UNIVERSIDAD DEL BIO-BIO

Facultad de Ciencias Empresariales
Departamento Sistema de Informaciones

MEMORIA PARA OPTAR A TITULO DE INGENIERO DE EJECUCIÓN EN COMPUTACIÓN E INFORMÁTICA

Investigación y desarrollo de la arquitectura orientada a microservicios para la implementación de módulos en un sistema web

Alumno: Ricardo Esteban Muñoz Hernández
Profesor Guía: Juan Carlos Parra Márquez

CONCEPCION, 2018

Agradecimientos

A mis padres por su apoyo incondicional, a las personas que siempre me dieron ánimos de continuar y a al líder en Zenta por brindarme conocimiento y habilidades técnicas.

Tabla de Contenidos

Resumen 1
Introducción 2
Planteamiento del problema 4
Problema a resolver 5
Estado del arte 7
Capítulo 1 9
1.1 Definición arquitectura monolítica 9
1.1.1 Principales características 10
1.1.2 Principales ventajas 10
1.1.3 Principales desventajas 11
1.1.4 Definición modelo vista controlador 11
1.1.5 Descripción de un modelo 14
1.1.6 Descripción de un controlador 15
1.1.7 Descripción de una vista 16
Capítulo 2 17
2.1 Definición general de la arquitectura orientada a microservicios 17
2.1.1 Principales ventajas de la arquitectura 18
2.1.2 Principales desventajas de la arquitectura 18
2.2 Definición microservicio 19
2.2.1 Principales atributos de un microservicio 20
2.2.2 Estructura de un microservicio 23
2.2.3 Descripción de las capas de un microservicio 24
2.2.4 Representación de la estructura interna de un microservicio 25
Capítulo 3 34
3.1 Definición de automatización para microservicios 34
3.1.1 ¿Qué son los contenedores? 34
3.1.2 Características de los contenedores 35
3.2 Diferencia entre virtualización y contenedores 35
3.3 Docker 36
3.3.1 Principales características de los contenedores Docker: 37
3.4 Kubernetes 38
3.4.1 Principales características de Kubernetes(Burns, Hightower, Beda, 2017) 39
Capítulo 4 40
4.1 Diferencias entre arquitectura monolítica y arquitectura orientada a microservicios. 40
4.1.1 Sistema monolítico. 40
4.1.2 Sistema orientado a microservicios. 41
Capítulo 5 43
5.1 Análisis y desarrollo de módulos 43
5.2 Descripción de los módulos 49
5.2.1 Módulo referenciados 49
5.2.2 Módulo rendiciones 50
5.3 Desarrollo del módulo referenciados 51
5.3.1 Objetivo general 51
5.3.2 Objetivos específicos 51

5.3.3 Descripción de interfaz. 51
5.3.4 Requerimientos funcionales 52
5.3.5 Requerimientos no funcionales. 52
5.3.6 Análisis del caso de uso modulo referenciados 53
5.3.7 Especificación de casos de uso 55
5.3.8 Diagrama de secuencia. 60
5.3.9 Modelamiento de datos 64
5.3.10 Ilustraciones de interfaz de usuario 65
5.3.11 Ilustraciones microservicio módulo referenciados 68
5.3.12 Pruebas al microservicio referenciados 73
5.3.13 Pruebas al módulo 75
5.4 Desarrollo del módulo rendiciones 82
5.4.1 Objetivo general 82
5.4.2 Objetivos específicos 82
5.4.3 Descripción de interfaz. 82
5.4.4 Requerimientos funcionales 83
5.4.5 Requerimientos no funcionales. 83
5.4.6 Análisis del caso de uso modulo rendiciones 84
5.4.7 Especificación de casos de uso 86
5.4.8 Diagrama de secuencia. 91
5.4.9 Modelamiento de datos 95
5.4.10 Ilustraciones de interfaz de usuario 96
5.4.11 Ilustraciones microservicio módulo rendiciones 99
5.4.12 Pruebas para el microservicio rendiciones 103
5.4.13 Pruebas al módulo 105
Conclusión 112
Posibles tareas a realizar 114
Bibliografía 115

Lista de tablas

Tabla1. Requisitos funcionales del módulo. 52
Tabla2. Requisitos no funcionales del módulo. 52
Tabla3.Antecedentes del caso de uso Ingresar Candidato. 56
Tabla4.Antecedentes del caso de uso Proceso Contratación. 57
Tabla5.Antecedentes del caso de uso Consultar Histórico. 58
Tabla6.Antecedentes del caso de uso Consultar Grafico. 59
Tabla7. Requisitos funcionales del módulo. 83
Tabla8. Requisitos no funcionales del módulo. 83
Tabla9.Antecedentes del caso de uso Ingresar rendición. 87
Tabla10.Antecedentes del caso de uso Recepcionar rendición. 88
Tabla11.Antecedentes del caso de uso Control rendición. 89
Tabla12.Antecedentes del caso de uso Confirmar rendición. 90

Lista de figuras

Figura 1. Componentes y procesos, SOA. 7
Figura 2. Evolución del acoplamiento en las distintas arquitecturas. 8
Figura 3.Representación de una Arquitectura Monolítica. 9
Figura 4. Patrón MVC (Blancarte,2014) 12
Figura 5. Interacción de los componentes (Blancarte, 2014). 12
Figura 6. Tabla person. 13
Figura 7. Conexión a una única base de datos (Modo Centralizado). 13
Figura 8. Modelo a nivel código. 14
Figura 9. Controlador a nivel código. 15
Figura 10. Vista de usuario 16
Figura 11. Arquitectura orientada a microservicios 17
Figura 12. Flexibilidad de tecnologías. (Newman, 2015) 20
Figura 13. Solo se le da escala a los microservicios que lo necesiten. (Newman, 2015) 20
Figura 14.Estructura general de un microservicio desarrollado con JHipster (Java). 21
Figura 15. Configuración de un microservicio a un motor de base de datos MySQL. 22
Figura 16.Configuración de un microservicio a un motor de base de datos PostgreSQL. 22
Figura 17.Diferentes microservicios para una misma aplicación. 22
Figura 18. Estructura interna de un microservicio (Clemson, 2014) 23
Figura 19. Representación de un dominio que referencia a la entidad person. 26
Figura 20. Representación de un dominio personalizado. 27
Figura 21. Representación de un repositorio que hereda métodos de JpaRepository. 28
Figura 22. Repositorio personalizado. 29
Figura 23. Representación de una clase java como capa de servicio. 30
Figura 24. Recurso básico, métodos CRUD. 31
Figura 25. Representación de un recurso personalizado. 32
Figura 26. Gateway con microservicios disponibles. 33
Figura 27. Recursos para solicitar peticiones básicas mediante HTTP. 33
Figura 28. Recursos para solicitar peticiones personalizadas mediante HTTP. 33
Figura 28.Recursos personalizados. 33
Figura 29. Virtualización vs Contenedores (Red Hat, 2018). 36
Figura 30. Contenedor Linux vs Docker. 37
Figura 31.Representación de un Sistema Monolítico 40
Figura 32.Representación de un Sistema orientado a microservicios 41
Figura 33. Productividad vs complejidad. (Fowler, 2015) 42
Figura 34. Modelo Iterativo 44
Figura 35. Diagrama de caso de uso para el módulo referenciados (Elaboración propia). 54
Figura 36. Caso de uso ingresar candidato al sistema (Elaboración propia). 55
Figura 37. Caso de uso proceso de contratación (Elaboración propia). 57
Figura 38. Caso de uso consultar histórico (Elaboración propia). 58
Figura 39. Caso de uso consultar gráfico (Elaboración propia). 59
Figura 40. Diagrama de secuencia Ingresar Candidato al sistema (Elaboración propia). 60
Figura 41. Diagrama de secuencia Proceso Contratación (Elaboración propia). 61
Figura 42.Diagrama de secuencia Consultar Histórico (Elaboración propia). 62
Figura 43.Diagrama de secuencia Consultar Grafico (Elaboración propia). 63

Figura 44.Diseño físico para el módulo referenciados (Elaboración propia). 64
Figura 45.Menú principal para el módulo referenciados (Elaboración propia). 65
Figura 46.Interfaz para mis referenciados (Elaboración propia). 65
Figura 46.a. Acción del botón "Añadir" (Elaboración propia). 66
Figura 47. Interfaz para el administrador de RRHH (Elaboración propia). 66
Figura 48. Interfaz para consultar gráfico (Elaboración propia). 67
Figura 49. Seleccionar cargo (Elaboración propia). 67
Figura 50. Consultar Histórico(Elaboración propia). 67
Figura 51. Dominio (Elaboración propia). 68
Figura 52. Dominio personalizado (Elaboración propia). 69
Figura 53. Repositorio (Elaboración propia). 69
Figura 54. Repositorio personalizado (Elaboración propia). 70
Figura 55. Capa de servicio (Elaboración propia). 71
Figura 56. Recurso (Elaboración propia). 71
Figura 57. Recurso personalizado (Elaboración propia). 72
Figura 58. Puerta de enlace para los recursos (Elaboración propia). 72
Figura 59. Puerta de enlace personalizada (Elaboración propia). 72
Figura 60. GET para personas referidas (Elaboración propia). 73
Figura 61. Resultado de la solicitud GET (Elaboración propia). 73
Figura 62. Solicitud a un recurso personalizado (Elaboración propia). 74
Figura 63. Resultado de un recurso personalizado (Elaboración propia). 74
Figura 64. Referido ingresado exitosamente (Elaboración propia). 75
Figura 65. Añadir referido (Elaboración propia). 75
Figura 66. Interfaz mis referenciados (Elaboración propia). 75
Figura 67. Visualización del curriculum (Elaboración propia). 76
Figura 68. Curriculum guardado exitosamente (Elaboración propia). 76
Figura 69. Acción guardar curriculum (Elaboración propia). 76
Figura 70. Interfaz administrador (Elaboración propia). 77
Figura 71. Cambiar estado a proceso (Elaboración propia). 77
Figura 72. Confirmación del cambio(Elaboración propia). 77
Figura 73. Solicitud en proceso(Elaboración propia). 77
Figura 74. Confirmación del cambio (Elaboración propia). 78
Figura 75. Aceptar solicitud (Elaboración propia). 78
Figura 76. Solicitud aceptada (Elaboración propia). 78
Figura 77. Detalles de la solicitud aprobada (Elaboración propia). 78
Figura 78. Solicitud rechazada (Elaboración propia). 79
Figura 79. Rechazar solicitud(Elaboración propia). 79
Figura 80. Mostrar grafico (Elaboración propia). 80
Figura 81. Gráfico de estadísticas por cargo (Elaboración propia). 80
Figura 82. Interfaz histórico (Elaboración propia). 80
Figura 83. Error ver curriculum (Elaboración propia). 81
Figura 84. Error detalles solicitud histórico (Elaboración propia). 81
Figura 85. Error cargar gráfico (Elaboración propia). 81
Figura 86. Error de flujo (Elaboración propia). 81
Figura 87. Diagrama de caso de uso para el módulo rendiciones (Elaboración propia). 85
Figura 88. Caso de uso ingresar rendición al sistema (Elaboración propia). 86

Figura 89. Caso de uso Recepcionar rendición (Elaboración propia). 88
Figura 90. Caso de uso Control rendición (Elaboración propia). 89
Figura 91. Caso de uso Confirmar rendición (Elaboración propia). 90
Figura 92. Diagrama de secuencia Ingresar rendición al sistema(Elaboración propia). 91
Figura 93. Diagrama de secuencia Recepcionar rendición (Elaboración propia). 92
Figura 94. Diagrama de secuencia Control rendición (Elaboración propia). 93
Figura 95. Diagrama de secuencia Confirmar rendición (Elaboración propia). 94
Figura 96. Diseño físico para el módulo rendiciones (Elaboración propia). 95
Figura 97. Menú principal para el módulo rendiciones (Elaboración propia). 96
Figura 98. Interfaz Mis rendiciones (Elaboración propia). 96
Figura 98.a.Interfaz Añadir rendición (Elaboración propia). 97
Figura 99. Interfaz Rendiciones recepción (Elaboración propia). 97
Figura 100. Interfaz Rendiciones gerencia (Elaboración propia). 98
Figura 101. Interfaz Rendiciones contabilidad (Elaboración propia). 98
Figura 102. Dominio (Elaboración propia). 99
Figura 103. Dominio personalizado (Elaboración propia). 100
Figura 104. Repositorio (Elaboración propia). 100
Figura 105. Consulta JPA en repositorio personalizado (Elaboración propia). 101
Figura 106. Capa de servicio (Elaboración propia). 101
Figura 107. Métodos del recurso (Elaboración propia). 102
Figura 108. Métodos de la capa de servicio (Elaboración propia). 102
Figura 109. Recurso para obtener las rendiciones (Elaboración propia). 103
Figura 110. Resultado de la solicitud (Elaboración propia). 103
Figura 111. Recurso personalizado (Elaboración propia). 104
Figura 112. Resultado de un recurso personalizado (Elaboración propia). 104
Figura 113. Rendición ingresada exitosamente. 105
Figura 114. Lista de rendiciones (Elaboración propia). 105
Figura 115. Añadir rendición (Elaboración propia). 105
Figura 116. Visualización de imagen (Elaboración propia). 106
Figura 117. Imagen guardada (Elaboración propia). 106
Figura 118. Seleccionar imagen (Elaboración propia). 106
Figura 119. Vista recepción con el estado recepción (Elaboración propia). 107
Figura 120. Rendición recepcionada. 107
Figura 121. Recepcionar rendición. 107
Figura 122. Vista recepción (Elaboración propia). 107
Figura 123. Rendición aceptada por gerencia (Elaboración propia). 108
Figura 124. Aceptar rendición (Elaboración propia). 108
Figura 125. Vista gerencia (Elaboración propia). 108
Figura 126. Vista gerencia con el estado aprobado (Elaboración propia). 108
Figura 127. Vista contabilidad (Elaboración propia). 109
Figura 128. Detalles de la rendición para confirmar el pago (Elaboración propia). 109
Figura 129. Confirmación del pago (Elaboración propia). 109
Figura 130. Mis rendiciones (Elaboración propia). 110
Figura 131. Rendición rechazada (Elaboración propia). 110
Figura 132. Confirmación de la solicitud. 110
Figura 133. Rechazar rendición (Elaboración propia). 110

Figura 134. Solicitud no disponible (Elaboración propia). 111
Figura 135. Campos incorrectos (Elaboración propia). 111
Figura 136. Campos incorrectos (Elaboración propia). 111
Figura 137. Ver imagen (Elaboración propia). 111
Figura 138. Ver imagen(Elaboración propia). 111

Resumen

Este documento plantea una nueva forma de desarrollo de sistemas, la cual está enfocada a la arquitectura orientada a microservicios. El objetivo de este informe es introducir al lector en el patrón de arquitectura, sus ventajas, características y además exponer el desarrollo e integración de nuevos módulos desarrollados bajo el patrón de arquitectura. Los conceptos definidos en este informe se demostrarán en el sistema informático llamado Zentown, de la consultora informática Zenta.

El informe se dividirá en tres grandes bloques:

- Definición de la arquitectura monolítica.
- Definición de la arquitectura orientada a microservicios.
- Análisis y desarrollo de módulos bajo el patrón arquitectura orientado a microservicios.

Introducción

El desarrollo ágil de software tiene como propósito impulsar el desarrollo más flexible, adoptar la capacidad de poder cambiar en tiempo de ejecución, añadir o eliminar funcionalidades, además de contar con versiones del producto funcional que se puede mostrar al cliente conforme con lo que se va mejorando y adaptando. Para lograr este desarrollo es necesario un enfoque ágil, flexible y con una retroalimentación constante del estado del proyecto o software en cuestión, básicamente esto se traduce en desarrollos incrementales e iterativos. Como indica (Laínez, 2014) en el desarrollo ágil, el software en construcción tiene que adoptar la capacidad de "acostumbrarse a los cambios", ofreciendo rapidez para realizar modificaciones a partir de la retroalimentación de los usuarios.

El patrón de arquitectura orientado a microservicios es un paradigma en el desarrollo de software, el cual propone que el desarrollo de aplicaciones deben ser fragmentadas de tal manera que mejore su disponibilidad, permite que se facilite el mantenimiento, las modificaciones y la escalabilidad cuando es necesario aumentar el número de procesos o módulos dentro de un software, esto permite que la arquitectura se acerque a la filosofía de los métodos ágiles.

Se conoce como un software o aplicación monolítica aquel producto informático que tiene todo sus componentes ejecutándose en una mismo servidor o algún tipo de contenedor para aplicaciones, además su código fuente es robusto y se encuentra en una misma base de código, esto quiere decir que no es fragmentado y se conecta a un único origen de dato. Como indica el proveedor internacional TI SaM Solutions, un software monolítico es una unidad cohesiva de componentes. Todo sistema informático monolítico de gran tamaño sufre inconvenientes de disponibilidad, mantenimiento y escalabilidad que se tornan difíciles a medida que el software crece, las actualizaciones de código se vuelven tediosas, obligando a los desarrolladores a necesitar más esfuerzo en adaptar las nuevas líneas de código, pasa exactamente lo mismo con la escalabilidad, se dificulta el

crecimiento, ya que si es necesario que el software crezca, los desarrolladores están obligados a replicar todo los componentes de la aplicación por cada nueva instancia de software necesitada, es decir, se tendrá que adaptar o escalar el software de manera general y no por partes. Por ende la arquitectura orientada a microservicios plantea romper el desarrollo tradicional y desarrollar bajo pequeñas aplicaciones orquestadas de forma automatizadas, con funcionalidades o propósitos específicos de tal manera que solo sea necesario modificar, adaptar o replicar partes pequeñas y no todo el sistema.

Planteamiento del problema

Pregunta de análisis: ¿Cómo se puede integrar el desarrollo de módulos en un sistema informático sin tener que detener el funcionamiento total del sistema?

