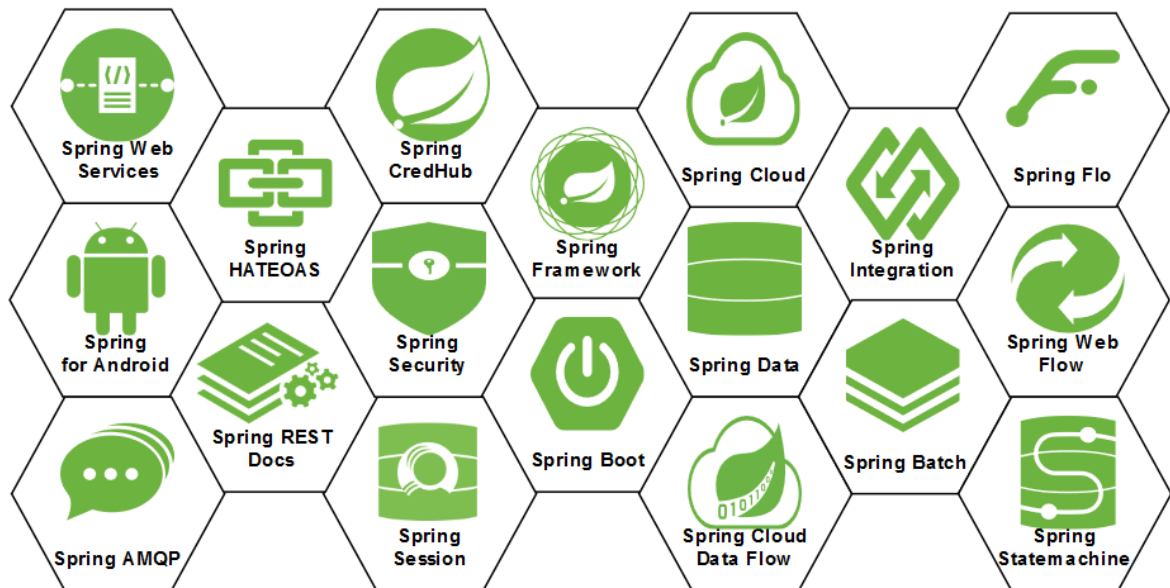


Ilustración 5-Proyecto que componen Spring

En la actualidad Spring está compuesto por un conjunto de proyectos de gran importancia para el desarrollo de aplicaciones, entre los que podemos destacar (Spring.io, 2020):

- Spring Framework: Proporciona un modelo integral de programación y configuración para aplicaciones empresariales modernas basadas en Java, en cualquier tipo de plataforma de implementación.
- Spring Boot: Facilita la creación de aplicaciones independientes basadas en aplicación de producción de Spring que se pueden "ejecutar".
- Spring Data: Su misión es proporcionar un modelo de programación familiar y consistente basado en Spring para el acceso a los datos, al mismo tiempo que conserva los rasgos especiales del almacén de datos subyacente.
- Spring Cloud: Proporciona herramientas para que los desarrolladores construyan rápidamente algunos de los patrones comunes en sistemas distribuidos, por ejemplo, gestión de configuración, descubrimiento de servicios, interruptores automáticos, enrutamiento inteligente, micro-proxy, bus de control, tokens únicos, bloqueos globales, elección de liderazgo, distribuido sesiones, estado del clúster.



- Spring Cloud Data Flow: Proporciona herramientas para crear topologías complejas para transmisión y procesamiento de datos por lotes. El procesamiento y transferencia de datos consisten en aplicaciones Spring Boot, creadas utilizando los marcos de microservicio Spring Cloud Stream o Spring Cloud Task.
- Spring Security: Es un Framework que se centra en proporcionar autenticación y autorización a las aplicaciones Java. Es el utilizado por defecto para asegurar aplicaciones basadas en Spring y al igual que todos los proyectos de Spring, el verdadero poder de Spring Security se encuentra en la facilidad con que se puede extender para cumplir con los requisitos personalizados.
- Spring Session: Proporciona una API e implementaciones para administrar la información de sesión de un usuario.
- Spring Integration: Permite la mensajería ligera dentro de las aplicaciones basadas en Spring y admite la integración con sistemas externos a través de adaptadores declarativos. Esos adaptadores proporcionan un mayor nivel de abstracción sobre el soporte de Spring para la comunicación remota, la mensajería y la programación. El objetivo principal de Spring Integration es proporcionar un modelo simple para construir soluciones de integración empresarial mientras se mantiene la separación de responsabilidades que es esencial para producir código testable y mantenible.
- Spring HATEOAS: Proporciona algunas API para facilitar la creación de servicios REST que siguen el principio HATEOAS cuando se trabaja con Spring y especialmente Spring MVC. HATEOAS es un acrónimo de Hypermedia As The Engine Of Application State (hipermedia como motor del estado de la aplicación), donde de acuerdo con un punto de acceso al API se puede determinar los recursos de acuerdo a la respuesta obtenida por el servidor.
- Spring REST Docs: Ayuda a documentar los servicios RESTful. Con este se produce documentación que sea precisa, concisa y bien estructurada.
- Spring Batch: Spring Batch proporciona funciones reutilizables que son esenciales en el procesamiento de grandes volúmenes de registros, incluidos el registro de logs, la gestión de transacciones, las estadísticas de procesamiento de tareas, el reinicio de tareas, la omisión y la gestión de recursos. También proporciona servicios y características técnicas más avanzadas



que permitirán trabajos por lotes de alto volumen y alto rendimiento a través de técnicas de optimización y partición.

- Spring AMQP: Aplica los conceptos básicos de Spring al desarrollo de soluciones de mensajería basadas en AMQP. Proporciona una "plantilla" como abstracción de alto nivel para enviar y recibir mensajes. Estas librerías facilitan la gestión de los recursos AMQP al tiempo que promueven el uso de inyección de dependencia y configuración declarativa.
- Spring for Android: Es un Framework diseñado para proporcionar componentes de Spring para su uso en aplicaciones de Android.
- Spring Statemachine: Herramienta para integrar los conceptos de máquinas de estado en las aplicaciones Spring.
- Spring CredHub: Proporciona una API para almacenar, generar, recuperar y eliminar credenciales de varios tipos de forma segura.
- Spring Flo: Es una biblioteca de JavaScript que ofrece un generador visual HTML5 embebido para automatización de procesos y gráficos simples. Esta biblioteca se usa como la base del generador de flujo en Spring Cloud Data Flow.
- Spring Web Flow: Se basa en Spring MVC y permite implementar los "flujos" de una aplicación web. Un flujo encapsula una secuencia de pasos que guían a un usuario a través de la ejecución de alguna tarea comercial. Abarca múltiples solicitudes HTTP, tiene estado, trata con datos transaccionales, es reutilizable y puede ser dinámico y de larga duración.
- Spring Web Services (Spring-WS): Es un producto de la comunidad de Spring enfocado en crear servicios web basados en documentos. Tiene como objetivo facilitar el desarrollo del servicio SOAP por contrato, lo que permite la creación de servicios web flexibles utilizando una de las muchas formas de manipular las cargas XML. El producto se basa en Spring, lo que significa que puede utilizar los conceptos de Spring, como la inyección de dependencia, como parte integral de su servicio web.

SPRING BOOT

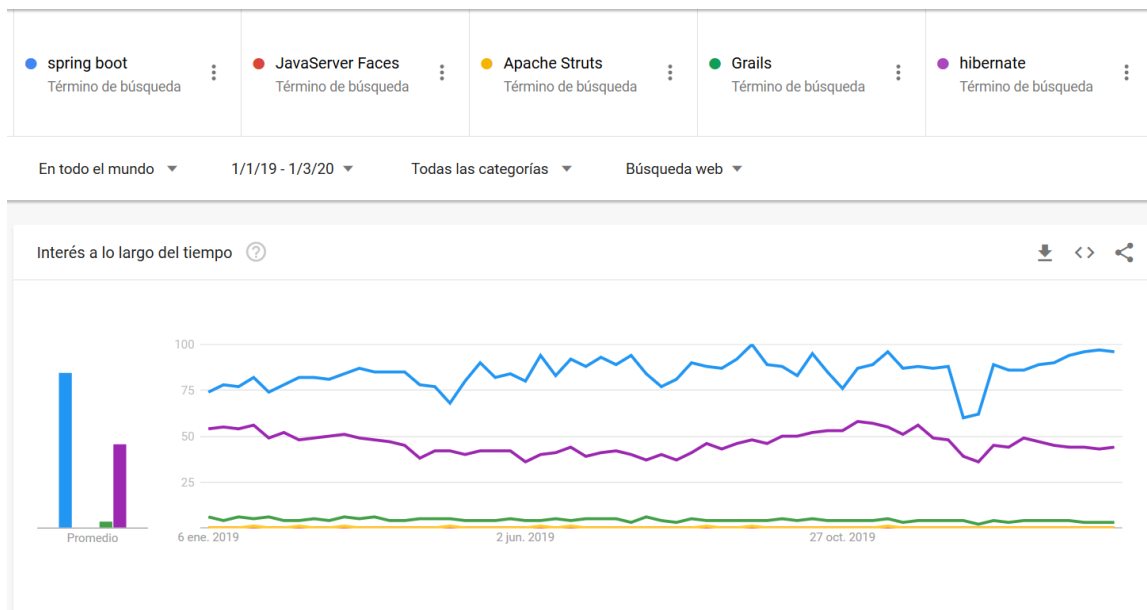


Ilustración 6-google trends de los framework más populares de Java

Spring Boot es una extensión de Spring Framework que sigue el enfoque de "Convención sobre configuración", que ayuda a construir aplicaciones basadas en Spring de manera rápida y fácil. El objetivo principal de Spring Boot es crear rápidamente aplicaciones basadas en Spring sin requerir que los desarrolladores escriban la misma configuración repetitiva una y otra vez (Prasad R., 2017).

Spring Boot está construido sobre Spring Framework y forma parte integral de este. Actualmente es el más utilizado dentro de conjunto de Framework, al igual que dentro del mundo Java.

El objetivo principal de Spring Boot es simplificar el desarrollo de aplicaciones por medio de la autogestión de un gran número de configuraciones, tareas y componentes que son necesarios para la ejecución de una aplicación. De esta manera, se logra que los desarrolladores se enfoquen en el desarrollo de la lógica de negocio del sistema.

Para cumplir con ese objetivo Spring Boot posee un conjunto de bibliotecas llamadas starters que contienen una colección de todas las dependencias relevantes preconfiguradas que se necesitan para iniciar una funcionalidad particular. Estas pueden ser usadas solo con incluirla como dependencia de un proyecto.

Spring Boot ofrece a los desarrolladores un conjunto de configuraciones automáticas por defecto que hacen que una aplicación web pueda desarrollarse y desplegarse de manera rápida, sencilla y a la vez segura, realizando una cantidad de pasos mínimas. Estas configuraciones pueden ser personalizadas de acuerdo con las necesidades requeridas por el sistema.



Adicional a las características antes mencionadas, Spring Boot también se destaca por lo siguiente:

- Posee todas las características de Spring Framework.
- Implementa el principio de diseño de Abierto-Cerrado, donde esta es cerrada a modificación, pero abierto a extensión cuando se desee tener un mayor control de sus componentes.
- Posee servidores de aplicaciones y contenedores de Servlet embebido.
- Eliminar la necesidad de configurar la aplicación por medio de código o XML. Para ello utiliza ficheros `.properties` o `.yml`.
- Es una gran opción para crear aplicaciones basadas en microservicios.
- Posee configuraciones automáticas para diferentes Frameworks como son Spring Security, Spring Batch y otros.
- Ofrece un amplio soporte a diferentes tecnologías como son bases de datos relaciones y no relacionales, almacenamiento en caché, mensajería, procesamiento por lotes y más.
- Ofrece métricas de la aplicación por medio de una librería llamada Actuator, utilizada para exponer información importante sobre la aplicación que se encuentra en ejecución a través del monitoreo.

Los componentes principales de Spring Boot son:

- Spring Boot Starters
- Spring Boot AutoConfigurator
- Spring Boot CLI
- Spring Boot Actuator





2.1. Spring Boot Starters

Los starters contienen muchas de las dependencias que se necesitan para poner en marcha un proyecto de una manera rápida y con un conjunto consistente y compatible de dependencias transitivas administradas. (Spring.io, 2020)

Los starters están compuestos por una agrupación de las dependencias relacionadas a las tecnologías y módulos más comunes que se usan en el día a día en el desarrollo de aplicaciones web con Spring Framework. Estos solucionan la tediosa tarea de tener que añadir cada una de las dependencias requeridas para implementar una tecnología dentro del proyecto y simplifican el fichero que contiene las configuraciones para la compilación de la aplicación.

Todos los starters inician con el prefijo de `spring-boot-starter` en el nombre y vienen con una configuración por defecto.

En la actualidad Spring Boot posee más de cincuenta starters divididos en tres grupos:

- **Spring Boot application starters:** Este grupo posee la mayoría de los starters que se pueden utilizar en las aplicaciones. Entre los más comunes están: `spring-boot-starter-web`, `spring-boot-starter-test`, `spring-boot-starter-security` y `spring-boot-starter-data-jpa`.
- **Spring Boot production starters:** Estos starters proveen un conjunto de dependencias relacionadas con la aplicación cuando se encuentra en un ambiente de producción. Actualmente solo existe `spring-boot-starter-actuator` que provee una librería de monitoreo y administración de aplicación.
- **Spring Boot technical starters:** Estos proveen un conjunto de dependencias a tecnologías alternativas a las utilizadas por defecto en Spring Boot.

Dentro de los starters a utilizar durante el presente trabajo se encuentra:

Starter	Descripción
<code>spring-boot-starter-web</code>	Starter que contiene las dependencias necesarias para crear aplicaciones web, aplicaciones RESTful y posee el contenedor de servlet Tomcat.



spring-boot-starter-thymeleaf	Starter utilizado para crear aplicaciones web con el motor de plantilla Thymeleaf para la capa de la vista.
spring-boot-starter-data-jpa	Suministra las dependencias necesarias para trabajar con Java Persistence API (JPA) e Hibernate.
spring-boot-starter-security	Provee la capa de seguridad a las aplicaciones creadas con Spring.
spring-boot-starter-test	Es incluido por defecto al crear los proyectos y posee todo el soporte a la creación de test con Spring.
spring-boot-starter-data-rest	Permite creación de recursos web basándose en los repositorios de datos que tiene la aplicación. Utiliza REST y Spring Data para realizar su función.

Tabla. Starters de Spring Boot



2.2. Spring Boot AutoConfigurator

La configuración automática que implementa Spring Boot es una de las características más importantes que posee el Framework porque es la encargada de proporcionar toda la configuración necesaria para que la aplicación se ejecute de manera adecuada, sin la necesidad de definir algún fichero XML o utilizar anotaciones especiales.

La clave importante para que Spring Boot funcione es la anotación `@EnableAutoConfiguration`, porque contiene la función de configuración automática, y aquí es donde todo comienza a suceder. Spring Boot utilizará la configuración automática basada en su classpath, sus anotaciones y su configuración para agregar la tecnología adecuada y crear una aplicación adecuada. (Gutiérrez, 2019).

Ésta configura los componentes basado en los siguientes criterios (Prasad Reddy, 2017):

- Busca y verifica la disponibilidad de una clase particular en el classpath.
- Verifica si existe o no un bean configurado para el componente de Spring.
- Verifica si existe propiedades relacionadas con el componente en el sistema.
- Verifica si existe un fichero de configuración.

Spring Boot implementa esta característica por medio de la anotación `@EnableAutoConfiguration`, la cual se utiliza al implementar la anotación `@SpringBootApplication` que habilita dos características necesarias adicionales a la autoconfiguración:

- `@ComponentScan`: define la búsqueda de los componentes de Spring en el paquete donde se encuentra la aplicación.
- `@Configuration`: Utilizada para definir que la clase puede registrar beans adicionales en el contexto o importar clases de configuración adicionales.

El siguiente es un ejemplo de uso de la anotación `@SpringBootApplication` utilizada para iniciar la aplicación a través de su método main:

`@SpringBootApplication`

```
public class SistemaGestionEventosTfmApplication {  
  
    public static void main(String[] args) {  
        SpringApplication.run(SistemaGestionEventosTfmApplication.class, args);  
    }  
}
```




Es importante mencionar que la configuración automática está diseñada para funcionar bien con "Starters", pero los dos conceptos no están directamente vinculados. Se pueden elegir dependencias fuera de los iniciadores. Spring Boot sigue haciendo todo lo posible para configurar automáticamente la aplicación. (Spring.io, 2020).



2.3. Spring Boot CLI

Spring Boot CLI es una herramienta que provee una interfaz de línea de comandos de Spring para ejecutar y probar la aplicación Spring Boot desde el símbolo del sistema. Ésta contiene los componentes esenciales para desarrollar una aplicación, como son los Starter y la autoconfiguración.

Spring Boot CLI permite ejecutar scripts Groovy, lo que significa que tiene una sintaxis familiar similar a Java sin tanto código repetitivo. También puede iniciar un nuevo proyecto o escribir su propio comando para él. (Spring.io, 2020)

Podemos ejecutar incluso aplicaciones web Spring con comandos simples de la CLI de Spring Boot.



2.4. Spring Boot Actuator

Es una librería que ofrece un conjunto de métricas e información importante sobre la aplicación que se encuentra ejecutándose en un ambiente de producción. Ésta permite administrar y monitorear partes específicas arrojando datos detallados sobre la configuración, auditoría y el estado del sistema.

Algunas de las características del actuador Spring son (Prasad Reddy, 2017):

- Se pueden visualizar los detalles de configuración del bean de aplicación.
- Muestra las asignaciones de URL de la aplicación, los detalles del entorno y los valores de los parámetros de configuración.
- Se visualizan las métricas de verificación de estado registradas.

Spring Boot incluye una serie de características adicionales para ayudarlo a monitorear y administrar su aplicación cuando la lleva a producción. Puede elegir administrar y monitorear su aplicación utilizando puntos finales HTTP o con JMX. La auditoría, el estado y la recopilación de métricas también se pueden aplicar automáticamente a su aplicación.



2.5. Herramientas para la compilación del proyecto: Maven o Gradle

En la actualidad el proceso de construcción, compilación y gestión del conjunto de dependencia que requiere una aplicación web son realizados por herramientas de gestión automatizadas minimizando el tiempo que invertimos como desarrolladores para mantener organizados y actualizados los proyectos.

Las herramientas más adecuadas para realizar esta tarea dentro del mundo java son Apache Maven y Gradle.

2.5.1. Apache Maven

Apache Maven es una herramienta de gestión y comprensión de proyectos de software. Basado en el concepto de un modelo de objeto de proyecto (POM), Maven puede administrar la construcción, los informes y la documentación de un proyecto a partir de una información central. (Maven, 2020).

Esta herramienta está presente durante todo el ciclo de vida del desarrollo, realizando tareas relacionada con la creación, búsqueda e integración de librerías, repositorios, compilación, empaquetado, documentación, ejecución de scripts, manejo de test y despliegue de aplicaciones en Java.

Maven se puede usar con multitud de añadidos aportados por la comunidad de desarrolladores, por ejemplo, hay plugins que se pueden usar para funciones como desplegar el proyecto en un servidor, pasar el proyecto a un analizador de calidad de código o generar la cobertura de test que tiene la aplicación (Autentia, 2018).

Un modelo de objeto de proyecto o POM es la unidad fundamental de trabajo en Maven. Es un archivo XML que contiene información sobre el proyecto y los detalles de configuración utilizados por Maven para construir el proyecto. Contiene valores predeterminados para la mayoría de los proyectos. (Maven, 2020).

Un ejemplo de los ficheros Pom.xml utilizado dentro de un proyecto java es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```



```
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.0.RELEASE</version>
  </parent>
  <groupId>com.ejemplo</groupId>
  <artifactId>tfm</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>tfm</name>
  <description>Pom ejemplo para TFM con Spring Boot</description>

  <properties>
    <java.version>1.8</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
      <exclusions>
        <exclusion>
          <groupId>org.junit.vintage</groupId>
          <artifactId>junit-vintage-engine</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
```



```
</plugin>  
</plugins>  
</build>
```

```
</project>
```

Dónde:

- Las primeras etiquetas definen el fichero xml y el conjunto de etiqueta utilizadas en maven.
- Las etiquetas “<groupId>” y “<artifactId>” son indicadores para definir el id del grupo del proyecto y el id del proyecto que se está creando. Junto se convierten en el identificador único del proyecto.
- En <version> se define cual es la versión actual del proyecto.
- <name> y <description> son campos orientativos para informar sobre el proyecto.
- La etiqueta “<properties>” sirve para definir propiedades a utilizar dentro del mismo fichero pom.xml o por otros componentes durante el proceso de construcción y compilación.
- La etiqueta “<dependencies>” posee una agrupación de dependencias y una dependencia se identifica por medio de los valores dentro de las etiquetas groupId, artifactId y scope.
- Dentro de <build> se encuentra los plugin y configuraciones utilizadas para la construcción y compilación del proyecto.

2.5.2. Gradle

Gradle es una herramienta de automatización de compilación de código abierto centrada en la flexibilidad y el rendimiento. Los scripts de compilación de Gradle se escriben usando un Domain Specific Language (DSL) Groovy o Kotlin (Gradle.org, 2020).

Esta herramienta es fácil de configurar y usar para cualquier tipo de aplicación web en diferentes lenguajes de programación, no tan solo para Java. Provee maneras potentes y flexibles para administrar las dependencias de un proyecto.

Entre las características que se pueden apreciar en su página oficial están:

- Altamente personalizable: Gradle está modelado de manera personalizable y extensible de las formas más fundamentales.
- Rápido: Gradle completa las tareas rápidamente reutilizando las salidas de ejecuciones anteriores, procesando solo las entradas que cambiaron y ejecutando tareas en paralelo.
- Potente: Gradle es la herramienta de compilación oficial para Android y viene con soporte para muchos idiomas y tecnologías populares.



Un fichero gradle posee la siguiente estructura:

```
plugins {
  id 'org.springframework.boot' version '2.3.0.RELEASE'
  id 'io.spring.dependency-management' version '1.0.9.RELEASE'
  id 'java'
}
group = 'com.ejemplo'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '1.8'
repositories {
  mavenCentral()
}
dependencies {
  implementation 'org.springframework.boot:spring-boot-starter'
  testImplementation('org.springframework.boot:spring-boot-starter-test') {
    exclude group: 'org.junit.vintage', module: 'junit-vintage-engine'
  }
}
test {
  useJUnitPlatform()
}
```

Donde al igual que con maven poseen plugins, group, version, propiedades y dependencias pero estructura de una manera más simple y sin utilizar el gran número de etiquetas que contiene el formato xml.

Ambas herramientas están soportadas por los entornos de desarrollos más utilizados para la creación de aplicaciones web y cumplen perfectamente la función de administrar las dependencias necesarias para el funcionamiento de los proyectos web, sin embargo, Maven posee una mayor integración con los IDE y en la actualidad es la herramienta de compilación más utilizada, como se puede apreciar en la siguiente figura:



Ilustración 7-Comparación entre Maven y Gradle



2.6. Spring Initializr

Spring Initializr es una herramienta proporcionada por Spring en su sitio web Spring.io para generar los proyectos de una manera rápida, como se muestra a continuación:

The screenshot displays the Spring Initializr configuration page. It is divided into several sections:

- Project:** Radio buttons for Maven Project and Gradle Project.
- Language:** Radio buttons for Java, Kotlin, and Groovy.
- Spring Boot:** Radio buttons for versions 2.3.1 (SNAPSHOT), 2.3.0, 2.2.8 (SNAPSHOT), 2.2.7, 2.1.15 (SNAPSHOT), and 2.1.14.
- Project Metadata:** Text input fields for Group (com.example), Artifact (demo), Name (demo), and Description (Demo project for Spring Boot). A text input field for Package name (com.example.demo).
- Packaging:** Radio buttons for Jar and War.
- Java:** Radio buttons for versions 14, 11, and 8.
- Dependencies:** A section with a button "ADD DEPENDENCIES... CTRL + B" and the text "No dependency selected".

At the bottom, there are three buttons: "GENERATE CTRL + G", "EXPLORE CTRL + SPACE", and "SHARE...".

Ilustración 8-Spring Initializr

Dentro de la herramienta se puede elegir entre las opciones que posee, desde el tipo de gestor de proyecto (maven o gradle), siguiendo por el lenguaje a utilizar, tipo de empaquetado, versión de Java hasta completar los datos informativos e identificativos del proyecto. También te permite añadir las dependencias a utilizar de un listado proporcionado por Spring.

Por último, cuando se haya colocado todo lo que se necesite para el proyecto, se presiona el botón Generate para obtener el proyecto con todos los ficheros, configuración y estructura base generada.

2.7. Estructura de un proyecto Spring Boot

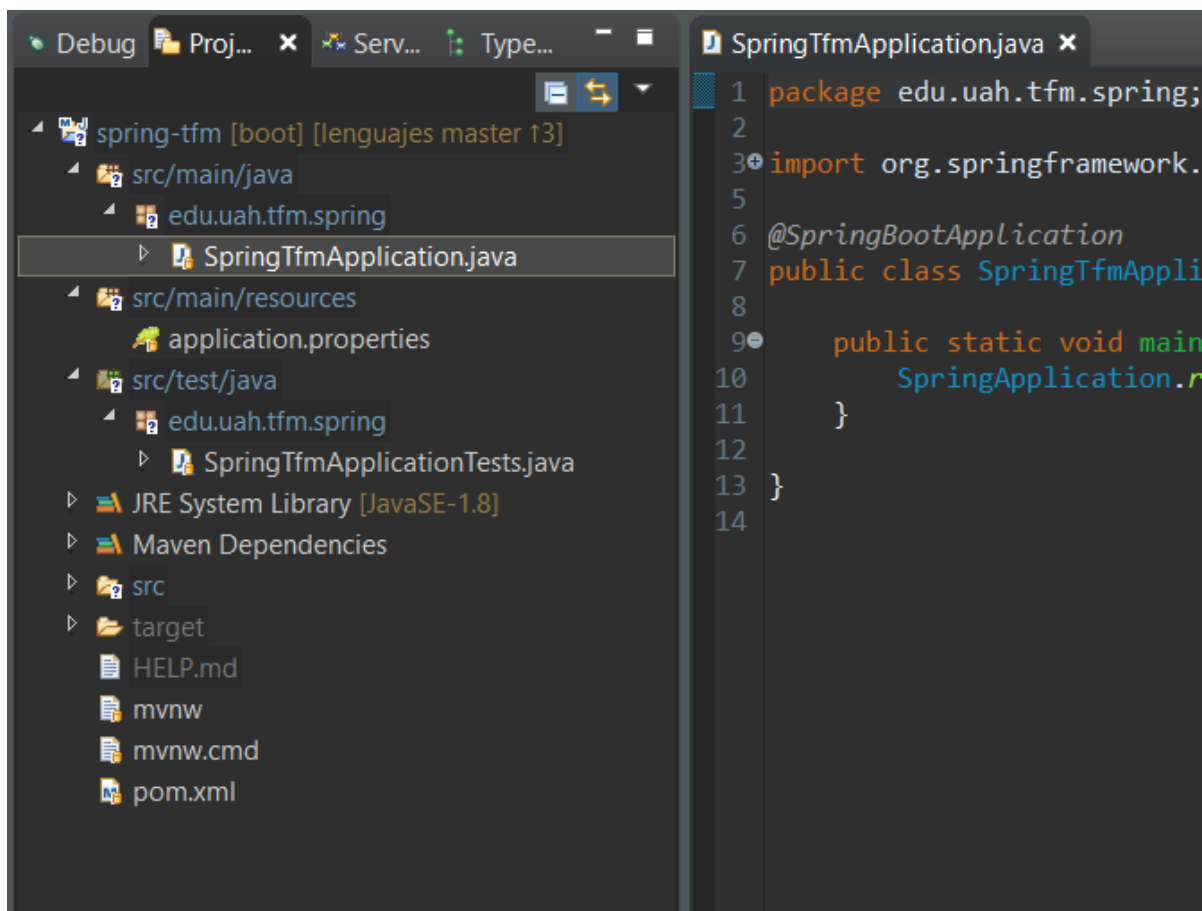


Ilustración 9-Estructura Proyecto Spring

La estructura de un proyecto con Spring Boot es similar a las utilizadas para cualquier otro proyecto realizado en Java Enterprise Edition (Jakarta EE) donde se tiene la jerarquía de paquetes que componen todo el código fuente de la aplicación dentro de src/main.

Sin embargo, a diferencia es otros Frameworks, en Spring Boot se crear por defecto una carpeta dentro del src/main/ llamada resources, en la cual se deben colocar todas las vistas, ficheros de configuración y los recursos externos que son utilizados por la aplicación.



2.8. Orientación a Aspecto con Spring

El módulo de orientación a aspecto de Spring es una de bondades que posee el framework desde sus inicios y está basada en la librería de Java AspectJ, la cual se convirtió en un estándar gracias a su facilidad de implementación en las aplicaciones.

A partir de la versión 2 de Spring se integra con la librería AspectJ, pero esta con el paso del tiempo y los avances realizados al framework, empezó a presentar limitaciones relacionadas con la gestión de beans, por lo cual Spring ofrece una solución a esas limitaciones a través del módulo Spring AOP y así complementar dicha librería.

Dentro de los conceptos que utiliza Spring en la orientación a aspecto están:

Termino	Definición
Aspecto	Es la clase que posee los mecanismos para tratar la preocupación transversal. Se anota con <code>@Aspect</code> para ser reconocida como un aspecto para Spring.
Tejido	Es el proceso de modificación o extensión de un programa dentro de la orientación a aspecto. Pueden ser de dos maneras: Estáticos en tiempo de compilación Dinámicos en tiempo de ejecución
Punto de unión	Es el momento donde interfiere la orientación a aspecto para realizar el proceso de tejido. Dentro de Spring corresponde al método interferido por el aspecto.
Objeto de destino	Es el objeto al que se le aplica el aspecto.
Consejo	En Spring, el consejo (advice) es la acción realizada por el aspecto en un punto de unión. Existen cinco tipos: <code>@Around</code> : permite realizar una acción antes y después de la invocación del punto de unión. <code>@Before</code> : permite realiza una acción antes del punto de unión. <code>@After</code> : se realiza una acción después de ejecutarse el punto de unión, sin importar si ha ocurrido una excepción en el punto de unión. <code>@AfterReturning</code> : se realiza la acción cuando se ha ejecutado el punto de unión y no ha ocurrido alguna excepción. <code>@AfterThrowing</code> : Se realiza una acción si ha ocurrido una excepción.



Pointcut	Son expresiones que sirven de filtro para identificar un punto de unión dentro del sistema se anota con <code>@Pointcut</code> .
----------	--

Uno de los requisitos técnicos a implementar en este TFM es la utilización de dos tipos de pool de conexiones a base de datos, uno que se realice para solo lectura y otro para escritura, de manera que todas las transacciones relacionadas con la inserción, actualización y eliminación de datos se realicen a través de una conexión correspondiente al pool de escritura, mientras que las transacciones de búsqueda o consulta se realicen a través del pool de lectura.