Se acostumbra a desarrollar sistemas con una arquitectura monolítica, la cual se enfoca en trabajar bajo un diseño centralizado todos sus procesos, flujos de datos y el almacenamiento de información, los cuales son implementados en un mismo componente de software. Generalmente están construidos a medida, esto implica que son eficientes y cumplen con lo requerido, pero por lo mismo existe una carencia de flexibilidad para soportar diferentes cambios o adaptaciones.

Reflexionando lo anterior, se propone una nueva forma de desarrollar sistemas o aplicaciones informáticas con la arquitectura orientada a microservicios, que permite romper el esquema de trabajar bajo un diseño centralizado, permitiendo lograr un desarrollo mucho más distribuido al segregar sus procesos.

Problema a resolver

Normalmente se desarrollan software con arquitectura monolítica, esto implica tener un software consolidado en una sola gran unidad. Con el transcurso del tiempo, el software necesita crecer, integrar nuevos módulos y adaptarse a los cambios, por ello un software bajo un desarrollo centralizado o monolítico se le dificultará la integración de cambios, ya que será necesario analizar el modelo de negocio e intentar adaptar los nuevos cambios al modelo ya implementado.

La arquitectura orientada a microservicios rompe el esquema de desarrollar software en una sola gran unidad, dividendo sus componentes, sus módulos y su funcionalidad total en pequeños microservicios que se orquestan de tal forma que se logra la funcionalidad total del sistema. Desarrollar módulos bajo este patrón de arquitectura puede ayudar en la siguiente manera:

- Desarrollo de forma distribuida.
  Indicador de eficiencia: Al desarrollar de forma distribuida en base a microservicios se logra el desacoplamiento de los componentes, permitiendo un sistema más escalable y legible.

- Manejo y control de versiones.
  Indicador de eficiencia: Cada microservicio debe estar alojado en un contenedor y cada contenedor se puede ir versionando de acuerdo a las necesidades, permitiendo manejar y controlar mejor los componentes de una aplicación.

- Fragmentación de código fuente.
  Indicador de eficiencia: El software se debe dividir en diferentes paquetes de código permitiendo un mayor manejo del software o módulos.

- Heterogeneidad en tecnologías.
  Indicador de eficiencia: Permite y soporta una mayor gama de tecnologías para el desarrollo del software o módulos.

- Disminuye el acoplamiento de los componentes de la aplicación, facilitando la flexibilidad y escalabilidad de forma independiente.
  Indicador de eficiencia: Cada microservicio debe tener funcionamiento autónomo, se tiene que desplegar de forma independiente de los demás componentes del software.

- Ofrece procesos de integración automatizados.
  Indicador de eficiencia: Las tareas de compilación, pruebas y despliegues se deben ejecutar de forma automática ante cualquier cambio en las bases de código.

- Mayor control y detección de fallos por medio de herramientas automatizadas.
  Indicador de eficiencia: Cada microservicio será monitoreado por herramientas automatizadas que permiten detectar errores y controlar de forma automática cada microservicio.

Estado del arte

Para iniciar el tema de los microservicios, primero se tiene que regresar al año 1994 en donde Alexander Pasik, un antiguo analista de Gartner definió el término SOA (Services Oriented Architecture) antes que se inventaran los servicios XML o Web. Pasik fue impulsado a crear el término por el motivo que cliente/servidor había perdido su significado clásico (Josuttis, 2007).

Según Hamza Naji y Mohammad Mikki un sistema SOA debe contar con tres servicios principales. Ver figura 1

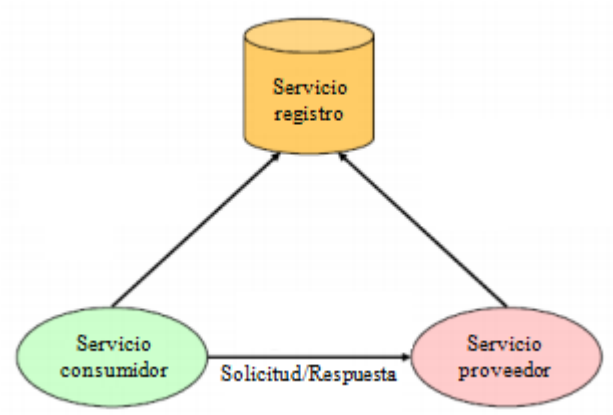


Figura 1. Componentes y procesos, SOA.

Conforme al centro de conocimiento de IBM (IBM Knowledge Center) algunas de las características de SOA son:

- Una unidad única ejecutable de código.
- Son diseñados para una tarea específica.
- Deben estar disponible en la red.
- Cada servicio debe tener una interfaz y acceder a ella debería ser suficiente para utilizar el servicio.

Además para aliviar los problemas de heterogeneidad, interoperabilidad y requisitos cambiantes, en el capítulo Service-oriented architecture desarrollador por IBM-2004, indica que se debe seguir estas características:

- Ligeramente acoplado.
- Ubicación transparente.
- Protocolo independiente.

Según el proveedor líder en soluciones de software de código abierto Red Hat, la arquitectura orientada a servicio se describe como aquellos componentes de aplicación que se comunican para proporcionar servicios a otros componentes a través de la red y además su funcionalidad se divide en servicios web. La SOA permite crear sistemas frágiles que dependen de una infraestructura compleja y además se crea una dependencia con el proveedor.

Por este motivo se dice que la arquitectura orientada a microservicios es un mejora de la arquitectura orientada a servicios (SOA), ya que se busca separar toda la lógica de negocio en pequeños procesos totalmente independientes del resto y así lograr un desacoplamiento total entre los componentes de la aplicación. Estos procesos se pueden comunicar con otros bajo peticiones sencillas y estandarizadas, basadas en HTTP. Además el formato de datos y protocolos pesados se sustituyen por JSON ligero a través de una API REST. Cada microservicio dispone de su propio almacén de datos, por ende la gestión y la persistencia centralizadas no son necesarias (Red Hat, 2016). Ver figura 2

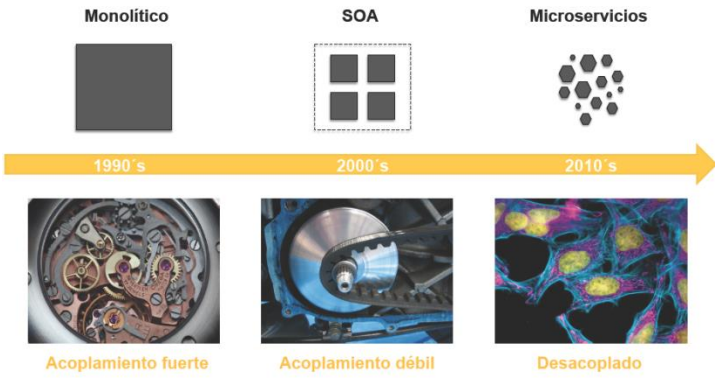


Figura 2. Evolución del acoplamiento en las distintas arquitecturas.

Capítulo 1

1.1 Definición arquitectura monolítica

El proveedor internacional de IT SaM Solutions indica que es un patrón de desarrollo con el cual se crea un software en una única base de código, es decir, solo se tiene que compilar una vez. Por esto es que una aplicación con un desarrollo monolítico se define como una unidad cohesiva de código, por ende sus componentes trabajan en conjuntos y utilizan el mismo espacio de memoria y recursos. Ver figura 3.

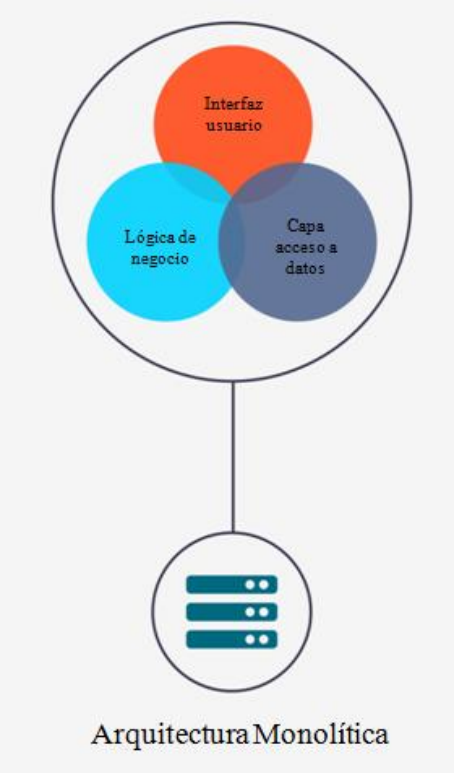


Figura 3.Representación de una Arquitectura Monolítica.

1.1.1 Principales características

- El código desarrollado para los procesos de todo el sistema se almacena en una única base de datos, siempre está la opción de poder dividir el código en diferentes clases y paquetes que permite obtener una mayor estructura, pero los módulos no se dividen, funcionan como una única unidad.
- Si se requiere algún cambio dentro de la aplicación es necesario analizar y modificar completamente el software, se debe integrar por partes pequeñas, es decir que las partes no pueden funcionar de manera independiente es necesario una integración completa al sistema.
- Los componentes dentro de una aplicación pueden comunicarse fácilmente entre sí mediante el uso de procedimientos internos.

A continuación se describirán una lista de ventajas y desventajas según la experiencia del equipo de profesionales experimentados en ámbitos informáticos Oodles Technologies:

1.1.2 Principales ventajas

- Se logra una velocidad de desarrollo temprano más rápido.
- Dado que todos los IDE están diseñados para admitir una sola aplicación, son fáciles de desarrollar e implementar en un servidor.
- Mientras el software no deba enfrentarse a cambios, la arquitectura funciona correctamente.
- Pruebas simples, como la aplicación es implementada solo una vez, por lo tanto todo el sistema es testeado en una sola instancia, no es necesario esperar dependencias o servicios adicionales, esto simplifica el proceso, es decir, menos tiempos.

1.1.3 Principales desventajas

- En las primeras fases del desarrollo, la arquitectura monolítica funciona bien, pero a medida que el tamaño de la aplicación comienza a crecer y se acumulan varios módulos, aparece la limitación de esta arquitectura.

- Disminución de iteraciones a largo plazo, es decir, que con el paso del tiempo el software será más difícil de mantener, se complicara la adaptación a los cambios,y en el peor de los casos no podrá seguir escalando.

1.1.4 Definición modelo vista controlador

Según la IJCA(2014) es un patrón de diseño que proporciona una forma de mantener vistas múltiples y sincronizadas conectadas a un controlador que maneja los eventos del usuario, sus componentes están estrechamente acoplados. El patrón de diseño divide las tareas e interacciones de un software en tres componentes generales:

- **Modelo:** Maneja el núcleo del sistema. Es el que interactúa directamente con el origen de datos y es independiente de la interfaz de usuario.
- **Vista:** Es el componente encargado de desplegar la información al usuario y es donde se habilitan las interacciones entre el software y usuario.
- **Controlador:** Es el componente que maneja la entrada del usuario desde la vista, validando la petición del usuario, el cual solicitara al modelo la información correspondiente a lo que solicito el usuario.

Ver figura 4 y 5.

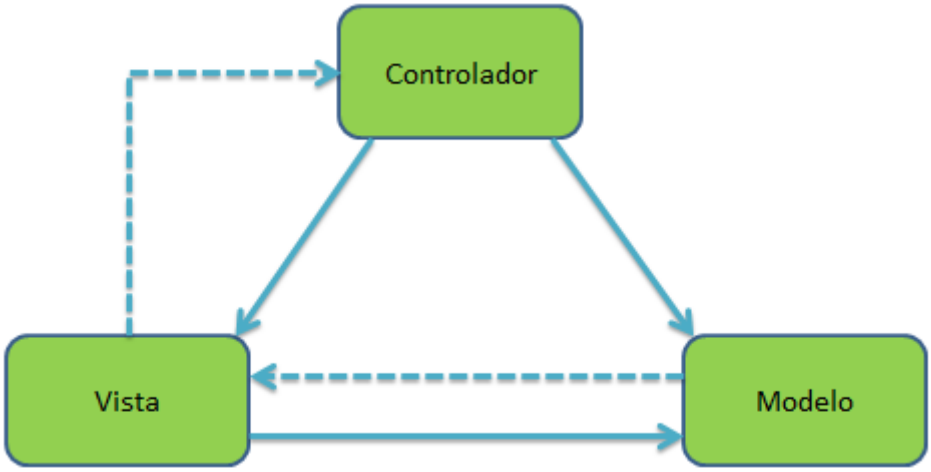


Figura 4. Patrón MVC (Blancarte,2014)

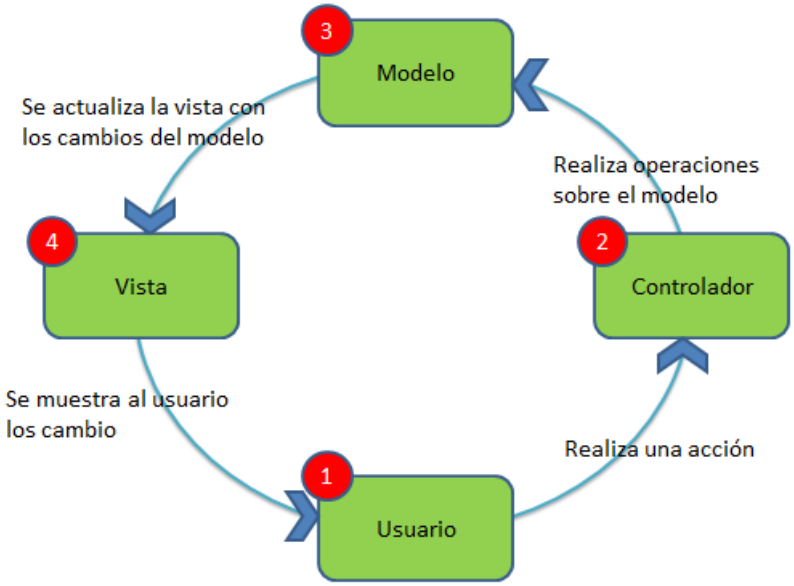


Figura 5. Interacción de los componentes (Blancarte, 2014).

Ilustraciones de una aplicación con arquitectura monolítica y con un patrón de diseño-MVC

A continuación se visualizará como está construido un software monolítico con un patrón de diseño Modelo Vista Controlador. El concepto MVC se concentrara en solo una tabla (person) de la aplicación para efectos de visualización. Ver figura 6

Tabla: person

	#	Nombre	Tipo
☐	1	rut_person	varchar(200)
☐	2	id_type_user	int(11)
☐	3	id_user	int(11)
☐	4	name_person	varchar(200)
☐	5	address_person	varchar(200)
☐	6	mail_person	varchar(200)
☐	7	url_photo	varchar(200)

Figura 6. Tabla person.

Variables de configuración para lograr una conexión a la base de datos

Para lograr la conexión es requerido configurar el host, username, password y database, con lo que se logra la conexión al origen de dato, cabe destacar que la aplicación solo utiliza un motor de base de datos (MySQL). Ver figura 7

```
'Datasources' => [
    'default' => [
        'className' => 'Cake\Database\Connection',
        'driver' => 'Cake\Database\Driver\Mysql',
        'persistent' => false,
        'host' => 'localhost',
        /**
         * CakePHP will use the default DB port based on the driver selected
         * MySQL on MAMP uses port 8889, MAMP users will want to uncomment
         * the following line and set the port accordingly
         */
        //'port' => 'non_standard_port_number',
        'username' => 'my_app',
        'password' => 'secret',
        'database' => 'my_app',
        'encoding' => 'utf8',
        'timezone' => 'UTC',
        'flags' => [],
        'cacheMetadata' => true,
        'log' => false,
```

Figura 7. Conexión a una única base de datos (Modo Centralizado).

1.1.5 Descripción de un modelo

Según la IJCA(2014) el modelo es el componente que encapsula los datos y es la única entidad que interactúa con la base de datos.

Se puede observar en la figura 8, como es un modelo a nivel de código:

A. Constructor de la clase en donde se declara que tabla se utilizará de la base de datos, por ende esta clase pasa a convertirse el modelo de la tabla Person.

B. Función de la clase en donde se declaran todos los atributos que contenga la tabla. Esta parte es la representación a nivel de código.

```php
class PersonTable extends Table
{

    /**
     * Initialize method
     *
     * @param array $config The configuration for the Table.
     * @return void
     */
    public function initialize(array $config)
    {
        parent::initialize($config);

        $this->setTable('person');
        //$this->setDisplayField('rut_person');
        $this->setPrimaryKey('rut_person');
    }

    /**
     * Default validation rules.
     *
     * @param \Cake\Validation\Validator $validator Validator instance.
     * @return \Cake\Validation\Validator
     */
    public function validationDefault(Validator $validator)
    {
        $validator
            ->scalar('rut_person')
            ->allowEmpty('rut_person', 'create');

        $validator
            ->integer('id_type_user')
            ->allowEmpty('id_type_user');

        $validator
            ->integer('id_user')
            ->allowEmpty('id_user');

        $validator
            ->scalar('name_person')
            ->allowEmpty('name_person');

        $validator
            ->scalar('address_person')
            ->allowEmpty('address_person');
```

Figura 8. Modelo a nivel código.

Se representa el componente modelo del patrón de diseño como una clase que contiene métodos y variables de configuración para hacer la equivalencia a la tabla "person" del modelo de datos.

1.1.6 Descripción de un controlador

De acuerdo con Blancarte (2014), el controlador es el puente entre la vista y el modelo, ya que desde la vista el usuario solicita alguna petición, el controlador toma esta petición y realiza las operaciones correspondiente.

En la siguiente figura se observa cómo es un controlador a nivel de código. Ver figura 9

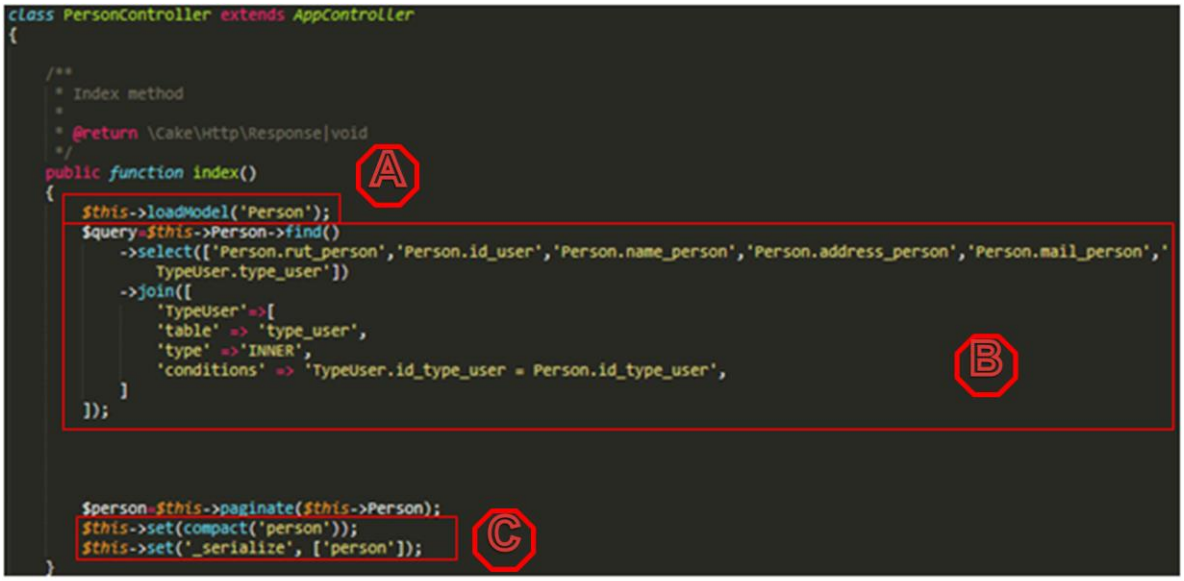


Figura 9. Controlador a nivel código.

A. Método propio del framework que facilita la instancia de los módulos dentro de la aplicación.

B. Flujo en donde se utiliza el modelo para poder obtener información desde la base de datos.

C. Métodos que proporcionan la utilidad de enviar variables a la vista.

1.1.7 Descripción de una vista

Interfaz de usuario para poder visualizar la información obtenida desde el controlador. Ver figura 10.

Figura 10. Vista de usuario

Capítulo 2

2.1 Definición general de la arquitectura orientada a microservicios

"...un estilo de arquitectura en el que el software se desarrolla como servicios pequeños e implementables de forma independiente que trabajan juntos en un entorno de negocio."

Sam Newman

Es un nuevo patrón de arquitectura para el desarrollo de software, el cual permite mantener un conjunto de microservicios independientes. Cada uno de estos microservicios tienen como objetivo mantener la funcionalidad de un software de forma modular, cada microservicio funciona de forma independiente, apuntando a un área específica del modelo de datos. Ver figura 11.

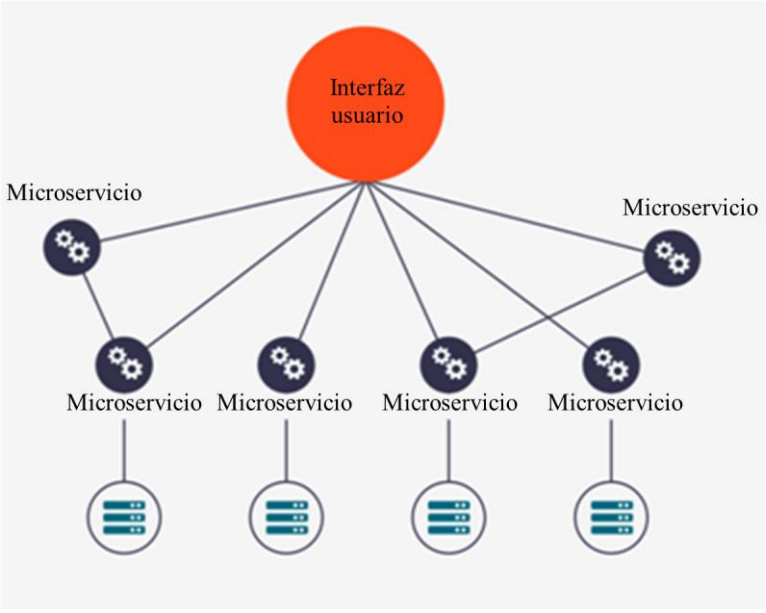


Figura 11. Arquitectura orientada a microservicios

Cada funcionamiento de un módulo está contenido en un microservicio independiente, que al hacerlos funcionar todos en conjunto se obtendrá el funcionamiento completo del software.

2.1.1 Principales ventajas de la arquitectura

- Si hay algún tipo de problema en un módulo del software, no es necesario que los desarrolladores deshabiliten toda la aplicación, solo basta con dar de baja el microservicio que contiene el módulo con problemas, así no se compromete el funcionamiento completo del software.
- La escalabilidad que brinda esta arquitectura es eficiente.
- Proporciona mayor rapidez para detectar algún tipo de incidencia.

2.1.2 Principales desventajas de la arquitectura

- Adaptar un software ya construido a este patrón de arquitectura es muy costoso.
- Mayor tiempo de esfuerzo en aprendizaje de los conceptos necesario para lograr un desarrollo distribuido.

2.2 Definición microservicio

Según Newman, 2015, los microservicios deberían ser caracterizado por ser pequeños y centrados en hacer de forma correcta una tarea específica. Cada microservicio será una pieza de funcionalidad en el software, en donde se limita explícitamente a que irá enfocado cada uno de ellos, por ende se evita que ese microservicio crezca demasiado, para que así no se presenten todas las dificultades que conlleva tener un código base robusto.

Como se menciona anteriormente, son pequeñas aplicaciones que contienen una lógica de negocio y se comportan de forma independiente entre sí, brindando una mayor modularidad al desarrollo de un sistema. El conjunto de estos componentes será necesario para que el software funcione correctamente. Con decir que son pequeñas aplicaciones o de código base pequeño indica que cada microservicio se tiene que caracterizar por ser un código limpio, de manera que se pueda reescribir o modificar en un plazo estimado de dos semanas.(Jon Eaves)

Cada microservicio es una entidad separada, dando como resultado que el sistema sea más distribuido, lo que se traduce en un sistema más fácil de analizar . Estos servicios deberían poder cambiar de forma independiente respecto el uno del otro y ser desplegados de forma automática sin que el desarrollador tenga desplegarlo o compilarlo. (Newman, 2015).

De acuerdo con James Lewis y Martin Fowler, 2014cada microservicio se debe ejecutar en su propio proceso y comunicándose con mecanismos livianos, a menudo con una API que soporte peticiones HTTP. Estos servicios se basan de acuerdo a las capacidades empresariales y se pueden implementar de forma independiente, mediante una plataforma de implementación completamente automatizada. Para ello hay herramientas de integración continua que brindan la opción de automatizar las tareas. Estos microservicios se pueden escribir en diferentes lenguajes de programación y utilizar diferentes tecnologías de almacenamiento de datos.

2.2.1 Principales atributos de un microservicio

- Heterogeneidad en las tecnologías:
    - Como el sistema estará compuesto por múltiples microservicios (Newman, 2015) indica que esto permite elegir la herramienta más adecuada para cada trabajo, en lugar de tener que seleccionar una única herramienta que da un enfoque más estandarizado. Ver figura 12.

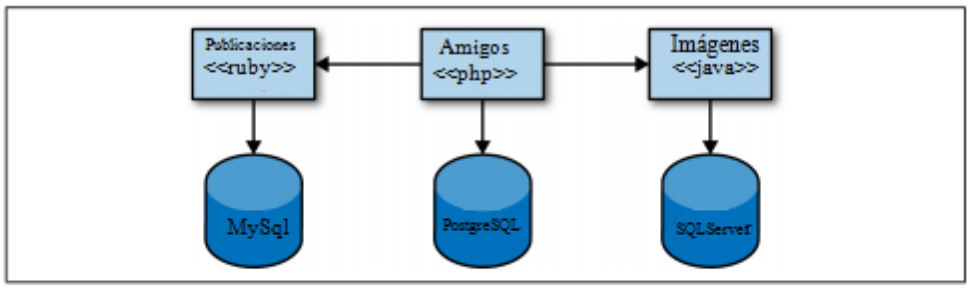


Figura 12. Flexibilidad de tecnologías. (Newman, 2015)

- Escalabilidad:
    - Con servicios más pequeños, podemos escalar solo aquellos servicios que necesitan modificaciones, permitiendo ejecutar servicios independientes sin dañar el sistema principal (Newman, 2015). Ver figura 13.

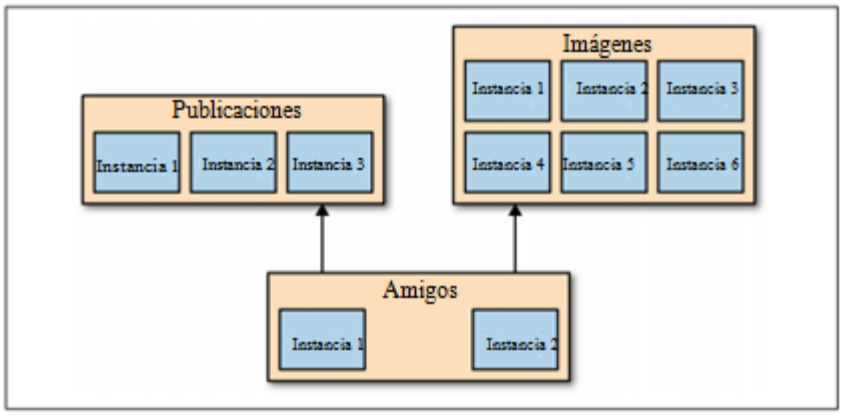


Figura 13. Solo se le da escala a los microservicios que lo necesiten. (Newman, 2015)

- Fácil integración e implementación automática con la ayuda de herramientas de orquestación e integración continua.

- Reusabilidad: Cualquier microservicio se puede usar y reutilizar para diferentes propósitos (Grebianuit De Bulten, 2017).

Ilustraciones de los atributos de un microservicio

Estructura general de un microservicio:

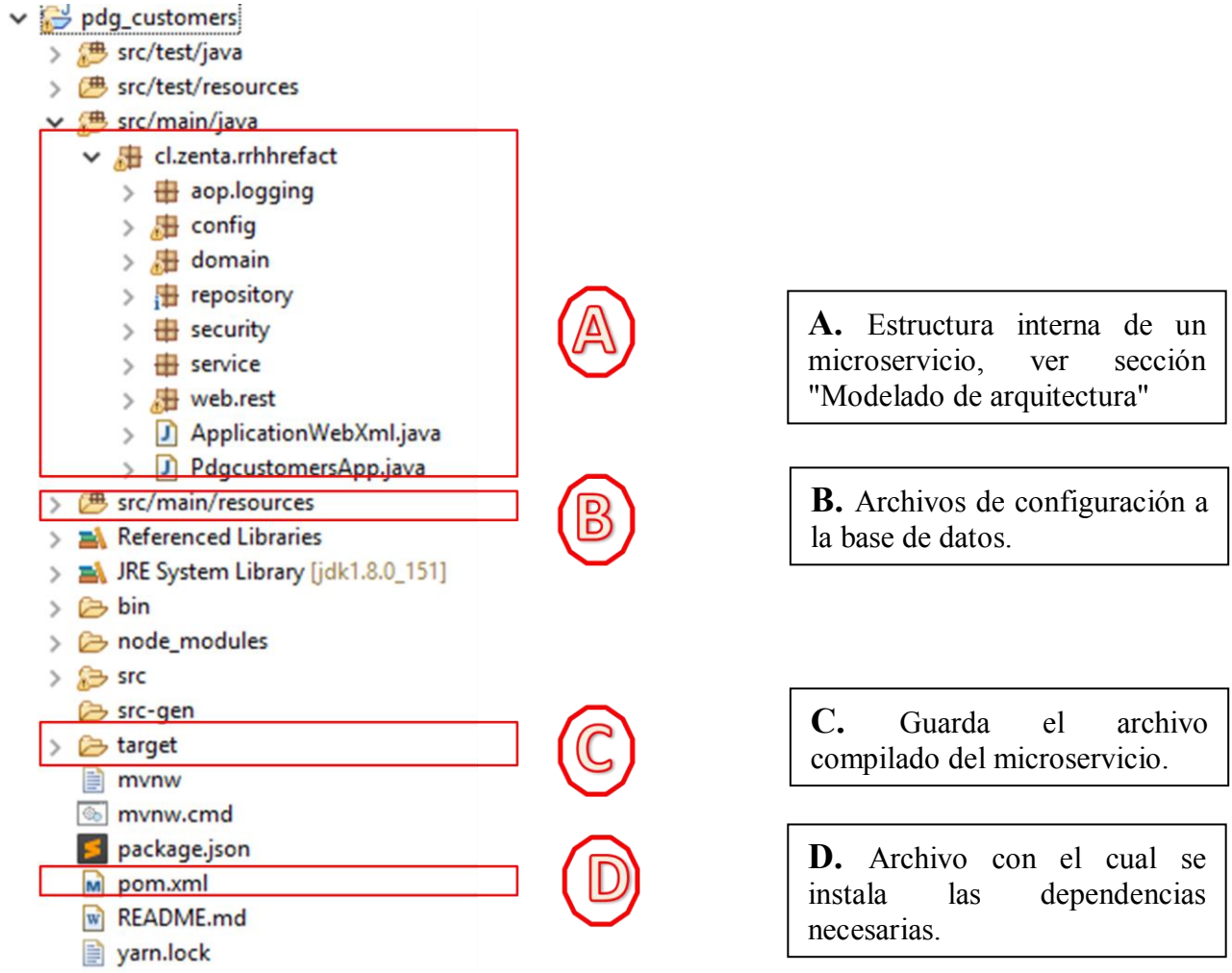


Figura 14.Estructura general de un microservicio desarrollado con JHipster (Java).

Se puede observar en las siguientes figuras la heterogeneidadde un microservicio respecto a la configuración de un motor de base de datos.

```
datasource:
    type: com.zaxxer.hikari.HikariDataSource
    url: jdbc:mysql://ec2-34-234-76-218.compute-1.amazonaws.com/dev_pdg?useUnicode=true&characterEncoding=utf8&useSSL=false
    username: zenta_ccp
    password: 12345678
    hikari:
        data-source-properties:
            cachePrepStmts: true
            prepStmtCacheSize: 250
            prepStmtCacheSqlLimit: 2048
            useServerPrepStmts: true
```

Figura 15. Configuración de un microservicio a un motor de base de datos MySQL.

```
datasource:
    type: com.zaxxer.hikari.HikariDataSource
    url: jdbc:postgresql://34.234.76.218:5432/rrhh_erp?currentSchema=rrhh_erp&assumeMinServerVersion=9.0
    username: jhipster
    password: rcXFZkxr5gmLpy95
```

Figura 16.Configuración de un microservicio a un motor de base de datos PostgreSQL.

Tener el funcionamiento del sistema separados en microservicio provee una mayor escalabilidad a la hora de ir integrando nuevos cambios, con una forma de orquestación automatizada se logra generar la funcionalidad total de un software, la siguiente ilustración muestra varios microservicios para una misma aplicación. Ver figura 17

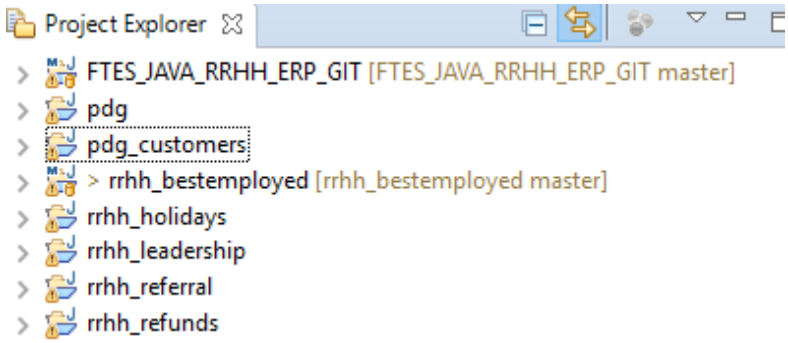


Figura 17.Diferentes microservicios para una misma aplicación.

2.2.2 Estructura de un microservicio

Los microservicios tienen una estructura interna, el cual se compone de unas capas que se mostraran en la siguiente figura:

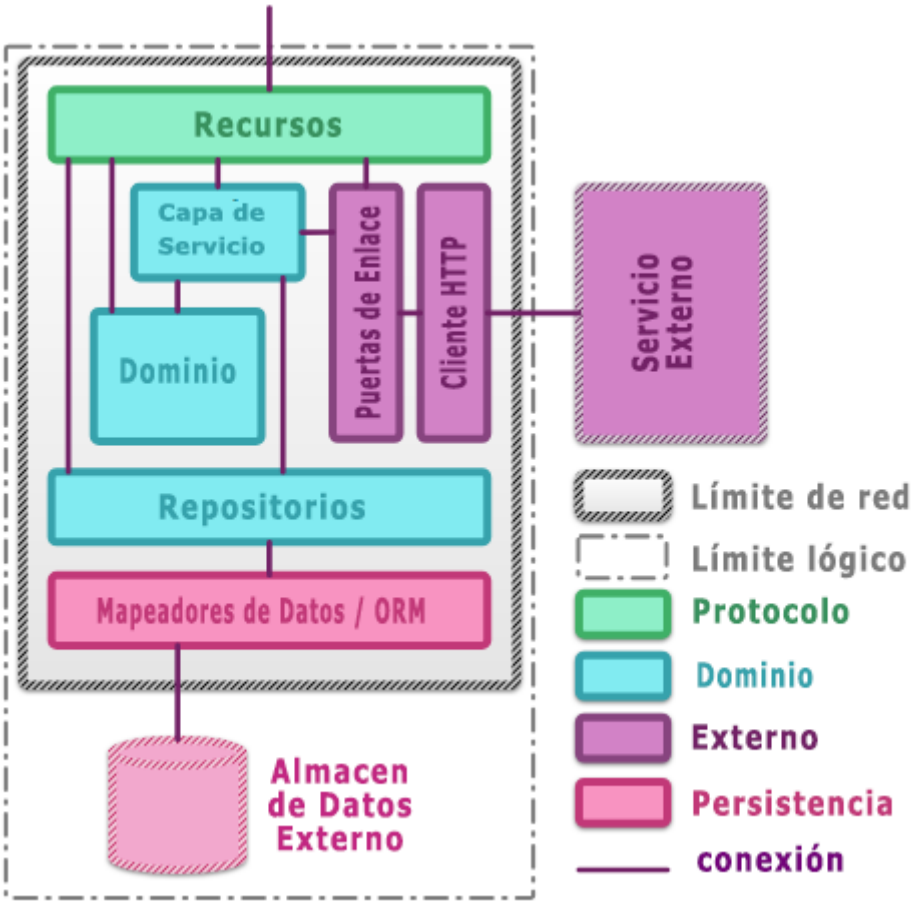


Figura 18. Estructura interna de un microservicio (Clemson, 2014)

2.2.3 Descripción de las capas de un microservicio

Según Clemson y Fowler (2014) colaboradores de ThoughtWorks, se indicará una pequeña descripción de cada capa de la arquitectura.