Una manera rápida de cumplir con este requerimiento es determinar en cada llamada a través de las clases repositorios si ésta corresponde a una búsqueda de datos o a una inserción. Sin embargo, esto genera una gran cantidad de código repetitivo en todas las clases componentes que realizan las llamadas. Sin embargo, través de la orientación a aspecto se puede abordar este problema de una manera más eficiente.

Con ayuda del módulo de orientación a aspecto que posee Spring Framework se crea un aspecto que interfiera los métodos con la anotación como se muestra a continuación:

```

@Aspect
@Component
@Order(0)
public class ReadOnlyDataSourceInterceptor {

    @Around("@annotation(transactional)")
    public Object proceed(ProceedingJoinPoint proceedingJoinPoint, Transactional
transactional) throws Throwable {
        if (transactional.readOnly()) {
            RoutingDataSource.setReadDataSource();
        } else {
            RoutingDataSource.setWriteDataSource();
        }
        return proceedingJoinPoint.proceed();
    }
}

public class RoutingDataSource extends AbstractRoutingDataSource{

    private static final ThreadLocal<DataSourceType> routeContext = new

```



```
ThreadLocal<>();

    public static void clearDataSource() {
        routeContext.remove();
    }

    public static void setWriteDataSource() {
        routeContext.set(DataSourceType.WRITE);
    }

    public static void setReadDataSource() {
        routeContext.set(DataSourceType.READ);
    }

    @Override
    protected Object determineCurrentLookupKey() {
        return routeContext.get();
    }
}

public enum DataSourceType {
    READ, WRITE
}
```

Como se puede observar, en la clase `ReadOnlyDataSourceInterceptor` realiza la acción del cambio de Datasource de lectura a escritura y viceversa de acuerdo con si la propiedad `readOnly` que posee la anotación `@transactional` es verdadera o falsa. Dicha clase es anotada con `@Aspect` para establecer que corresponde a un aspecto, `@Component` para que sea reconocido como un bean de Spring y `@Order` para definir el orden en que se ejecuta.

Se utiliza la anotación `@Around("@annotation(transactional)")` para realizar la acción deseada a todos los métodos que tengan la anotación `@transactional` en su definición.

Por último, el enum `DataSourceType` y la clase `RoutingDataSource` sirve para establecer el Datasource deseado, el cual será utilizado al momento de establecer la conexión con la base de datos.



2.9. Inversión de control e Inyección de dependencia

La inversión de control e inyección de dependencias son dos conceptos que han sido claves para el desarrollo y gran éxito que ha tenido Spring Framework a lo largo del tiempo. Estos mantienen una estrecha relación entre sí y son parte esencial del núcleo del Framework.

2.9.1. Inversión de control

La inversión de control es un principio de diseño de la ingeniería de software en el cual el control de los objetos de una aplicación se transfiere a un contenedor encargado de suplir la instancia que necesita el sistema para funcionar. Esto marca una diferencia a lo establecido en la programación tradicional donde la instanciación de los objetos se realizaba en la clase donde se quería utilizar.

Este principio utiliza otro principio de diseño llamado Hollywood (“Don’t call us, we’ll call you”, en español “No nos llames, nosotros te llamaremos”), el cual establece que las estructuras padres o superiores son las que deben llamar a las hijas o inferiores. Por lo que, para implementar en Java este principio solo basta con tener una interfaz que defina la estructura, una clase hija que implemente dicha interfaz y definir un objeto de la interfaz en la clase concreta que se utilice.

Sus ventajas son:

- Desacoplar la ejecución de una tarea de su implementación.
- Facilitar el cambio entre diferentes implementaciones.
- Hacer los sistemas más modulares.
- Una mayor facilidad para probar un programa aislando un componente y logrando que estos se comuniquen a través de interfaces.

Los Frameworks que utilizan la inversión de control deben seguir el principio de abierto cerrado, donde las entidades que componen el sistema deben estar abierta para extensión y cerrada para modificación.

El contenedor encargado de la gestión y control de los objetos dentro de Spring se llama Contenedor de Inversión de Control (IoC, por sus siglas en inglés) y forma parte del núcleo del Framework.

El contenedor Spring IoC tiene la responsabilidad de conectar los objetos definidos en el Framework para construir una aplicación y lo hace leyendo una configuración proporcionada por el



desarrollador. El contenedor Spring IoC es, por lo tanto, una autoridad externa que pasa una dependencia a un objeto dependiente que lo usará. (Cosmina, 2020)

2.9.2. Inyección de dependencia

Es un patrón de diseño orientado a objetos que busca reducir notablemente el acoplamiento entre las estructuras del sistema, donde se establece que las instancias de los objetos son suministradas por otro componente o contenedor sin que la clase tenga que establecer una dependencia directa con ese objeto, de manera, que en el código escrito por el desarrollador no se crea el objeto o se instancia, sino que recibe dicha instancia con toda la información que necesita.

La implementación de este patrón favorece en gran medida el mantenimiento, crecimiento y futuras extensiones del código, ya que permite ampliar las funcionalidades sin cambiar las estructuras ya definidas en el sistema. Utiliza un contenedor de dependencia o control para realizar la inyección de los objetos necesarios de acuerdo con la relación de dependencia definida previamente del objeto que la necesite.

Para la implementación de este patrón es recomendable utilizar interfaces para definir el comportamiento que debe tener la clase concreta y abstraer la relación entre dicha clase concreta y el objeto que la requiere, logrando de esta manera, el desacoplamiento entre clases y las bases para una programación orientada a interfaces.

Este patrón establece que:

- Las clases de alto nivel no deberían depender de las clases de bajo nivel. Ambas deberían depender de las abstracciones.
- Las abstracciones no deberían depender de los detalles. Los detalles deberían depender de las abstracciones.

En Spring, el proceso que realiza el contenedor de IoC para proporcionar las dependencias es llamado inyección y ocurre en tiempo de ejecución, cuando la aplicación se arma después de compilarse, y esto permite mucha flexibilidad, porque la funcionalidad de una aplicación se puede ampliar modificando una configuración externa sin una compilación completa de dicha aplicación. (Cosmina, 2020)

En Spring, la inyección de dependencia se basa en dos conceptos de Java: JavaBeans e interfaces. JavaBeans proveen un mecanismo estándar para crear objetos en java que se pueden configurar de



diversas maneras. Las interfaces son tecnologías que ofrecen un diseño claro que da como resultado una aplicación flexible.

Entre los beneficios de utilizar la inyección de dependencia están:

- Reducción de código redundante: Una de las grandes ventajas que ofrece la inyección de dependencia es la habilidad de reducir drásticamente la cantidad de código que se escribe para tener los componentes de una aplicación unidos.
- Simplificación en la configuración de la aplicación: Al utilizar inyección de dependencia se simplifica enormemente el proceso de configuración de una aplicación. Esto es gracias a una variedad de opciones para configurar esas clases que fueron inyectables en otras clases. Además, se facilita el intercambio de una implementación de una dependencia por otra.
- Gestión de dependencias comunes en un repositorio único: Cuando se utiliza la inyección de dependencia, toda la información sobre esas dependencias comunes está contenida en un repositorio único, lo que produce que la gestión de dependencias sea mucho más simple y menos propensa a errores.
- Mejoras en las pruebas: El diseño que propone la inyección de dependencia hace posible reemplazar las dependencias fácilmente dando como resultado una manera más sencilla y completa de realizar las pruebas.

Buen diseño de aplicaciones: El diseño sugerido por el patrón de inyección de dependencia significa, en general, diseñar basado en interfaces.

2.9.2.1. Tipos de inyecciones

Dentro de Spring Boot existen varios tipos de inyección de dependencia definidas a continuación:

1. Inyección de dependencia del constructor

La inyección de dependencia del constructor ocurre cuando las dependencias de un componente se le proporcionan en su constructor o constructores. El componente declara un constructor o un conjunto de constructores, tomando como argumentos sus dependencias, y el contenedor de inversión de control se encarga de pasar las dependencias en el momento de la instanciación.

La inyección de constructor es útil cuando con total seguridad se tiene una instancia de la clase de dependencia antes de usar el componente en Spring. Ayudando de esta manera al uso de objetos inmutables. Sin embargo, como consecuencia del uso de este tipo de inyección, un objeto no se



puede crear sin sus dependencias; lo cual puede producir una excepción si en el contenedor no posee ese tipo de objeto.

2. Inyección de dependencia a través de métodos setter

En esta inyección de dependencia, el contenedor de inversión de control inyecta las dependencias de un componente a través de métodos de setter. Como ventaja se puede crear un objeto sin sus dependencias, y se pueden proporcionar más tarde llamando al método setter.

La inyección de Setter es útil cuando la dependencia es opcional. Es decir, cuando no se proporcionan los valores por defecto. Otro beneficio de la inyección de setter es que permite que se declare la dependencia en una interfaz, a diferencia de la dependencia al constructor.

3. Inyección de campos

Hay un tercer tipo de inyección de dependencia utilizada en Spring llamada inyección de campo. Como su nombre lo indica, la dependencia se inyecta directamente en el campo, sin necesidad de constructor ni setter. Esto se hace anotando el miembro de la clase con “@Autowired”.

El término *autowire* es la versión corta para la inyección automática de dependencias. La anotación @Autowired indica que Spring debería encargarse de inyectar esa dependencia. (Cosmina, 2020)

Esto simplifica y reduce el código que se debe escribir para obtener una dependencia, pero hacer uso de esta, implica las siguientes desventajas:

- Es cierto que es mucho más sencillo añadir dependencias a las clases utilizando este tipo de inyección, pero se debe tener cuidado de no violar el principio de responsabilidad única. Ya que mientras más dependencias existan más se carga las clases con responsabilidades que al momento de realizar una refactorización resultará más difícil dividir dichas responsabilidades.
- La responsabilidad de inyectar dependencias se pasa al contenedor en Spring, pero la clase debe comunicar claramente el tipo de dependencias necesarias mediante una interfaz pública, a través de métodos o constructores. Al usar inyecciones de campo, no queda claro si el tipo de dependencia es realmente necesaria y si la dependencia es obligatoria o no.



- Al usar `@Autowired` se introduce una dependencia del propio contenedor de Spring a la entidad o al bean, por lo que este ya no es un POJO (Plain Old Java Object) y no se puede instanciar de forma independiente.
- No se puede utilizar en campos declarados como `final`, ya que ese tipo de campos solo se puede inicializar mediante la inyección del constructor.
- Presenta dificultades al escribir pruebas, ya que las dependencias deben inyectarse manualmente.

4. Inyección de valores simples

La inyección de valores es sencilla, se realiza colocando la anotación `@value` a las propiedades del bean al que se desea obtener el valor.

5. Inyección de valores utilizando SpEL (Spring Expression Language)

Spring Expression Language (SpEL) permite evaluar una expresión dinámicamente en tiempo de ejecución y luego usarla en el `ApplicationContext` de Spring. Esta fue introducida a partir de la versión 3 de Spring y es muy útil durante la creación de las vistas con JSP u otros motores de plantilla.

En el siguiente fragmento de código de la aplicación, se puede visualizar los distintos tipos de inyecciones:

```
@Service("ticketService")  
public class TicketService {  
  
@Autowired  
private ITicketDao ticketDao;  
  
@Value("2")  
private int cantidadDeTicket;  
  
@Value("F5")  
private String asiento;  
  
@Value("false")  
private boolean esConcierto;
```



```
@Value("#{configuracion.defecto.url}")
private String urlPorDefecto;

@Value("#{configuracion.defecto.contador + 1}")
private int contador;

@Resource(name="listadoEventos")
private List listadoEventoPruebas;

private IAsientoDao asientoDao;

private EventoService eventoService;

public TicketService(IAsientoDao asientoDao) {
    this.asientoDao = asientoDao;
}

public void setEventoService(EventoService eventoService) {
    this.eventoService = eventoService;
}
}
```



2.10. Beans en Spring Boot

En Spring, los objetos que forman la columna vertebral de la aplicación y que son administrados por el contenedor Spring IoC se llaman beans. Un bean es un objeto que se crea, se instancia, se ensambla y administra mediante el contenedor de inversión de control de Spring IoC. (Spring.io, 2020).

Un bean es un simple objeto de la aplicación que se encuentra dentro del alcance de Spring, el cual es inyectado cuando es requerido a través de la inyección de dependencia en función de la configuración existente en el proyecto.

Cada bean en el contexto de la aplicación posee un identificador único. El cual puede ser definido manualmente al momento de declarar el objeto o nombrado de manera automática por el contenedor de inversión de control.

Para definir un Bean en Spring Boot se utiliza la anotación `@Bean`, indicando que ese método produce un bean y que será administrado por el contenedor IoC de Spring. Luego, este método es ejecutado durante la configuración de Java y su valor de retorno se registra como un bean dentro de una Factoría de Bean.

La interfaz `BeanFactory` es responsable de administrar los componentes, dependencias y ciclos de vida de los beans. Por lo que cuando se necesita acceder a cualquier bean por la aplicación, se debe interactuar con el contenedor de inyección de dependencia de Spring a través de la interfaz `BeanFactory`.

2.10.1. Ciclo de vida de un Bean

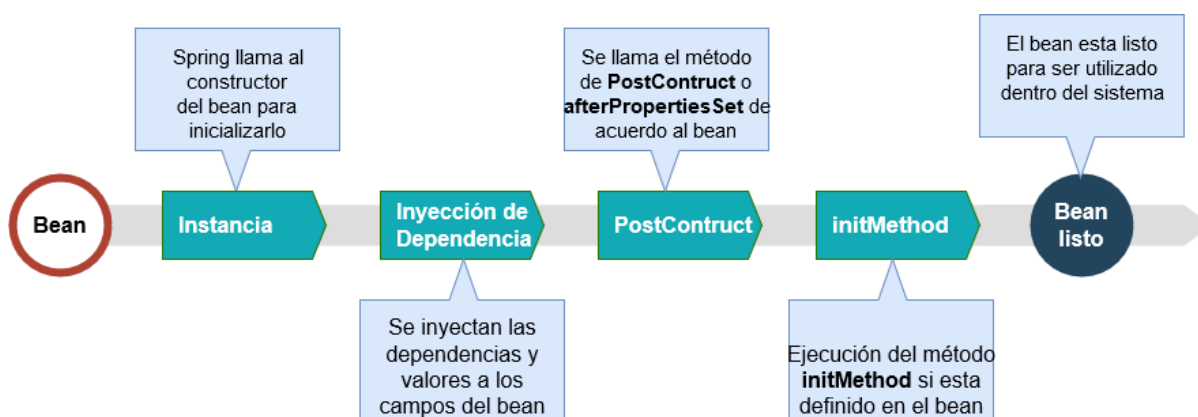


Ilustración 10-Ciclo de vida de un Bean



El ciclo de vida de un bean en Spring es importante y a la vez complejo, ya que al ser manejados por un contener deben pasar por un conjunto de paso desde su creación para poder ser utilizados y luego se añaden otros pasos más antes de ser destruido.

El ciclo de vida de un bean se compone de los siguientes pasos:

1. Spring llama al constructor del bean para inicializarlo.
2. Se inyectan las dependencias y valores a los campos del bean.
3. Se ejecuta el método anotado con `@PostConstruct` si existe dentro del bean.
4. Si no se declaró un método con `@PostConstruct`, se ejecuta el método `afterPropertiesSet()` si el bean posee una implementación de la interfaz `InitializingBean`.
5. Se ejecuta el método `initMethod` si ha sido definido durante la declaración del bean.
6. El bean ha sido creado y puede ser utilizado.
7. Si el bean tiene declarado un método con la anotación `@PreDestroy` es ejecutado al momento de destruir el bean.
8. Se ejecuta el método `destroy()` si el bean posee una implementación de la interfaz `DisposableBean` justo antes de destruir el bean.



2.11. Usando CommandLineRunner y ApplicationRunner con Spring Boot

CommandLineRunner y ApplicationRunner son interfaces proporcionadas por Spring Boot que sirven para ejecutar un bloque de código específico justo antes de que finalice el cargado por completo de la aplicación y componentes.

Estas interfaces se pueden utilizar en múltiples escenarios:

- Cuando se desea ejecutar un código de inicialización de algún componente específico.
- Para obtener información de algún servicio externo.
- Ejecutar algún proceso en batch.
- Registrar cualquier mensaje.

Para utilizarla se debe crear una clase que sea un componente para que sea reconocida por Spring y que implemente una de estas interfaces y se sobrescriba su método “run”, donde se coloca el código que se desea ejecutar:

```
@Component  
@Order(1)  
public class CommandRunner implements CommandLineRunner{  
  
    @Override  
    public void run(String... args) throws Exception {  
  
    }  
}
```

O implementando ApplicationRunner:

```
@Component  
@Order(2)  
public class MiApplicationRunner implements ApplicationRunner{  
  
    @Override  
    public void run(ApplicationArguments args) throws Exception {  
  
    }  
}
```



La anotación `@Order` sirve para establecer en qué orden Spring ejecutará cada componente que implemente estas interfaces.

Ambas interfaces tienen el mismo propósito, pero se diferencian en el argumento que se recibe en el método `run`. `CommandLineRunner` acepta el arreglo de `String` que se pasan al iniciar el servidor como su argumento, mientras que `ApplicationRunner` acepta un `ApplicationArguments` como argumento, el cual contiene todos los argumentos que se pasan al método “main” que inicia la aplicación.



2.12. Anotaciones de estereotipos en Spring

Spring Framework contiene una gran cantidad de anotaciones que han logrado simplificar el proceso de desarrollo para que en pocas líneas de código se pueda tener un sistema funcionando con interfaz, manejo de peticiones y acceso a datos. De igual manera se ha disminuido el manejo de configuración a través de los ficheros .xml con el objetivo de simplificar el proceso de desarrollo.

Entre las anotaciones principales destacan los estereotipos, siendo la base principal de todo el desarrollo web para la creación de aplicaciones con Spring. Estas anotaciones se encuentran en el paquete de spring “org.springframework.stereotype” y está conformado por las siguientes anotaciones:

- **@Component**: Es una anotación estereotipo genérica, utilizada para que Spring interprete a la clase que la utiliza como parte de sus componentes, lo agregue al contenedor y pase a ser uno de los beans que esta gestiona.
- **@Controller**: Es la anotación que realiza la funcionalidad de controlar las clases y gestionar la comunicación entre la vista y los demás componentes de la aplicación.
- **@Repository**: Esta anotación es utilizada para los componentes inmersos en la persistencia y acceso a datos implementando el patrón de diseño Repositorio para el contacto con las bases de datos.
- **@Service**: Su función es implementar la lógica de negocio a través de procesos o funcionalidades en donde establece relación con los repositorios que posee la aplicación.



2.13. Persistencia de Datos con Spring Boot

En la actualidad la persistencia de datos es parte fundamental en aplicaciones web empresariales, ya que sin esta no se podría almacenar la información, ni hacer uso de ella posteriormente. Para solventar esa necesidad existen las bases de datos, las cuales permiten almacenar un gran conjunto de datos de forma estructurada para luego extraerlos en el momento que se necesiten.

En el mercado existen muchos tipos y modelos de bases de datos, pero los más utilizados en las aplicaciones son las bases de datos dinámicas y los modelos de base de datos relacionales (SQL) y no relacionales (NoSQL). Entre las que se destacan están:

SQL	NoSQL
Oracle	MongoDB
MySQL	Elasticsearch
Microsoft SQL Server	Redis
PostgreSQL	Apache Cassandra
Db2	DynamoDB
SQLite	
MariaDB	

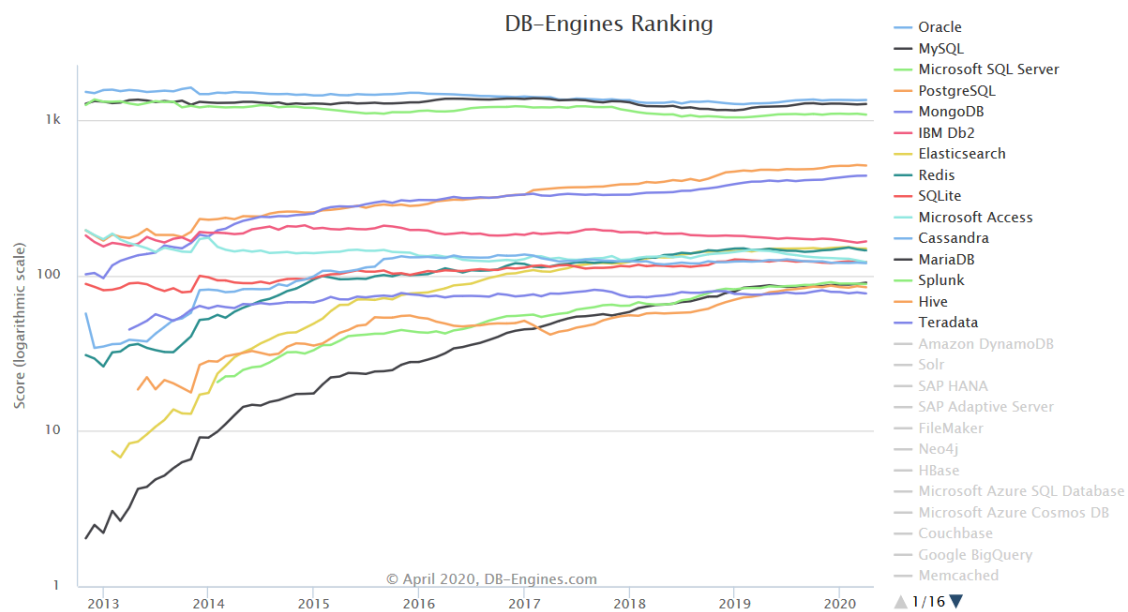


Ilustración 11-Ranking Bases de Datos – Fuente: https://db-engines.com/en/ranking_trend

Para dar soporte al conjunto de base de datos SQL y NoSQL listadas en el punto anterior, el equipo de Spring desarrolló un proyecto llamado Spring Data que ofrece un modelo de acceso a los datos basados en la misma estructura consistente del Framework, es decir, que permite la configuración automática, la creación de beans y un gran número de herramientas para persistir y manipular los datos.

Este facilita el uso de tecnologías de acceso a datos, bases de datos relacionales y no relacionales, marcos de reducción de mapas y servicios de datos basados en la nube. Es un proyecto general que contiene muchos subproyectos que son específicos para una base de datos determinada. (Spring.io, 2020).

Entre sus características están:

- Posee un potente repositorio y abstracciones personalizadas para el mapeo de objetos.
- Permite realizar consulta dinámica solo con nombrar los métodos de acuerdo con los criterios de Spring Data.
- Soporte para auditorías de las entidades basado en los campos de fecha de creación y modificación.
- Posibilidad de personalizar de los repositorios añadiendo el código deseado.
- Integración avanzada con los controladores creados bajo Spring MVC.

De acuerdo con la página oficial de Spring, sus módulos principales son:



Módulo	Descripción
Spring Data Commons	Es un módulo común que posee los conceptos básicos de Spring y el código base que sustentan cada módulo Spring Data.
Spring Data JDBC	Módulo de Spring Data que brinda soporte al API de conectividad a bases de datos de Java (JDBC, por sus siglas en inglés).
Spring Data JPA	Módulo de brinda soporte al API de persistencia de Java, llamada JPA por sus siglas en inglés.
Spring Data KeyValue	Repositorios basados en mapas y SPI para crear fácilmente un módulo Spring Data para almacenes de datos en forma de clave-valor.
Spring Data LDAP	Módulo de Spring Data que brinda soporte con LDAP.
Spring Data MongoDB	Módulo de soporte a los objetos basados en documento de la base de datos MongoDB.
Spring Data Redis	Módulo de soporte para el motor de base de datos en memoria basado en hashes Redis.
Spring Data REST	Módulo para la creación automática de servicios REST basado en la exportación de los repositorios de datos con Spring.
Spring Data para Apache Cassandra	Módulo para configuración y acceso a Apache Cassandra o aplicaciones Spring orientadas al manejo de datos a gran escala y con alta disponibilidad.
Spring Data para Apache Geode	Módulo de configuración y acceso sencillos a Apache Geode para aplicaciones Spring altamente consistentes, de baja latencia y orientados a datos.
Spring Data para Apache Solr	Módulo de configuración y acceso a Apache Solr para la creación de aplicaciones Spring orientadas a la búsqueda.
Spring Data para Pivotal GemFire	Módulo de configuración y acceso a Pivotal GemFire para la creación de aplicaciones Spring orientadas a datos altamente consistentes, de baja latencia y alto rendimiento.



Para el desarrollo de este trabajo se abordará la persistencia y gestión de los datos por medio de bases de datos relacionales, por lo cual, se utilizará el módulo de Spring Data JPA en todo lo relacionado con el acceso a datos.

2.13.1. Spring Data JPA

Spring Data JPA es un módulo de Spring Data para la gestión de acceso a datos que requieren las aplicaciones web de una manera sencilla y basada en las tecnologías y estructuras utilizadas por Spring.

Este módulo crea una capa de abstracción al API de persistencia de Java (JPA), mejorando en gran medida los procesos utilizados para ejecutar consultas, manipular datos, realizar auditoria y reduciendo la cantidad de código repetitivo que se necesita para establecer el acceso a datos en las diferentes tecnologías de base de datos relacionales.

Entre sus características están:

- Soporte para construir repositorios basados en Spring y JPA.
- Utiliza la librería Querydsl para la creación de sentencias y consultas JPA seguras.
- Auditoría transparente de la clase de dominio por medio de anotaciones.
- Soporte de paginación, ejecución dinámica de consultas y capacidad de integrar código de acceso a datos personalizado.

Es importante destacar que JPA es una especificación estándar aplicado a Java para manejar y mejorar la persistencia de los objetos y el acceso dentro de la aplicación. Esta fue publicada por primera vez en mayo del año 2006 por el proceso de comunidad de Java y poco tiempo después se implementó como parte del lenguaje Java bajo el paquete javax.persistence, otorgando los siguientes beneficios:

- API para dar soporte a las entidades y objetos.
- Implementación de un ORM para el mapeo de objeto relacional.
- Metadatos de los objetos y relaciones existentes en la base de datos.
- Lenguaje de consulta desde Java (JPQL).
- Criterios API a partir de su versión dos para mejorar las consultas.



Mapeo de Objeto-Relacional (ORM) hace referencia a proceso de conversión de entidades entre un sistema de base de datos relacional y una aplicación.

Como parte de la implementación de Java para JPA se utilizan las siguientes anotaciones:

Anotación	Definición
@Entity	Declara la clase como una entidad que representa una tabla de la base de datos.
@Table	Utilizada para indicar el nombre de la tabla.
@Id	Se usa para identificar el campo que corresponde a la clave primaria de la tabla.
@IdClass	Utilizada para establecer claves compuestas formada por una clase Java.
@EmbeddedId	Indica que la clave primaria es compuesta y se encuentra en otra clase Java.
@Column	Indica que el campo corresponde a una columna de la tabla.
@Temporal	Utilizada para el mapeo de fechas entre Java y la base de datos.
@Transient	Indica que el campo no se persiste en la base de datos.
@ManyToOne	Establece la relación mucho a uno de la tabla en el campo de relación con otro objeto en Java.
@OneToMany	Establece la relación uno a mucho tomando como referencia en campo que une a ambas tablas.
@ManyToMany	Establece la relación mucho a mucho tomando como referencia en campo que une a ambas tablas.
@JoinColumn	Establece que existe una unión entre el campo y otra tabla.
@OneToOne	Establece la relación uno a uno tomando como referencia el campo que une a ambas tablas.
@JoinTable	Sirve para establecer la unión entre tablas. La condición de dicha unión puede ser compuesta.

Con el paso del tiempo, Hibernate, que era una herramienta de software que poseían características similares y buscaban el mismo objetivo, empezó a cumplir con la especificación

establecidas de JPA, hasta el punto de convertirse en una implementación certificada de dicha especificación.

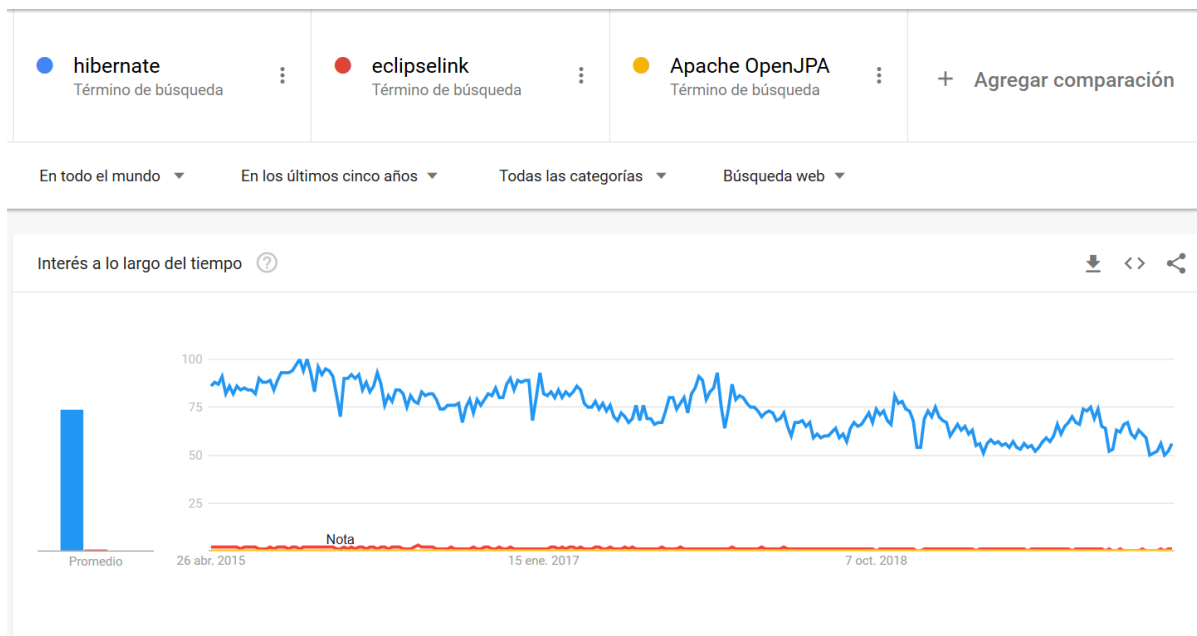


Ilustración 12-Orm en Java

En la actualidad Hibernate es el framework más utilizado en las aplicaciones Java para el mapeo de las entidades de una base de datos relacional con las clases POJO de la aplicación gracias a las siguientes características:

- **Flexibilidad:** Ofrece diversas maneras de realizar la configuración de la base de datos y permite extender su comportamiento. Al igual que se adapta a los esquemas de base de datos utilizados.
- **Simplicidad:** Toda la herramienta está basada en configuraciones simples y el uso de anotaciones dentro de la aplicación.
- **Alto rendimiento:** Implementa estrategias de inicialización, obtención de datos, generación de sentencias SQL y sellado de tiempo para otorgar un alto rendimiento en las transacciones.
- **HQL:** Es el lenguaje de consulta de datos en donde se puede definir un conjunto de sentencias simples o complejas utilizando el API “Criteria” que luego son compiladas y cacheadas.
- **Completo:** Utiliza un gran número de componentes basados en la orientación a objeto y manejo de tipos de datos.
- **Es de código abierto:** Cuenta con el apoyo de toda la comunidad que sigue aportando para mejorar el Framework.