Protocolo

- **Recursos**
  - Los recursos actúan como mapeadores entre el protocolo expuesto por el servicio y los objetos que son representados por el dominio. Tienen como objetivo verificar las peticiones de red y proveer una respuesta específica al protocolo de acuerdo al resultado de la solicitud. Un recurso cumple con la funcionalidad de recibir una petición y una vez que esté validada, llama a la capa del servicio el cual empieza a procesar la petición.

Dominio

- **Capa de servicio**
  - La capa de servicio actúa como un mediador por si la aplicación requiere coordinar varios servicios para poder completar alguna petición más personalizada. Si la petición no necesita una lógica de negocio mayor, esta petición pasa directo al repositorio que corresponde.
- **Repositorios**
  - El repositorio actúa junto a la colección de entidades de dominios. Este repositorio tiene como objetivo almacenar en memoria las listas de los objetos del dominio que serán necesario para poder manipular la base de datos.
- **Dominio**
  - El dominio representa un objeto que asocia a una entidad de la base de datos.

Persistencia

- **Mapeadores de dato / ORM**
    - Para poder persistir los objetos del dominio entre las peticiones, se utiliza un mapeo objeto relacional. La base de datos es acoplada al sistema, pero como se encuentra por fuera de los límites de la red pueden presentarse problemas de latencia y riesgo de interrupción.

Externo

Si un servicio requiere funcionalidad de otro servicio, se necesita cierta lógica para comunicarse con el servicio externo. Una puerta de enlace encapsula un mensaje para transmitirlo hacia un servicio remoto, transformando solicitudes y respuestas desde y hacia objetos del dominio. Todo tipo de conexión a servicios externo tienen la posibilidad de algún tipo de riesgo, esto es porque cruzan el límite de la red. Para ello la puerta de enlace debería contener la lógica necesaria para mitigar los posibles fallos.

2.2.4 Representación de la estructura interna de un microservicio

Para efecto de visualización se mostrara la estructura interna de un microservicio desarrollado en lenguaje Java con la gama de tecnologías que provee Jhipster.

Dominio

De acuerdo con Clemson y Fowler (2014) el dominio representa un objeto que hace referencia a una entidad del origen de dato. Ver figura 19.

Person.java

```
10
11 /**
12  * A Person.
13  */
14 @Entity
15 @Table(name = "person", schema="rrhh_erp")
16 @Cache(usage = CacheConcurrencyStrategy.NONSTRICT_READ_WRITE)
17 public class Person implements Serializable {
18
19     private static final long serialVersionUID = 1L;
20
21     @Id
22     @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "sequenceGenerator")
23     @SequenceGenerator(name = "sequenceGenerator")
24     private Long id;
25
26     @Column(name = "name")
27     private String name;
28
29     @Column(name = "lastname")
30     private String lastname;
31
32     @Column(name = "second_name")
33     private String secondName;
```

Figura 19. Representación de un dominio que referencia a la entidad person.

Además de representar una tabla o entidad del origen de dato, en el caso que se necesite un dominio más personalizado, con alguna respuesta de mayor complejidad, se pueden añadir más dominios con formato personalizados. Ver figura 20.

CustomPerson.java

```java
import java.time.ZonedDateTime;

public class CustomPerson {

    private Long id;
    private Long idCommune;
    private String address;
    private String allergies;
    private String bloodType;
    private String cardTickets;
    private String civilStatus;
    private String contactEmergency;
    //private ZonedDateTime created;
    private String diaseases;
    private ZonedDateTime dob;
    private String documentNumber;
    private Long documentType;
    private String email;
    private int gender;
    private String glasses;
    private String group;
    private String name;
    private String lastname;
    private String secondName;
    private String motherLastname;
    private ZonedDateTime modified;
    private String nacionality;
    private String personalEmail;
    private String phone;
    private String phoneEmergency;
    private String visa;
    private List<ResponsabilityCustom> responsabilities;
    private List<AnswerCustom> answers;
    private CustomAfpPerson afpPerson;
    private CustomIsaprePerson isaprePerson;

    public Long getId() {
        return id:
```

Figura 20. Representación de un dominio personalizado.

Repositorio

El repositorio permite utilizar una colección de objeto(dominio) para lograr manipular la base de datos. La ilustración 21 muestra una interface de java que hereda métodos de la tecnología JPA que brinda las operaciones básica a una base de datos. Ver figura 21.

PersonRepository.java

```java
package com.zentagroup.rrhh.repository;

import com.zentagroup.rrhh.domain.Person;


/**
 * Spring Data JPA repository for the Person entity.
 */
@SuppressWarnings("unused")
@Repository
public interface PersonRepository extends JpaRepository<Person,Long> {

}

```

Figura 21. Representación de un repositorio que hereda métodos de JpaRepository.

En el caso que se requiera alguna operación más compleja a la base de datos, es posible utilizar un repositorio que se adapte a las necesidades. Ver figura 22.

PersonCustomRepository.java

```java
package com.zentagroup.rrhh.repository.custom;

import com.zentagroup.rrhh.domain.Person;


/**
 * Spring Data JPA repository for the Person entity.
 */
@SuppressWarnings("unused")
@Repository
public interface PersonCustomRepository extends JpaRepository<Person,Long> {

    @Query(value = "select p from Person p where p.documentNumber = ?1")
    Person findPersonByDocumentNumber(String rut);

    @Query("SELECT p FROM PersonPositionLevel ppl JOIN ppl.person p WHERE ppl.id = ?1")
    Person getPersonByPositionLevel(Long positionLevelId);

    @Query("select p from Person p where lower(p.name) like lower(concat('%',?1,'%')) or lower(p.secondName)
    List<Person> findByNameFree(String name);

}

```

Figura 22. Repositorio personalizado.

Capa de servicio

Cuando hay necesidad de desarrollar una lógica de negocio más personalizada, la capa de servicio permite manipular diferentes tipos de dominios y repositorios. Ver figura 23.

PersonCustomService.java

```
1  package com.zentagroup.rrhh.service.custom;
2
3  import com.zentagroup.rrhh.domain.Address;
18
19
20 @Service
21 @Transactional
22 public class PersonCustomService {
23     private final Logger log = LoggerFactory.getLogger(PersonCustomService.class);
24
25     private final PersonCustomRepository personRepository;
26     private final AddressRepository addressRepository;
27     private final CommuneRepository communeRepository;
28     private final DocumentTypeRepository documentTypeRepository;
29     private final PersonPositionLevelCustomRepository personPositionLevelCustomRepository;
30
31     public PersonCustomService(PersonCustomRepository personRepository, AddressRepository addressRepository,
32                                PersonPositionLevelCustomRepository personPositionLevelCustomRepository) {
33         this.personRepository = personRepository;
34         this.addressRepository = addressRepository;
35         this.communeRepository = communeRepository;
36         this.documentTypeRepository = documentTypeRepository;
37         this.personPositionLevelCustomRepository = personPositionLevelCustomRepository;
38     }
39
40     public Person createPerson(CustomPerson customPerson) {
79
80     public BasicInfoPersonCustom getPersonByDocumentNumber(String rut){
88
89 }
90
```

Figura 23. Representación de una clase java como capa de servicio.

Recursos

Para poder desplegar la información solicitada por una petición de algún protocolo externo, se utilizan los recursos. Estos actúan como mapeadores entre el protocolo externo y los demás componentes de la estructura interna. Ver figura 24.

PersonResource.java

```
 */
@RestController
@RequestMapping("/api")
public class PersonResource {

    private final Logger log = LoggerFactory.getLogger(PersonResource.class);

    private static final String ENTITY_NAME = "person";

    private final PersonRepository personRepository;

    public PersonResource(PersonRepository personRepository) {
        this.personRepository = personRepository;
    }

     * GET  /people : get all the people.
    @GetMapping("/people")
    @Timed
    public List<Person> getAllPeople() {
        log.debug("REST request to get all People");
        return personRepository.findAll();
    }

     * POST  /people : Create a new person.
    @PostMapping("/people")
    @Timed
    public ResponseEntity<Person> createPerson(@RequestBody Person person) throws URISyntaxException {
        log.debug("REST request to save Person : {}", person);
        if (person.getId() != null) {
            return ResponseEntity.badRequest().headers(HeaderUtil.createFailureAlert(ENTITY_NAME, "idexists",
        }
        Person result = personRepository.save(person);
        return ResponseEntity.created(new URI("/api/people/" + result.getId()))
            .headers(HeaderUtil.createEntityCreationAlert(ENTITY_NAME, result.getId().toString()))
            .body(result);
```

Figura 24. Recurso básico, métodos CRUD.

Como en los casos anteriores, también existe la posibilidad de construir un recurso personalizado de acuerdo a las necesidades del sistema.. Ver figura 25.

PersonCustomResource.java

```
 */
@RestController
@RequestMapping("/api/custom/")
public class PersonCustomResource {

    private final Logger log = LoggerFactory.getLogger(PersonCustomResource.class);

    private static final String ENTITY_NAME = "person";

    private final PersonRepository personRepository;
    private final CommuneRepository communeRepository;
    private final AddressRepository addressRepository;
    private final DocumentTypeRepository documentTypeRepository;
    private final PersonCustomService personCustomService;
    private final ResponsabilityRepository responsabilityRepository;
    private final PersonCustomRepository personCustomRepository;
    private final ContractCustomRepository contractCustomRepository;
    private final AnswerRepository answerRepository;
    private final QuestionRepository questionRepository;
    private final AfpPersonCustomService afpPersonCustomService;
    private final IsaprePersonCustomService isaprePersonCustomService;

    private EntityManager entityManager;

    public PersonCustomResource(PersonCustomService personCustomService,PersonRepository personRepository, CommuneRepository

    public ResponseEntity<HashMap<String,HashMap<String,String>>> getInfoByDocument()

    @GetMapping("/idPersonByDocumentNumber/{rut}")
    public ResponseEntity<BasicInfoPersonCustom> getPeopleByDocumentNumber(@PathVariable String rut)
    {
        BasicInfoPersonCustom person=personCustomService.getPersonByDocumentNumber(rut);
        if(person.getPerson().getActive()==1){
            return ResponseEntity.ok().body(person);
        }
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
    }

```

Figura 25. Representación de un recurso personalizado.

Puerta de enlace

Provee la funcionalidad de hacer peticiones a los recursos del microservicio y además es la puerta de enlace para acceder a todos los microservicios necesarios para el funcionamiento del software. Ver figuras 26-27-28.

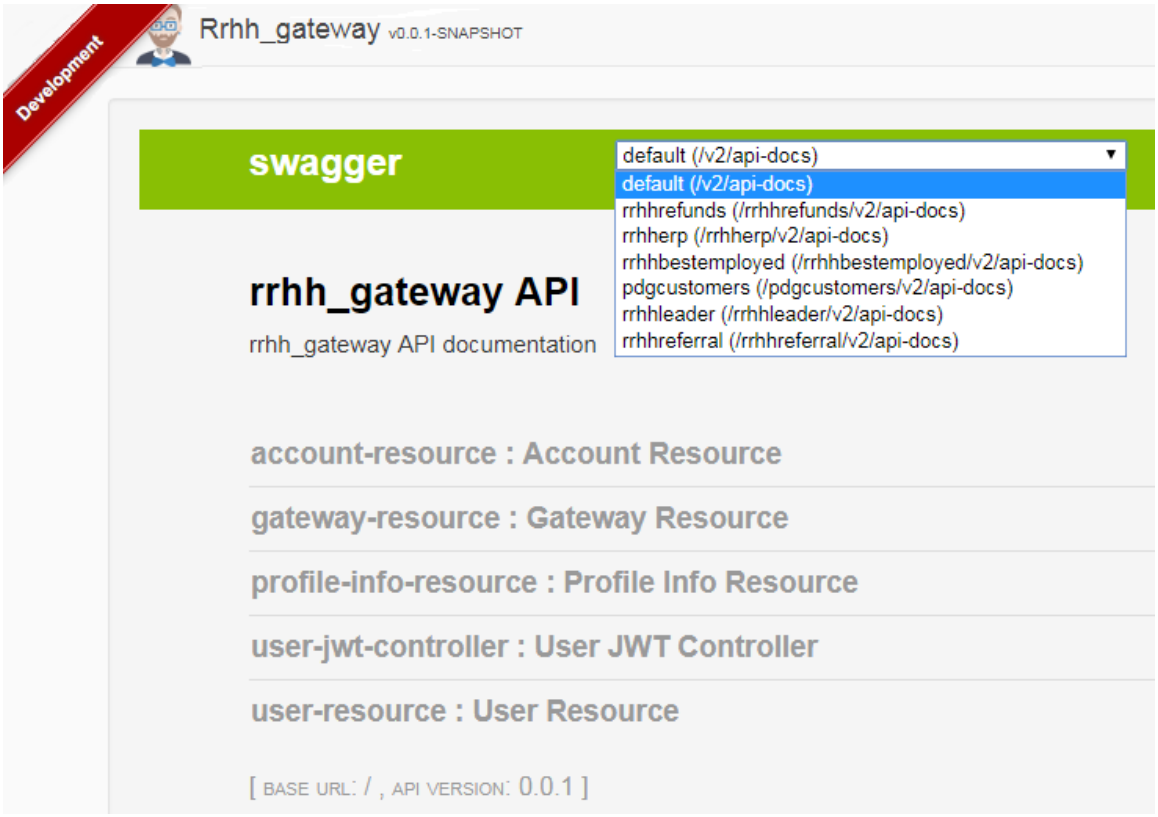


Figura 26. Gateway con microservicios disponibles.

Imágenes que representan el despliegue de los recursos que provee la puerta de enlace o gateway.

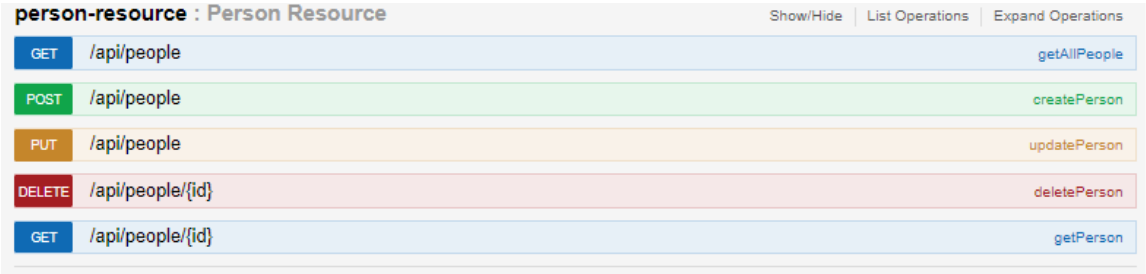


Figura 27. Recursos para solicitar peticiones básicas mediante HTTP.

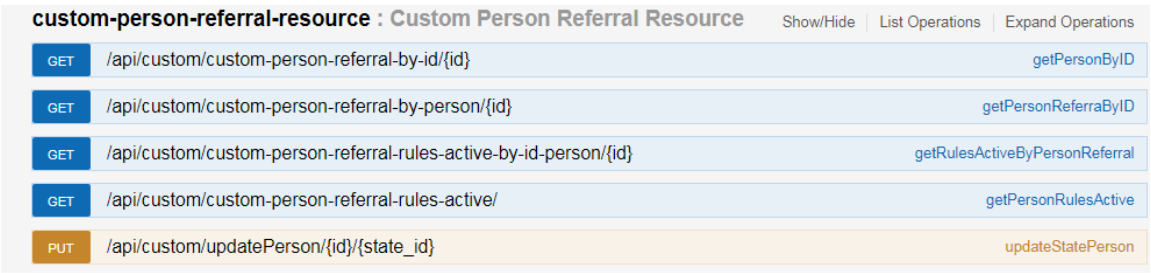


Figura 28. Recursos para solicitar peticiones personalizadas mediante HTTP.

Capítulo 3

3.1 Definición de automatización para microservicios

Para lograr una arquitectura orientada a microservicios es muy importante preguntarse: ¿Cómo es posible controlar y/o integrar miles de microservicios al mismo tiempo?. Según Newman, 2015,uno de los aspectos más importante que debe contener la arquitectura, es la integración y automatización de los microservicios, en donde cada microservicio tendrá su autonomía. Esto permite cambiar y liberarlos de forma independiente. Como indica Microsoft Azure (Mayo, 2018) es necesario manipular alguna herramienta de orquestación para contenedores, en donde cada contenedor tendrá alojado de forma independiente un microservicio.

3.1.1 ¿Qué son los contenedores?

Según Microsoft Azure (Mayo, 2018) los contenedores son componentes encapsulados individualmente que se pueden ejecutar como instancias aisladas en un mismo sistema operativo, esto quiere decir que cada aplicación o microservicio tendrá su tiempo de ejecución, dependencias y bibliotecas alojadas en un contenedor. El uso de contenedores beneficia la portabilidad y el aislamiento de los componentes en un sistema. En unos de los artículos de Red Hat (2018) indica que los contenedores Linux, son un conjunto de procesos que están separados del resto del sistema. Cada microservicio alojado en un contenedor se podrá ejecutar desde una imagen, la cual es una versión compilada del microservicio, esta contiene las dependencias necesarias para el funcionamiento correcto, brindando portabilidad y consistencia a la hora del cambio de entornos de trabajo, es decir el paso desde el ambiente de desarrollo al ambiente de producción.

3.1.2 Características de los contenedores

Algunas características de los contenedores según Microsoft Azure (Mayo, 2018)

- Pequeño: Utilizan un único espacio de almacenamiento.
- Rápido: No se debe iniciar un sistema operativo completo para la ejecución de un contenedor, esto beneficia la rapidez de uso.
- Portabilidad: Un contenedor que aloje una imagen de un microservicio se puede portar para que se ejecute en una nube o directamente en un maquina física. Esta característica ayuda en el proceso de cambio de etapas desde desarrollo a producción.

3.2 Diferencia entre virtualización y contenedores

La virtualización permite crear múltiples entornos simulados o recursos dedicados en un solo sistema de hardware físico. El sistema llamado hipervisor se conecta directamente al sistema de hardware físico y esto permite dividir un sistema en entornos virtualizados, separados y distintos, esto es conocido como máquinas virtuales. Con esto se puede lograr un entorno de desarrollo dividido, pero al tener múltiples máquinas virtuales ejecutándose en un mismo hipervisor(software que permite la virtualización de las máquinas virtuales) puede ser costoso, además no es tan ligero como usar contenedores. Los contenedores se ejecutan desde el mismo sistema operativo, es decir, comparten el mismo kernel y aíslan los procesos de los microservicios del resto del sistema. Además los contenedores comparten el sistema operativo para que los servicios se mantengan ligeros y se ejecuten en paralelo.

En la siguiente figura se puede observar la diferencia entre virtualización y contenedores:

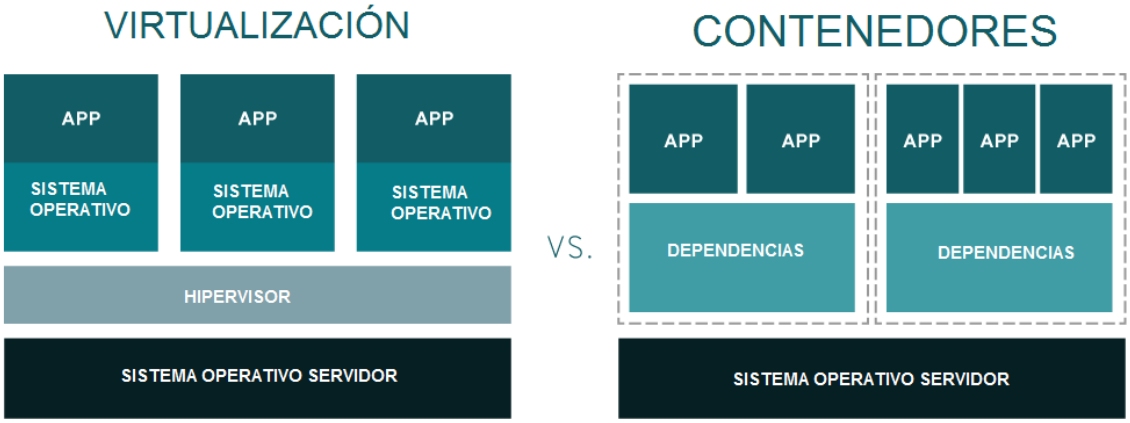


Figura 29. Virtualización vs Contenedores (Red Hat, 2018).

3.3 Docker

En el 2008 apareció una nueva tecnología de uso libre para el control de contenedores llamada Docker. Esta tecnología combina el trabajo de los contenedores tradicionales (Linux) con herramientas mejoradas para desarrolladores, con el fin de facilitar el manejo de los contenedores. Docker es actualmente el proyecto y el método más conocido para implementar los contenedores Linux(RedHat, 2018).

La tecnología Docker usa las funciones del kernel de Linux para segregar procesos, permitiendo que se puedan ejecutar de manera independiente. Esta independencia es la intención de los contenedores, la capacidad de ejecutar varios procesos o microservicios separados unos de los otros. Además Docker ofrece un modelo de implementación basado en imágenes, esto permite compartir, mover un microservicio o un conjunto de microservicios con todas sus dependencias en diferentes entornos de desarrollo.

Ilustración entre un contenedor tradicional Linux vs Docker:

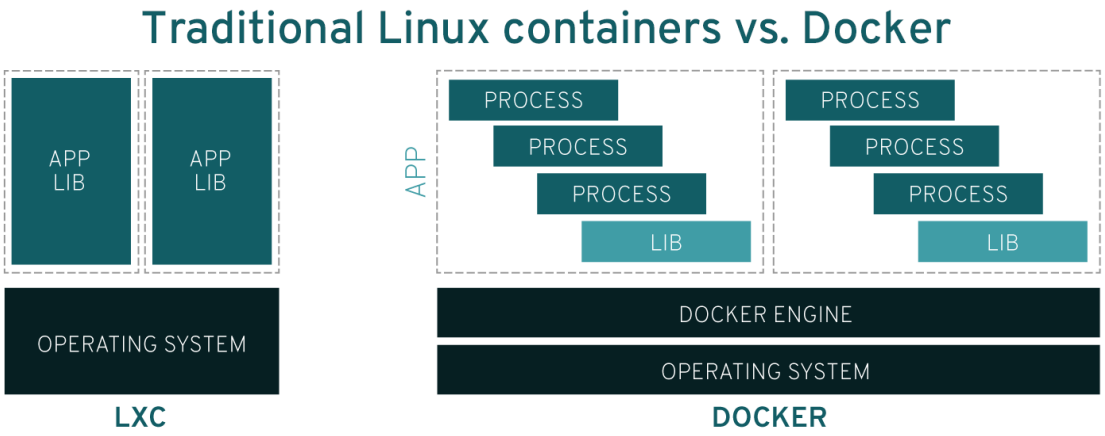


Figura 30. Contenedor Linux vs Docker.

3.3.1 Principales características de los contenedores Docker:

- Modularidad: Capacidad de tomar una parte de una aplicación, poder actualizarla y repararla, sin necesidad de modificar toda la aplicación.
- Control de versiones y reutilización de componentes: Brinda la opción de rastrear diferentes versiones de un mismo contenedor, además de inspeccionar, diferenciar o retroceder a versiones anteriores.
- Compartir: Se puede tener un repositorio remoto que almacene todos los contenedores y a si poder compartir el despliegue de información.

Docker en sí mismo facilita la gestión de contenedores únicos, pero en caso de que se comience a necesitar cada vez más contenedores de manera progresiva, la gestión de los contenedores se vuelve complicada. Cuando el número de contenedores de una aplicación incrementa constantemente, es donde aparece la orquestación de los contenedores.

3.4 Kubernetes

Es una plataforma de uso libre creada por Google en el año 2014, fue diseñada para la orquestación de contenedores y la automatización en el despliegue, escalado y administración eficiente de las aplicaciones, con funcionalidades tales como re-arranque automático, auto-replicación y auto-escalado.

Los contenedores y Kubernetes alientan a los desarrolladores a construir aplicaciones distribuidas, esto lleva a los principios de la infraestructura inmutable, que consiste en que una vez ya creado un componente, este ya no cambiara mediante modificaciones de los desarrolladores(Burns, Hightower, Beda, 2017).

Es posible pensar en esta herramienta como:

- Una plataforma de contenedores
- Una plataforma de microservicios.
- Una plataforma portátil en la nube.

Esta plataforma está orientada a entregar un entorno para el control de contenedores, en donde la persistencia de los servicios de red, almacenamiento y disponibilidad está diseñado para ser gestionados de forma automática y simplificada.

3.4.1 Principales características de Kubernetes(Burns, Hightower, Beda, 2017)

- Velocidad: Consiste en la cantidad de información que se puede enviar mientras los servicios se encuentran altamente disponibles, en este caso los contenedores y Kubernetes proporcionan la herramienta necesaria para realizar cambios rápidamente mientras los servicios se encuentran disponibles.

- Crecimiento: El desacoplamiento de componentes hace que sea más fácil escalar una aplicación, ya que tiene la capacidad de integrar nuevos componentes sin la necesidad de reconfigurar ninguno de los otros. Además el crecimiento es mayor ya que cada equipo de desarrollo puede enfocarse en un microservicio.

- Abstracción: Cuando los desarrolladores eligen construir aplicaciones en términos de contenedores y utilizar Kubernetes para la orquestación de estos, se logra una abstracción en la portabilidad y transferencia entre distintos tipos de entornos, es decir funciona igual en una maquina física o en una nube.

- Eficiencia: Kubernetes proporciona herramientas para automatizar la distribución de microservicios, asegurando niveles de utilización más altos que las herramientas tradicionales. Para crear un entorno de pruebas para el desarrollador, en el pasado habría significado el uso de tres máquinas, con Kubernetes se puede crear rápida y económicamente como un conjunto de contenedores que se ejecutan en una vista de un cluster compartido.

Cluster: conjunto de máquinas físicas o virtuales utilizados para el control de los microservicios

Capítulo 4

4.1 Diferencias entre arquitectura monolítica y arquitectura orientada a microservicios.

4.1.1 Sistema monolítico.

Figura que representa como está construido un sistema monolítico con un patrón de diseño modelo-vista-controlador.

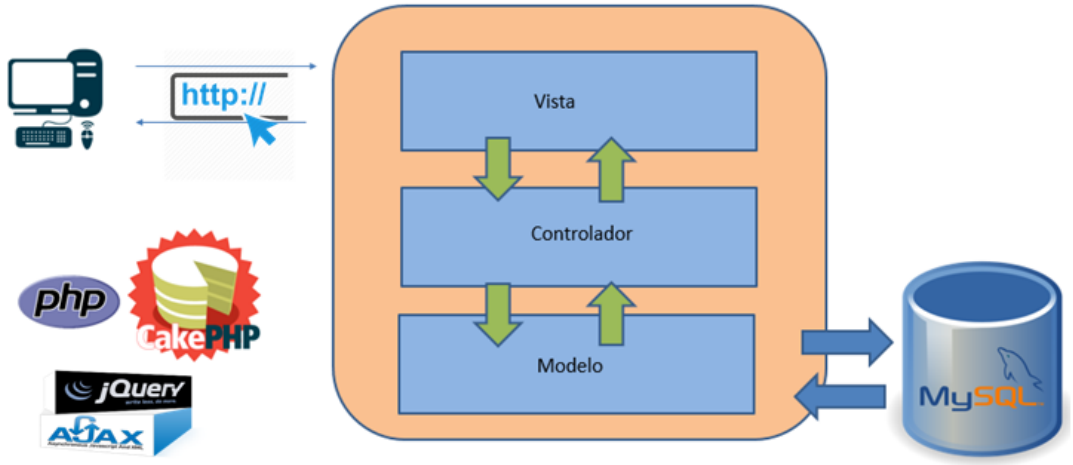


Figura 31.Representación de un Sistema Monolítico

- Única unidad cohesiva de procesos.
- Aplicación desarrollada a medida y bien definida.
- Entornos de desarrollo rígidos.
- Poca flexibilidad a tecnologías.
- Alto acoplamiento de componentes.
- Dificultad de mantener cuando el software se vuelve más complejo.
- Desarrollo temprano más rápido.

4.1.2 Sistema orientado a microservicios.

Imagen que representa un sistema bajo la arquitectura orientada a microservicios

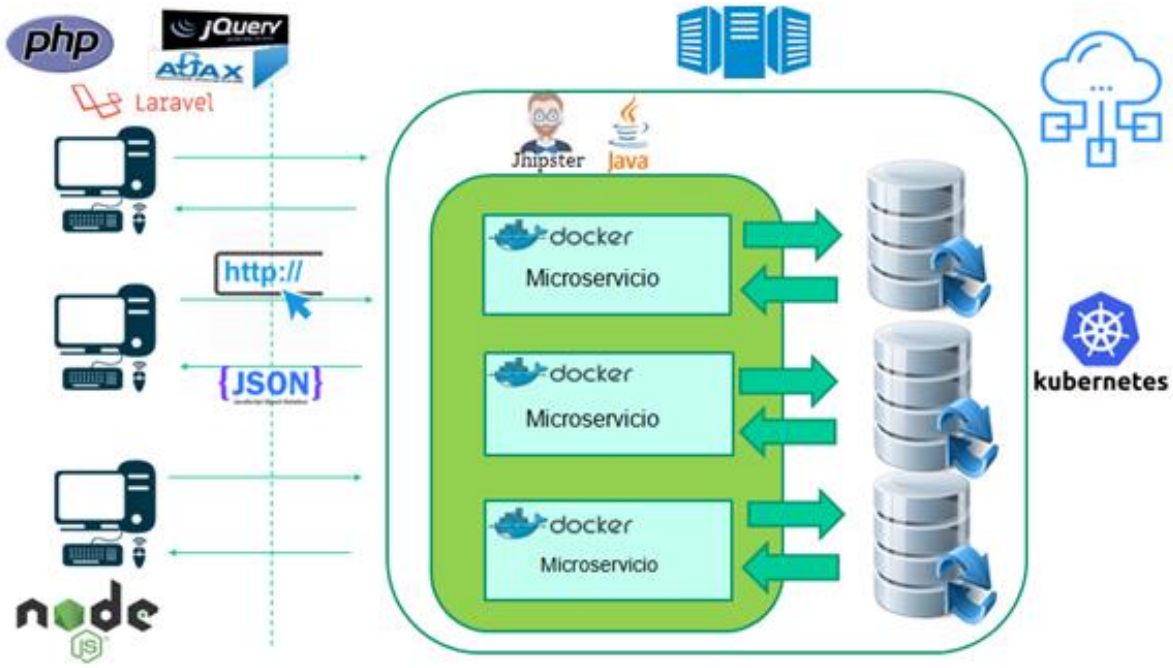


Figura 32.Representación de un Sistema orientado a microservicios

- Tiempo de esfuerzo en la adaptación a los conceptos.
- Desarrollo distribuido.
- Alto desacoplamiento de los componentes.
- Numerosas opciones de tecnologías.
- Procesos automatizados.

Es importante tener en cuenta las variables de productividad y complejidad que están presentes a la hora de comparar aplicaciones monolíticas y aplicaciones con microservicios. El porqué de un sistema monolítico se vuelve menos productivo de mantener a medida que el sistema aumenta su complejidad, mientras que un sistema basado en microservicios es más productivo de mantener a medida que aumenta su complejidad, se ve reflejado en la siguiente figura (Fowler, 2015).

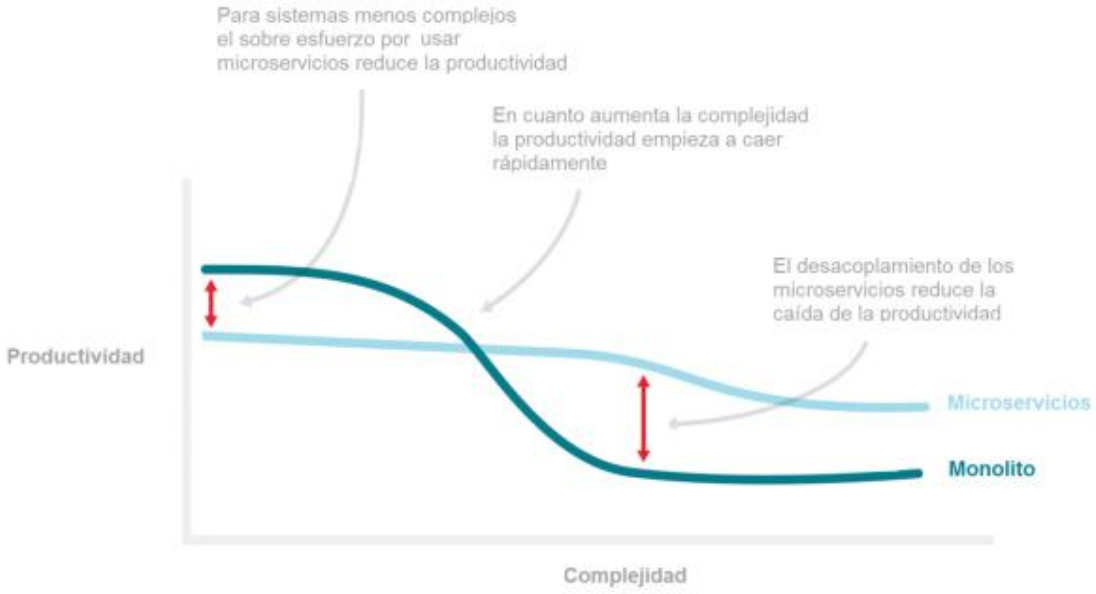


Figura 33. Productividad vs complejidad. (Fowler, 2015)

En el caso de los sistemas monolíticos, están construidos con un alto acoplamiento entre sus componentes, los efectos que esto produce son la baja sensibilidad a fallos, altos costos de escalabilidad, despliegues menos automáticos y riesgosos, además se dificulta las tareas de mantenimiento. Por esto se dice que entre más complejo se vuelve un sistema monolítico la productividad de mantener un software decrece.

Capítulo 5

5.1 Análisis y desarrollo de módulos

Objetivo general

Desarrollare integrar módulos bajo la arquitectura orientada a microservicios para el sistema Zentown.

Objetivos específicos

- Definir y aplicar los conocimientos de la investigación para el desarrollo de módulos bajo el enfoque de la arquitectura orientada a microservicios.
- Implementar nuevas tecnologías para lograr el desarrollo correcto de los microservicios.
- Desarrollar los siguientes módulos bajo el patrón de arquitectura para el sistema Zentown:
    - Módulo 1: Referenciados.
    - Módulo 2: Rendiciones.

Ambiente de ingeniería de software

Metodología de desarrollo.

En el proceso del desarrollo de los módulos se utiliza la metodología "Iterativo Incremental". Ya que la empresa adopta esta forma de trabajo, una vez tomados los requerimientos, se trabajan en base a los métodos de análisis, diseño, codificación y pruebas, pero aplicados de forma iterativa hasta que el cliente esté satisfecho con sus requerimientos. Cabe destacar que existe un constante contacto con el cliente. Ver figura 33.