En la mayoría de las aplicaciones creadas con Java y Spring Boot se utiliza Hibernate como proveedor de implementación de la especificación JPA.

2.13.2. Patrones de diseños utilizados

Los principales patrones utilizados por Spring Data JPA junto con Hibernate Para el manejo de la persistencia son Repositorio y DAO.

2.13.2.1. Patrón Repositorio

Un Repositorio crea la ilusión de una colección en memoria de todos los objetos del tipo que maneja. Configura el acceso a través de una interfaz global bien conocida, proporciona métodos para agregar y remover objetos que encapsulen las operaciones reales y proporciona métodos que seleccionan objetos basados en algún criterio y que devuelvan objetos o colecciones de objetos completamente instanciados cuyos valores de atributos cumplen los criterios. (Evans, 2003)

El patrón repositorio plantea la creación de una interfaz que gestione el acceso a los datos definiendo métodos para obtener, crear, actualizar y eliminar objetos que pasan a ser persistidos en una fuente de almacenamiento por medio de las llamadas internas que realizan estos métodos. También plantea la realización de sentencias de búsqueda filtradas dentro de los objetos utilizando el patrón Specification.

El objetivo de este patrón es desacoplar la de datos del resto de capas en una aplicación y establecer un modelo de dominio para tratar todo lo relacionado con el acceso a datos, donde en lugar de analizar el modelo relacional de una base de datos, se analice como gestionar de manera eficiente los objetos del sistema y su interacción con las demás capas y componentes.

Spring implementa este patrón proveyendo un conjunto de interfaces para realizar el acceso a datos y las operaciones con los objetos. Entre las que se destacan:

- **Repository:** Es la interfaz principal que administra los objetos del dominio y sus tipos.
- **CrudRepository:** Extiende de la interfaz **Repository** y proporciona la funcionalidad de CRUD a la clase que administra por medio de sus métodos.
- **PagingAndSortingRepository:** Esta interfaz extiende de **CrudRepository** y provee métodos adicionales para obtener entidades utilizando paginación y clasificación.
- **JpaRepository:** Extiende de **PagingAndSortingRepository** y provee un conjunto de métodos para ampliar el funcionamiento de los repositorios.



2.13.2.2. Patrón DAO

El patrón Data Access Object (DAO) es un patrón estructural que establece que se debe utilizar un Objeto de Acceso a Datos para abstraer y encapsular el acceso a los datos. Este objeto gestiona la conexión con la base de datos para obtener y guardar los datos. De esta manera, los componentes de una aplicación se mantienen aislados de la capa de persistencia.

Los DAOs ocultan completamente los detalles de implementación de la fuente de datos a sus clientes. Ya que la interfaz expuesta por los DAOs a sus clientes no cambia cuando la implementación de la fuente de datos subyacente cambia, este patrón permite adaptarse a diferentes esquemas de almacenamiento sin afectar a sus clientes (Deepak Alur et. Al, 2001).

Para implementar este patrón dentro de Spring Boot es necesario apoyarse en el patrón Repositorio con las interfaces que posee Spring para la administración de los datos.

Elementos que posee el patrón

Elemento	Descripción
Cliente	Es el objeto que requiere el acceso a los datos. Este puede ser una clase controlador, un objeto, un servicio o cualquier componente.
DataAccessObject	Es el objeto que se encarga de las operaciones con la fuente de datos. Este puede utilizar un objeto de transferencia (DTO) para realizar su función.
Fuente de Datos	Puede ser una base de datos o cualquier sistema que provea almacenamiento.
ResultSet	Es el resultado obtenido al realizar una consulta en la fuente de datos.
Data	Es el objeto utilizado por el DataAccessObject para realizar operaciones y retorna información.

Es importante mencionar que tanto el patrón Repositorio como el DAO cumplen objetivos distintos, donde su diferencia radica en que el patrón Repositorio busca obtener, proporcionar y manipular los objetos dentro de un dominio y el patrón DAO busca almacenar y extraer los datos de una base de datos.



2.13.3. Implementando Spring Data JPA

Para utilizar Spring Data JPA en Spring Boot se necesita añadir las siguientes dependencias al proyecto que se desea establecer el acceso a datos:

- `spring-boot-starter-data-jpa`: contiene el conjunto de librería a utilizar en Spring Boot para el acceso a datos.
- La dependencia del motor de base de datos a utilizar si pertenece a una de las bases de datos embebidas por Spring.
- En caso de utilizar una base de datos no embebida por Spring, se debe añadir la dependencia del Driver de conexión que controla la comunicación entre Java y la base de dato.

Las bases de datos embebidas son muy útiles durante la fase de desarrollo, ya que son fáciles de configurar, rápidas, con tiempos de inicio cortos y permiten desarrollar una aplicación sin necesidad de instalar todo el ambiente externo de base de datos que requiere un entorno empresarial. Estas generalmente siguen el esquema de almacenamiento de datos en memoria, lo cual las hace más rápida en comparación con el esquema de almacenamiento en disco.

Sin embargo, estas bases de datos dentro del contexto de Spring tienen la limitación de que mantienen los datos en memoria de manera temporal y no los persiste en disco, por lo cual son utilizadas con propósitos de desarrollo y pruebas.

Spring incluye las siguientes bases de datos de manera embebida:

- H2
- HyperSQL Database (HSQLDB)
- Apache Derby

A continuación, se mostrará el uso de Spring Data JPA de dos maneras:

- 1- Utilizando H2 como base de datos embebida que posee Spring.
- 2- Utilizando MySQL como una base de datos externa.

2.13.3.1. Utilizando la base de datos H2

Se debe añadir la dependencia del motor de base de datos y Spring Data JPA al fichero `pom.xml`:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```



```
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>
```

Para mostrar el funcionamiento se creará la clase Usuario con las anotaciones necesarias.

```
@Entity
@Table(name="usuario")
public class Usuario {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name="nombre")
    private String nombre;

    @Column(name="apellido")
    private String apellido;

    @Column(name="email")
    private String email;

    @Column(name="telefono")
    private String telefono;
    //Métodos getter y setter omitidos
}
```

Como se ha mencionado en el listado anterior, están presentes las siguientes anotaciones:

1. `@Entity` para establecer que la clase corresponde a una entidad de la base de datos.
2. `@Table` para determinar el nombre de la tabla a la que hace relación la clase.
3. `@Id` para establecer que campo corresponde a la clave primaria de la clase.
4. `@GeneratedValue` para indicar la estrategia a utilizar para generar la clave al momento de la inserción de datos. En el ejemplo se puede apreciar que se ha colocado “`GenerationType.IDENTITY`” para indicar que su generación será autoincrementar.
5. `@Column` para establecer que el campo corresponde a una columna de la base de datos. Si no se coloca el nombre, esta toma el nombre del campo como nombre de la columna por defecto.



Luego de tener la clase entidad que representa la tabla en la base de dato, creamos una interfaz que extienda de `CrudRepository` para que nos provea lo métodos que se necesitan para persistir los datos. Al momento de extender de esta es necesario colocar la clase de tipo entidad utilizada para enlazar los datos y el tipo de datos de la clave primaria.

```
import org.springframework.data.repository.CrudRepository;

public interface UserDao extends CrudRepository<Usuario, Long>{

}
```

Luego de tener la entidad y las clases que establecerán la comunicación con la fuente de datos, debemos configurar h2 en el fichero `application.properties` ubicado en el directorio `/resources`, colocando lo siguiente:

```
spring.datasource.url=jdbc:h2:mem:tfmdb
spring.datasource.driverClassName=org.h2.Driver
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.datasource.username=sa
spring.datasource.password=sa
spring.jpa.hibernate.ddl-auto=create-drop
spring.h2.console.enabled=true
```

Dónde:

- `Spring.datasource.url` se utiliza para determinar la dirección en la que se encuentra la fuente de datos.
- `Spring.datasource.driverClassName` determinar la clase utilizada para el controlar todos los procesos de comunicación y conexión con la base de datos.
- `spring.jpa.database-platform` especifica la clase Java que controlará el proceso de mapeo con el orm Hibernate
- `Spring.datasource.username` y `spring.datasource.password` se utilizan para colocar el usuario y contraseña necesaria para la conexión a la base de datos.
- `spring.jpa.hibernate.ddl-auto` es utilizado para definir que al momento de ejecutar el sistema se creen las tablas automáticamente a partir de la definición que tengan las entidades Java del proyecto.
- `spring.h2.console.enabled` habilita la consola para visualizar las estructuras y datos de la base de datos H2.



Con fines de validar esta implementación se añade la dependencia `spring-boot-starter-web` al proyecto para poder visualizar los datos con H2 y se ejecuta el sistema para que se pueda ver la consola.

Para visualizar la consola, se debe acceder a la siguiente dirección con un navegador web (<http://localhost:8080/h2-console/>):

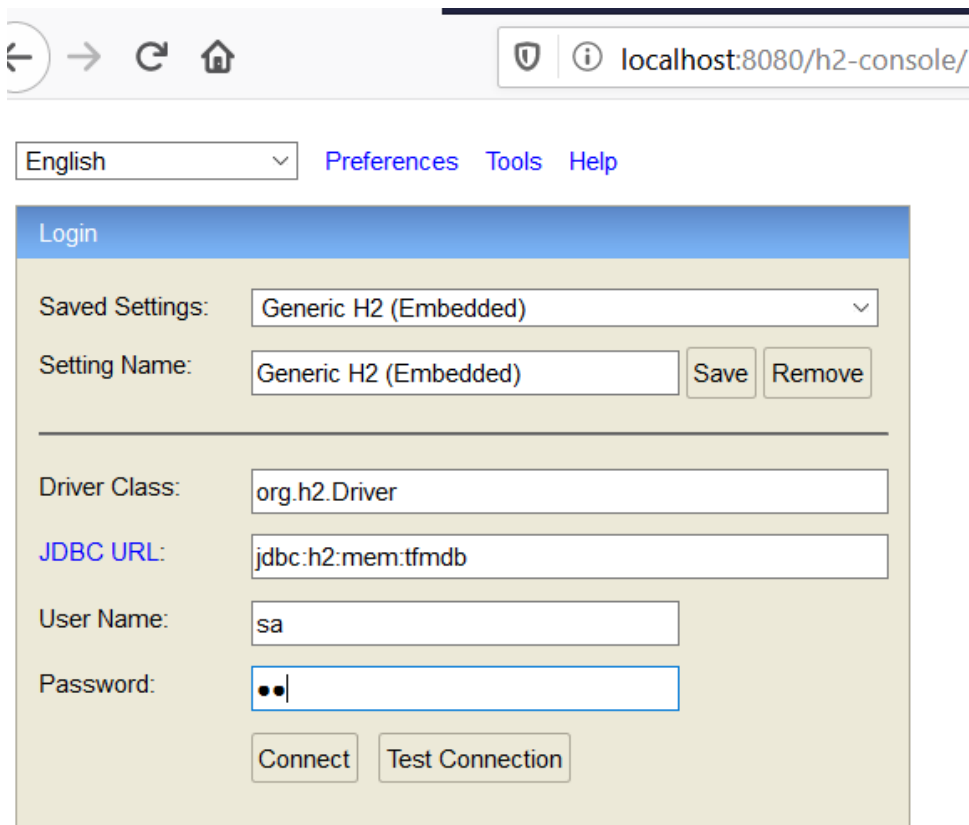


Ilustración 13-Formulario Login Base de Datos H2

Donde luego de rellenar los campos JDBC URL, User Name y Password con los mismos datos que en el fichero `application.properties` se inicia sesión en la consola, mostrando lo siguiente:

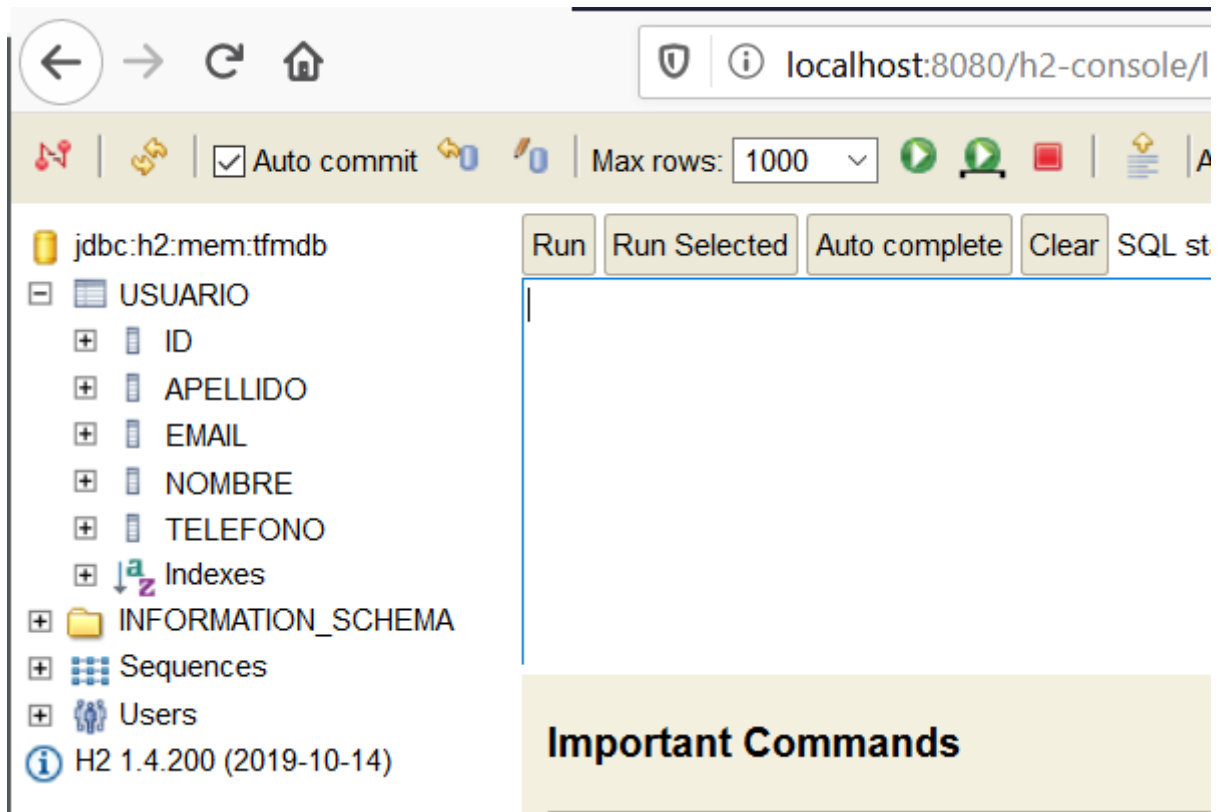


Ilustración 14-Estructura tabla Base de Datos H2

Donde se puede observar que ha sido creada la estructura de la base de datos.

2.13.3.2. Utilizando la base de datos MySQL

Para utilizar MySQL es necesario realizar la instalación de la base de datos ya sea descargando el fichero instalador de su página oficial o por medio de alguna herramienta o software que contenga la base de datos MySQL de manera embebida.

Para fines de implementación se asume que se tiene la base de datos MySQL instalada y configurada. Para acceder a esta a través de un entorno gráfico Oracle provee una herramienta de visualización, administración, gestión y mantenimiento llamada MySQL Workbench, en la cual se creará un esquema llamado “alcala”:

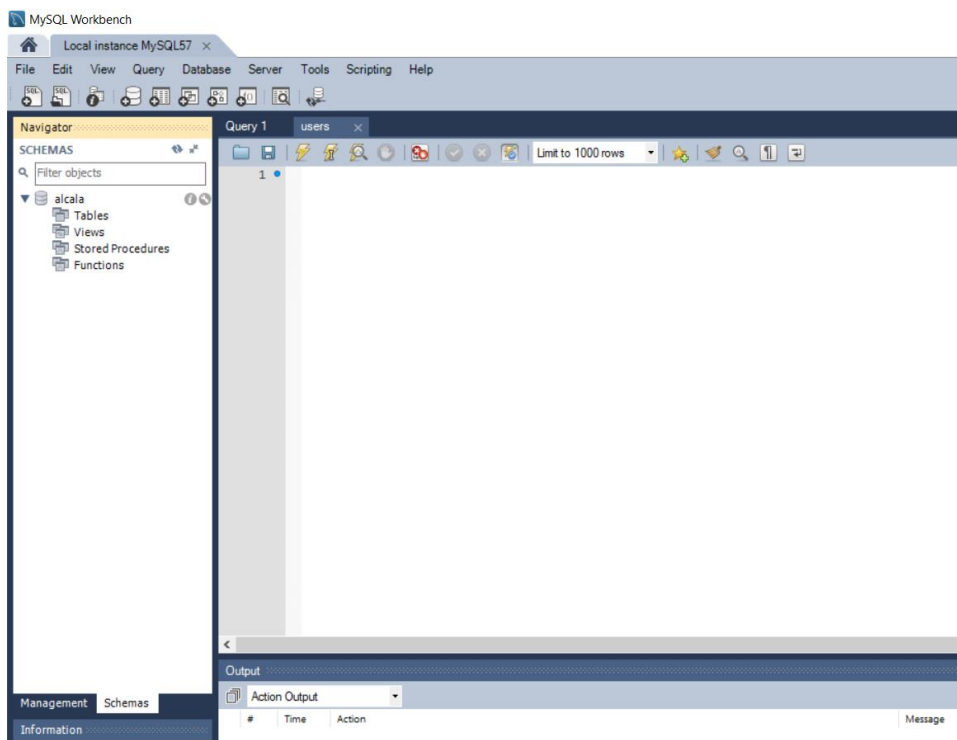


Ilustración 15-Estructura Base de Datos

Para realizar la implementación con MySQL se puede reutilizar el proyecto anterior con H2, realizando solo cambios en la dependencia a incluir y el fichero de configuración para establecer la conexión con la base de datos:

Primero se removerá la dependencia de H2 y se añade el conector de MySQL:

```
<dependency>  
  <groupId>mysql</groupId>  
  <artifactId>mysql-connector-java</artifactId>  
  <scope>runtime</scope>  
</dependency>
```

Luego se realizan los cambios necesarios en el fichero `application.properties` para establecer la conexión con la base de datos de acuerdo a la configuración previamente realizada en MySQL:

```
spring.datasource.url=jdbc:mysql://localhost:3306/alcala?useSSL=false&serverTimezone=  
UTC  
spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver  
spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect  
spring.datasource.username=root
```

```
spring.datasource.password=root  
spring.jpa.hibernate.ddl-auto=create-drop
```

Al ejecutar el proyecto se crea la estructura de base de datos por defecto:

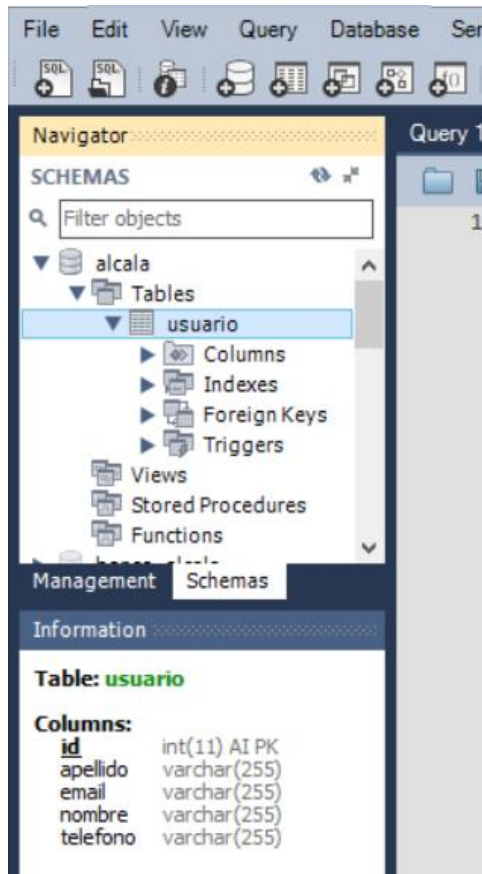


Ilustración 16-Tabla usuario

Spring Boot dentro de su configuración establece que si encuentra un fichero import.sql dentro de directorio /resources intentará ejecutar su contenido en la base de datos. Para implementarlo se debe crear el fichero import.sql y colocar los datos dentro del mismo:

```
INSERT INTO USUARIO (nombre,apellido, email, telefono)  
values\('Santiago','Ramirez','santiago.ramirez@edu.uah.es','99999999'\);
```

```
INSERT INTO USUARIO (nombre,apellido, email, telefono)  
values\('Juan','Aybar','juan.aybar@test.es','99999999'\);
```

Al ejecutar la aplicación se puede observar la ejecución del script en la consola del IDE y el Workbench:



```

ariDataSource : HikariPool-1 - Starting...
ariDataSource : HikariPool-1 - Start completed.
Dialect       : HHH0000400: Using dialect: org.hibernate.dialect.MySQL5InnoDBDialect
SchemaCreatorImpl : HHH000476: Executing import script 'file:/C:/lenguajes/spring/workspace/microservicios/spring-tfm/target/classes/import.sql'
FormInitiator  : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
EntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
nt.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''

```

id	apellido	email	nombre	telefono
1	Ramirez	santiago.ramirez@edu.uah.es	Santiago	999999999
2	Avbar	juan.avbar@test.es	Juan	999999999

Ilustración 17-Resultado sentencia de consulta a tabla usuario

2.13.4. Auditoria a las entidades con JPA

Spring Data JPA ofrece un conjunto de componentes utilizados a través de anotaciones para auditar las inserciones y actualizaciones que se realizan en las tablas. Estas anotaciones permiten insertar la fecha de creación y fecha de modificación en los registros que poseen las tablas.

Para utilizar dicha auditoria dentro de un proyecto Spring Boot, se debe incluir la dependencia de Spring Data JPA que contiene las anotaciones que aparecen en las clases que se muestran a continuación:

```

@Entity
@Table(name="usuario")
@EntityListeners(AuditingEntityListener.class)
public class Usuario {

```

```

    @Id

```




```
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Integer id;

@Column(name="nombre")
private String nombre;

@Column(name="apellido")
private String apellido;

@Column(name="email")
private String email;

@Column(name="telefono")
private String telefono;

@Column(name="fecha_creacion")
@CreatedDate
private Date fechaCreacion;

@Column(name = "fecha_modificacion")
@LastModifiedDate
private Date fechaModificacion;

public Usuario(String nombre, String apellido, String email, String telefono) {
    this.nombre = nombre;
    this.apellido = apellido;
    this.email = email;
    this.telefono = telefono;
}

public Usuario() {

}
}

@SpringBootApplication
@EnableJpaAuditing
public class SpringTfmApplication implements CommandLineRunner{

    @Autowired
    private UserDao userDao;
```



```
public static void main(String[] args) {
    SpringApplication.run(SpringTfmApplication.class, args);
}

@Override
public void run(String... args) throws Exception {
    Usuario testU = new Usuario("Prueba", "Prueba", "test@test.com", "669669699");
    userDao.save(testU);
}
}
```

- `@EntityListeners(AuditingEntityListener.class)` de la clase `Usuario` sirve de observador e interceptor al momento de que el `EntityManager` realice cualquier tipo de operación con la entidad y de esta manera coloca fecha correspondiente a los campos anotados con `@CreatedDate` y `@LastModifiedDate`.
- `@CreatedDate` y `@LastModifiedDate` son las anotaciones que sirve para determinar que campos de tipo fecha corresponden a la auditoria de tablas dentro de la entidad. `@CreatedDate` establece la fecha de creación del registro y `@LastModifiedDate` la última fecha de modificación.
- `@EnableJpaAuditing` se coloca en la clase que contiene la anotación `@SpringBootApplication` y el método `main` de proyecto para habilitar los componentes de auditorías.

Con fines de demostración se ha implementado la interfaz `CommandLineRunner` con el método `run` para crear un `Usuario` y almacenarlo en la base de datos llamado el método `userDao.save` bajo el esquema de Spring Data JPA, arrojando como resultado lo siguiente:



The screenshot displays a database management interface. On the left, the 'Navigator' pane shows a tree view of the 'alcala' database schema, with the 'usuario' table selected. Below it, the 'Information' pane provides details for the 'usuario' table, including its columns and their data types: 'id' (int(11) AI PK), 'apellido' (varchar(255)), 'email' (varchar(255)), 'nombre' (varchar(255)), and 'telefono' (varchar(255)).

The main window shows a query editor with the SQL statement: `SELECT * FROM alcala.usuario;`. Below the editor, the 'Result Grid' displays the following data:

id	apellido	email	fecha_creacion	fecha_modificacion	nombre	telefono
1	Ramirez	santiago.ramirez@edu.uah.es	NULL	NULL	Santiago	999999999
2	Avbar	juan.avbar@test.es	NULL	NULL	Juan	999999999
3	Prueba	test@test.com	2020-05-17 09:30:35	2020-05-17 09:30:35	Prueba	669669699

Ilustración 18- Resultado de la consulta realizada a tabla usuario

2.14. Aplicaciones Web con Spring Boot

2.14.1. Arquitectura MVC en Spring

El principal objetivo de Spring desde sus inicios ha sido facilitar el desarrollo de aplicaciones web empresariales y para ello ha utilizado Spring Web MVC como módulo principal, encargado de dar soporte a la arquitectura modelo vista controlador y gestionando las peticiones que se realizan a la aplicación a través de la clase DispatcherServlet.

La arquitectura modelo vista controlador corresponde a la siguiente ilustración:

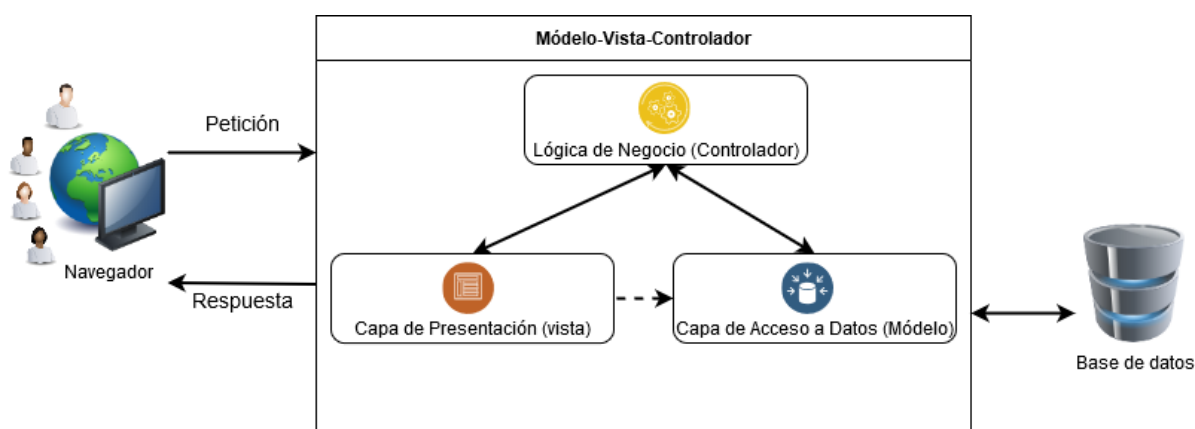


Ilustración 19-Arquitectura modelo vista controlador

En la ilustración se puede apreciar las tres capas:

- Acceso a Datos o Modelo: Esta capa contiene entidades que representan la información que maneja el sistema y todos los componentes relacionados el acceso y persistencia de datos.
- Vista o Presentación: Contiene los componentes que forman la interfaz de usuario del sistema y su interacción con los usuarios.
- Controlador: Esta capa trabajo actúa como intermediaria entre la capa de modelo y la vista. Posee todos los componentes utilizados para crear, transformar y mostrar los datos del sistema.

Esta arquitectura fue diseñada para desacoplar los componentes de una aplicación en capas que interactúen entre sí, de manera que se pueda cambiar la implementación de una de sus capas sin afectar a las demás.

Adicional a este modelo, Spring Web MVC utiliza el patrón Front Controller para el manejo de las peticiones. Este patrón establece que las solicitudes realizadas a la aplicación se gestionen desde un único punto de acceso a través de un manejador (handler) como se muestra a continuación:

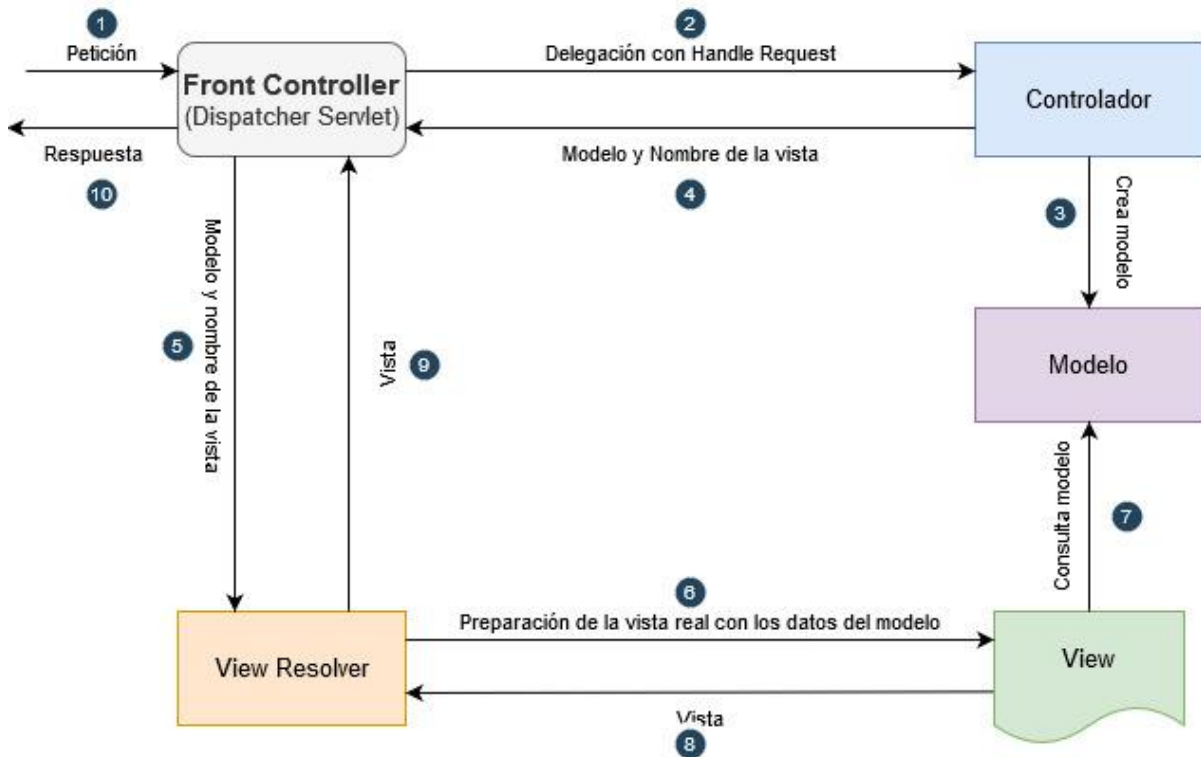


Ilustración 20-Patrón Front Controller en Spring MVC

Dónde:

- Las peticiones HTTP que recibe la aplicación son manejadas por el componente Front Controller, el cual en el caso de Spring posee el nombre de DispatcherServlet.
- El DispatcherServlet determina a que controlador y método debe delegar la petición a través de la interfaz HandlerMapping analizando la URL a la que se ha realizado la petición.
- Se ejecuta el método del controlador determinado que procesa la petición y devuelve un resultado al DispatcherServlet que contiene el nombre de la vista que se debe cargar y el mapa de objeto que ésta utiliza.
- Luego el DispatcherServlet utiliza las interfaces View y ViewResolver definida en el proyecto para enlazar el nombre de la vista con las vistas reales y luego gestionar la preparación de los datos obtenidos por el controlador para pasarlo a la vista ya enlazada.
- Por último, el DispatcherServlet, arroja retorna una respuesta que contiene la vista con los datos cargados.

Este proceso se realiza por cada llamada que se haga a un recurso web que posea un controlador y una vista dentro de la aplicación.



2.14.1.1. Spring Boot MVC Auto-Configuration

Spring Boot proporciona toda la configuración automática necesaria para que crear las aplicaciones web con el módulo de Spring MVC. Esta utiliza los valores por defecto para inicializar los componentes utilizados y se puede personalizar en cualquier momento de acuerdo con las necesidades del sistema. Esta personalización generalmente se realiza a través de propiedades que se colocan en el fichero `application.properties`.

Entre los componentes que se configuran por defecto están:

- Soporte a contenido estático: Spring Boot por defecto expone como recurso todo el contenido estático que encuentra en las carpetas `/static`, `/public` o `/resources` del proyecto. Este contenido puede estar formado por ficheros HTML, CSS, JavaScript, imágenes, videos y cualquier otro fichero. Esta configuración por defecto se puede personalizar a través de las propiedades `spring.mvc.static-path-pattern` o `spring.resources.static-locations`.
- Soporte a formato JSON por medio de la librería Jackson y el conversor `HttpMessageConverters` para tratar el formato JSON como predestinado. También proveen serializadores y deserializadores para JSON por defecto que se pueden personalizar extendiendo de las clases abstractas `JsonSerializer<T>` y `JsonDeserializer<T>`.
- Muestra de errores por medio de páginas en blanco y sin etiquetas, las cuales se puede personalizar creando un fichero de la vista en la siguiente ruta dentro del proyecto: `src/main/resources/public/error/location`.
- Configuración por defecto para los motores de plantillas a través de starter. Spring Boot posee starter para FreeMarker, Groovy Templates, Thymeleaf y Moustache.

2.14.1.2. Spring Boot Web Starter

Para crear una aplicación web con Spring Boot, se debe añadir la dependencia de Spring Boot Web Starter al proyecto, la cual posee una agrupación de las dependencias utilizadas en Spring para el desarrollo web.

Está compuesta por los siguientes módulos:

- `spring-boot-starter`
- `spring-boot-starter-json`
- `spring-boot-starter-tomcat`
- `spring-boot-starter-validation`
- `spring-web`: Está compuesto por las características esenciales para crear la aplicación web ofreciendo un contexto de aplicación orientada a la web y el manejo de servlet y contenedor de Spring.
- `spring-webmvc`: Este módulo posee todos los componentes para implementar la arquitectura modelo vista controlador de Spring y crear los servicios REST.



Para añadir Spring Boot Web Starter a la aplicación, se debe colocar en el fichero Pom.xml lo siguiente:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Al añadir esta dependencia podemos utilizar todos los componentes que Spring posee para crear la aplicación web.

2.14.1.2.1. Añadiendo el modelo

En esta sección se utilizará la entidad y componentes creados y configurados en la sección anterior del presente trabajo que aborda la demostración del uso de Spring Data JPA como el componente de acceso a datos de la aplicación.

Para esta demostración se utilizará la entidad Usuario y la clase UsuarioDao que implementa la interfaz CrudRepository:

```
@Entity
@Table(name="usuario")
@EntityListeners(AuditingEntityListener.class)
public class Usuario {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name="nombre")
    private String nombre;

    @Column(name="apellido")
    private String apellido;

    @Column(name="email")
    private String email;

    @Column(name="telefono")
```



```

private String telefono;

@Column(name="fecha_creacion")
@CreatedDate
private Date fechaCreacion;

@Column(name = "fecha_modificacion")
@LastModifiedDate
private Date fechaModificacion;

public Usuario(String nombre, String apellido, String email, String telefono) {
    this.nombre = nombre;
    this.apellido = apellido;
    this.email = email;
    this.telefono = telefono;
}

public Usuario() {
}
}

import org.springframework.data.repository.CrudRepository;

public interface UserDao extends CrudRepository<Usuario, Long>{
}

```

2.14.1.2.2. Añadiendo la vista

La vista de una aplicación realizada con el módulo spring-web es gestionada a través de las implementaciones a la interfaz ViewResolver. Entre las implementaciones que Spring posee de esta interfaz están:

ViewResolver	Descripción
XmlViewResolver	Esta implementación de ViewResolver es utilizada para resolver o enlazar los nombres de las vistas definidas como beans de Spring en ficheros XML.



ResourceBundleViewResolver	Esta implementación se encarga de cargar las vistas a partir de un fichero de configuración “.properties” colocado en la raíz dentro del classpath.
UrlBasedViewResolver	Esta implementación es utilizada para dar soporte a la configuración y mapeo de las vistas físicas a partir del nombre de vista que devuelve el controlador. Entre sus métodos están: <ul style="list-style-type: none"> • SetPrefix: para añadir un prefijo a las rutas que debe conformar la vista física. • SetSuffix: para añadir la extensión o sufijo al nombre de vista retornado en el controlador. • SetViewClass: Establece la clase y tecnología que se encargará de la gestión de las vistas.
InternalResourceViewResolver	Es una subclase de UrlBasedViewResolver que ofrece soporte a la tecnología de la vista JavaServer Pages (JSP) y Servlets por medio de la clase InternalResourceView, la cual es una clase envoltorio de un JSP.
FreeMarkerViewResolver	Esta implementación extiende de UrlBasedViewResolver y ofrece soporte al motor de plantilla FreeMarker.
ContentNegotiatingViewResolver	Esta implementación se encarga de determinar el tipo de vista y ViewResolver que se debe utilizar de acuerdo al contenido solicitado. Para esto verifica en todos los ViewResolver si existe la vista solicitada desde el controlador y si el tipo contenido es el solicitado.

Por defecto, Spring utiliza el InternalResourceViewResolver como manejador de la vista. La cual soporta las páginas creadas con JSP.

Entre las tecnologías de las vistas y motores de plantillas a los que Spring ofrece soporte están:

- JSP y JSTL
- Thymeleaf
- Freemarker
- Apache Velocity
- Groovy Markup

Realizando un análisis de tendencia utilizando la plataforma Google Trends se puede apreciar el uso que han recibidos estas tecnologías durante los últimos cinco años:

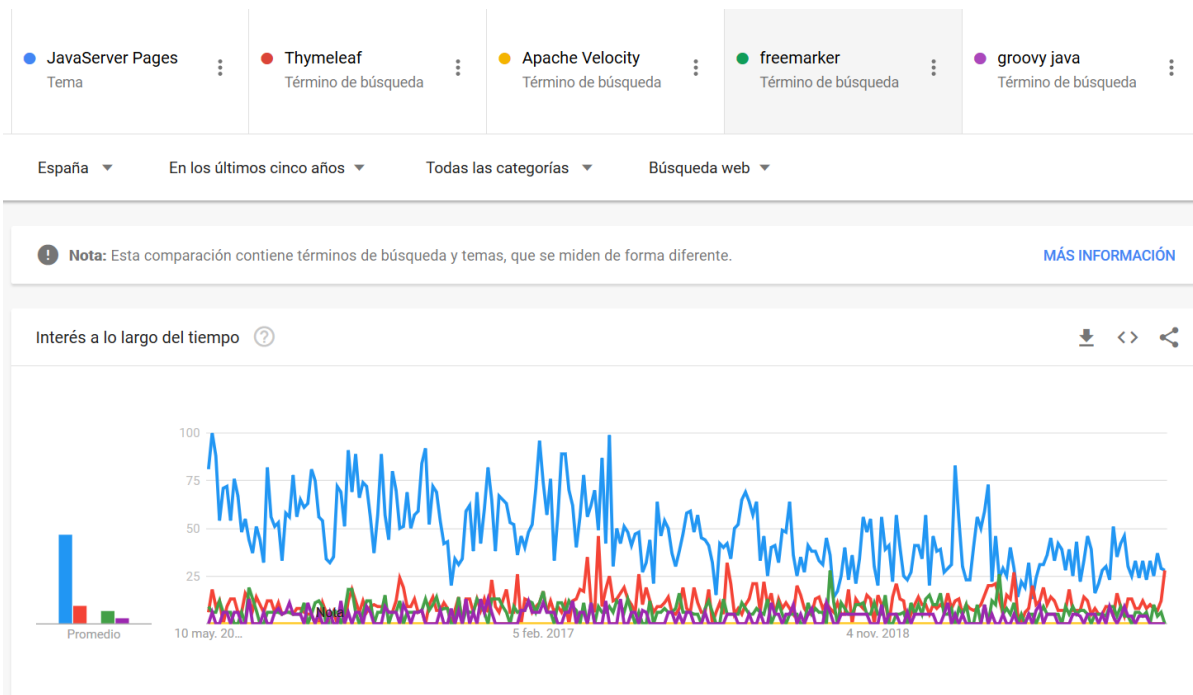


Ilustración 21-Comparación tecnologías de las vistas en Java

2.14.1.2.3. JSP Y JSTL

Java Server Pages (JSP) es una tecnología utilizada para desarrollo de las páginas web en Java y JavaServer Pages Library (JSTL) es un componente que posee una colección de etiquetas JSP útiles para ampliar el poder de JSP. Estos componentes han sido las tecnologías utilizadas por defecto para la creación de las páginas dinámicas (“la vista”) que componen una aplicación web desde los inicios de Java EE. Dicha tecnología se basadas en estructuras HTML y XML.

La dependencia spring-boot-starter-web incluye por defecto el contenedor de servlets Tomcat de forma embebida para ejecutar las aplicaciones sin necesidad de realizar configuración adicional pero dicho contenedor no incluye el manejo y renderización de JSP por defecto. Para ello es necesario añadir la dependencia llamada “tomcat-embed-jasper”, la cual proporciona el soporte necesario para los ficheros .JSP:

```
<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
  <scope>provided</scope>
</dependency>
```



Para utilizar JSTL también es necesario incluir la siguiente dependencia:

```
<dependency>
  <groupId>jstl</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>
```

Spring Boot Starter Web utiliza `InternalResourceViewResolver` como `ViewResolver` para soportar los `jsp`, por lo que a nivel de propiedades, se deben definir en el fichero `application.properties` el prefijo y sufijo que tendrán las vistas a ser resueltas:

```
spring.mvc.view.prefix=/WEB-INF/jsp/
spring.mvc.view.suffix=.jsp
```

Al definir estas propiedades se establece que todos los nombres de las vistas que se retornen desde el controlador se buscarán en la siguiente ruta dentro del proyecto: `/src/main/webapp/WEB-INF/jsp/` y que la extensión que tendrán los ficheros de las vistas serán `.jsp`.

Sin embargo, como se apreció en la sección mencionada anteriormente sobre la estructura de un proyecto Spring Boot, la carpeta `webapp` no existe. Para solucionar esto se existen varias alternativas:

- 1- Crear la carpeta `webapp` dentro de `main` y colocar dentro las subcarpetas y los ficheros.
- 2- Cambiar la ruta de búsqueda por defecto y colocar la que entendamos. Para ello debemos crear una clase que implemente la interfaz `WebServerFactoryCustomizer<TomcatServletWebServerFactory>` para personalizar la configuración que tenemos del servidor web o contenedor de servlets:

@Configuration

```
public class TomcatConfig implements
WebServerFactoryCustomizer<TomcatServletWebServerFactory>{
```

@Override

```
public void customize(TomcatServletWebServerFactory factory) {
  factory.addContextCustomizers(new TomcatContextCustomizer() {
```

@Override

```
public void customize(Context context) {
  try {
    context.setDocBase(new ClassPathResource("").getFile().getAbsolutePath());
```



```
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
  
    }  
});  
}  
}
```

Esta posee un método llamado `customize` que tiene como argumento una clase `Fabrica` del `Servlet` con su conjunto de configuraciones. Debido a que deseamos personalizar el contexto de búsqueda para los ficheros `jsp` se añade un nuevo contexto personalizado donde se determinará que la ruta base será la que se desee. Para fines de este trabajo la colocaremos en `/resources`.

Cabe destacar que el uso de `JSP` como tecnología para la vista ha ido disminuyendo debido a la popularidad que van adquiriendo los motores de plantillas como `Thymeleaf` y `Freemarker` y también a los nuevos `Framework` y lenguajes de programación orientados a la web que se basan en la segmentación de concepto de `back-end` y `front-end`.

2.14.1.2.4. Thymeleaf

`Thymeleaf` es un moderno motor de plantillas en `Java` que plantea que las plantillas `HTML` naturales se puedan previsualizar en un navegador haciendo doble clic, lo que es muy útil para el trabajo independiente en plantillas de interfaz de usuario (por ejemplo, por un diseñador) sin la necesidad de un servidor en ejecución. (Spring.io, 2020)

Este motor de plantilla cada día gana más popularidad dentro del entorno de desarrollo `Java` y `Spring Boot` gracias a que permite el desarrollo de plantillas de forma independiente y basada en el simple `HTML` que se ha venido utilizando desde los inicios de web. Entre otras ventajas que este ofrece están:

- Permite utilizar `Spring Expression Language (SpEL)`.
- Es más legible que otras tecnologías de las vistas como `JSP`.
- Se integra perfectamente con los módulos de `Spring Framework`.
- No es necesario ejecutar la aplicación para visualizar la página `HTML`.
- Posee un conjunto de librería para manejo de objetos.

`Spring Boot` provee un `starter` para dar soporte por completo a esta tecnología, solo se debe añadir la dependencia `"spring-boot-starter-thymeleaf"` y se realizará la autoconfiguración por defecto, la cual incluye como prefijo para las vistas `"/resources/templates/"` y `"html"` como sufijo:



```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

Un ejemplo de una vista creada con Thymeleaf es la siguiente:

```
<html xmlns:th="http://www.thymeleaf.org/">
<head><title>Hola Mundo!</title></head>
<body>
<h1>Hola Mundo</h1>
<h2 th:text="{cuerpo}"></h2>
</body>
</html>
```

Donde se coloca el atributo único de Thymeleaf (th) a utilizar con xmlns y la expresión th:text se pasa el valor de la variable cuerpo al encabezado <h2>.

Durante el desarrollo de este Trabajo de Fin de Máster se utilizará este motor de plantilla para dar solución a la vista gracias a todas las bondades descritas en los párrafos anteriores.

2.14.1.3. Añadiendo los controladores

Para definir un controlador en Spring Boot se coloca la anotación @Controller encima del nombre de la clase. Dicha anotación sirve para especificarle a Spring que se trata de un componente de tipo controlador donde se definirán los métodos utilizados para manejar las solicitudes HTTP que delegue el DispatcherServlet:

```
import org.springframework.stereotype.Controller;

@Controller
@RequestMapping("/evento")
public class EventoController {

}
```



Si se desea especificar a partir de que ruta dentro de la aplicación se podrán acceder a los métodos que este controlador posee se utiliza la anotación `@RequestMapping`, que permite establecer dicha ruta de acceso. Por ejemplo, si el servidor donde se ejecuta la aplicación fuera localhost, la ruta para acceder sería `http://localhost:8080/evento`.

Las clases controladores en Spring están compuestas por métodos que se convierten en los recursos web que posee la aplicación y pueden ser accedidos a través de una ruta única establecida en su definición. Estas son gestionadas por una implementación de la interfaz `HandlerMapping`, la cual controla las rutas de las peticiones para determinar a qué clase controlador corresponde.

Estos métodos también utilizan la anotación `@RequestMapping` para establecer el tipo de solicitud HTTP y la ruta que tendrá el recurso web que se desea crear a través de dicho método. La ruta especificada puede estar formada por expresiones regulares, por comodines y se puede combinar con parámetros.

Los atributos que puede contener esta anotación son:

Atributo	Descripción
name	Atributo para asignar un nombre al mapeo.
value	Establece la ruta de acceso al método.
path	Es un alias de Value. Posee la misma función.
method	Atributo para definir el tipo de petición HTTP recibirá el método.
params	Define la asignación de los parámetros que recibe el método.
headers	Establece y asigna el encabezado que recibe el método.
consumes	Establece el tipo de formato del contenido que acepta y recibe el método.
produces	Define el tipo de formato de datos que producirá el servicio.

A partir de la versión 4.3 de Spring Framework se añadieron anotaciones específicas para tipo de petición HTTP manejada por Spring para simplificar el proceso de mapeo de los métodos del controlador y sustituyendo con esto la anotación `@RequestMapping`:



Anotación	Descripción
@GetMapping	Es una anotación utilizada para manejar la asignación de peticiones HTTP GET al método dentro del controlador.
@PostMapping	Es utilizada para establecer que el método puede ser ejecutado solo a través de una petición HTTP POST.
@PutMapping	Utilizada para actualizar la información a través de una petición HTTP PUT.
@DeleteMapping	Anotación utilizada para definir que el tipo de petición que manejará el método es HTTP DELETE.
@PatchMapping	Utilizada para soportar las modificaciones parciales de información realizadas a través de HTTP PATCH.

Todas estas anotaciones son envoltorios de @RequestMapping, siendo @GetMapping y @PostMapping las más utilizadas. De manera que para la siguiente demostración se utilizará las anotaciones más comunes:

```
@GetMapping(value = "/ver")
public String verEvento() {
    return "ver";
}
```

```
@PostMapping(value = "/guardar")
public String guardar(Model modelo) {
    //llamadas a funciones para guardar
    return "index";
}
```

```
@RequestMapping(value = {"/home", "/index", "/"}, method = RequestMethod.GET,
headers = {"content-type=text/plain"})
public String home() {
    return "home";
}
```

```
@RequestMapping(value = "/listado")
public String listado() {
```



```
    return "listado";  
}
```

2.14.2. Servicios REST con Spring Boot

Dentro de Spring Boot existen diferentes maneras de crear un API REST, donde implementando un conjunto de anotaciones y unas pocas líneas de código se puede crear una aplicación que posea un gran número de servicios capaces de mostrar toda la informa sobre una base de datos.

La primera de estas maneras es utilizando Spring MVC, la cual permite crear un servicio REST a través de clases tipo controlador anotada con `@Controller` y utilizando las anotaciones `@RequestBody` y `@ResponseBody` en la definición de los métodos. A continuación, un ejemplo de cómo emplear estas anotaciones:

```
@Controller  
@RequestMapping("/api/usuario")  
public class UsuarioController {  
  
    @Autowired  
    private UsuarioDao usuarioDao;  
  
    @GetMapping(value = "/listado-rest")  
    public @ResponseBody ClienteList listarRest() {  
        return new usuarioDao.findAll();  
    }  
  
    @PostMapping(value = "/guardar", consumes =  
        {MediaType.APPLICATION_JSON_VALUE})  
    public Usuario crearUsuario(@RequestBody Usuario usuario) {  
        usuarioDao.save(usuario);  
        return usuario;  
    }  
}
```

La anotación `@RequestBody` convierte la información recibida en el cuerpo de la petición web en un objeto Java definido a través de la deserialización de dicho cuerpo. Esta información puede ser recibida en varios formatos, pero generalmente se utiliza un formato Json.



La anotación `@ResponseBody` le indica al controlador que el valor de retorno del método estará vinculado al cuerpo de respuesta web por medio de la serialización de dicho valor. Esta serialización se puede realizar en varios formatos, siendo el formato `Json` el más utilizado. Luego de que se serializa la respuesta esta es colocada en el objeto `HttpResponse` para finalizar el proceso de comunicación y transmisión de la petición.

A partir de la versión 4.0 de Spring Framework se introdujo la anotación `@RestController`, la cual define un tipo de componente que agrupa las anotaciones `@Controller` y `@ResponseBody` para que al crear los servicios no sea necesario colocar `@ResponseBody` en cada método.

`@RestController` se anota en la definición de la clase:

```
@RestController  
@RequestMapping("/api/rest/usuario")  
public class UsuarioRestController {  
  
    @Autowired  
    private UsuarioDao usuarioDao;  
  
    @GetMapping(value = "/listar")  
    public List<Usuario> listar() {  
        return usuarioDao.findAll();  
    }  
}
```

Tanto si se usa `@RestController` como `@Controller` si se desea establecer un servicio que retorne un fichero se deben utilizar los siguientes mecanismos:

Para la creación de un servicio que nos retorne un fichero Spring ofrece la clase `ResponseEntity`, la cual representa una entidad tanto de solicitud como respuesta HTTP, que posee encabezados, cuerpo y un código de estado de tipo `Enum` de la clase `HttpStatus`.

El primer paso para utilizar esta clase es definirla como tipo de respuesta en el método del controlador con el tipo de objeto que se quiere retornar:

```
@GetMapping(value = "/foto/ejemplo")  
public ResponseEntity<Resource> obtenerFotoEjemplo() throws
```



```
MalformedURLException {
```

```
    Path pathFoto = Paths.get("src//main//resources//static/uploads//ejemplo.png");
```

```
    Resource recurso = new UrlResource(pathFoto.toUri());
```

```
    return ResponseEntity.ok()
```

```
        .header()
```

```
            HttpHeaders.CONTENT_DISPOSITION,
```

```
            "attachment; filename=\"" + recurso.getFilename() + "\"")
```

```
        .body(recurso);
```

```
    }
```

Como se puede observar este método cumple la función de retornar una imagen de ejemplo alojada en una de las carpetas del proyecto. De modo que se puede evidenciar lo siguiente:

- `ResponseEntity<Resource>`: Permite definir el retorno del método como tipo `Resource` para retornar un fichero.
- `UrlResource`: Obtiene el fichero como un recurso que puede manejar Spring.
- `ResponseEntity.ok().header(...).body`: establece una respuesta con el código de estado 200, pasa la información de recurso por el header y por último pasa el recurso en el body.



2.15. Spring Data REST

Spring Data REST forma parte del proyecto de Spring Data y ofrece la creación automática de servicios REST basado en la exportación de los repositorios de datos como un recurso. Para implementarlo se debe añadir la dependencia “spring-boot-starter-data-rest” al proyecto y esta se encargará de buscar todos los repositorios existentes para luego proceder con la creación de los servicios.

Esta utiliza una de las siguientes estrategias para el descubrimiento de los repositorios:

Nombre	Descripción
DEFAULT	Expone todas las interfaces de repositorio encontradas. Solo toma en consideración si existe el atributo <code>exported=false</code> al utilizar la anotación <code>@RepositoryRestResource</code> .
ALL	Expone todos los repositorios independientemente de su visibilidad y anotación.
ANNOTATION	Solo se exponen los repositorios que poseen la anotación <code>@RepositoryRestResource</code> y no tengan seteada el atributo <code>exported</code> como <code>false</code> .
VISIBILITY	Se exponen solo los repositorios públicos.

Fuente (<https://docs.spring.io/spring-data/rest/docs/current/reference/html/reference>)

La anotación `@RepositoryRestResource` es opcional y se utiliza para personalizar los servicios REST a crear.

Spring Data REST se construye sobre los repositorios de Spring Data y los exporta automáticamente como recursos REST. Spring Data REST es en sí misma una aplicación Spring MVC y está diseñada de tal manera que debería integrarse con sus aplicaciones Spring MVC existentes con poco esfuerzo (Spring.io, 2020).

Luego de tener la dependencia añadida se debe definir una ruta base para los servicios en el fichero `properties`, configurando la siguiente propiedad:



`spring.data.rest.basePath=/ruta`

Al ejecutar el proyecto son creados los servicios, los cuales se pueden visualizar desde la raíz de la aplicación, donde se muestran los enlaces a cada una de las entidades disponibles en formato Json:

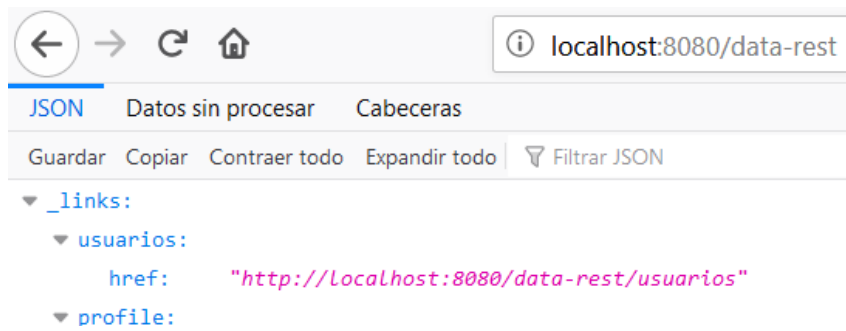


Ilustración 22-Ejemplo de Spring Data Rest



2.16. Asegurando la aplicación con Spring Security

La capa de seguridad es una de las características más importante que debe tener una aplicación web empresarial, esta capa está relacionada con las limitaciones de acceso que se les otorga a los usuarios, ya sea para que visualice todo el sistema o solo parte de este de acuerdo con una definición y asignación de roles.

Cuando se habla de la seguridad de una aplicación, existen dos términos que son claves que están presente en el proceso: La autenticación y la autorización.

La autenticación hace referencia a que mecanismo utiliza la aplicación web para verificar la identidad de quién está intentando acceder a un recurso en particular. Una forma común de autenticar a los usuarios es exigiéndole que ingrese un nombre de usuario y una contraseña. Una vez que se realiza la autenticación y se conoce la identidad, inicia el proceso de autorización (Spring.io, 2020).

La autorización protege los recursos de la aplicación para que solo puedan obtenerlos a quienes se les haya concedido los permisos previamente. Está se basa Roles y asignación de permiso a usuarios, donde los usuarios con más permisos suelen ser llamados administradores y pueden realizar un conjunto de funciones dentro del sistema que los demás tipos de usuario no pueden.

La seguridad provee a la aplicación de un conjunto de herramientas necesarias para encriptar contraseñas, manejar los diferentes tipos de autenticación y la protección ante diversos tipos de ataques contra el sistema. Por lo cual, Framework como Spring Security ofrece una gran ventaja para las aplicaciones web.

Spring Security forma parte del ecosistema de Spring y es considerado un Framework que complementa a los demás proyectos de Spring, ofreciendo un conjunto de servicios, componentes y configuraciones personalizables de seguridad para proteger las aplicaciones web y hacerlas seguras.

Spring Security proporciona un amplio conjunto de capacidades que se pueden agrupar en cuatro áreas de interés: autenticación, autorización de solicitudes web, autorización de llamadas a métodos y autorización de acceso a objetos de dominio individuales (Cosmina, 2020).

Entre las características encontradas en la plataforma web de Spring están:

- Soporte completo y extensible para autenticación y autorización



- Protección contra ataques como la fijación de sesiones, clickjacking, falsificación de solicitudes entre sitios, etc.
- Integración API Servlet
- Integración opcional con Spring Web MVC

Para tener las bondades de Spring Security dentro de una aplicación con Spring Boot hay que añadir la dependencia “spring-boot-starter-security”, la cual posee el conjunto de dependencia relacionada con Spring Security:

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-security</artifactId>  
</dependency>
```

Gracias a que este Framework funciona de manera autónoma, con solo incluir esta dependencia y sin realizar alguna configuración extra, la aplicación ya implementa una seguridad básica, donde antes de poder acceder a uno de los recursos que posee el proyecto en ejecución, aparecerá una ventana de inicio de sesión para autenticar al usuario.

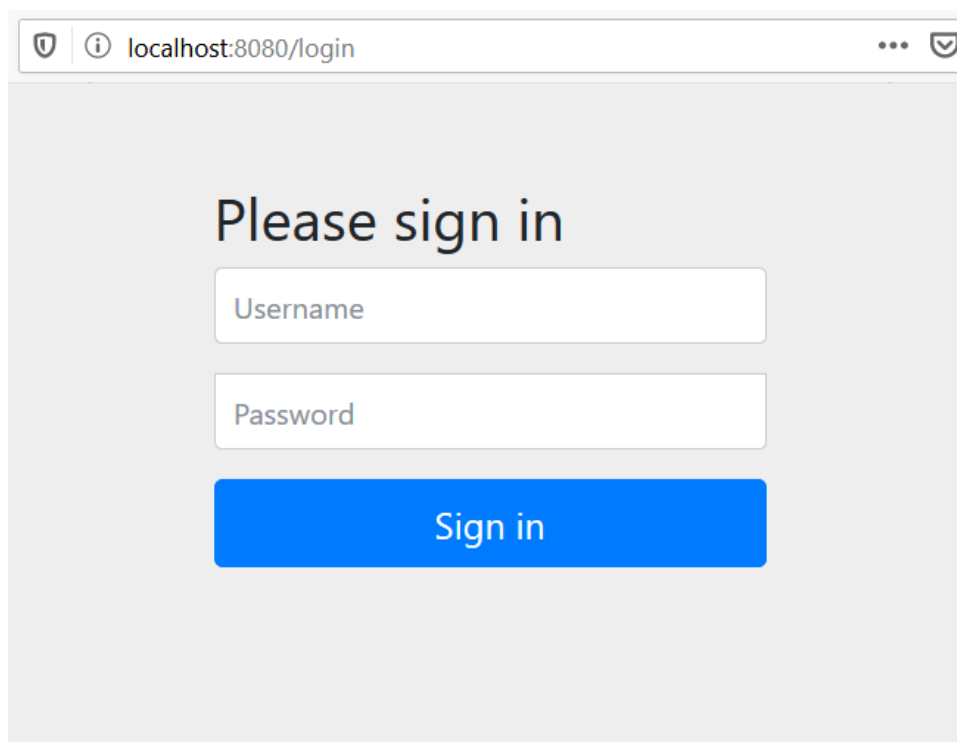


Ilustración 23-Ventana Login por defecto Spring Security

El usuario por defecto será “user“ y la contraseña se imprimirá en la consola:



```
Using generated security password: a9d7ca5c-eb90-49a7-bdbe-9030065c0770
```

Ilustración 24-Contraseña generada al ejecutar la aplicación

Si se desea cambiar la configuración por defecto con relación al usuario y contraseña solo se debe rellenar las siguientes propiedades en el fichero `application.properties`:

```
spring.security.user.name=administrador  
spring.security.user.password=4dm1nIst%d0R
```

Spring security posee una configuración por defecto que puede personalizarse implementando

Spring Boot implementó la configuración automática predeterminada de Spring Security en `SecurityAutoConfiguration`. Para cambiar la configuración de seguridad de la aplicación web predeterminada y proporcionar su propia configuración de seguridad personalizada, puede crear una clase de configuración que amplíe `WebSecurityConfigurerAdapter` y se anote con `@EnableWebSecurity`.