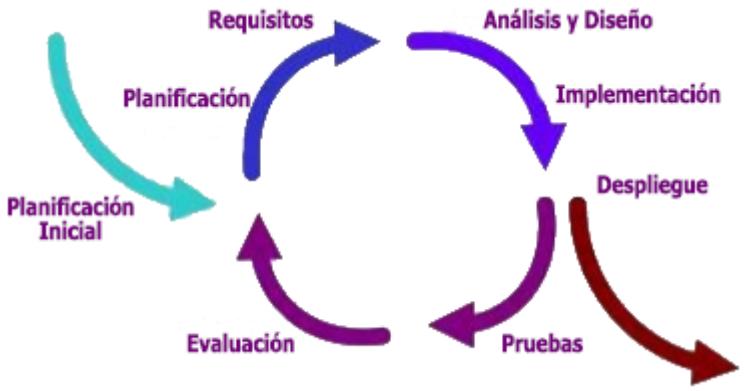


Figura 34. Modelo Iterativo

Ventaja de esta metodología:

- Por cada iteración, se afinan los detalles que pueda tener el sistema o los módulos en desarrollo.
- Una mayor mitigación de riesgos a la hora de entregar el producto final o módulos en este caso.
- Una mejor retroalimentación para los requerimientos.

Tecnologías.

Se describirá brevemente cada una de las tecnologías a utilizar durante el desarrollo de los módulos.

- **PHP Versión 5.4:** Es el acrónimo de Hyper text Pre-Processor. Es un lenguaje gratuito que nos permite generar códigos en el lado del servidor.
- **HTML:** Es la sigla que define a Hyper Text Markup Language. Es un lenguaje de etiquetas para elaborar páginas web
- **JavaScript:** Es un lenguaje de programación interpretado por el navegador, esto quiere decir que se utiliza en el lado del cliente para mejorar la interfaz y páginas web dinámicas.
- **JQuery:** Es una biblioteca de JavaScript que permite simplificar la interacción con los elementos HTML, permite manipular el DOM, manejar eventos, interacción con AJAX.
- **AJAX:** El acrónimo de **A**synchronous **J**avaScript **A**nd **X**ML, es una técnica que permite hacer llamadas desde el lado del cliente asíncronas con el lado del servidor, permitiendo realizar cambios en el cliente sin tener que recargar la página completa.
- **CSS:** Las siglas de Cascading Style Sheet. Es una hoja de estilo que permite controlar el aspecto visual a las páginas web creada con HTML.
- **Java:** Es un lenguaje de programación orientado a objetos con el fin de de permitir a los desarrolladores codificar un programa una vez y ejecutarlo en cualquier dispositivo, conocido en inglés como el acrónimo **WORA** o "*write once, run anywhere*", esto quiere decir que no es necesario recompilar el código para ejecutar dicha aplicación.

- **Jhipster:** Es un generador de código basado en Yeoman, para crear aplicaciones o en este caso microservicios que vienen siendo pequeñas aplicaciones, con una lista tecnologías tanto para el lado del cliente y del servidor.
- **MySQL:** Es una de las base de datos de código abierto más popular en el mercado, como un gestor de base de datos relacional, multiusuario y multihilo.
- **PostgreSQL:** Es un potente sistema de base de datos objeto-relacional de código abierto, con más de treinta años de desarrollo activo, le ha valido una sólida reputación de fiabilidad robustez de las características y rendimientos.
- **Git:** Es un software de control de versiones, el cual define como control de versiones a la gestión de diversos cambios que se realizan sobre algún software. Como desarrollador constantemente se está actualizando el código, entonces Git nos brinda la opción de subir lo desarrollado a una nube, permite subir los cambios, descargarlos y actualizarlos.
- **Docker:** Es una tecnología que segrega procesos, es decir, que se puede ejecutar varios procesos de forma independiente, cada proceso(microservicio) es "guardado" en un contenedor , con esto se define la funcionalidad que nos brinda docker, mantener varios contenedores ligeros, portables e independientes para alojar todo lo que necesite el software en desarrollo. Docker es un software de uso libre.
- **WampServer:** Es un entorno de desarrollo web que permite tener nuestro propio servidor o host local, cabe destacar que es un software libre.
- **Sublime Text:** Un sofisticado editor de texto para código multiplataforma.
- **WinSCP:** Es un software libre, permite la transferencia segura de archivos entre dos sistemas informáticos, el local y uno remoto que ofrezca servicios ssh.
- **SourceTree:** Es un cliente GUI(graphical user interface) para manejar los repositorios GIT.
- **PgAdmin III:** Es una aplicación grafica para gestionar el gestor de base de datos PostgresSQL, es un software libre.

- **PHPMyAdmin:**Es un software de código abierto, diseñado para manejar la administración y gestión de bases de datos MySQL a través de una interfaz gráfica de usuario, escrito en PHP.
- **Eclipse:** Es una plataforma de desarrollo el cual no tiene como objetivo trabajar con un lenguaje especifico, sino más bien un software genérico, pero tiene popularidad entre los desarrolladores codificar en java usando el plug-in JDT que viene incluido en la versión estándar del software.
- **Framework Laravel Versión 5.4:**Es una herramienta para desarrollar aplicaciones con PHP además proporciona potentes herramientas necesarias para construir aplicaciones robustas y que puede ser utilizado tanto para proyectos a nivel empresarial como para proyectos más sencillos, lo que significa que es perfecto para todos los tipos de proyectos.

Descripción del sistema

El sistema Zentown es un software desarrollado por la consultora Zenta, el cual es enfocado a una red social corporativa. Tiene la visión de ayudar a agilizar procesos y automatizar otros. También permite manejar documentos personales y certificados. Contiene un inicio que muestra secciones de información, como por ejemplo una sección de noticias destacadas, una sección en donde se visualizan las publicaciones realizadas por los usuarios del sistema, entre otras. Permite realizar publicaciones y adjuntar imágenes en ellas, además los usuarios pueden reaccionar y dejar comentarios en una publicación de otro usuario. El inicio de Zentown contiene otra sección donde se informa el ganador del talento del mes, se da a conocer a los nuevos colaboradores que ingresan a la empresa y contiene una galería de fotos en la que se va mostrando fotografías tomadas en eventos y actividades de la empresa. El propósito principal del desarrollo de este software es potenciar aún más la comunicación entre los colaboradores.

Bajo este software se aplicará los conceptos estudiados de la arquitectura orientada a microservicios. Para esto se crearan e integrarán nuevos módulos al software.

5.2 Descripción de los módulos

Introducción a la necesidad que requiere cada módulo a desarrollar.

5.2.1 Módulo referenciados

El módulo referenciados va enfocado a un programa de referidos, este programa de referidos tiene la intención de reclutar nuevos colaboradores bajo diferentes modalidades, existen dos tipos de modalidades para el programa, la modalidad libre consiste en que cualquier colaborador puede referir libremente candidatos de acuerdo al perfil de la empresa y la modalidad On Demand que es utilizada para referir a personas específicas, con habilidades más particulares, con diferente nivel de experiencias. Bajo estas modalidades existen diferentes reglas asociadas al cargo de la persona candidata, un ejemplo de regla para el cargo Analista o Desarrollador, si es que el candidato bajo este cargo supera los seis meses de permanencia en la empresa al colaborador referente será premiado con un bono de $60.000 (sesenta mil pesos), una vez superado el año y medio de permanencia, nuevamente se premiara al colaborador referente con un bono de $80.000 (ochenta mil pesos), otro ejemplo de reglas para los cargo Arquitecto de software, Líder de proyecto o Gerente, si el candidato supera los seis meses de permanencia al colaborador que refirió se le premiara con un bono de $80.000 (ochenta mil pesos), si el candidato supera el año y medio de permanencia se premiara nuevamente con un bono de $100.000 (cien mil pesos).

5.2.2 Módulo rendiciones

El módulo rendiciones es para ayudar a la gestión de boletas y facturas de compras, en donde a los colaboradores autorizados se les permite realizar compras y luego rendir aquella compra con su respectiva boleta o factura.

En un plazo establecido por la empresa, a los colaboradores se les devuelve el monto final de las compras realizadas durante el mes.

Para ser efectivo el pago de las rendiciones del mes completo, es necesario un control para las boletas o facturas que se rinden, primero el colaborador debe acercarse a recepción para entregar las boletas o facturas para que la persona en recepción pueda ir a entregar estas rendiciones a gerencia, una vez que gerencia revisa y aprueba las rendiciones estas son trasladadas a contabilidad, en donde contabilidad da la última confirmación que todo está en orden y efectúa el pago.

Para esto se necesita desarrollar un módulo que permita agilizar el proceso, permitiendo al colaborador subir de forma digital las boletas o facturas.

El proceso que se describe anteriormente será controlador atreves de un módulo para rendiciones.

5.3 Desarrollo del módulo referenciados

El módulo para el programa referidos cuenta con los siguientes objetivos.

5.3.1 Objetivo general

Diseñar e implementar un módulo bajo la arquitectura orientada a microservicios, que permita referenciar a personas como nuevos colaboradores.

5.3.2 Objetivos específicos

- El módulo permitirá a un colaborador a ingresar a un nuevo candidato.
- El módulo permitirá realizar el proceso de contratación del nuevo candidato.
- El módulo permitirá identificar que reglas tendrá activa cada candidato.
- El módulo permitirá tener un registro de los candidatos y sus respectivas reglas.

5.3.3 Descripción de interfaz.

Interfaz Mis referenciados

El módulo presenta una interfaz de usuario que contiene la lista de los candidatos de acuerdo al colaborador que este iniciado en el sistema, además incluye un botón para añadir un nuevo candidato ingresando los datos y el curriculum.

Interfaz Administrador

Interfaz de usuario que contiene una lista con todos los candidatos ingresado al sistema, permitiéndole al administrador con diferentes botones controlar el proceso de contratación de un candidato.

Interfaz Histórico

Interfaz de usuario que visualizara a todos los colaboradores contratados bajo el programa referidos, permitiendo ver en qué regla se encuentra cada colaborador.

Interfaz Grafico

Interfaz de usuario que incluye la lista de cargos disponibles en el sistema, el cual al seleccionar un cargo se filtrara la información y mostrara un gráfico con estadísticas para ver cuántos candidatos existen por el cargo seleccionado.

5.3.4 Requerimientos funcionales

Id	Nombre	Descripción
RF-01	Ingresar candidato	Registrar un nuevo candidato.
RF-02	Flujo contratación	Poner en espera, Aceptar o rechazar al nuevo candidato.
RF-03	Identificar reglas	Asignar reglas automáticamente al nuevo candidato.
RF-04	Grafico	Grafico para ver el % de candidatos por un cargo.
RF-05	Historial	Mantener un historial de los candidatos con reglas vigentes.

Tabla 1. Requisitos funcionales del módulo.

5.3.5 Requerimientos no funcionales.Acá puede ir otra idea del documento.

Id	Nombre	Descripción
RNF01	Arquitectura	Se desarrollara bajo la arquitectura orientada a microservicios.
RNF02	Lenguaje de programación	Se utilizara principalmente Java, PHP, JavaScript.
RNF03	Gestor de Base de Datos.	Se utilizara PostgreSQL y MySQL.

Tabla 2. Requisitos no funcionales del módulo.

5.3.6 Análisis del caso de uso modulo referenciados

En esta sección del informe se identifican los actores principales que interactuaran con el módulo además se describe el contexto del caso de uso. Por otro lado se ilustrará diagramas de secuencias para visualizar la interacción entre el sistema y el microservicio.

Actores

- Administrador
    - Tiene acceso total al módulo.
- Recursos Humanos
    - Se le permite las acciones de aprobar, rechazar al candidato y ver el historial de reglas para todos los candidatos contratados.
- Colaborador o Referente
    - Se le permite ingresar un nuevo candidato al sistema.

Descripción

En la figura 34 se aprecia como los diferentes actores pueden interactuar con las distintas funcionalidades del módulo, en donde el actor "Colaborador" realiza una interacción que le permite ingresar un nuevo candidato, se puede ver como el actor "Recurso Humanos" puede controlar el proceso de un candidato, además de poder ver el histórico de los colaboradores, el actor "Administrador" tiene acceso a todas las acciones nombradas anteriormente y además puede interactuar con el grafico de estadísticas.

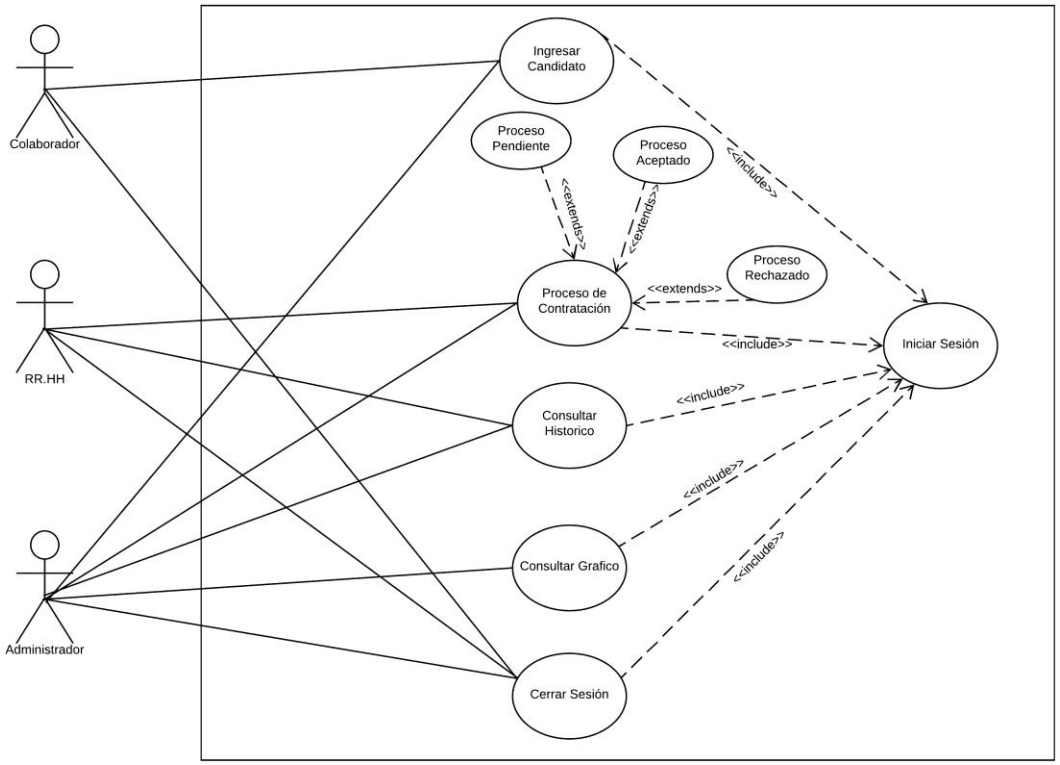


Figura 35. Diagrama de caso de uso para el módulo referenciados (Elaboración propia).

5.3.7 Especificación de casos de uso

A continuación se detallan todos los casos de uso explicando que función cumple, su pre y su post condiciones y el flujo de eventos respectivo.

Caso de uso Ingresar Candidato:

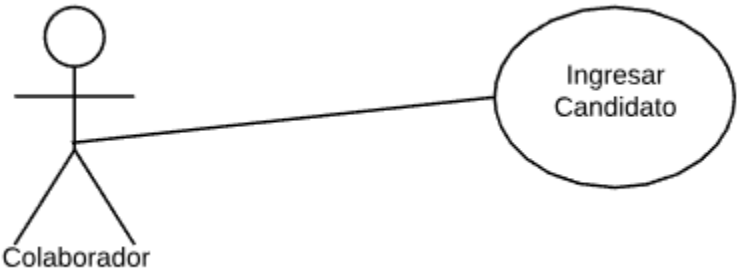


Figura 36. Caso de uso ingresar candidato al sistema (Elaboración propia).

En la tabla **3** se detallara el caso de uso Ingresar Candidato.

Tabla3. Antecedentes del caso de uso Ingresar Candidato.

Nombre	Ingresar Candidato
ID	CU-001
Descripción	El usuario debe ingresar su candidato al sistema Zentown, para que la solicitud quede en estado pendiente para su aprobación, posteriormente el administrador la acepta y queda asociado el usuario al referenciado con el bono correspondiente.
Actores Principales	Usuario que ingresa candidato
Actores Secundarios	-
Pre-Condiciones	Ingresar al sistema
Flujo principales	1) El usuario ingresa su nombre de usuario y contraseña al sistema Zentown. 2) El sistema comprueba que el usuario tiene un perfil asociado al sistema. 3) El usuario selecciona dentro del menú, mis referenciados. 4) El usuario selecciona añadir referencia. 5) El usuario ingresa los datos del referenciado al sistema Zentown. 6) El usuario ingresa el curriculum en formato PDF al sistema Zentown. 7) La solicitud queda en estado pendiente para ser aprobada en el sistema Zentown.
Post-Condiciones	Solicitud en estado pendiente para un candidato
Flujo alternativo	2.- El sistema comprueba que el usuario no tiene un perfil asociado -Responde con mensaje de reintentar ingresar al sistema.

Caso de uso Proceso contratación:

En la tabla **4** se detallara el caso de uso Proceso Contratación.

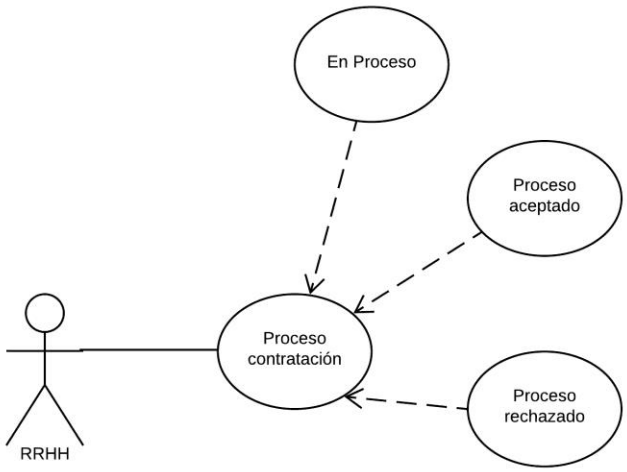


Figura 37. Caso de uso proceso de contratación (Elaboración propia).