2.16.1. Open Web Application Security Project (OWASP)

Entre los estándares encontrados en la red de internet está el “Open Web Application Security Project” (OWASP) que en su traducción al español es “Proyecto abierto de seguridad de aplicaciones web”. Este proyecto tiene como objetivo principal determinar y combatir las causas que hacen que el software sea inseguro. Según su página web “www.owasp.org”, este proyecto es dirigido y gestionado por la fundación OWASP, la cual, inició su funcionamiento el 1 de diciembre de 2001, establecida como organización sin fines de lucro en los Estados Unidos.

OWASP es una comunidad abierta dedicada a permitir que las organizaciones conciban, desarrollen, adquieran, operen y mantengan aplicaciones confiables. Todas las herramientas, documentos, foros y capítulos de OWASP son gratuitos y están abiertos a cualquier persona interesada en mejorar la seguridad de la aplicación. Abogamos por acercarnos a la seguridad de las aplicaciones como un problema de personas, procesos y tecnología porque los enfoques más efectivos para la seguridad de las aplicaciones incluyen mejoras en todas estas áreas.

Esta comunidad ha publicado desde el año 2003 un documento llamado OWASP Top 10 que contiene una lista con los diez riesgos principales de seguridad que se deben mitigar en las aplicaciones web. Dicho documento se actualiza cada tres años, donde su última actualización ha sido en el año 2017 con los siguientes riesgos:



Riesgo	Definición	Mitigación
A1- Inyección	Es una vulnerabilidad relacionada con la inserción de un código SQL malicioso que afecta a la base de datos para provocar un comportamiento no deseado dentro del sistema.	-Validar las entradas realizadas por los campos de la vista y evitar los caracteres que puedan romper la sentencia. - Utilizar queries parametrizados.
A2- Pérdida de autenticación	Es una vulnerabilidad crítica debido a la suplantación de identidad de un usuario en la aplicación web.	-Manejo de sesiones. -Implementación de un proceso de autenticación. -Uso de cookies.
A3- Exposición a datos sensibles	Hace referencia a la falta de protección antes los atacantes sobre los datos sensibles y confidenciales que contiene una aplicación web.	-Establecer un cifrado para las credenciales. -Uso de Hash y Salt para encriptar las contraseñas.
A4- Entidades Externas XML (XXE)	Es una vulnerabilidad que provoca que la información confidencial de una aplicación web se almacene en una entidad externa a la aplicación.	-Deshabilitando el parseo de documentos basados en DTD.
A5 - Pérdida de Control de Acceso	Vulnerabilidad que permite a los atacantes realizar acciones dentro de la aplicación a las que no deberían tener acceso.	-Una correcta definición de roles. -Delimitación de acceso a recursos de acuerdo al rol. -Modulo para validar y autorizar.
A6 - Pobre Configuración de Seguridad.	Permite acceder a la configuración del sistema y obtener datos sensibles debido a una mala configuración de la aplicación, falta de actualización o por no cambiar la configuración por defecto.	-No utilizar cuentas por defecto para acceso al sistema y a la parte de administración. -Restringir el acceso público a la configuración de la aplicación. -Mantener el sistema y sus componentes actualizados.
A7- Cross Site Scripting (XSS)	Es un tipo de inyección similar al riesgo A1, donde se inyecta código JavaScript desde la vista para obtener	-Validación de datos y procesos. -Codificación de la data.



	datos sensibles sobre la aplicación y redireccionar a los usuarios a otros sitios web.	-Uso de políticas de seguridad para el contenido.
A8-Deserialización insegura	Vulnerabilidad donde el atacante ejecuta de forma remota un código que modifica el comportamiento del sistema a través de la manipulación de datos y estructuras serializadas.	-No permitir o cargar datos serializados no confiables. -Validación y manejo de excepciones y errores en el proceso de deserialización.
A9-Componentes con vulnerabilidades conocidas	Vulnerabilidad donde el atacante compromete el sistema debido al uso de librerías y Framework de terceros con fallas conocidas.	-Mantener actualizados los componentes de terceros usados en la aplicación. -Crear una estructura que envuelva el acceso a las características del componente de tercero utilizado.
A10-Monitoreo y registro insuficiente.	Debido a la falta de monitoreo de los eventos que transcurren en una aplicación y la poca información almacenada sobre estos, se puede pasar por alto fallos y violaciones de seguridad.	-Implementado una correcta estructura de monitoreo que permita auditar el funcionamiento del sistema.

Mitigación de brechas de seguridad a través de Spring Security.

Para personalizar la seguridad de una aplicación que implementa Spring Security existe una clase llamada `WebSecurityConfigurerAdapter` que contiene un conjunto de métodos encargados de proporcionar la configuración de seguridad por defecto para las aplicaciones; ésta puede ser extendida para mejorar o personalizar la seguridad.

Autenticación y Autorización.

Para mitigar las vulnerabilidades relacionadas con la autenticación de los usuarios y autorización de los recursos a los que pueden acceder dichos usuarios Spring Security permite sobrescribir el método `configure` de la clase `WebSecurityConfigurerAdapter` para adaptarlo a las necesidades y requisitos de seguridad exigidos, como se muestra a continuación:



```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
            .antMatchers("/", "/inicio").permitAll()
            .antMatchers("/admin*").hasRole("ADMIN")
            .anyRequest().authenticated()
            .and()
            .formLogin();
    }

    @Bean
    @Override
    public UserDetailsService userDetailsService() {
        PasswordEncoder encoder =
        PasswordEncoderFactories.createDelegatingPasswordEncoder();
        UserDetails user =
        User.withUsername("tfm").password("12345678").passwordEncoder(encoder::encode).role
        s("USER")
            .build();
        UserDetails admin =
        User.withUsername("admin").password("87654321").passwordEncoder(encoder::encode).r
        oles("ADMIN")
            .build();
        return new InMemoryUserDetailsManager(user,admin) ;
    }
}
```

La anotación `@EnableWebSecurity` que se encuentra al inicio de la clase `WebSecurityConfig` se utiliza para habilitar el soporte de seguridad web de Spring Security y proporcionar la integración de seguridad de la aplicación con Spring MVC.

El método `configure()` posee múltiples declaraciones para manejar diferentes aspectos de la seguridad, dentro de las que se encuentra la implementación que recibe como el argumento un objeto de tipo `HttpSecurity` que hace referencia a la configuración base de seguridad para las peticiones http que se realizan a la aplicación y obtención o restricción a los recursos web.



A través de la llamada al método `authorizeRequests().antMatchers()` se puede permitir o denegar el acceso a una ruta específica dentro de la aplicación; gracias a que retorna un objeto de tipo `ExpressionUrlAuthorizationConfigurer` encargado de interceptar las llamadas y determinar si se tienen los accesos necesarios para ver ese bloque.

Al realizar la llamada al método `.formLogin()`, Spring Security habilita el formulario de inicio de sesión por defecto para que se muestre al intentar acceder a los recursos protegidos de la aplicación.

El método `userDetailsService()` que retorna un objeto de tipo `UserDetailsService` es heredado de la clase `WebSecurityConfigurerAdapter` y permite definir las credenciales de usuarios y los roles de la aplicación para que se carguen en memoria cada vez que esta se ejecute. En dicho método se hace una llamada al codificador de contraseña para hacerla más segura ante ataques.

Ataques CSRF y protección con Spring Security.

Cross Site Request Forgery (CSRF), también conocido como XSRF es una de las vulnerabilidades listadas por OWASP y consiste en suplantar las acciones de los usuarios a través del uso de las sesiones autorizadas que dichos usuarios poseen sobre determinados sitios web de confianza.

Este tipo de ataque se realiza mientras un usuario se encuentra dentro de un sitio web de confianza donde ha iniciado sesión y decide visitar otro sitio web, el cual sin dicho usuario saberlo se ejecuta un script o aplicación maliciosa que realiza peticiones http a la dirección de confianza utilizando la ip, cookies y sesión del usuario para realizar una acción.

Para evitar este tipo de ataques por parte de las aplicaciones web a donde acceden los usuarios, se plantean las siguientes opciones:

- Utilizar el patrón de sincronizador de token, donde el servidor genera un token CSRF a partir del id de la sesión, el cual se coloca en los formularios HTML y al momento de enviar el formulario el servidor verifica si se ha proporcionado el token, en caso contrario rechaza la petición.
- Establecer un atributo `SameSite` en la cookie de sesión. Este atributo solo se puede acceder a través del dominio que lo generó y de esta manera se restringe la gestión de cookie por parte de aplicaciones y sitios web de terceros.



Para solventar esta vulnerabilidad Spring Security provee el método `csrf()` de la clase `HttpSecurity`, el cual aplica el patrón de sincronización de token a los formularios de la aplicación.

@Override

```
protected void configure(HttpSecurity http) throws Exception {  
    http  
        .authorizeRequests()  
        .antMatchers("/", "/inicio").permitAll()  
        .antMatchers("/admin*").hasRole("ADMIN")  
        .anyRequest().authenticated()  
        .and()  
        .formLogin()  
        .and()  
        .csrf();  
}
```



2.17. Principios y patrones de diseño utilizados por Spring

2.17.1. Principios SOLID

Los principios SOLID hacen referencia al acrónimo conformado por un conjunto de principios utilizados en la programación orientada a objetos que buscan que los desarrolladores escriban el código del sistema con gran calidad, con bajo acoplamiento, que se entienda por otros miembros del equipo y que pueda ser mantenido sin dificultad.

Existen también otras ventajas de utilizar estos principios, las cuales son:

- Mayor cohesión entre el código creado para realizar una funcionalidad. De manera que mientras más alta sea la cohesión, más sencilla será la depuración y mantenimiento.
- Reutilización de las funcionalidades creada sin la necesidad de ajustarla a cada bloque o modulo dentro de un sistema.
- Mayor robustez, permitiendo un manejo de errores, fallos y excepciones sencillo y eficiente.

Los cinco principios que componen dicho acrónimo son:

- Single Responsibility Principle (Principio de Responsabilidad Única).
- Open-Close Principle (Principio de Abierto-Cerrado)
- Liskov Substitution Principle (Principio de Sustitución de Liskov)
- Interface segregation (Segregación de Interfaces)
- Dependency Inversion (Inversión de dependencia)

Estos principios fueron mencionados por primera vez en el año 2000, en el artículo llamado Principios de diseño y patrones de diseño escrito por Robert C. Martin y más adelante Michael Feathers los denominó con el acrónimo de SOLID.

La base fundamental de los principios SOLID es hacer que el software desarrollado sea fácilmente adaptable y extensible, lo cual favorece en gran manera a los requisitos cambiantes que se presentan al utilizar metodologías ágiles como Scrum, Kanban, Lean y otros.

2.17.1.1. Principio de responsabilidad única (Single Responsibility Principle)

Como su nombre lo indica, este principio establece que las clases, componentes o servicios creados por los desarrolladores deben tener una responsabilidad única sin importar lo simple o compleja que sea la responsabilidad que posea. Implementar este principio ofrece los siguientes beneficios:

- Dividir el código en componentes legibles y mantenibles.
- Alta cohesión en el código al establecer que todo lo relacionado con una funcionalidad de una clase o bloque esté de acuerdo con su responsabilidad y evitando la alta dependencia entre las clases.



- Reducción de errores durante el desarrollo y posterior mantenimiento o actualización a las funcionalidades.
- Detectar fallos y errores de una manera más sencilla y focalizada a partir del log de errores, sin la necesidad de recorrer todo el código de una aplicación.
- Mejor organización de las clases y los paquetes.

Este principio se basa en una técnica fundamental utilizada por la orientación a objetos llamada divide y vencerás que consiste en descomponer sistemas y componentes complejos en un conjunto de funcionalidades o componentes más pequeños que puedan ser desarrolladas y testadas de manera individual para luego ensamblarlas para conformar la aplicación.

Dicho de otro modo, las clases con más de una responsabilidad única deben dividirse en clases más pequeñas y eso se logra a través de la abstracción y delegación, donde la clase que contiene el conjunto de responsabilidades debe delegar una o más responsabilidades a otras clases, introduciendo orden y claridad al código.

Existen unos patrones de diseño que facilitan la aplicación del principio de responsabilidad única, entre estos está el patrón Decorador. El patrón decorador permite que la funcionalidad se divida entre clases con áreas únicas de interés, de manera que se encapsule en diferentes clases de decoradores.

Dentro del Framework Spring existen un gran número de clases que se adhieren al principio de responsabilidad única, las cuales forman los componentes que se utilizan a diario y definen la manera en que desarrollan las aplicaciones, entre estos están:

- La mayoría de las anotaciones utilizadas en Spring, incluyendo las anotaciones estereotipos mencionados anteriormente en este trabajo.

La división en capas que plantea Spring (Service, Repository y Controller) ayudan al cumplimiento de este principio.

2.17.1.2. Open-Close Principle (Principio de Abierto-Cerrado)

Este principio establece que un módulo, clase o función debe abrirse para extensiones, pero cerrarse para modificaciones. Es decir, cuando las funcionalidades necesitan cambiar, esos cambios no deberían afectar la implementación existente, sino que debe existir la manera en que se pueda extender desde ella para añadir la actualización.

Este principio fue definido por Bertrand Meyer en 1998 de la siguiente manera:

“Las entidades de software deben estar abiertas para la extensión, pero cerradas para la modificación.” (Meyer, 1998).

Y mejorado por Robert C. Martin en 2003 ofreciendo más detalles como se muestra a continuación:



“Abierto para la extensión: Esto significa que el comportamiento del módulo puede extenderse y a medida que cambian los requisitos de la aplicación, podemos ampliar el módulo con nuevos comportamientos que satisfagan esos cambios. En otras palabras, podemos cambiar lo que hace el módulo”. (Martín, 2003).

“Cerrado por modificación. Es decir, ampliar el comportamiento de un módulo no debe producir cambios en el código fuente del módulo existente, de manera que el bloque de código de origen debe permanecer intacto”. (Martín, 2003)

“El principio abierto / cerrado es una guía para el diseño general de clases e interfaces y cómo los desarrolladores pueden construir código que permita el cambio a lo largo del tiempo. Con cada sprint que pasa, los nuevos requisitos son inevitables y deben ser aceptados. Sin embargo, reconocer que el cambio es algo bueno es solo una parte de la respuesta. Si el código que ha producido hasta este punto no está diseñado para permitir el cambio, el cambio será difícil, lento, propenso a errores y costoso” (Hall, 2017).

Entre sus ventajas están:

- Lograr que el código desarrollado permitirá adaptarse de forma sencilla a los futuros cambios que pueden aparecer.
- El código se vuelve más legible y mantenible.
- Forma parte de las buenas prácticas definidas por el mercado.
- Evitar el impacto a otras funcionalidades y componente al momento de realizar un cambio.

Todo esto se logra a través de un bajo acoplamiento y haciendo uso de la herencia entre componentes, donde las clases y funcionalidades no poseen ese estrecho vínculo con otras partes del código, permitiendo que los nuevos cambios puedan extender del código existente y ampliar su comportamiento.

A través de la herencia se abstrae el diseño de las clases utilizando interfaces y clases abstractas que sean implementadas por clases concretas para evitar que la funcionalidad existente en las clases concretas se vea obligada a cambiar y de esta manera, simplemente extendiendo de la clase abstracta o implementado la interfaz se puede añadir el cambio deseado.

Los componentes que son desarrollados por el equipo del Framework Spring se adhieren al principio de abierto/cerrado, haciendo parte de su filosofía y diseñando sus módulos basados en este principio.

2.17.1.3. Liskov Substitution Principle (Principio de Sustitución de Liskov)

El principio de sustitución de Liskov fue definido por la informática Barbara Liskov, en el año 1988 y establece que:

“Si S es un subtipo de T, entonces los objetos de tipo T pueden reemplazarse por objetos de tipo S, sin interrumpir el programa” (Liskov, 1988).



Lo cual hace referencia específicamente a la herencia y al polimorfismo estableciendo que las instancias de las clases padres puedan reemplazar a las clases hijas sin afectar el funcionamiento de cualquier sistema que aplique la herencia entre clases.

Para abordar el concepto de la herencia entre clases de una manera sencilla dentro de los lenguajes de programación, se puede definir como la creación de un objeto basado en otro objeto o clase, donde el objeto creado hereda todas las propiedades, características y comportamiento de la clase padre de forma automática. Esta clase hija posee la capacidad de ampliar la funcionalidad existente en la clase padre.

El polimorfismo es la capacidad de un subtipo de ser tratado como si fuera una instancia de la clase padre (Hall, 2017). Esto quiere decir que en cualquier parte del código que se acepta un objeto de tipo padre, se aceptará el objeto de tipo hijo por igual sin la necesidad de convertir el objeto de tipo hijo a un tipo padre.

En la superficie, el principio de sustitución de Liskov es una de las facetas más complejas de los principios SÓLIDOS. Se requiere un conocimiento fundamental de los contratos y la variación para construir reglas que lo guíen hacia un código más adaptable.

Este ofrece las siguientes ventajas:

- Permite que el código desarrollado sea más adaptable.
- Al implementarlo se cumple con el principio abierto/cerrado y como el principio de responsabilidad única.
- Produce un sistema con una jerarquía de clases comprensible.
- Vuelve el código reutilizable.
- Facilita el desarrollo de pruebas unitarias.

Dentro de Spring Framework, la mayor parte del comportamiento definido en los componentes está basado y definido por clases abstractas, facilitando el cumplimiento con este principio.



2.17.1.4. Interface segregation (Segregación de Interfaces)

El principio de segregación de interfaces fue planteado por Robert C. Martin en el libro llamado “Agile Software Development, Principles, Patterns, and Practices” publicado en el año 2002 y establece que:

“Los clientes no deberían verse obligados a depender de métodos que no utilizan”. (Martin, 2002).

Donde el término “los clientes” dentro del lenguaje Java hace referencia a las implementaciones concretas de las interfaces dentro del sistema, dejando a entender que las interfaces deben ser simples, pequeñas y cumplir con una única responsabilidad.

De una manera más precisa, esto plantea que si existen clases hijas que implementen una interface y éstas no utilizan o implementen todos los métodos que posea dicha interface, la interfaz se debe segregar a través de la separación de las funcionalidades definidas, siendo colocada en otra interfaz que satisfaga todos los métodos de las clases que la implementen.

Otro de los planteamientos consiste en que al momento de desarrollar una nueva funcionalidad no se debe añadir nuevos métodos a una interfaz existente, sino más bien crear otra interfaz que sea implementada por la clase que la necesite.

Con esto se logra que una interfaz no esté llena de métodos que no son requeridos por las clases que la implementan y se evita que las implementaciones estén obligadas a contener métodos sobrescritos sin ninguna funcionalidad o vacíos.

Ventajas:

- Contribuye con el principio de responsabilidad única.
- Favorece la alta cohesión en el código.
- Evita que los cambios en las interfaces se apliquen a las implementaciones que tienen los métodos vacíos.
- Plantea un diseño arquitectónico legible y flexible.
- Siguiendo este principio nos aseguraremos que nuestros componentes son fácilmente escalables sin necesidad de impactar a componentes que no deberían.



2.17.1.5. Dependency Inversion (Inversión de dependencia)

El principio de inversión de dependencia fue publicado por Robert C. Martin en el año 1996 y establece que:

1. Las clases de alto nivel no deberían depender de las clases de bajo nivel. Ambos deberían depender de abstracciones.
2. Las abstracciones no deberían depender de los detalles. Los detalles deben depender de las abstracciones. (Martin, 1996).

La aplicación de estos enunciados hace que sea posible mantener los módulos del sistema desacoplados, evitando la estrecha dependencia que comúnmente se produce al enlazar varios componentes, sobre todo entre los módulos de alto y bajo nivel.

Cuando se habla sobre módulo y clase de alto nivel, se hace referencia a una clase abstracta o interfaz que definen el comportamiento que debe seguir ese módulo, mientras que una clase de bajo nivel se define como una implementación concreta de dicha interfaz o clase abstracta, la cual describe el comportamiento.

Este principio tiene como objetivo que el código desarrollado no dependa directamente de las clases de bajo nivel, sino que la dependencia se realice a las clases abstractas e interfaces.

Las ventajas resultantes de la implementación de este principio son:

- Un código menos acoplado.
- Un diseño de abstracciones robustas y reutilizables.
- Módulos más coherentes.
- Un código fácil de mantener y extensible.

Spring hace uso de este principio a través del patrón Inyección de Dependencia.

MICROSERVICIOS





3.1. Computación en la Nube

Hoy en día, la nube es uno de los términos más utilizados tanto por personas del área de la tecnología y relacionados a esta, así como por cualquier usuario que desea subir, enviar o publicar una foto a través de internet.

En principio, la nube era definida dentro de la era tecnológica como el internet, es decir, un conjunto de redes de comunicación interconectadas, la cual se puede apreciar en todos los diagramas donde está presente el internet con el símbolo de nube, pero con el paso del tiempo, el concepto de la nube se ha usado para referirse a la computación en la nube.

La computación en la nube es un modelo para permitir el acceso de red ubicuo, conveniente y bajo demanda a un grupo compartido de recursos informáticos configurables (por ejemplo, redes, servidores, almacenamiento, aplicaciones y servicios) que se pueden aprovisionar y liberar rápidamente con un mínimo esfuerzo de administración o interacción del proveedor de servicios. (NIST, 2011).

Este modelo ofrece una evolución a las tecnologías utilizadas en internet, ofreciendo servicios e infraestructura tecnológicas a través de este conjunto de redes de comunicación. Entre estos servicios se puede encontrar el almacenamiento, el correo electrónico, el procesamiento y analítica de datos, entre otros.

Esto ha traído consigo un cambio de paradigma en el manejo de la infraestructura informática, donde el uso de un modelo de arquitectura basado en el conjunto de equipos informáticos ubicados en las instalaciones de las empresas se convierte en un modelo donde todos los servicios e infraestructura son suministrados por proveedores de la nube sin necesidad de recurrir a los espacios físicos.

Existen diferentes tipos de nube, entre las que se destacan:

Tipo	Descripción
Pública	Está compuesta por todos los servicios ofrecidos por proveedores externos a través de la Internet pública, donde todo el mundo puede contratarlos.
Privada o Corporativa	Al igual que la anterior, ofrece servicios informáticos, pero solo a través de redes internas o internet donde solo tienen acceso un número limitados de usuarios.



Híbrida	Es una combinación de las dos anteriores, donde de acuerdo con un conjunto de factores, una parte de los servicios corresponden a una nube pública y la otra a una nube privada.
---------	--

Al igual que los tipos de nube, existen un conjunto de categorías dentro de la nube de acuerdo con la parte que se desea contratar y administrar. Estos son:

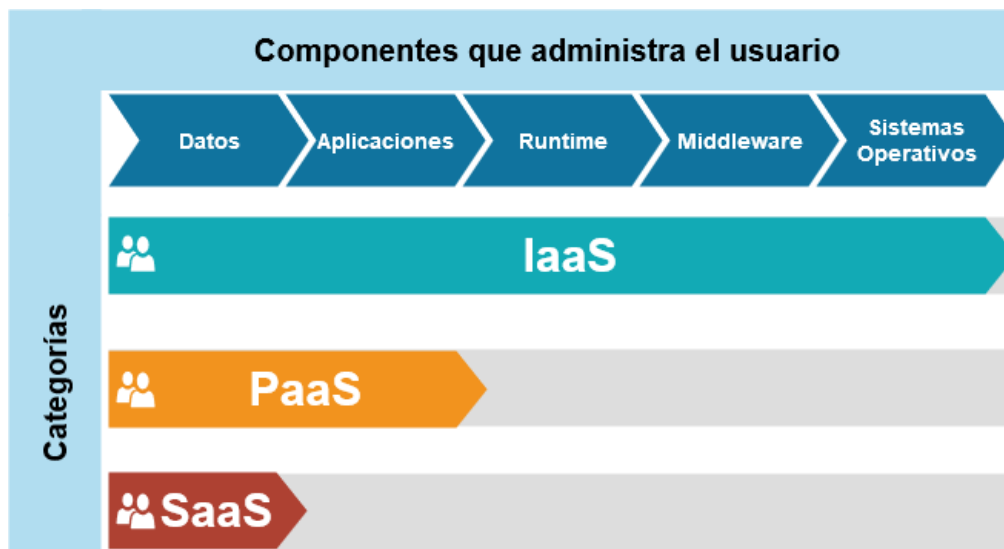


Ilustración 25-Categoría de infraestructura de la nube

Como se puede visualizar en la ilustración de acuerdo con el grado de administración que se desea tener están:

Categoría	Descripción
Infraestructura como servicio (IaaS)	Ofrece los recursos de hardware o infraestructura compuestos por servidores, equipos de hardware, sistemas de almacenamientos y otros equipos como un servicio.
Plataforma como Servicios (PaaS)	Ofrece todo un ambiente configurado de acuerdo a la demanda del cliente para desarrollo, pruebas, despliegue e instalación de aplicaciones. Esta plataforma viene con su sistema operativo ya configurado, donde dicho cliente solo administra las aplicaciones.
Software como Servicio (SaaS)	Ofrece toda una aplicación como un servicio, de manera todos los recursos necesarios para que el software funcione son suplidos por el



	proveedor y el cliente solo tiene que encargarse de la administración de dicha aplicación. En la actualidad existen mucho CRMs y ERPs bajo este modelo.
--	---

Beneficios de la computación en la nube

Gracias a los siguientes beneficios las empresas están dejando de lado la computación tradicional para migrar a la computación en la nube:

- Reducción de Costos.
- Disponibilidad.
- Recursos ilimitados.
- Cero mantenimiento.
- Seguridad.
- Seguridad de la información.
- Rapidez de aprovisionamiento y auto aprovisionamiento.
- Elasticidad y escalabilidad.



3.2. Microservicios

Un microservicio es un componente de software que posee funcionalidades específicas para realizar operaciones y cumplir con la lógica de negocios que componen una plataforma web empresarial. Estos forman parte de la aplicación, pero se ejecutan de manera independiente, desacoplada y autónoma, comunicándose con los demás componentes mediante operaciones de red.

Un microservicio debe aplicar el principio de responsabilidad única y tener un solo objetivo y propósito que cumplir. Debido a su pequeño tamaño y capacidad de trabajo, debe entenderse fácilmente y su característica más importante es la independencia de los otros servicios. Este debería poder funcionar solo, donde la implementación, el escalado y las pruebas deben separarse de otros servicios. De manera que puede ser reemplazado con facilidad. (Thönes, 2015)

Los microservicios surgieron como componentes de una aplicación web para facilitar el desarrollo de sistemas de software distribuidos, donde sus componentes se encuentran separados entre sí e interactúan a través de las redes de comunicación.

Desde el momento en que empezaron a crearse los microservicios y las empresas fueron implementándolos a través de la descomposición de funcionalidades y módulos de sistemas existentes, el término de arquitectura de microservicios tomó mucho auge, este se usó para denominar a los sistemas de software en donde todos los módulos que lo componen son microservicios.

La creación de esta arquitectura es gracias a que con los avances del desarrollo web, los sistemas monolíticos empezaron a representar un estancamiento, debido al aumento de la complejidad que suponía realizar una actualización, estos sistemas se volvían cada vez más difíciles de mantener y añadir nuevas funcionalidades.

Para ir solventando esta dificultad, se segmentaban los sistemas en distintos módulos, sin embargo, aunque estuvieran divididos, existía un gran acoplamiento entre estos y en muchos casos desarrollar un cambio específico afectaba a todo el software.

Los sistemas monolíticos son muy difíciles de mantener después de un largo período de cambios continuos. Ahora pueden dividir las características del monolito e implementar un microservicio para reemplazar la parte de la aplicación. Con la separación en servicios, es posible mejorar la escalabilidad. (Thönes, 2015).



Como evolución a la arquitectura monolítica, surgió la arquitectura orientada a servicio (SOA por sus siglas en inglés), la cual planteaba descomponer ciertas partes de los sistemas monolíticos en servicios bien definidos para reducir el tamaño de los sistemas monolíticos y que estos establecieran comunicación entre sí. Esto representó un alivio para los sistemas, sin embargo, los protocolos de comunicación entre los servicios produjeron varios problemas.

La arquitectura orientada a microservicios es planteada como una evolución de la arquitectura orientada al servicio (SOA), la cual consiste en el desarrollo de servicios independientes entre sí para cumplir con los requisitos del negocio que forman parte de una aplicación.

La diferencia más importante entre SOA y microservicios es el nivel al que apunta la arquitectura. Cuando SOA considera toda la empresa, los microservicios representan una arquitectura para un sistema individual en el nivel del proyecto. En SOA, el servicio creado es responsable de la gestión de otros servicios, mientras que una aplicación basada en microservicio, estos son independiente uno del otro y son organizados y gestionados por un componente externo a ellos, pero que pertenezca al ecosistema que conforma la aplicación. (Wolff E., 2006)

Entre estos componentes a los que hace mención Wolff, podemos encontrar el componente Spring Cloud Netflix Eureka que es capaz de registrar y mantener la comunicación entre microservicios que pertenezcan al mismo ecosistema.

Entre las ventajas que ofrece utilizar la arquitectura de microservicios están:

- **Desacoplamiento:** Al ejecutarse de manera independiente y poseer su propia estructura, estos no requieren que todo el sistema este cargado para realizar sus funciones.
- **Reducción de coste de cambio:** Los microservicios realizan funciones específicas de forma independiente, lo que facilita la realización de cambios sin afectar a los demás entes de la aplicación, al igual que no es necesario detener o volver a desplegar toda la aplicación, solo con desplegar el servicio funcionaría correctamente.
- **Diversidad de tecnologías:** Existe un mínimo de gestión centralizada de estos servicios, que pueden escribirse en diferentes lenguajes de programación y utilizar diferentes tecnologías de almacenamiento de datos (Fowler M. Lewis J, 2014).
- **Reutilización:** Existen muchos microservicios que ofrecen solución a problemas comunes en distintas aplicaciones facilitando su reutilización.
- **Tolerancia a fallos:** La mayoría de los microservicios implementan el patrón Circuit Breaker que les provee de una eficiente tolerancia a fallos.
- **Escalabilidad:** Gracias al desacoplamiento y la buena definición de estructuras que estos poseen es sencillo escalar elásticamente los microservicios.



3.3. Spring Cloud

Spring Cloud es un Framework para el desarrollo y administración de aplicaciones distribuidas en la nube. Este Framework proporciona un conjunto de herramientas para dar solución a los problemas comunes a los que se enfrentan los desarrolladores al trabajar en un entorno distribuido. En la actualidad la mayoría de los entornos distribuidos se basan en una arquitectura de microservicios.

Entre las ventajas que ofrece Spring Cloud están:

- Permite a los desarrolladores enfocarse en el desarrollo de la lógica de negocios y la creación de servicios.
- Modularidad y descomposición de todo el entorno para que al realizar cambios a un componente específico, los demás no se vean afectados.
- Implementación y mantenimiento de forma sencilla y flexible a través de ficheros de configuración.
- Escalabilidad de las aplicaciones.

Con Spring Cloud se puede desarrollar servicios y aplicaciones de una manera rápida, que sean capaces de establecer una gestión de configuración, descubrir servicios, realizar enrutamiento inteligente y establecer un componente intermedio o proxy para las peticiones, mantener sesiones distribuidas y otros componentes necesarios para el correcto funcionamiento de una plataforma en la nube.

Estos componentes funcionarán bien en cualquier entorno distribuido, incluida la computadora portátil del desarrollador, los centros de datos básicos y las plataformas administradas como Cloud Foundry (Spring.io, 2020).

En la web principal de Spring Cloud se mencionan las características que posee el Framework, como se visualiza a continuación:

- Configuración distribuida / versionada.
- Servicio de registro y descubrimiento.
- Enrutamiento.
- Llamadas de servicio a servicio.
- Balanceo de carga.
- Disyuntores.



- Cerraduras globales.
- Gestión de clúster.
- Mensajería distribuida.

Es importante mencionar que el primer conjunto de herramientas lanzadas para desarrollar las aplicaciones distribuidas basadas en una arquitectura de microservicio en Java fue desarrollado por el equipo de Netflix con el nombre de Netflix OSS, las cuales buscaban solucionar los problemas que presentaban los sistemas distribuidos a gran escala.

Poco tiempo después, Spring Framework desarrolló su conjunto de herramientas Cloud basándose en la integración de los componentes de Netflix OSS; donde trabajando en conjunto con el equipo de la compañía de Streaming se formó Spring-Cloud-Netflix como herramienta principal para el desarrollo en un entorno en la nube.

Spring Cloud está compuesto por veintitrés proyectos, de los cuales, para la realización de este trabajo se utilizarán los siguientes:

- Spring Cloud Config: Es usado para gestionar toda la configuración distribuida de los microservicios a través de un fichero de configuración.
- Spring Cloud Netflix Eureka: Es utilizado para gestionar el registro y estado de todos los microservicios que componen la aplicación.
- Spring Cloud Netflix Zuul: Sirve para tener un único punto de acceso a los componentes y microservicios que posee la aplicación.

Spring-Cloud-Netflix sigue siendo el proyecto más utilizado de Spring Cloud, sin embargo, a partir de la versión Greenwich Spring ha decidido solo dar mantenimiento y no añadir nuevas funciones a la mayoría de los componentes de Netflix con el objetivo de añadir los nuevos componentes de Spring que buscan sustituir el conjunto de componentes de Netflix OSS.

Es importante resaltar que en la actualidad, la mayoría de los componentes creados para reemplazar a Netflix no poseen un alto grado de popularidad y de uso.

3.3.1. Spring Cloud Netflix Eureka

Dentro de un sistema distribuido es necesario que los componentes que tengan algún tipo de intercambio de información con los demás componentes del entorno sepan la ubicación y como



comunicarse entre ellos. Lo cual resulta una tarea difícil para cada servicio cuando éstas ubicaciones se encuentran en constante cambio.

Spring Cloud Netflix Eureka se creó para resolver el problema antes mencionado, ofreciendo un servidor que mantenga el registro y la localización de los microservicios y los demás componentes existentes en el ecosistema. Además, es el encargado de ofrecer datos relevantes de cada uno de sus servicios como es el estado y cualquier metadato que se configure.

Este servidor sirve como administrador de direcciones, gestor de registro y descubrimiento de microservicios desplegados y eliminados, donde cada microservicio debe configurarse como un cliente de Eureka y enviar una petición solicitando el registro en dicho servidor al momento de arrancar.

Para mantener el registro y estado en el servidor de Eureka, cada microservicio debe mantenerse notificando al servidor, enviando una petición con sus datos cada treinta segundos o el tiempo configurado en Eureka. Esto se denomina “heartbeats” o latidos y si el servidor de Eureka deja de recibir estos heartbeats en un intervalo de tiempo determinado expulsa, cambia el estado o elimina esa instancia obsoleta. El intervalo de tiempo por defecto es noventa segundos.

Entre los beneficios que ofrece Eureka están:

- Localización de microservicios: permite que un microservicio pueda conectarse con otro de los componentes del ecosistema tan solo con un identificador y Eureka se encargará ofrecer la localización, puerto y cualquier otro metadato configurado.
- Clusterización: permite configurarse como un cluster en el despliegue de instancias según sea necesario.
- Estado de todo el entorno: permite conocer los estados de los componentes que forman el ecosistema ofreciendo una interfaz gráfica para ello.
- Agrupación: permite realizar agrupaciones de microservicios.

3.3.2. Spring Cloud Config

Una de las inquietudes con la que se encuentra el equipo de desarrollo al crear una aplicación dentro de un entorno distribuido es la de como configurar todos los componentes del ambiente de manera correcta, sencilla, centralizada y segura para evitar tener que crear y repetir la configuración en cada componente de forma manual o tener que ir uno por uno cuando se modifica dicha configuración.



Para dar respuesta a esta inquietud se creó la herramienta Spring Cloud Config, que proporciona un servidor de configuración, que almacena y administra todas las propiedades de configuración de los distintos ambientes y componentes que conforman la arquitectura basada en microservicios, y de esta manera se tiene un lugar central que los componentes deben llamar para obtener la configuración necesaria.

Este componente utiliza por defecto un repositorio git para administrar las propiedades de configuración, pero también, ofrece la opción de tener un fichero local o remoto desde el cual obtiene las propiedades.

Spring Cloud -Config se puede integrar con Netflix Eureka y en el momento en que arranca cada microservicio, se consultará el servidor de Eureka para obtener el servidor de configuración y por consiguiente las propiedades necesarias para arrancar correctamente.

3.3.3. Netflix Zuul

Zuul es un componente dentro del entorno distribuido que sirve como puerta de enlace entre la red externa y el conjunto de APIS internas que se tiene dentro del ecosistema. Este componente fue desarrollado por Netflix con el objetivo de añadir un conjunto de funciones de seguridad y enrutamiento a través de filtros para el inmenso volumen de peticiones que recibían a diario.

Zuul es la puerta de entrada para todas las solicitudes de dispositivos y sitios web al backend de la aplicación de transmisión Netflix. Como una aplicación de servicio perimetral, Zuul está diseñada para permitir enrutamiento dinámico, monitoreo, resistencia y seguridad (Zuul Wiki Github, 2020).

Entre las ventajas que ofrece Zuul según la documentación que se encuentra en la página oficial de Zuul en GitHub, están:

- Autenticación y seguridad: identifica los requisitos de autenticación para cada recurso y rechaza las solicitudes que no los satisfacen.
- Información y monitoreo: seguimiento de datos significativos y estadísticas generales para dar una visión precisa de las llamadas en producción.
- Enrutamiento dinámico: solicitudes de enrutamiento dinámicas a diferentes clústeres de back-end según sea necesario.
- Pruebas de estrés: aumenta gradualmente el tráfico a un clúster para medir el rendimiento.
- Load Shedding: asignando capacidad para cada tipo de solicitud y eliminando solicitudes que exceden el límite.



- Manejo de respuesta estática: crear algunas respuestas directamente en el borde en lugar de reenviarlas a un clúster interno.
- Resistencia Multiregion: solicitudes de enrutamiento en las regiones de AWS para diversificar el uso de (Elastic Load Balancer).

DESARROLLO DEL PROTOTIPO





4.1. Prototipo a desarrollar

El objetivo de este prototipo es implementar el contenido y las buenas prácticas aprendidas durante el master para el desarrollo de este trabajo de fin de master.

La aplicación a desarrollar durante todo el TFM consistirá en una plataforma web de ventas de boletos, artículos promocionales y ticket para eventos que contiene los siguientes módulos:

- Módulo de acceso.

Este módulo será el encargado de resolver todas las solicitudes de accesos que realizarán los usuarios dentro de la aplicación. En este se validan si las credenciales del usuario son correctas y se determina el rol que corresponde.

También se encarga del registro de usuario realizado tanto desde la parte pública como usuario final, como en el formulario interno al que solo pueden acceder los usuarios que posean el rol de administrador.

- Módulo de gestión de eventos.

Este módulo es el encargado de manejar los servicios para el listado, creación y modificación de eventos para el usuario administrador. Para el usuario final será el encargado de ofrecer los listados y los servicios para la gestión de los comentarios de cada evento.

En este módulo también se manejarán la creación, modificación y listado de artículos promocionales para ambos roles.

- Módulo de pago y factura.

Este módulo contendrá todos los servicios necesarios para el usuario puede realizar el pago de los productos y boletas seleccionadas.

También contendrá los servicios para la gestión del carrito de compra de cada usuario, ofreciendo la funcionalidad de añadir y eliminar productos.



- Módulo de reportes.

Este módulo será encargado de mostrar los diversos reportes que se pueden generar dentro de la aplicación. Solo estará disponible para los usuarios que posea un rol de administrador. Dentro de este módulo estará toda la estructura de los reportes.

- Módulo de notificaciones.

El módulo de notificaciones es el encargado tramitar y enviar a cada uno de los usuarios el mensaje específico de acuerdo a la acción o cambio de estado dentro de las entidades principales de la aplicación.



4.2. Historias de Usuario

No. 1	
Título	Registrar usuarios
Narrativa	Como un usuario final deseo registrarme en la aplicación para acceder al sistema.
Criterio de Aceptación	<ul style="list-style-type: none"> - Tener un formulario de registro dentro de la aplicación. - El formulario debe tener los campos de correo como nombre de usuario, nombre completo, contraseña y repite la contraseña. - El formulario debe tener un botón para enviar los campos rellenos. - Al enviar el formulario se debe mostrar el siguiente mensaje: “Datos recibidos, le enviaremos un mail para validar y activar la cuenta a la dirección de correo suministrada”. - Debe llegar un mail al correo suministrado con los datos necesario para activar la cuenta.

No. 2	
Título	Registrar usuario administrador
Narrativa	Como un usuario administrador quiero registrar otros usuarios para que accedan al sistema.
Criterio de Aceptación	<ul style="list-style-type: none"> - Tener un formulario de registro. - El formulario se debe acceder a través del menú de usuario. - El formulario debe tener los campos de correo como nombre de usuario, nombre completo. - El formulario debe tener un botón para enviar los campos rellenos. - Al enviar el formulario se debe mostrar el siguiente mensaje: “Datos recibidos, se enviará un mail para validar y completar la cuenta a la dirección de correo suministrada”. - Debe llegar un mail al correo suministrado en el formulario con los datos necesario para completar la cuenta.



No. 3	
Título	Iniciar sesión en la aplicación
Narrativa	Como un usuario final quiero acceder a la aplicación para ver mi perfil y administrar mis boletas.
Criterio de Aceptación	<ul style="list-style-type: none"> - Tener un formulario de inicio de sesión. - El formulario debe tener los campos de correo y contraseña. - El formulario debe tener un botón para enviar los campos rellenos. - Se debe mostrar un mensaje de “Credenciales Incorrectas” si los datos son incorrectos. - Se debe mostrar una ventana con información sobre el perfil del usuario y el conjunto de opciones si las credenciales suministradas son correctas.

No. 4	
Título	Cambiar de idioma en la aplicación
Narrativa	Como un usuario final quiero cambiar el idioma de la aplicación para ver el contenido en el idioma de mi preferencia.
Criterio de Aceptación	<ul style="list-style-type: none"> - Tener botones para los idiomas español e inglés en la parte superior de la ventana de la aplicación. - Al presionar uno de los botones, se debe cambiar el contenido de la aplicación al idioma correspondiente.

No. 5	
Título	Visualizar los eventos
Narrativa	Como un usuario final quiero visualizar los eventos próximos para ver los detalles y comprar una boleta.
Criterio de Aceptación	<ul style="list-style-type: none"> - Mostrar un slider con los eventos próximos en el centro superior de la ventana.



	- El componente de slider debe tener los botones hacia atrás y hacia adelante.
--	--

No. 6	
Título	Comentar y Visualizar los comentarios de los eventos
Narrativa	Como un usuario final que ha iniciado sesión quiero comentar y ver todos los comentarios que tiene un evento para compartir con los demás interesados.
Criterio de Aceptación	<ul style="list-style-type: none"> - Mostrar el bloque de comentarios en detalle de eventos. - Posicionarlo en la parte inferior del detalle. - Se debe tener un campo para colocar el comentario. - Se debe tener un botón para enviar el comentario. - El comentario enviado se debe colocar al final de la lista.

No. 7	
Título	Seleccionar boletas.
Narrativa	Como un usuario final que ha iniciado sesión quiero seleccionar las boletas que desee para asistir a un evento específico.
Criterio de Aceptación	<ul style="list-style-type: none"> - Mostrar botón para seleccionar boleta en la ventana que muestra el detalle del evento. - Al presionar el botón mostrar ventana para seleccionar cantidad de boletas y los asientos si el evento posee asientos enumerados. - Si el evento no posee asiento, solo mostrar campo cantidad de boletas. - La ventana debe tener para añadir las boletas al carrito. - Al presionar el botón se deben enviar los datos al carrito de compra.

No. 8	
Título	Carrito de compra
Narrativa	Como un usuario final que ha iniciado sesión quiero visualizar mi carrito de compra para proceder a comprar las boletas y producto seleccionados.



Criterio de Aceptación	<ul style="list-style-type: none"> - Mostrar un botón para acceder al carrito de compra. - El botón debe estar en la parte superior de la pantalla. - Al presionar el botón se debe mostrar un listado con todas las boletas y productos previamente añadidos por el usuario. - Debe tener un botón de continuar que envíe todos los productos en el carrito y muestre la ventana de pago.
------------------------	--

No. 9	
Título	Visualizar artículos promocionales
Narrativa	Como un usuario final quiero visualizar todos los artículos promocionales de temporada para visualizar su detalle.
Criterio de Aceptación	<ul style="list-style-type: none"> - Mostrar en el bloque un carrusel con todos los artículos promocionales. - Este componente de carrusel debe estar por debajo del componente que muestra los eventos. - Al presionar sobre el artículo promocional, se debe acceder a una ventana donde se muestra su detalle.

No. 10	
Título	Seleccionar artículos promocionales
Narrativa	Como un usuario final que ha iniciado sesión quiero añadir los artículos promocionales que desee para proceder a comprarlos.
Criterio de Aceptación	<ul style="list-style-type: none"> - En la ventana que muestra el detalle de los artículos promocionales, debe aparecer el botón de añadir al carrito. - Al presionar el botón se debe añadir el artículo al listado de producto del carrito de compra.

No. 11	
Título	Comprar los productos del carrito de compra



Narrativa	Como un usuario final que ha iniciado sesión quiero pagar las boletas y artículos que están en el carrito de compra para adquirirlos.
Criterio de Aceptación	<p>- Mostrar ventana de pago al presionar continuar en el carrito de compra.</p> <p>- La ventana de pago debe contener un formulario con los campos:</p> <ul style="list-style-type: none"> • Número de tarjeta de crédito. • Nombre del titular de la tarjeta. • Fecha de vencimiento. • CCV. <p>- El formulario debe tener un botón para enviar la información contenida en los campos anteriores.</p> <p>- Al presionar el botón enviar se debe enviar un mail con la información de los productos comprados en la aplicación.</p>

No. 12	
Título	Visualizar productos comprados.
Narrativa	Como un usuario final que ha iniciado sesión quiero visualizar las boletas y artículos comprados por la aplicación para tener constancia de las compras realizadas.
Criterio de Aceptación	<p>- Mostrar la opción de productos comprados dentro del menú de opciones del usuario.</p> <p>- Al presionar la opción se debe mostrar una ventana que contenga un listado con todos los artículos y boletas compradas.</p> <p>- Cada registro de dicho listado debe tener el nombre del producto, la cantidad, la fecha de compra, el monto total y una opción para acceder a su factura.</p>

No. 13	
Título	Recibir factura por mail.
Narrativa	Como un usuario final quiero recibir un mail que contenga la factura adjunta al momento de realizar una compra de boletas o artículos dentro de la aplicación para tener constancia de la transacción.



Criterio de Aceptación	<ul style="list-style-type: none"> - Al procesar los productos del carrito de compra debe llegar un correo con la información sobre los productos comprados. - En mail debe recibirlo el usuario que ha realizado la compra. - El mail debe contener la factura como un fichero adjunto.
------------------------	---

No. 14	
Título	Realizar pago por PayPal.
Narrativa	Como un usuario final quiero pagar los productos a comprar por la plataforma de pago PayPal para tener otro medio de pago.
Criterio de Aceptación	<ul style="list-style-type: none"> - Al procesar los productos del carrito de compra debe mostrarse una ventana que muestren la opción de pago con PayPal. - Al seleccionar esta opción, se debe abrir una ventana emergente proveniente de PayPal para que el usuario rellene los campos nombre de usuario y contraseña. - Se debe mostrar en todo momento el flujo de la plataforma de pago PayPal. - Al finalizar el proceso de pago con PayPal se debe retornar a la aplicación mostrando información sobre la compra realizada.

No. 15	
Título	Administrar locales para eventos.
Narrativa	Como un usuario administrado quiero añadir, modificar o eliminar locales donde se realizan los eventos para que al crear un evento se posea el detalle del local.
Criterio de Aceptación	<ul style="list-style-type: none"> - Se debe mostrar en el menú de administración, una opción para acceder a la gestión de locales. - Al presionar la opción se debe redirigir a una ventana que contenga el listado de locales registrados en la aplicación. - Cada registro debe tener la opción de editar los datos. - Al presionar el botón se debe mostrar un formulario con los siguientes campos en modo editable: <ul style="list-style-type: none"> • Nombre del local



	<ul style="list-style-type: none"> • Dirección • Radio que indique si tiene una cantidad de personas permitidas. • Cantidad máxima de personas en caso de tener un máximo. <p>- El formulario debe tener un botón para guardar los cambios.</p> <p>- Al presionar el botón se deben almacenar los cambios en la base de datos y redirigir al listado de locales.</p> <p>- En la parte superior del listado debe estar un botón para añadir un nuevo local.</p> <p>- Al presionar este botón se desplegará el mismo formulario que en editar, pero sin los campos rellenos.</p> <p>- Al completar el formulario se debe mostrar un mensaje que diga “Local registrado correctamente”</p> <p>- Se debe redirigir al usuario a la ventana con el listado de locales.</p>
--	--

No. 16	
Título	Administración de eventos.
Narrativa	Como un usuario administrador quiero añadir, modificar y eliminar eventos para que los usuarios finales los puedan visualizar y adquirir boletas.
Criterio de Aceptación	<ul style="list-style-type: none"> - El menú de administración debe contener la opción de gestión de eventos. - Al presionar esa opción se debe desplegar una ventana que muestre el listado de eventos que ha sido creado por dicho administrador. - El listado debe tener la opción de modificar e inhabilitar los eventos. - Al presionar la opción modificar, se despliega una ventana con la información general del evento en un formulario en modo de edición. - El formulario debe tener el botón de guardar. - Al presionar el botón de guardar, se debe actualizar la información del evento. - En la parte superior del listado se debe mostrar un botón para añadir nuevos eventos. - Al presionar debe aparecer un formulario con los siguientes campos: <ul style="list-style-type: none"> • Nombre del evento • Descripción



	<ul style="list-style-type: none"> • Fecha • Fecha límite de venta de boletas • Cargado de imágenes <p>- El formulario debe tener la opción de continuar, donde se le presenta al usuario las siguientes opciones:</p> <ul style="list-style-type: none"> • Campo de tipo radio para determinar si el evento se realizará con asientos asignados. • Campo para colocar las restricciones o requisitos que tendrá el evento. • Si se ha marcado que el evento se realizará con asientos asignados, se desplegará un mapa de acuerdo al local elegido para realizar el evento. • En el mapa se podrá seleccionar los asientos no disponibles. <p>- El formulario tendrá la opción de guardar.</p> <p>- Al presionar el botón de guardar, se almacenará en la base de datos y se mostrará al usuario un mensaje expresando que el evento ha sido creado.</p> <p>- Al crear el evento se debe redirigir a la ventana de listado de eventos.</p>
--	--

No. 17	
Titulo	Administrar artículos promocionales.
Narrativa	Como un usuario administrado quiero añadir, modificar o eliminar artículos promocionales para ofrecer a los usuarios finales artículos relacionados con los eventos.
Criterio de Aceptación	<p>- Se debe mostrar en el menú de administración, una opción para acceder a la gestión de artículos.</p> <p>- Al presionar la opción se debe redirigir a una ventana que contenga el listado de artículos registrados en la aplicación.</p> <p>- El listado mostrará la información sobre el artículo y si hace referencia a un evento específico.</p> <p>- Cada registro debe tener la opción de editar los datos.</p> <p>- Al presionar el botón se debe mostrar un formulario con los siguientes campos en modo editable:</p> <ul style="list-style-type: none"> • Nombre del artículo. • Cantidad disponible del artículo. • Imágenes del artículo. • Descripción



	<ul style="list-style-type: none"> • Combo para seleccionar si pertenece a un evento en específico. - El formulario debe tener un botón para guardar los cambios. - Al presionar el botón se deben almacenar los cambios en la base de datos y redirigir al listado de artículos promocionales. - En la parte superior del listado debe aparecer un botón para añadir un nuevo artículo. - Al presionar este botón se desplegará el mismo formulario que en editar, pero sin los campos rellenos. - Al completar el formulario se debe mostrar un mensaje que diga “Artículo registrado correctamente” - Se debe redirigir al usuario a la ventana con el listado de artículos.
--	--

No. 18	
Titulo	Cerrar Sesión
Narrativa	Como un usuario final quiero cerrar sesión dentro del sistema para tener control sobre mi actividad dentro de la aplicación.
Criterio de Aceptación	<ul style="list-style-type: none"> - Se debe mostrar un botón en la parte superior de aplicación para cerrar sesión. - Al presionar el botón se debe mostrar una ventana emergente para confirmar si desea cerrar sesión. - Si el usuario confirma, se cierra la sesión dentro del sistema y se redirige a la página principal. - Si el usuario cancela el cierre de sesión, permanecerá en la página actual.

No. 19	
Titulo	Reportes
Narrativa	Como un usuario administrado quiero generar reporte sobre los eventos, artículos y usuarios.
Criterio de Aceptación	<ul style="list-style-type: none"> - Se debe mostrar en el menú de administración, una opción para acceder a los reportes.



	<p>- Al presionar la opción se debe redirigir a una ventana que contenga las siguientes opciones de reporte:</p> <ul style="list-style-type: none"> • Reporte de eventos y venta de boletas • Reporte sobre venta de artículos • Reporte de usuarios. <p>- Cada uno de estos reportes debe tener un botón para generar la información y mostrarla al usuario.</p> <p>- Al presionar algunos de estos reportes se debe desplegar las opciones disponibles para cada reporte, como se muestra a continuación:</p> <ul style="list-style-type: none"> • Para el reporte de Eventos y venta de boletas se mostrará el listado de eventos para que el usuario escoja y se genere el reporte presionando el botón generar. • Para el reporte sobre venta de artículos, se mostrará un calendario donde se seleccione la fecha desde y hasta. Estas fechas serán las utilizadas para la generación del reporte. • Para el reporte de usuarios, se mostrará toda la información relevante de los usuarios.
--	--

No. 20	
Titulo	Notificaciones
Narrativa	Como un usuario final quiero recibir notificaciones sobre las actualizaciones que se producen a los eventos y sobre mi interacción con la aplicación web para estar al tanto de las novedades.
Criterio de Aceptación	<p>- Se debe mostrar las notificaciones en un recuadro en la parte superior de la aplicación.</p> <p>- Este recuadrado aparecerá al presionar un botón que debe tener un icono que haga referencia a notificaciones.</p> <p>- Las notificaciones a tomar en consideración dentro de la aplicación son las siguientes:</p> <ul style="list-style-type: none"> • Al momento de realizar una compra de boleta o artículo. • Al momento que se relaciones un artículo promocional con un evento específico en el cuál haya comprado una boleta. • Cuando se produzcan cambio de estado en los eventos.



4.3. Requerimientos no funcionales

Para el desarrollo de este prototipo se han especificado los siguientes requerimientos no funcionales:

Requisito No Funcional No. 1	
Nombre	Diseño responsivo
Descripción	-Las vistas en el sistema se desarrollará bajo un diseño responsivo, que permita adaptar la apariencia a los diferentes dispositivos.
Prioridad	Alta
Dependencia	No aplica

Requisito No Funcional No. 2	
Nombre	Carga de usuario
Descripción	-El sistema debe trabajar adecuadamente con más de 1000 sesiones de usuarios de manera simultánea.
Prioridad	Alta
Dependencia	No aplica

Requisito No Funcional No. 3	
Nombre	Seguridad
Descripción	-El sistema debe tener medidas de seguridad para: <ul style="list-style-type: none">• Gestión de contraseñas• Conexión entre microservicios• Gestión de las brechas de seguridad planteadas en OWASP.
Prioridad	Alta
Dependencia	No aplica

4.4. Diseño Tecnológico del prototipo

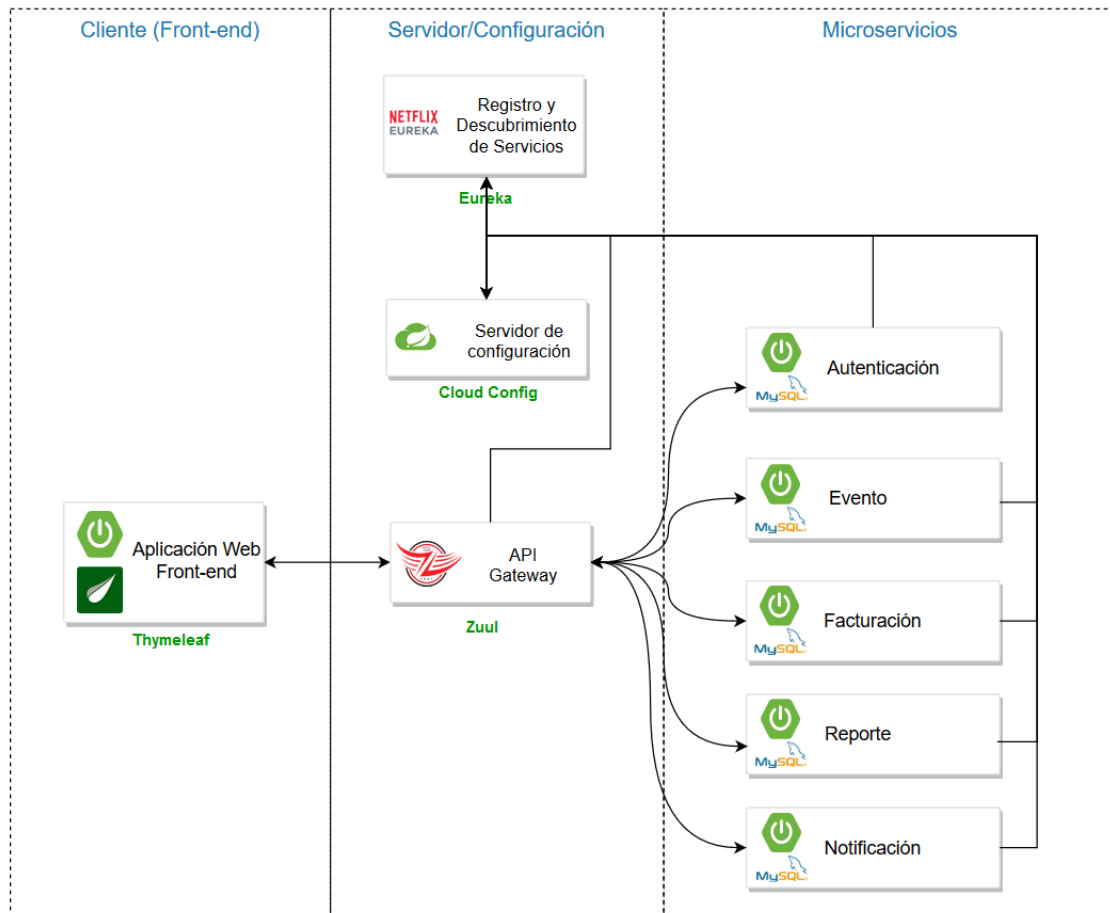


Ilustración 26 - Infraestructura Tecnológica

El proyecto estará dividido a nivel técnico de la siguiente manera:

4.4.1. Front-End

Para la parte cliente (front-end) se utilizará el motor de plantilla Thymeleaf.

Este se integra perfectamente con Spring Boot utilizando la dependencia de Spring Web y Thymeleaf. Con dicho motor de plantilla se crearán todas las vistas que corresponden a la aplicación y se realizará el llamado al API formada por la arquitectura de microservicios (back-end) para poblar las vistas con la información recibida.



También se utilizarán un conjunto de librerías y Framework que son compatibles con Thymeleaf como:

- Framework
 - Bootstrap
- Librerías JavaScript
 - JQuery
 - SweetAlert
 - Moment
 - Polyfill
 - Daterangepicker
 - Fileinput
- Plantillas
 - AdminLTE Bootstrap

4.4.2. Servidores de configuración

Para la parte de configuración se utilizarán las siguientes herramientas:

- Netflix Zuul: Este componente de un entorno distribuido se utilizará como puerta de enlace entre la aplicación front-end desarrollada con Spring Boot-Thymeleaf y el conjunto de API ofrecidas por los microservicios.
- Spring Cloud Config: Se utilizará como servidor de configuración para que cada microservicio pueda obtener la configuración necesaria para su arranque y correcto funcionamiento.

Spring Cloud Netflix Eureka: Se configurará dentro del entorno el servidor de registro y descubrimiento para que los microservicios realicen la comunicación entre ellos de una manera sencilla y rápida.

4.4.3. Microservicios

En la capa de microservicios se creará uno para dar respuesta a cada módulo que compone la aplicación. Estos servicios ofrecerán el API que se consultará desde el aplicativo Front-End, teniendo como ruta base el nombre del dominio y `/api/"versión"`, donde el parámetro versión corresponde a la versión desplegada en el API manager.

Por ejemplo: `/api/1.0.0/"nombre-base-recurso"`



4.4.3.1. Autenticación

Este microservicio contendrá el api para realizar toda la gestión de accesos a la aplicación.

Las rutas de servicios RESTFul que ofrecerá este microservicio son:

GET

/login	Recurso que retorna un usuario a partir de la verificación de un token.
/users	Retorna el listado de usuarios.
/users/@id:[0-9]+	Retorna el usuario especificado en el id.

POST

/token	Recurso para obtener un token de acceso a la aplicación a partir de las credenciales suministradas en el body de la petición
/users	Recurso para la crear un nuevo usuario.

PATCH

/users/@id:[0-9]+	Recurso para actualizar información sobre el usuario.
-------------------	---

4.4.3.2. Gestión de eventos

En este microservicio se tendrá toda la lógica de negocio relacionada con los eventos tanto para los administradores como los usuarios finales. Así como la gestión de inventario.