Nombre	Proceso Contratación
ID	CU-002
Descripción	El usuario de RRHH debe ver todas las solicitudes pendientes de los candidatos, para luego cambiarlas a estado de espera, posteriormente del análisis al perfil del candidato, el usuario de RRHH debe aceptar o rechazar la solicitud, dejando un comentario de por qué se acepta o rechaza
Actores Principales	Usuario de RRHH
Actores Secundarios	Administrador
Pre-Condiciones	Ingresar al sistema y que existan solicitudes
Flujo principales	1) Las solicitudes registradas deben ser puesta en espera, posteriormente ser aceptadas o rechazadas
Post-Condiciones	Solicitud en estado aceptado o rechazado para un candidato
Flujo alternativo	-

*Tabla 4.*Antecedentes del caso de uso Proceso Contratación.

Caso de uso Consultar histórico:

En la tabla **5** se detallara el caso de uso Consultar Histórico.

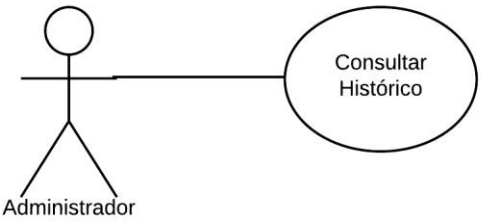


Figura 38. Caso de uso consultar histórico (Elaboración propia).

Nombre	Consultar histórico
ID	CU-003
Descripción	Ver todas las personas con sus respectivas reglas asociadas.
Actores Principales	Administrador
Actores Secundarios	-
Pre-Condiciones	Ingresar al sistema y que existan personas referidas
Flujo principales	1) Ver personas y reglas asociadas
Post-Condiciones	Lista de personas referidas
Flujo alternativo	-

Tabla 5. Antecedentes del caso de uso Consultar Histórico.

Caso de uso Consultar gráfico:

En la tabla **6** se detallara el caso de uso Consultar Grafico.

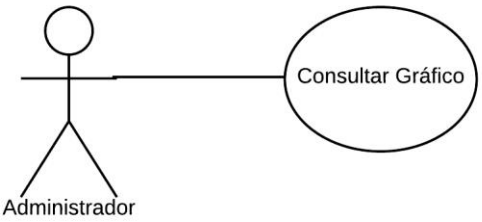


Figura 39. Caso de uso consultar gráfico (Elaboración propia).

Nombre	Consultar gráfico
ID	CU-004
Descripción	Ver gráfico con estadísticas de cuantas personas referenciadas existe por algún tipo de cargo.
Actores Principales	Administrador
Actores Secundarios	-
Pre-Condiciones	Ingresar al sistema y que existan personas referidas
Flujo principales	2) Se debe seleccionar un cargo para visualizar el gráfico con las estadísticas
Post-Condiciones	Gráfico con las estadísticas.
Flujo alternativo	1.- Cargo seleccionado no tiene personas asociadas. -Responde con un mensaje de que no existe información

*Tabla 6.*Antecedentes del caso de uso Consultar Grafico.

5.3.8 Diagrama de secuencia.

Ingresar Candidato:

En la Figura 40 se muestra la interacción que tiene el usuario con el sistema para poder ingresar un nuevo candidato, además se ilustra la interacción del sistema con el microservicio.

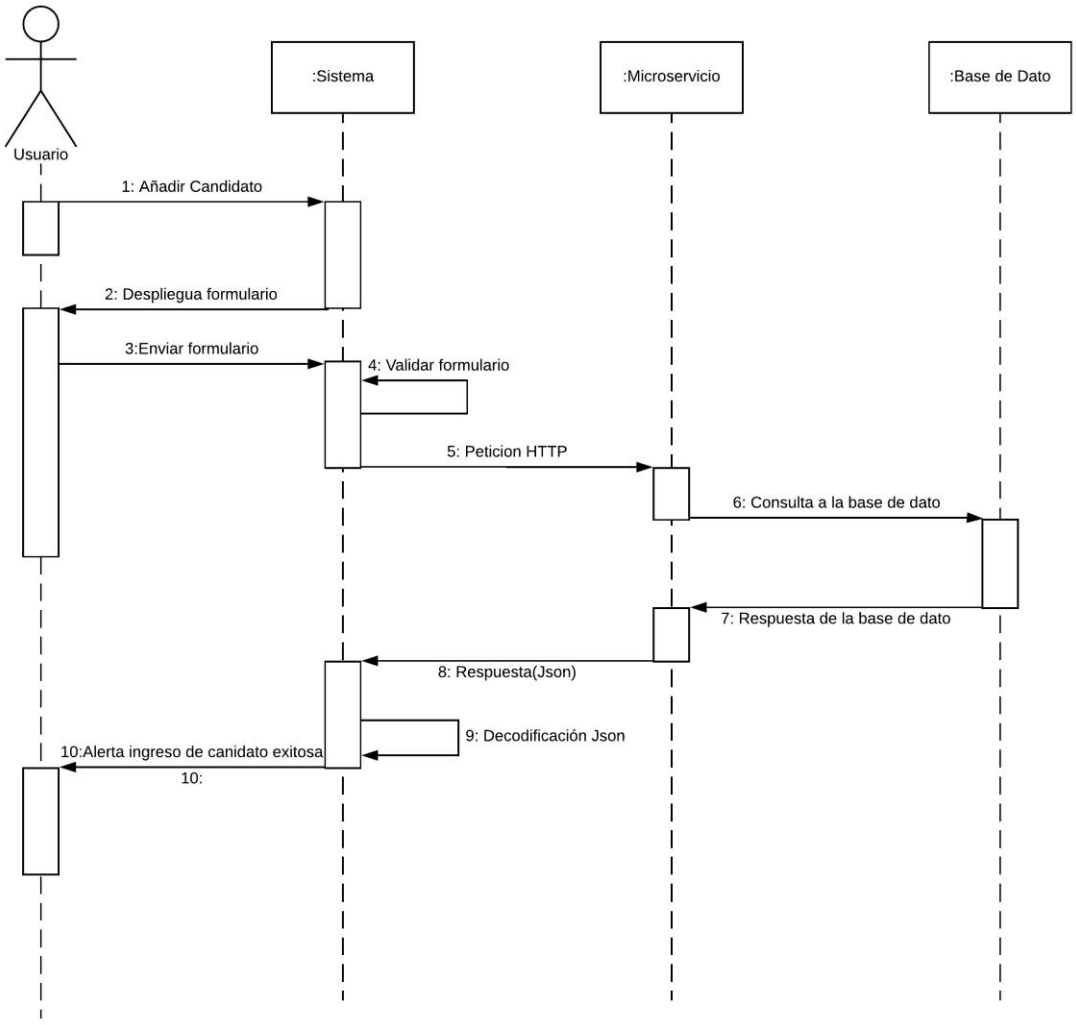


Figura 40. Diagrama de secuencia Ingresar Candidato al sistema (Elaboración propia).

Proceso Contratación:

En la Figura 41 se ilustra la interacción que tiene el usuario del rol RRHH con los diferentes flujos para la solicitud de un candidato y como el sistema interactúa con el microservicio.

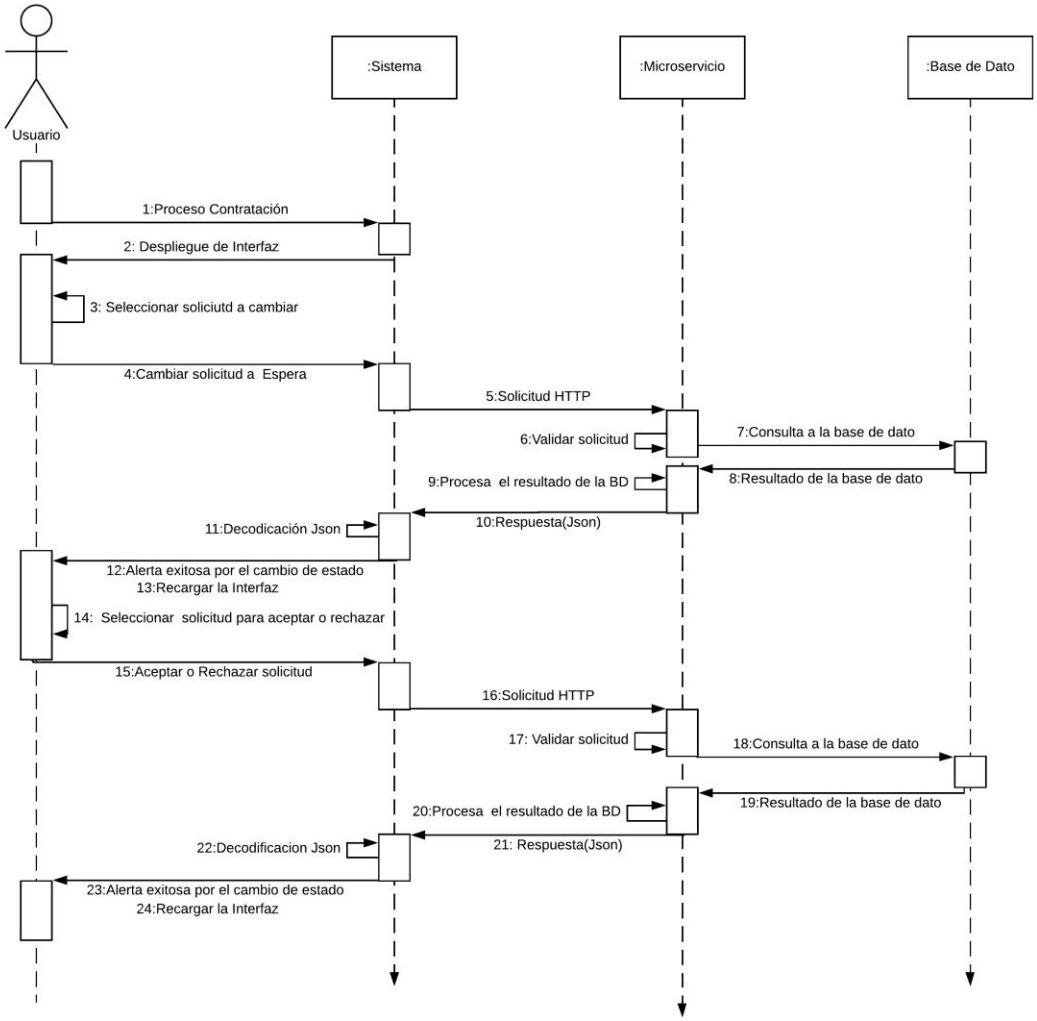


Figura 41. Diagrama de secuencia Proceso Contratación (Elaboración propia).

Consultar Histórico

En la Figura 42 se ilustra la interacción del usuario de RRHH y/o Administrador para consultar por el historial, además se ilustra la interacción del sistema con el microservicio.

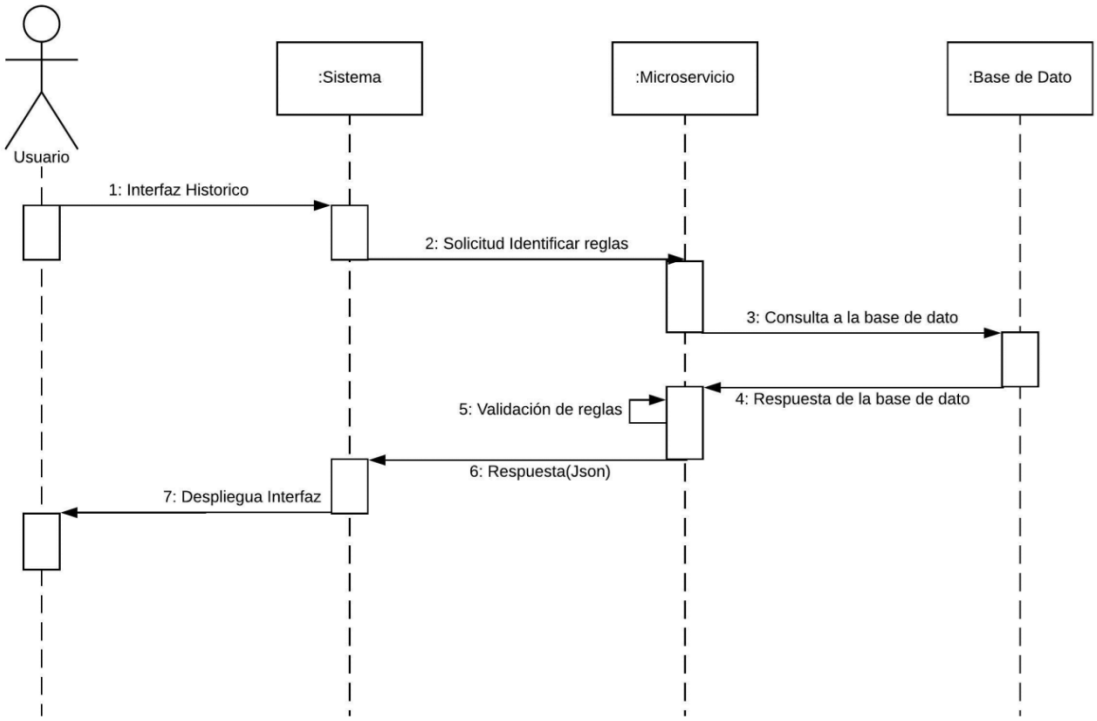


Figura 42.Diagrama de secuencia Consultar Histórico (Elaboración propia).

Consultar Gráfico

En la Figura 43 se ilustra la interacción del usuario Administrador para consultar por el grafico, además se ilustra la interacción del sistema con el microservicio.

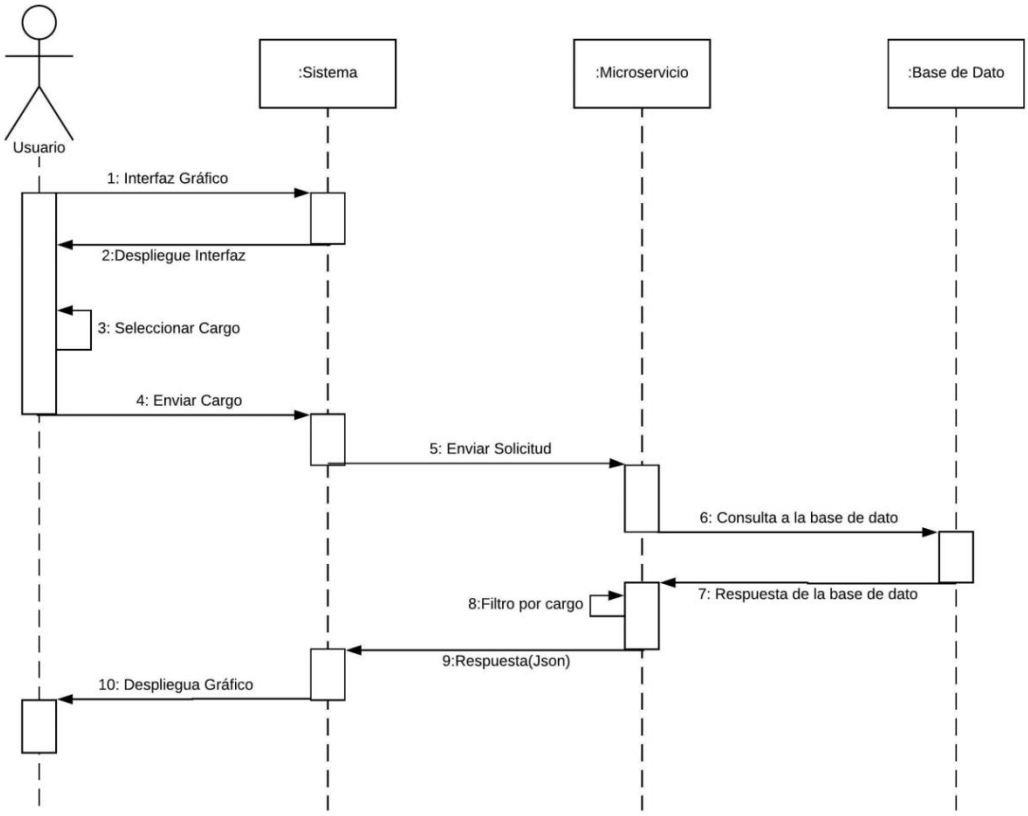


Figura 43.Diagrama de secuencia Consultar Grafico (Elaboración propia).

5.3.9 Modelamiento de datos

Haciendo énfasis a lo que dice Newman(2015), el microservicio debe ser pequeño e independiente y es por esto que el microservicio solo contendrá datos y lógicas de negocios respecto al módulo referenciados.

Diseño físico

Diseño físico con el cual se desarrolla el microservicios para el módulo referenciados.