Las rutas de servicios RESTFul que ofrecerá este microservicio son:



GET

/events	Devuelve el listado de eventos.
/events/@id:[0-9]+	Devuelve el evento especificado en el id.
/events/@id:[0-9]+/comments	Devuelve todos los comentarios de un evento.
/events/@id:[0-9]+/comments/@id2:[0-9]+	Devuelve el comentario especificado en el id del evento específico.
/events/@id:[0-9]+/products	Devuelve el listado de productos de un evento.
/events/@id:[0-9]+/products/@id2:[0-9]+	Retorna el producto especificado en el campo id2 del evento especificado en el campo id.
/events/@id:[0-9]+/images	Devuelve el listado de imágenes de un evento.
/events/@id:[0-9]+/images/@id2:[0-9]+	Retorna la imagen del evento específico.
/events/@id:[0-9]+/places	Devuelve el listado de lugares para ese evento.
/events/@id:[0-9]+/places/@id2:[0-9]+	Devuelve un lugar específico del evento colocado en el campo id.
/events/@id:[0-9]+/places/@id2:[0-9]+/seats	Devuelve el mapa de asiento de un local.
/products	Retorna el listado de productos promocionales.
/products/@id:[0-9]+	Retorna el artículo promocional especificado.
/products/@id:[0-9]+/comments	Retorna el listado de comentarios de un artículo promocional.
/products/@id:[0-9]+/comments/@id2:[0-9]+	Devuelve el comentario especificado en el campo id2 del artículo promocional especificado en el campo id.
/products/@id:[0-9]+/images	Devuelve el listado de imágenes de un producto.
/products/@id:[0-9]+/images/@id2:[0-9]+	Retorna la imagen del producto específico.



POST

/events	Recurso que servirá para crear un nuevo evento.
/events/@id:[0-9]+/images	Recurso para almacenar la imagen de un evento.
/events/@id:[0-9]+/comments	Recurso para almacenar un comentario de un evento.
/events/@id:[0-9]+/places/@id2:[0-9]+/seats	Recurso para almacenar el mapa de asientos de un evento.
/products	Recurso para crear un nuevo producto.
/products/@id:[0-9]+/comments	Recurso para almacenar un comentario de un producto.
/products/@id:[0-9]+/images	Recurso para almacenar la imagen de un producto.

PATCH

/events/@id:[0-9]+	Recurso que actualiza información y estado sobre el evento.
/events/@id:[0-9]+/images	Recurso que actualiza la información de las imágenes de un evento.
/events/@id:[0-9]+/products/@id2:[0-9]+	Recurso que actualiza la información de un producto de un evento.
/events/@id:[0-9]+/comments/@id2:[0-9]+	Recurso que actualiza la información de un comentario de un evento.
/products/@id:[0-9]+	Recurso que actualiza la información de un producto en específico.
/products/@id:[0-9]+/images	Recurso que actualiza la información de las imágenes de un producto.
/products/@id:[0-9]+/comments/@id2:[0-9]+	Recurso que actualiza la información de un comentario de un producto.



4.4.3.3. Facturación

Este microservicio se encarga de la gestión de carrito de compra, pago y facturación de la toda la aplicación.

Las rutas de servicios RESTFul que ofrecerá este microservicio son:

GET

/carts/me	Recurso que retorna el carrito de compra del usuario actual.
/carts/me/products	Recurso que retorna el listado de productos del carrito de compra.
/carts/me/products/@id:[0-9]+	Recurso que retorna un producto específico del carrito de compra de acuerdo a su id.
/invoices	Recurso que retorna el listado de facturas realizadas de toda la aplicación.
/invoices/@id:[0-9]+	Recurso que retorna una factura específica de acuerdo al campo id.
/invoices/me	Recurso que retorna todas las facturas que posee el usuario actual.
/payments	Recurso que retorna todos los pagos realizados por los usuarios dentro de la aplicación.
/payments/@id:[0-9]+	Recurso que retorna el registro de pago específico de acuerdo al campo id.
/payments/me	Recurso que retorna el listado de todos los pagos realizados por el usuario actual.

POST

/carts	Recurso para crear un nuevo registro de carrito de compra para el usuario actual.
/carts/me/products	Recurso para añadir un producto al carrito de compra del usuario actual.



/carts/me/checkout	Recurso para guardar el registro de pago del carrito actual.
--------------------	--

PATCH

/carts/me/products/@id:[0-9]+	Recurso que actualiza el producto en el carrito de compra del usuario actual.
-------------------------------	---

DELETE

/carts	Recurso que eliminar todos los productos seleccionados en el carrito de compra del usuario actual.
/carts/me/products/@id:[0-9]+	Recurso que elimina un producto específico del carrito actual.

4.4.3.4. Reportes

Será el encargado a través de su API de suministrar todos los datos necesarios para la creación de reportes y graficas que posea la aplicación.

Las rutas de servicios RESTFul que ofrecerá este microservicio son:

GET

/reports	Recurso que muestra el listado de reportes realizados en la aplicación.
/reports/@id:[0-9]+	Recurso que retorna un reporte de acuerdo al campo id suministrado.
/reports/@id:[0-9]+/params	Recurso que retorna los parámetros utilizados para generar el reporte especificado en el campo id.

POST



/reports	Recurso que sirve para crear un nuevo registro de reporte.
----------	--

4.4.3.5. Notificaciones

Este servicio se encargará del envío de notificaciones a los distintos usuarios y envío de correos.

Las rutas de servicios RESTFul que ofrecerá este microservicio son:

GET

/notifications	Recurso que devuelve un listado con todas las notificaciones del usuario actual.
----------------	--

POST

/notifications	Recurso que genera una notificación para el usuario específico o listado de usuarios.
----------------	---

4.4.4. Otros componentes usados para el desarrollo de la aplicación.

Dependencias

- Spring Boot.
- Spring Cloud.
- Lombok.
- Spring Email.
- Spring Devtools.
- Spring Security.
- Spring Web.
- Spring Data JPA.

Almacenamiento de datos

Para el almacenamiento de datos se utilizará MySQL en su versión 8, siendo la versión más reciente. Esta base de datos es de código abierto y se acopla a la perfección con el lenguaje Java y SpringBoot.

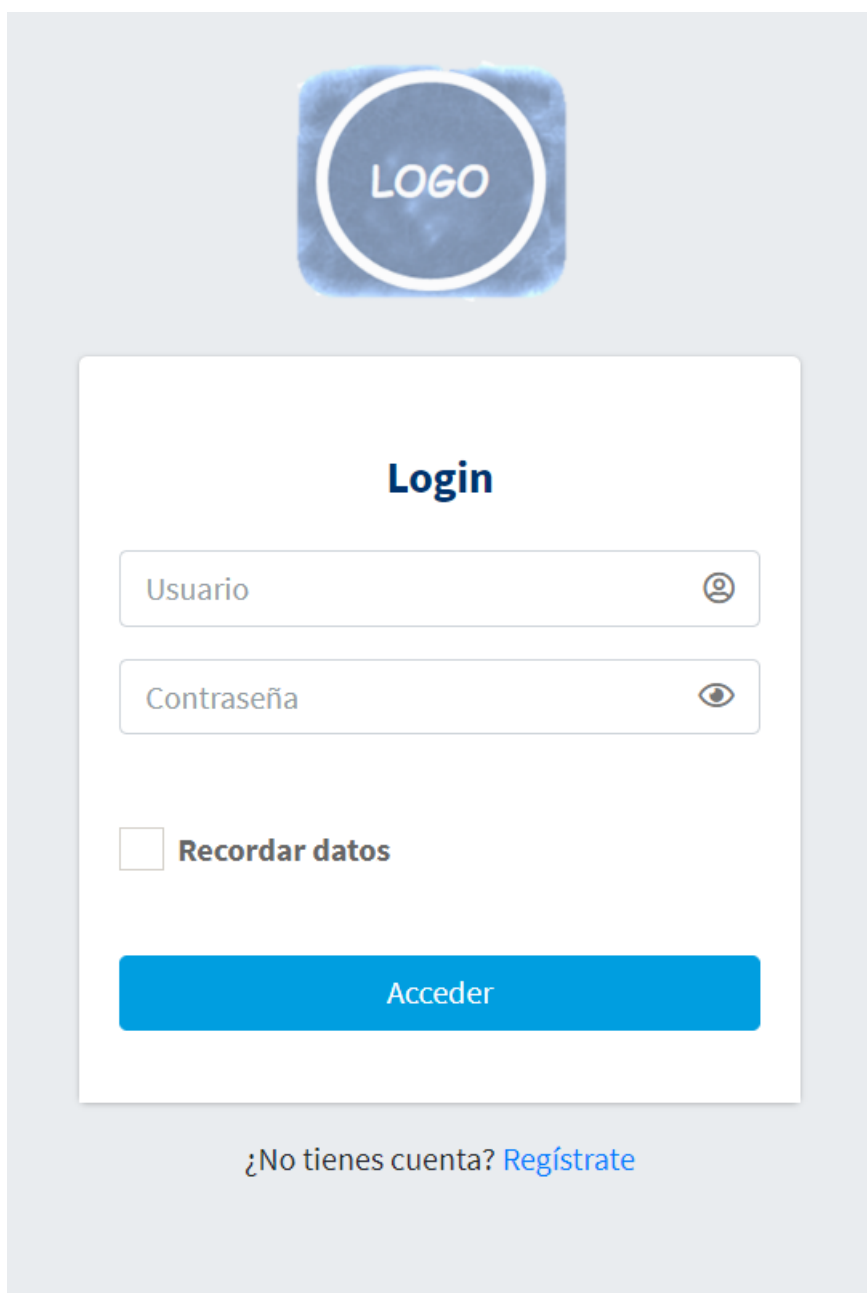


Seguridad de la aplicación

Para la seguridad de la aplicación cliente (Front-end) se utilizará Spring Security aplicando las medidas seguridad y mitigación de las vulnerabilidades presentadas en el listado OWASP Top 10.


Para el manejo de la seguridad a nivel de API se utilizará Netflix Zuul para restringir el acceso a los servicios internos a que solo estén visibles desde Zuul y de esta manera evitar el acceso directo a los microservicios.


4.4.5. Diseño de la interfaz



LOGO

Login

Usuario 

Contraseña 

Recordar datos

Acceder

¿No tienes cuenta? [Regístrate](#)

Ilustración 27 - Ventana de inicio de sesión

Esta ventana muestra el formulario de inicio de sesión que tiene la aplicación. Los campos de dicho formulario son validados con las medidas de seguridad que ofrece Spring Security.

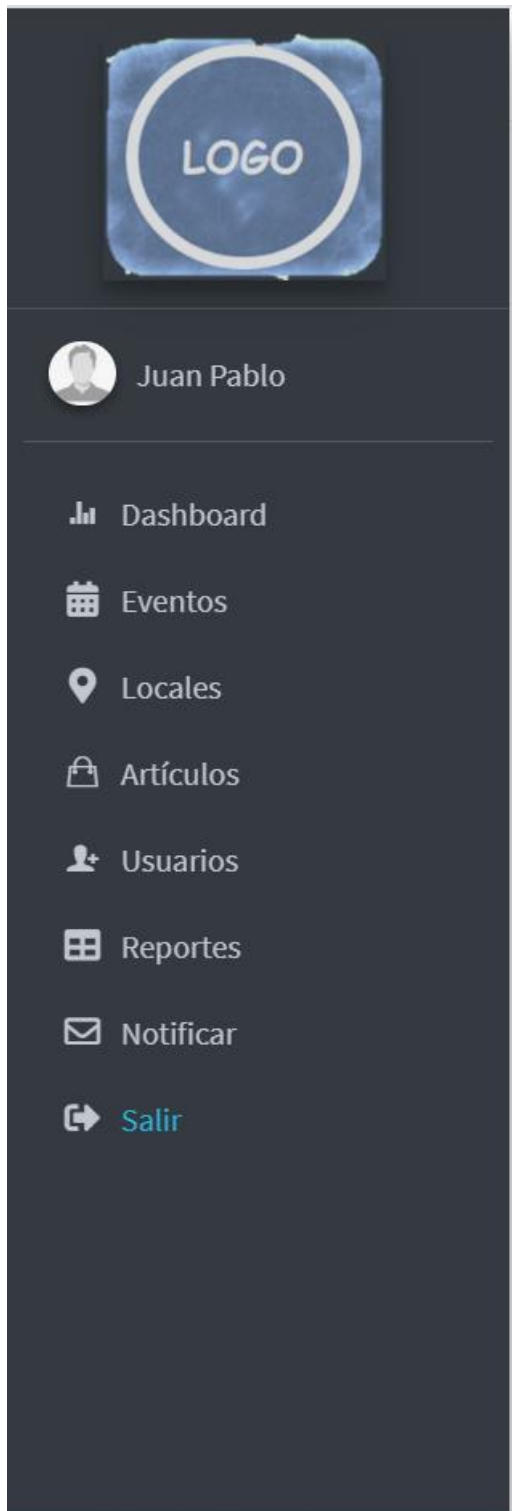


Ilustración 28 - Menú administración

Esta ilustración corresponde al menú principal del usuario que inicia sesión con el rol de administración de la plataforma.

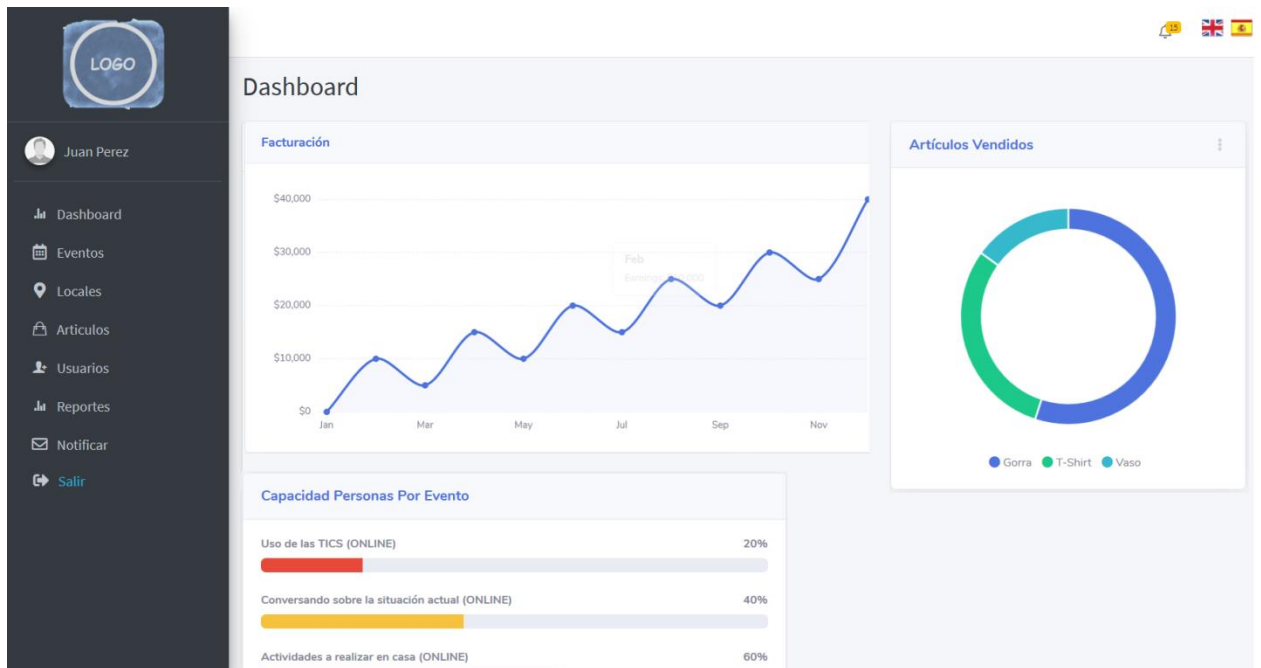


Ilustración 29 - Dashboard

Esta ilustración muestra el dashboard del administrador, donde esté puede observar las gráficas esenciales que indiquen los resultados actuales del negocio.

Eventos

[+Añadir](#)

Nombre del evento	Imagen	Estado	Fecha inicio	Fecha fin	
Lorem ipsum		Eliminado	02/02/2020 09:54:58	05/05/2020 01:08:18	
Lorem ipsu		Terminado	02/02/2020 09:54:58	05/05/2020 01:08:18	
TICS		Terminado	02/02/2020 09:54:58	05/05/2020 01:08:18	
Lorem ipsu		Activo	02/02/2020 09:54:58	05/05/2020 01:08:18	

Ilustración 30 - Ventana listado de eventos

En esta ventana se muestra un listado de los eventos que están en la aplicación.

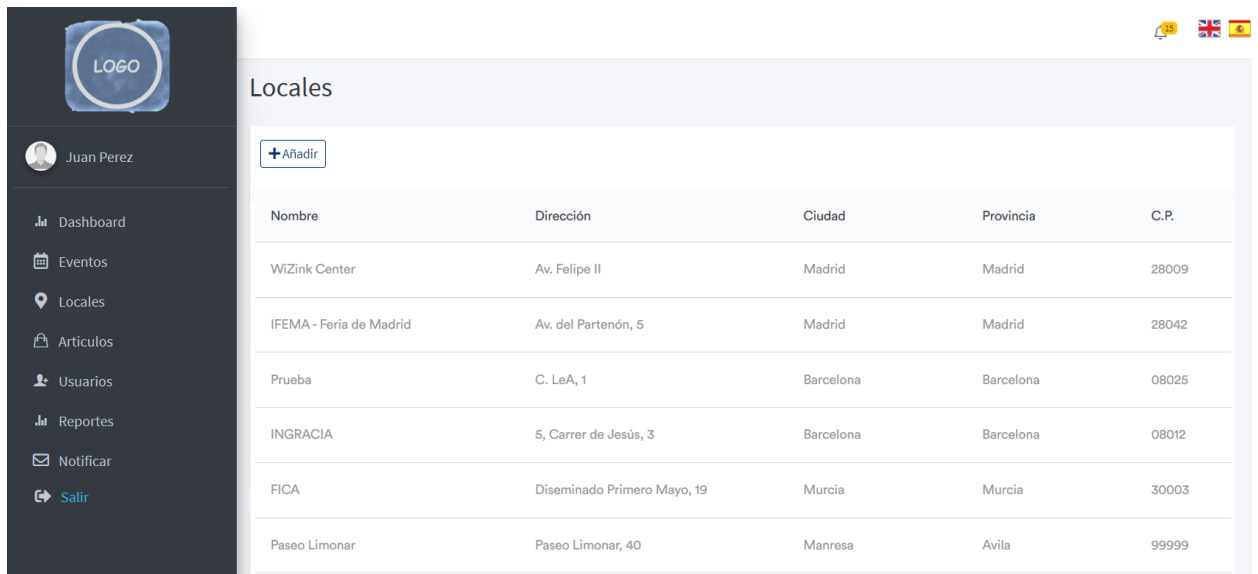


Ilustración 31 - Ventana listado de locales

Esta ventana corresponde al listado de los locales en la parte de administración.

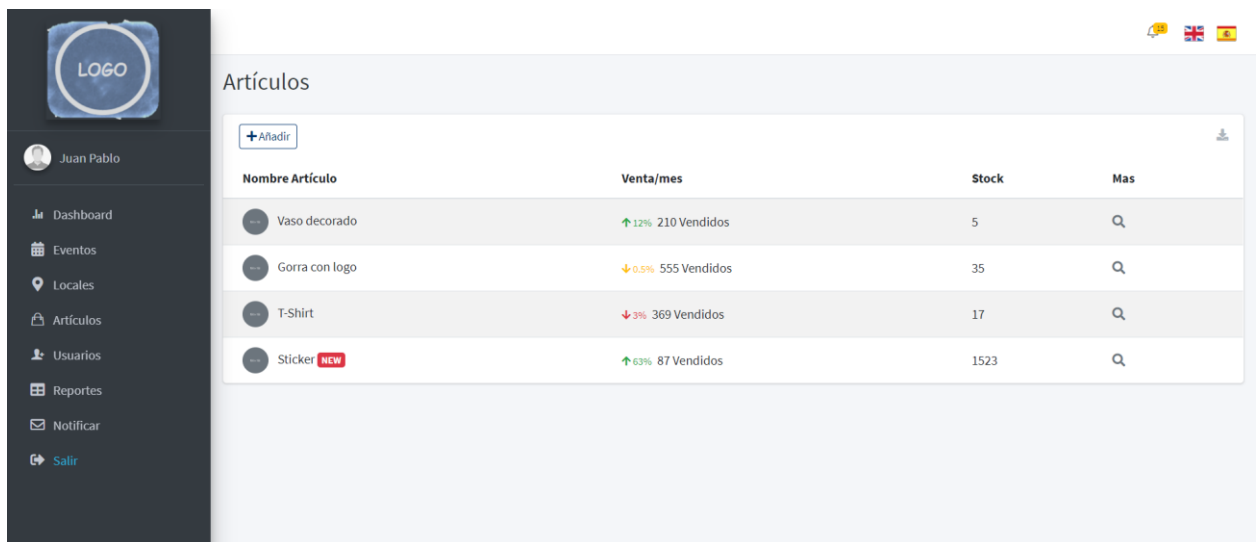


Ilustración 32 - Ventana listado de artículos

En esta ventana se pueden observar los artículos promocionales que están dados de alta en la aplicación.

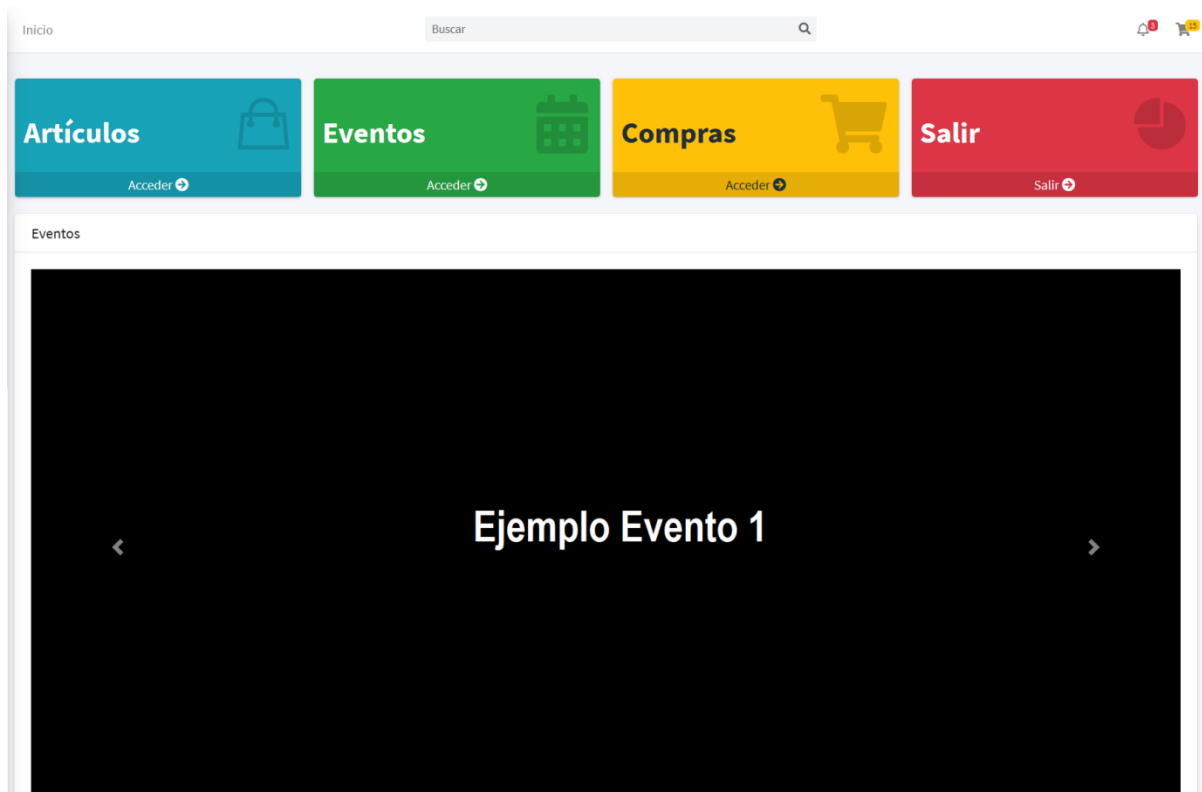


Ilustración 33 - Ventana principal rol usuario

Esta corresponde a la ventana principal del usuario final, es decir todo aquel que solo tenga el rol de usuario. En esta se muestran los últimos eventos que han sido dados de alta en la aplicación.

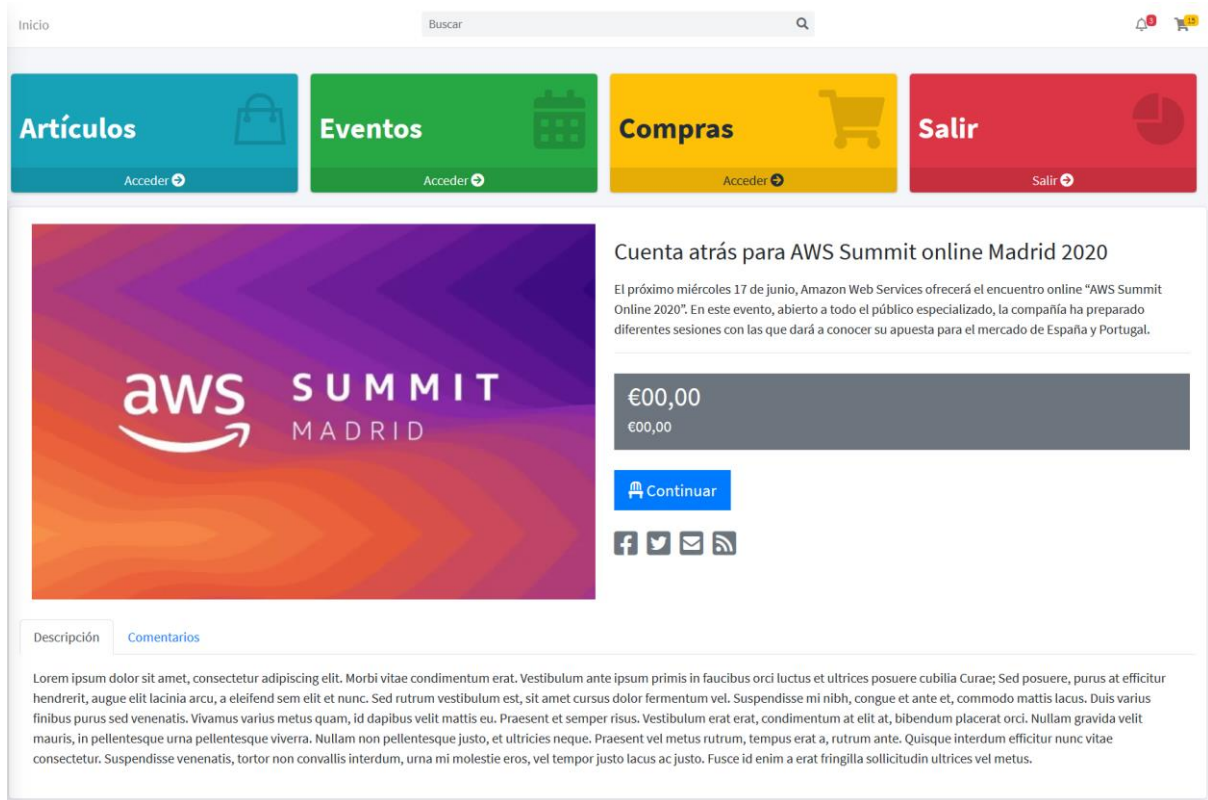


Ilustración 34 - Ventana detalle evento

Esta ventana muestra el detalle de uno de los eventos que se muestra en la ventana principal.

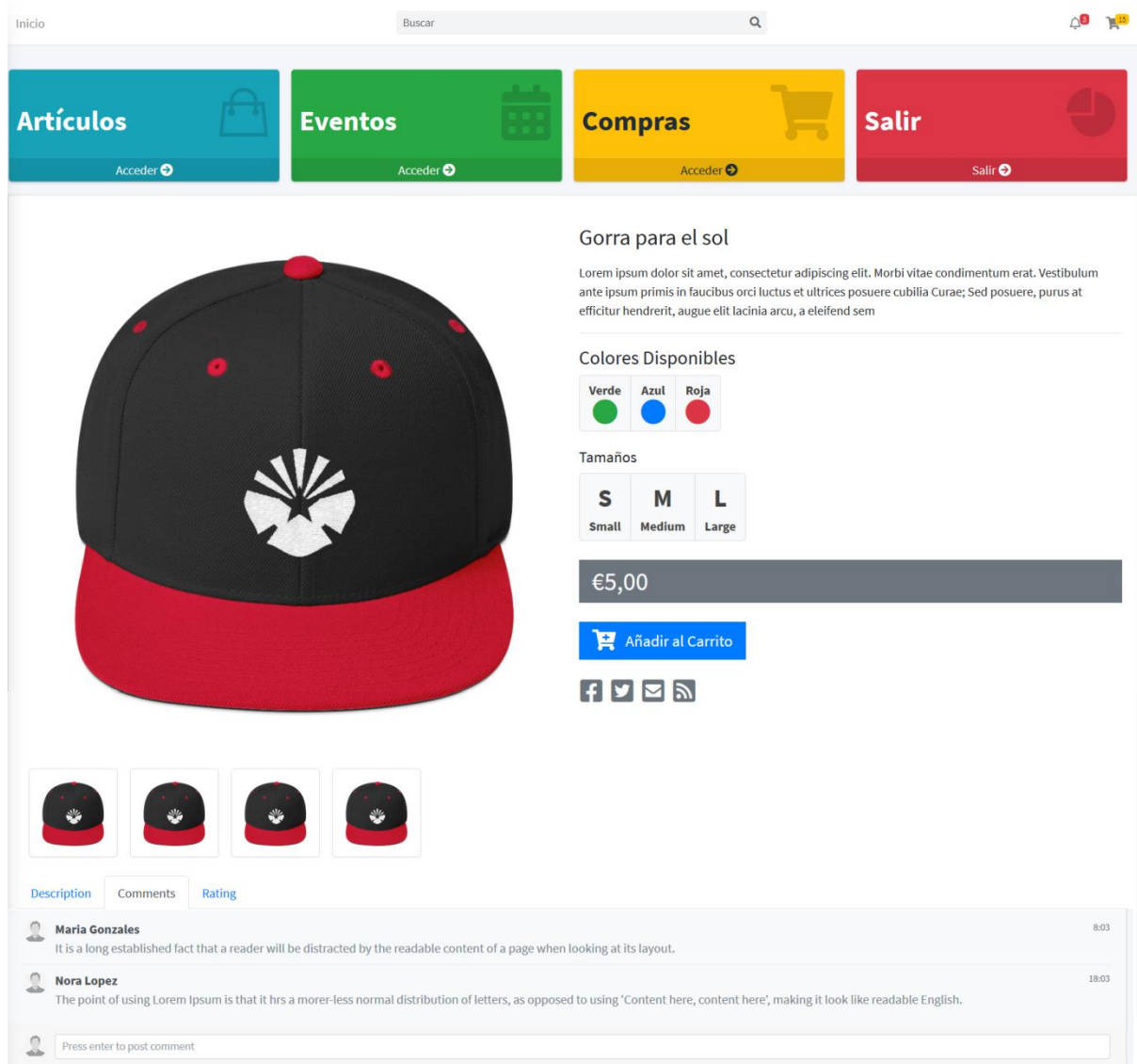


Ilustración 35 - Ventana detalle artículo promocional

En esta ventana se muestra un detalle de uno de los productos promocionales y los comentarios realizados por los usuarios.

4.4.6. Diagramas de Base de Datos

De acuerdo al planteamiento realizado para la elaboración de este prototipo basado en una arquitectura de microservicios, se ha realizado un diagrama de Entidad-Relación de base de datos de manera independiente para cada microservicio.



4.4.6.1. Autenticación

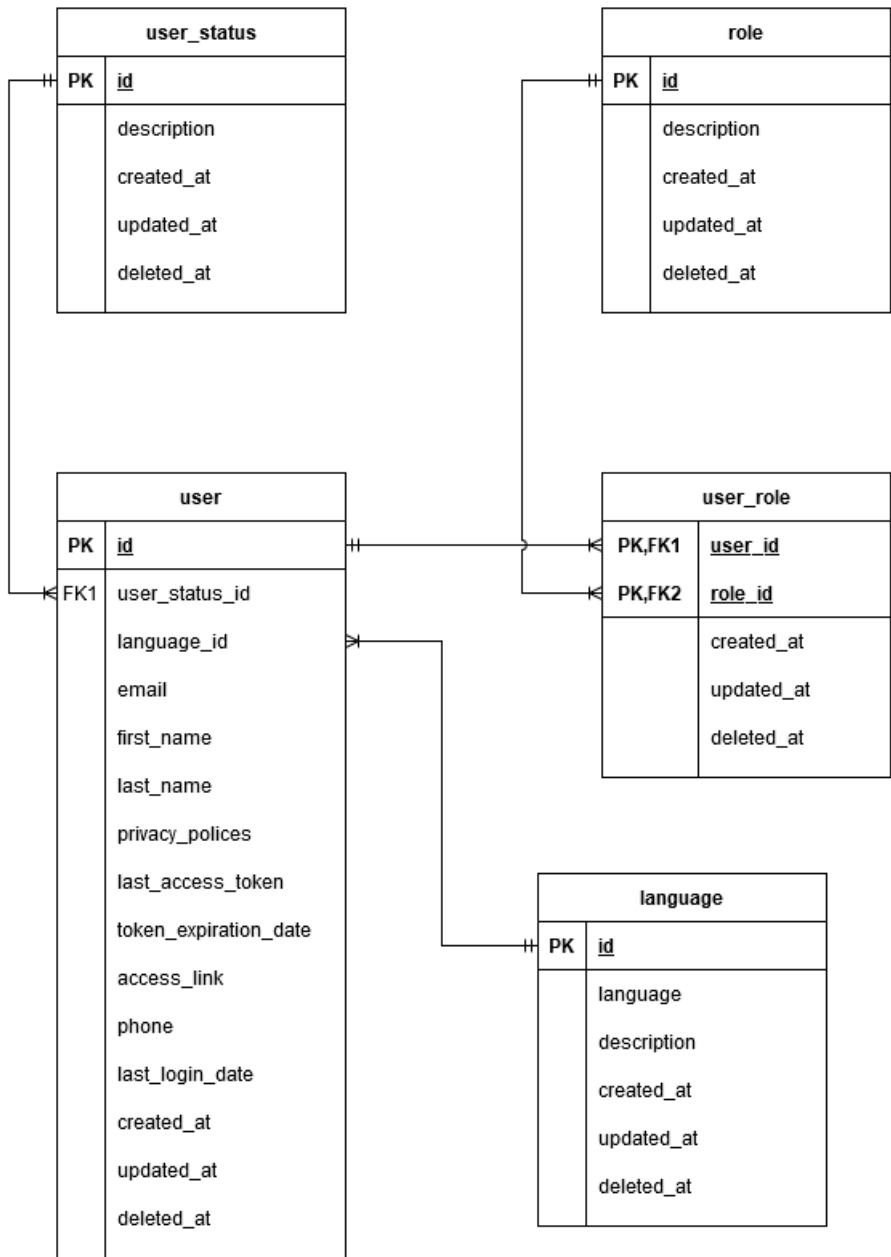
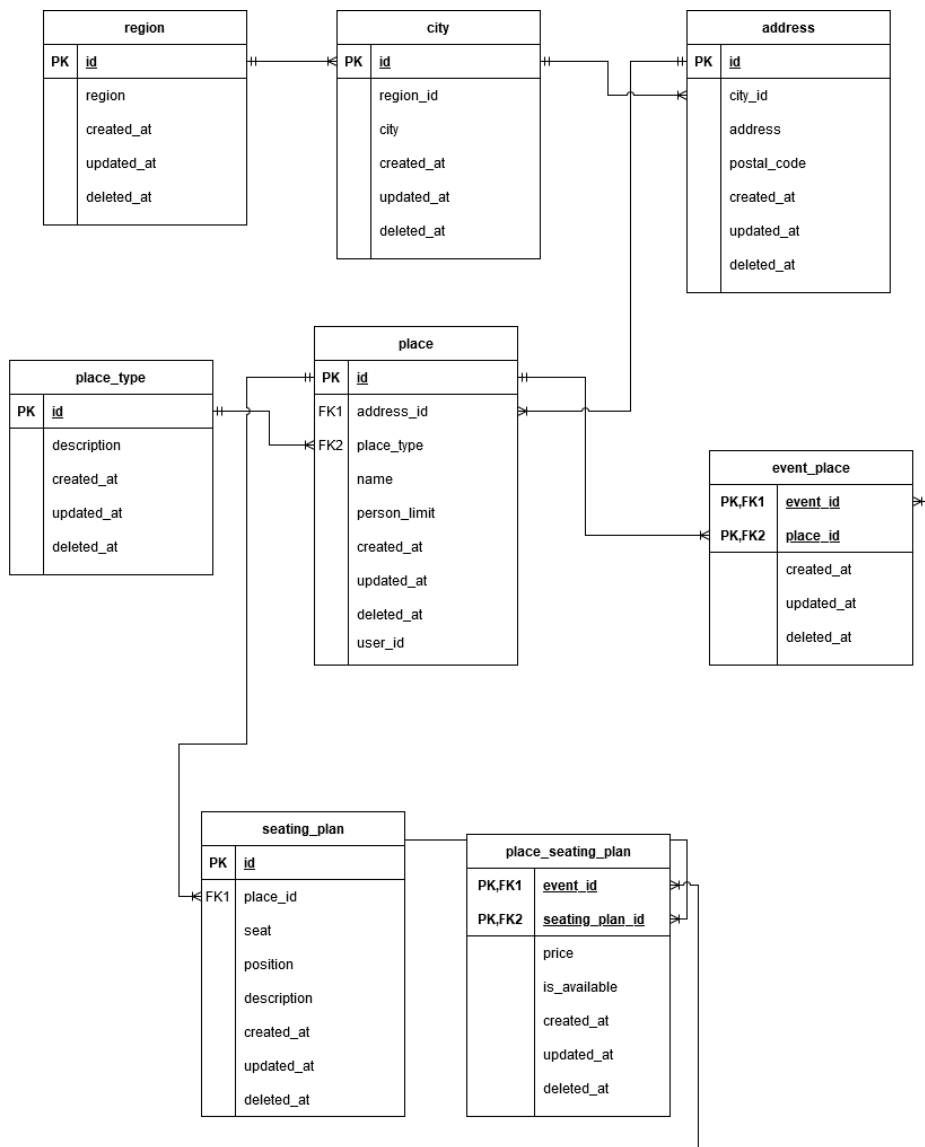


Ilustración 36 - Diagrama entidad-relación Autenticación

Como se puede apreciar, el diagrama de entidad relación de la base de datos a creada para el microservicio de autorización hace referencia a los usuarios, su estatus, sus roles y su idioma.



4.4.6.2. Gestión de eventos



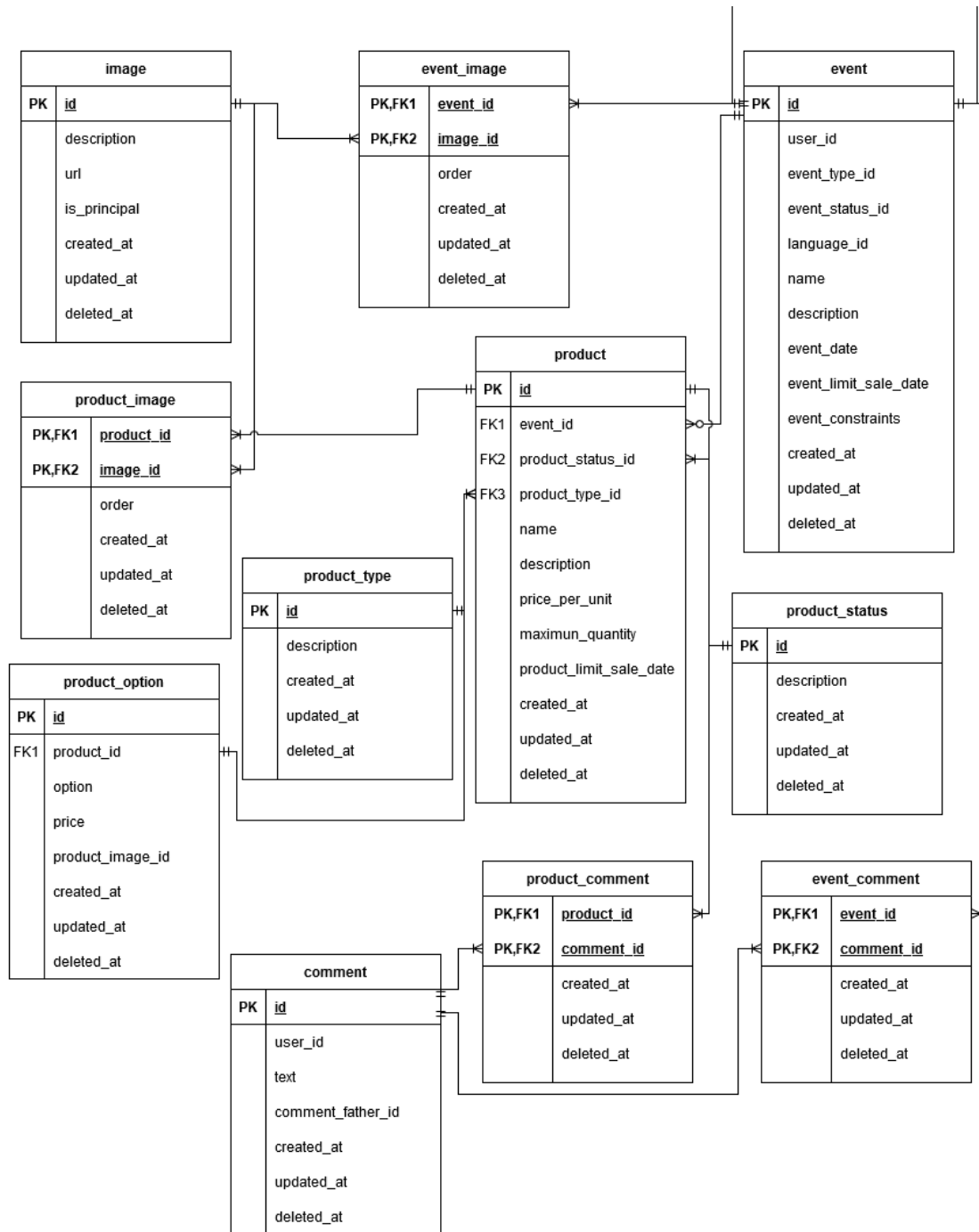


Ilustración 37 - Diagrama entidad-relación eventos

Como se puede apreciar, el diagrama de entidad relación de la base de datos a creada para el microservicio de autorización hace referencia a los eventos, los productos, los comentarios, los locales y las imágenes gestionadas en la aplicación. Este es el diagrama más complejo de todos los microservicios que componen el prototipo.



4.4.6.3. Facturación

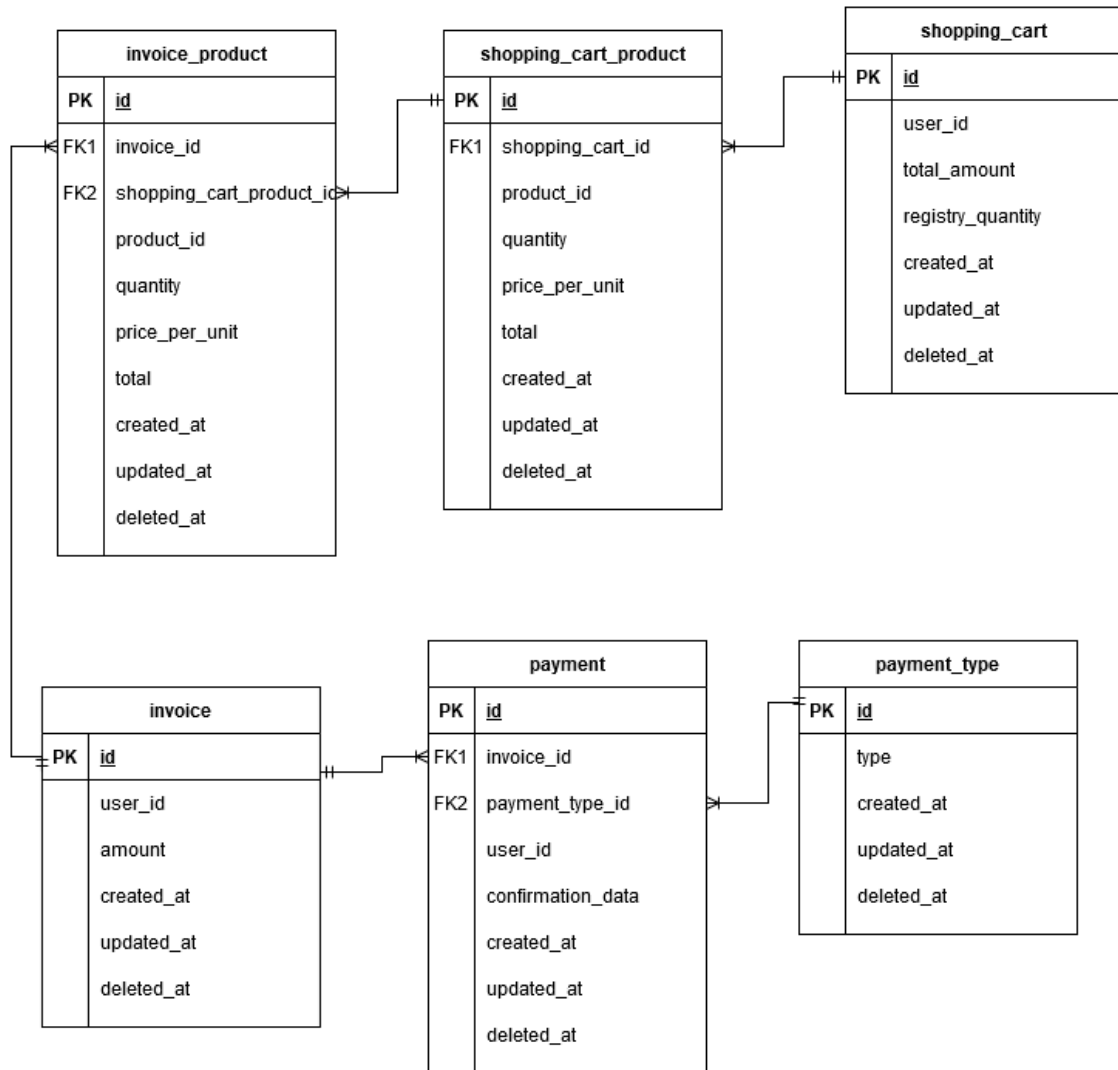


Ilustración 38 - Diagrama entidad-relación facturación

Como se puede apreciar, el diagrama de entidad relación de la base de datos a creada para el microservicio de facturación se hace referencia al carrito de compra, la factura y el pago.



4.4.6.4. Reportes

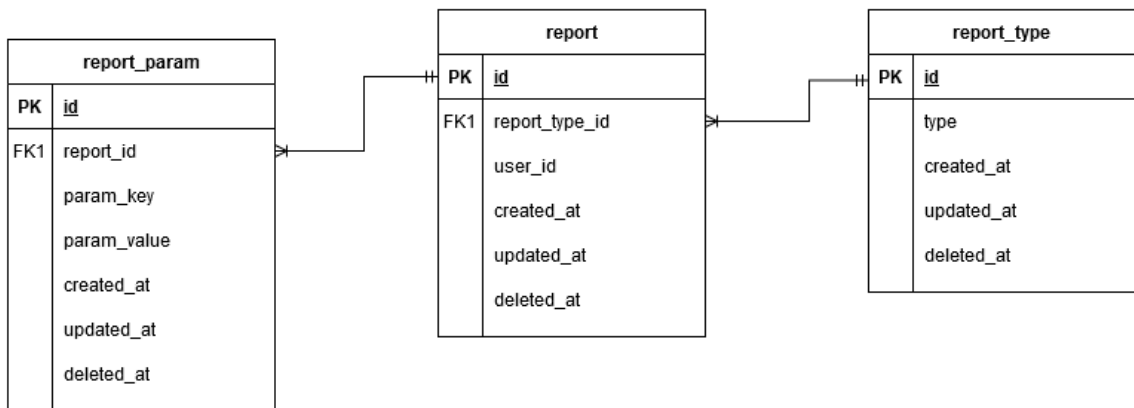


Ilustración 39 - Diagrama entidad-relación reportes

Como se puede apreciar, este es el diagrama de entidad relación más sencillo que posee la aplicación y corresponde al microservicio de reporte.



4.4.6.5. Notificaciones

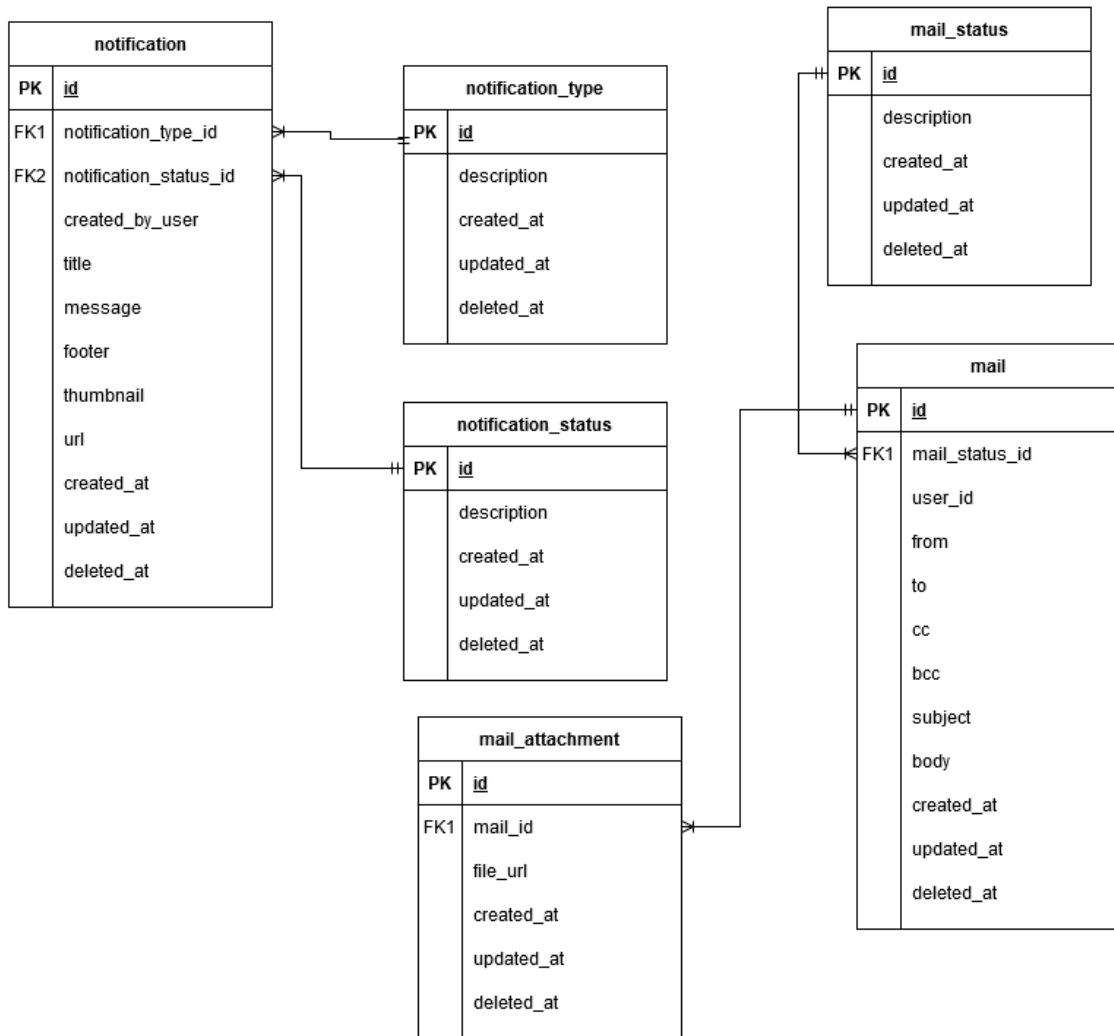


Ilustración 40 - Diagrama entidad-relación notificación

Como se puede apreciar, el diagrama de entidad relación de la base de datos a creada para el microservicio de notificación se hace referencia al envío de notificaciones por mail y por medio de la aplicación.

CONCLUSIÓN



Los microservicios han llegado para cambiar todo el paradigma actual de los servicios que existen en la web, ofreciendo una solución práctica, extensible y capaz de satisfacer todas las necesidades a nivel de sistemas web que tiene el cliente y además ofrece un gran número de oportunidades para la comunicación entre plataformas y dispositivos.

Spring framework y todos los proyectos que lo componen se han convertido en la herramienta principal para la creación y mantenimiento de aplicaciones empresariales con una arquitectura basada en microservicios en el lenguaje de programación Java. Estos proyectos van desde la creación de una página HTML hasta el desarrollo de todo un ecosistema que puede contener múltiples microservicios y componentes que forman la base de muchas de las aplicaciones que utilizamos en nuestra actualidad.

Como se ha podido apreciar en este trabajo, el auge y desarrollo de plataformas web cada día se hace más popular gracias a los avances de los frameworks y entornos de desarrollo, los cuales, reducen considerablemente el tiempo y costo de creación de las aplicaciones.

Otros de los puntos importantes es que estos emplean las mejores prácticas para solucionar un conjunto de problemas y situaciones que se presentan en el día a día del desarrollo de aplicaciones web; para dar como resultado la creación de sistemas con los más altos estándares de calidad del mercado, capaces de satisfacer las necesidades del negocio y mitigar las vulnerabilidades de seguridad que afectan a muchas aplicaciones.

En la actualidad, más y más empresas se suman a la era de los microservicios, dejando de lado sus arquitecturas monolíticas para pasar a tener toda su infraestructura en la nube. Esto puede representar un coste de implementación al principio, pero con el paso del tiempo ofrece muchas ventajas, como son: un mejor nivel de organización de servicios, reducción de espacio físico, costes y consumo de recursos.

Luego de analizar y estudiar Spring y sus proyectos principales se puede afirmar que el uso de este conjunto de Frameworks garantiza el desarrollo de aplicaciones empresariales con excelente calidad y capaces de ampliar su funcionalidad con facilidad sin importar el tamaño o la cantidad de funcionalidades que éstas posean.

Spring también ofrece a las empresas una reducción de tiempo y coste de desarrollo gracias al conjunto de componentes y configuraciones que pertenecen al framework y que con solo incluirlos en la aplicación, se evita que el desarrollador tenga que crearlos desde cero.



Este framework posee una gran comunidad de desarrolladores, que introducen nuevos proyectos mejoras constantemente, siendo una de las más activas dentro del mundo Java y marcando una diferencia notable en comparación con los otros frameworks en Java.

La comunicación e integración es esencial entre dispositivos con acceso a internet, tales como, teléfono móvil, portátil, SmartTV, Tablet u otros para que la información que se muestra a los usuarios sea la misma en todo momento. Esta comunicación e integración se realiza de manera instantánea entre aplicaciones gracias a los microservicios creados por grandes plataformas como Google, Netflix, Facebook o Instagram.

El conjunto de proyectos que forman Spring sirven de base para la creación de componentes que mantienen la integración y comunicación entre un gran número de plataformas, aplicaciones móviles y web.

PRESUPUESTO



En esta sección se incluye un presupuesto estimado del costo de desarrollar e implementar la aplicación prototipo definida en este trabajo.

Actividad	Horas	Precio
Desarrollo de los componentes de configuración para los microservicios.	40	800,00€
Desarrollo de todos los microservicios.	80	1600,00€
Compra y configuración de los servidores para colocar los componentes del ecosistema.	10	200€
Compra y configuración dominio Web.	1	15€
		Total: 2.615,00€

BIBLIOGRAFÍA

- Prasad Reddy S. (2017), *Beginning Spring Boot 2: Applications and Microservices with the Spring Framework*, Apress, California.
- Cosmina I. (2020), *Pivotal Certified Professional Core Spring 5 Developer Exam: A Study Guide Using Spring Framework 5*
- Gutierrez F. (2019), *Pro Spring Boot 2: An Authoritative Guide to Building Microservices, Web and Enterprise Applications, and Best Practices*.
- Marten Deinum, Daniel Rubio, Josh Long (2017), *Spring 5 Recipes: A Problem-Solution Approach*, Apress, California.
- Sam Newman (2015), *Building Microservices: Designing Fine-Grained Systems*, O'Reilly Media, Sebastopol, California.
- Alexandru Jecan (2017), *Java 9 Modularity Revealed: Project Jigsaw and Scalable Java Applications*, Apress, California.
- Relan Kunal (2019), *Building REST APIs with Flask: Create Python Web Services with MySQL*, Apress, California.
- Laddad R. (2003), *AspectJ in Action: Practical Aspect-oriented Programming*, Manning, New York.
- Evans E. (2003), *Domain-Driven Design: Tackling Complexity in the Heart of Software*, Addison Wesley, Boston.
- Deepak Alur et. Al, (2001), *Core J2EE Patterns: Best Practices and Design Strategies*, Prentice Hall, Nueva Jersey.
- Wolff E. (2006), *Microservices: Flexible Software Architecture*, Addison Wesley, Boston.
- Meyer B. (1998), *Object-Oriented Software Construction*, Prentice-Hall, New Jersey.

- McLean Hall G. (2017), Adaptive Code via C#: Agile coding with design patterns and SOLID principles, O'Reilly Media, Sebastopol, California.
- LISKOV B. (1988), Data Abstraction and Hierarchy. SIGPLAN Notices.
- Martin F. (1996), Analysis Patterns: Reusable Object Models, Addison Wesley, Boston.
- Fowler R. (2002), Agile Software Development, Principles, Patterns, and Practices, Pearson, Harlow.
- Spring (2020), Spring, (<https://spring.io/>) (Consultado, 15 de marzo de 2020).
- Mozilla (2020), World Wide Web, 31 de enero 2020. (https://developer.mozilla.org/es/docs/Glossary/World_Wide_Web) (Consultado el 3 de marzo de 2020).
- Martínez et al., (2009), Introducción a las arquitecturas de componentes y a Java EE (2009). (http://ocw.uc3m.es/ingenieria-telematica/software-de-comunicaciones/transparencias/3_cmpnts-JavaEE.pdf) (Consultado el 9 de marzo de 2020).
- Nupur Choudhury / (IJCSIT) International Journal of Computer Science and Information Technologies (2014), World Wide Web and Its Journey from Web 1.0 to Web 4.0 (2014). (<http://ijcsit.com/docs/Volume%205/vol5issue06/ijcsit20140506265.pdf>) (Consultado el 2 de marzo de 2020).
- World Wide Web Consortium (W3C) (2011), Answers for Young People, 15 de agosto de 2011, (<https://www.w3.org/People/Berners-Lee/Kids.html>). (Consultado el 5 de marzo de 2020).
- Tim O'Reilly (2006), Web 2.0 Compact Definition: Trying Again, 10 de diciembre de 2006, (<http://radar.oreilly.com/2006/12/web-20-compact-definition-tryi.html>). (Consultado el 2 de marzo de 2020).
- World Wide Web Consortium (W3C) (2009), W3C Semantic Web Frequently Asked Questions, 12 de noviembre de 2009, (<https://www.w3.org/RDF/FAQ>). (Consultado el 5 de marzo de 2020).

- Oracle (2010), Chapter 2 Understanding Java™ Platform, Enterprise Edition, 2010, (<https://docs.oracle.com/cd/E19798-01/821-1770/6nmnum0kp/index.html>). (Consultado el 07 de marzo de 2020)
- Viewnext (2018), Arquitectura de microservicios vs arquitectura monolítica, 3 de mayo de 2018, (<https://www.viewnext.com/arquitectura-de-microservicios-vs-arquitectura-monolitica/>). (Consultado el 10 de marzo de 2020).
- Dragoni N. et al. (2016) Microservices: Yesterday, Today, and Tomorrow. 13 de junio de 2016 (https://www.researchgate.net/publication/305881421_Microservices_yesterday_today_and_tomorrow). (Consultado el 12 de marzo de 2020).
- Salesforce (2015), SaaS: Power Your Business with Software as a Service, (<https://www.salesforce.com/saas/>) (Consultado el 14 de marzo de 2020).
- Maven, Welcome to Apache Maven (<https://maven.apache.org/>) (Consultado el 20 de marzo de 2020).
- Autentia (2018), Por qué trabajar con Maven y Gradle, 31 de octubre de 2018, <https://www.autentia.com/2018/10/31/por-que-trabajar-con-maven-y-gradle/> (Consultado el 20 de marzo de 2020).
- Gradle, Accelerate developer productivity, (<https://gradle.org/>) (Consultado el 25 de marzo de 2020).
- Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Videira Lopes, C., Loingtier, J. M., & Irwin, J. (1997). Aspect-Oriented Programming. Recuperado de <https://www.cs.ubc.ca/~gregor/papers/kiczales-ECOOP1997-AOP.pdf>
- Wikipedia, Programación orientada a aspectos (https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_aspectos), (Consultada el 01 de abril de 2020).
- Thönes, Johannes (2015). “Microservices.” In: Software, IEEE, <https://ieeexplore.ieee.org/document/7030212>, (Consultado el 15 de abril de 2020).

- Lewis J. Fowler M. (2014), Microservices: a definition of this new architectural term, <https://martinfowler.com/articles/microservices.html> (Consultado el 20 de abril de 2020).
- National Institute of Standards and Technology (NIST) (2011), The NIST Definition of Cloud Computing, (<https://csrc.nist.gov/publications/detail/sp/800-145/final>) (Consultado el 25 de abril de 2020).
- Netflix /zuul (2020) (<https://github.com/Netflix/zuul/wiki>) (Consultado el 20 de mayo de 2020)