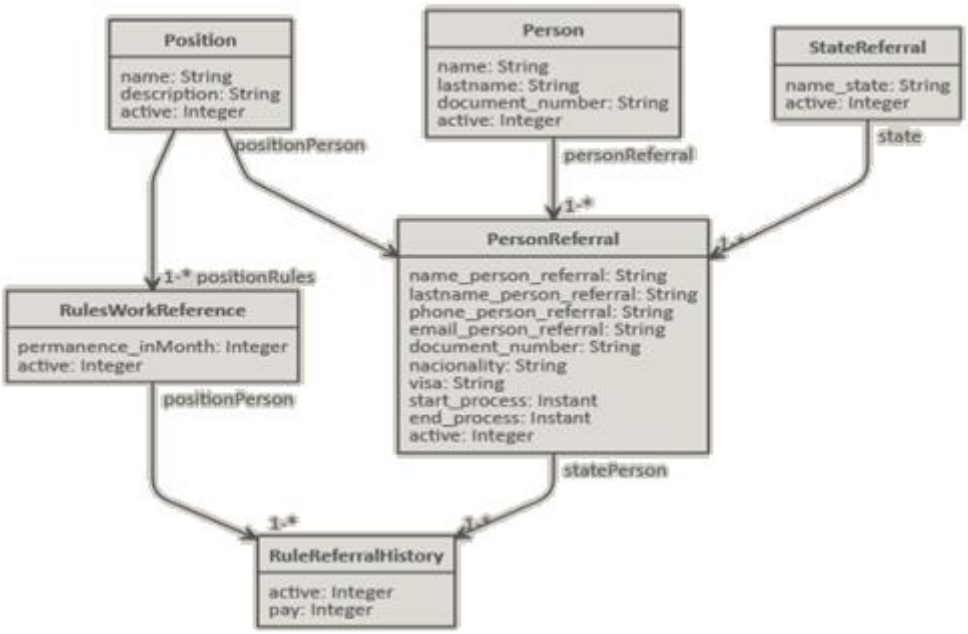
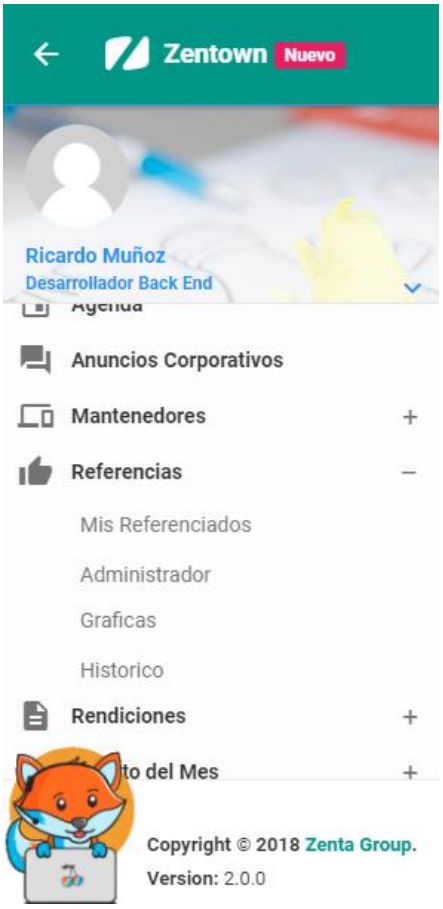


Figura 44.Diseño físico para el módulo referenciados (Elaboración propia).

5.3.10 Ilustraciones de interfaz de usuario



Menú principal del sistema en donde se permite acceder al módulo referenciados o "referencias". Este menú permite acceder a las vistas:

- Mis referenciados: Ingresar nuevo candidatos.
- Administrador: Proceso contratación.
- Graficas: Consultar gráfico.
- Histórico: Consultar histórico.

Figura 45.Menú principal para el módulo referenciados (Elaboración propia).

Interfaz para mis referenciados

Vista para ver a los candidatos que postulo el usuario ingresado en el sistema, ver en la figura 46, 46.a la acción de ingresar un nuevo candidato.

Figura 46.Interfaz para mis referenciados (Elaboración propia).

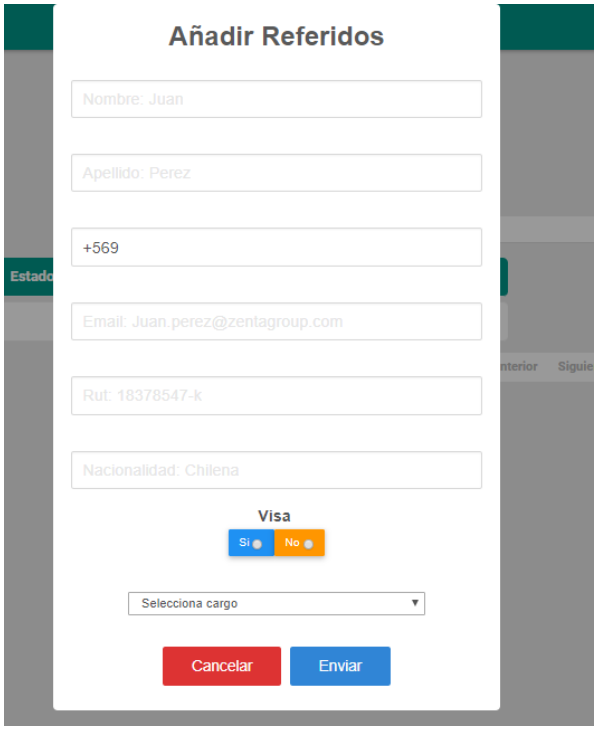


Figura 46.a. Acción del botón "Añadir" (Elaboración propia).

Interfaz para el administrador

Vista para el administrador de RRHH donde se le listara todas las solicitudes de nuevos candidato y tendrá la opciones una vez hayan solicitudes de cambiar el estado a pendiente, aceptar o rechazar. Ver figura 43.

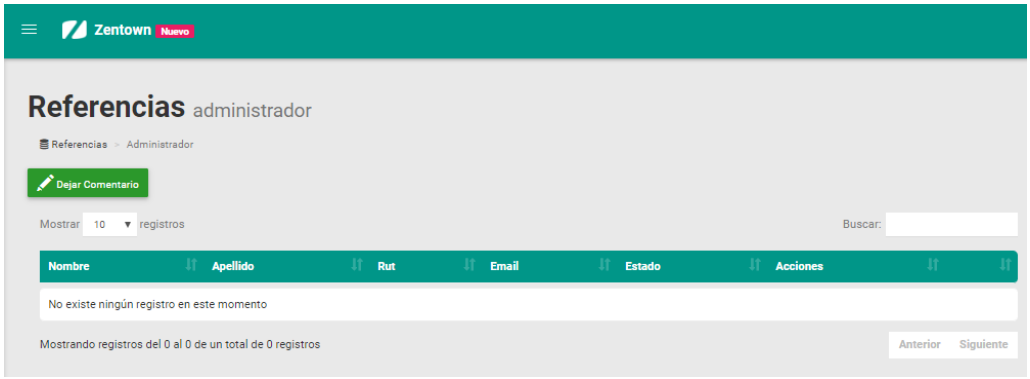


Figura 49. Interfaz para el administrador de RRHH (Elaboración propia).

Interfaz para consultar grafico

Vista para consultar un grafico que provee estadísticas en base a los cargos de las personas candidatas ingresadas en el módulo.

Figura 47. Interfaz para consultar gráfico (Elaboración propia).

Figura 48. Seleccionar cargo (Elaboración propia).

Interfaz para consultar histórico

Vista que muestra un historial de las personas contratadas por el módulo y además visualiza las respectivas reglas de cada persona.

Figura 50. Consultar Histórico(Elaboración propia).

5.3.11 Ilustraciones microservicio módulo referenciados

Se listaran imágenes de cómo está desarrollado a nivel código el microservicio, se hará referencia a los conceptos previamente descritos en el documento. Se ejemplificara solo la tabla PersonReferral del modelo físico.

Dominio

Clase java que representa la tabla PersonReferral del modelo físico

PersonReferral.java

```java
package cl.zenta.rrhhrefact.domain;


import javax.persistence.*;

/**
 * A PersonReferral.
 */
@Entity
@Table(name = "person_referral", schema="rrhh_referral")
@Document(indexName = "personreferral")
public class PersonReferral implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "sequenceGenerator")
    @SequenceGenerator(name = "sequenceGenerator")
    private Long id;

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

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

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

    @Column(name = "email_person_referral")
    private String email_person_referral;
```

Figura 51. Dominio (Elaboración propia).

Dominio personalizado

Clase java que representa un dominio personalizado para PersonReferral

CustomPersonReferral.java

```java
public class CustomPersonReferral {
    /*CAMPOS PERSON*/
    private Long person_id;
    private String namePerson;
    private String lastname;
    /*CAMPOS PERSON-REFERRAL*/
    private Long id;
    private String name_person_referral;
    private String lastname_person_referral;
    private String phone_person_referral;
    private String document_number;
    private String email_person_referral;
    private String nacionality;
    private String visa;
    private ZonedDateTime start_process;
    private ZonedDateTime end_process;
    private String cv_url;
    private Integer active;
    /*CAMPOS POSITION*/
    private Long position_id;
    private String name;
    /*rules*/
    private Long idrwr;
    private Integer permanence_inmonth;
    private Integer bonus_amount;
    /*CAMPOS STATE*/
    private String name_state;

    public CustomPersonReferral() {
    }


    public CustomPersonReferral(Long person_id, String namePerson, String lastname, Long id,

    public CustomPersonReferral(Long person_id, String namePerson, String lastname, Long id,
```

Figura 52. Dominio personalizado (Elaboración propia).

Repositorio

Interface de java para representar un repositorio que hereda de JpaRepository y permite acceder a los métodos básicos que opera una base de datos (CRUD).

PersonReferralRepository.java

```java
package cl.zenta.rrhhrefact.repository;

import cl.zenta.rrhhrefact.domain.PersonReferral;


/**
 * Spring Data JPA repository for the PersonReferral entity.
 */
@SuppressWarnings("unused")
@Repository
public interface PersonReferralRepository extends JpaRepository<PersonReferral, Long> {

}
```

Figura 53. Repositorio (Elaboración propia).

Repositorio personalizado

Si existe la necesidad de consultar a la base de datos con peticiones más personalizadas, un repositorio personalizado ayudara en este proceso.

CustomPersonReferralRepository.java

```
1  package cl.zenta.rrhhrefact.repository.custom;
2
3  import java.util.List;
12
13
14 public interface CustomPersonReferralRepository extends JpaRepository<PersonReferral,Long> {
15     @Query(value="SELECT new cl.zenta.rrhhrefact.domain.custom.CustomPersonReferral("
16                 + "pp.id,"
17                 + "pp.name, "
18                 + "pp.lastname, "
19                 + "pr.id, "
20                 + "pr.name_person_referral, "
21                 + "pr.lastname_person_referral, "
22                 + "pr.phone_person_referral, "
23                 + "pr.document_number, "
24                 + "pr.email_person_referral, "
25                 + "pr.nacionality, "
26                 + "pr.visa, "
27                 + "pr.start_process, "
28                 + "pr.end_process, "
29                 + "pr.cv_url, "
30                 + "pr.active, "
31                 + "p.id, "
32                 + "p.name, "
33                 + "sr.name_state) "
34                 + "FROM PersonReferral pr "
35                 + "JOIN pr.position p "
36                 + "JOIN pr.stateReferral sr "
37                 + "JOIN pr.person pp "
38                 + "WHERE pp.id= ?1 ")
39     List<CustomPersonReferral> getPersonReferralByID(Long idPerson);
```

Figura 54. Repositorio personalizado (Elaboración propia).

Capa de servicio

Capa de servicio para realizar lógicas de negocios necesarias para el módulo.

CustomPersonReferralService.java

```
        this.personRepository=personRepository;
        this.stateRepository=stateRepository;
        this.rwrRepository=rwrRepository;
        this.rrhRepository=rrhRepository;
        this.customRrhRepository=customRrhRepository;
    }


    public List<CustomPersonReferral> getPersonReferralByID(Long id)
    {
        List<CustomPersonReferral> listPersonReferralByID = new ArrayList<CustomPersonReferral>();

        try
        {
            listPersonReferralByID=personCustomRepository.getPersonReferralByID(new Long(id));
        }
        catch (Exception e)
        {
            log.error(e.getMessage(),e);
        }

        return listPersonReferralByID;
    }
```

Figura 55. Capa de servicio (Elaboración propia).

Recurso

Clase java que representa los recursos listados en la puerta de enlace, métodos básicos para crear, leer, actualizar y eliminar en tabla PersonReferral.

PersonReferralResource.java

```
import com.codahale.metrics.annotation.Timed;

/**
 * REST controller for managing PersonReferral.
 */
@RestController
@RequestMapping("/api")
public class PersonReferralResource {

    private final Logger log = LoggerFactory.getLogger(PersonReferralResource.class);

    private static final String ENTITY_NAME = "personReferral";

    private final PersonReferralRepository personReferralRepository;

    private final PersonReferralSearchRepository personReferralSearchRepository;

    public PersonReferralResource(PersonReferralRepository personReferralRepository, PersonReferralSearchRepository personReferralSearchRepository) {

     * POST  /person-referrals : Create a new personReferral.
    public ResponseEntity<PersonReferral> createPersonReferral(@RequestBody PersonReferral personReferral) throws URISyntaxException {

     * PUT  /person-referrals : Updates an existing personReferral.
    public ResponseEntity<PersonReferral> updatePersonReferral(@RequestBody PersonReferral personReferral) throws URISyntaxException {

     * GET  /person-referrals : get all the personReferrals.
    public List<PersonReferral> getAllPersonReferrals() {

     * GET  /person-referrals/:id : get the "id" personReferral.
    public ResponseEntity<PersonReferral> getPersonReferral(@PathVariable Long id) {

     * DELETE  /person-referrals/:id : delete the "id" personReferral.
    public ResponseEntity<Void> deletePersonReferral(@PathVariable Long id) {

     * SEARCH  /_search/person-referrals?query=:query : search for the personReferral corresponding
    public List<PersonReferral> searchPersonReferrals(@RequestParam String query) {

}
```

Figura 56. Recurso (Elaboración propia).

Recurso personalizado

Cuando es necesario listar un nuevo método en la puerta de enlace se requiere un recurso personalizado.

CustomPersonReferral...

```
@GetMapping("/custom-person-referral-by-person/{id}")
public ResponseEntity<List<CustomPersonReferral>> getPersonReferraByID(@PathVariable Long id)
{
   List<CustomPersonReferral> t = personService.getPersonReferralByID(id);
    return ResponseEntity.ok().body(t);
}
```

Figura 57. Recurso personalizado (Elaboración propia).

Recursos básicos en la puerta de enlace

Puerta de enlace que lista los recursos del microservicio.

Este es el enlace de los recursos entre un microservicio y una aplicación web, toda petición de una aplicación web hacia un microservicio se hace mediante peticiones HTTP.

Figura 58. Puerta de enlace para los recursos (Elaboración

Recursos personalizados en la puerta de enlace

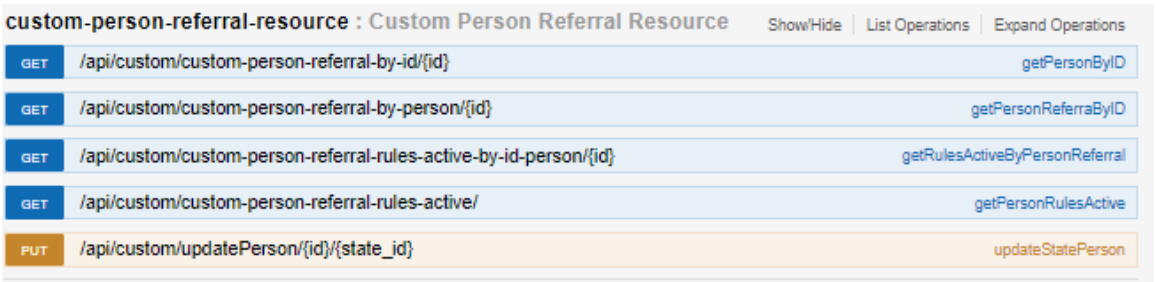


Figura 59. Puerta de enlace personalizada (Elaboración propia).

5.3.12 Pruebas al microservicio referenciados

Se realizaran pruebas de entrada y salida al microservicio que contiene la lógica para el módulo referenciados.

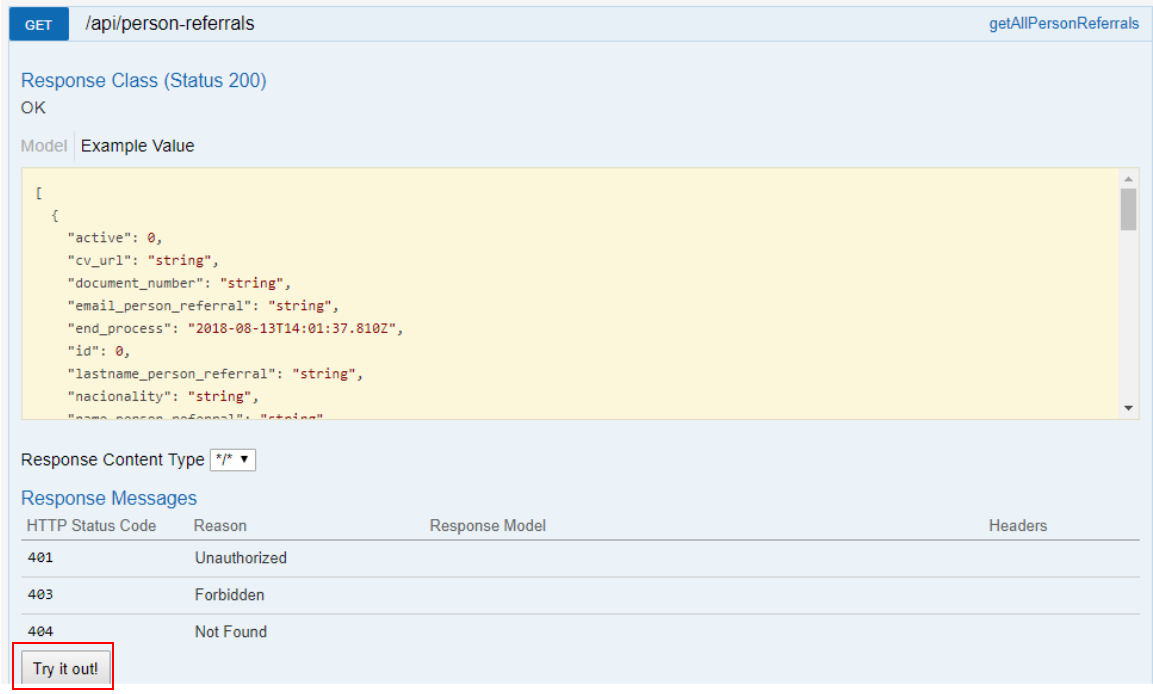


Figura 60. GET para personas referidas (Elaboración propia).

Al hacer click en "Try itout" tendremos la respuesta del recurso GET para la tabla PersonReferral.

Figura 61. Resultado de la solicitud GET (Elaboración propia).

Prueba de entrada y salida a un recurso personalizado, se obtendrá las reglas activas que tenga una persona referida.

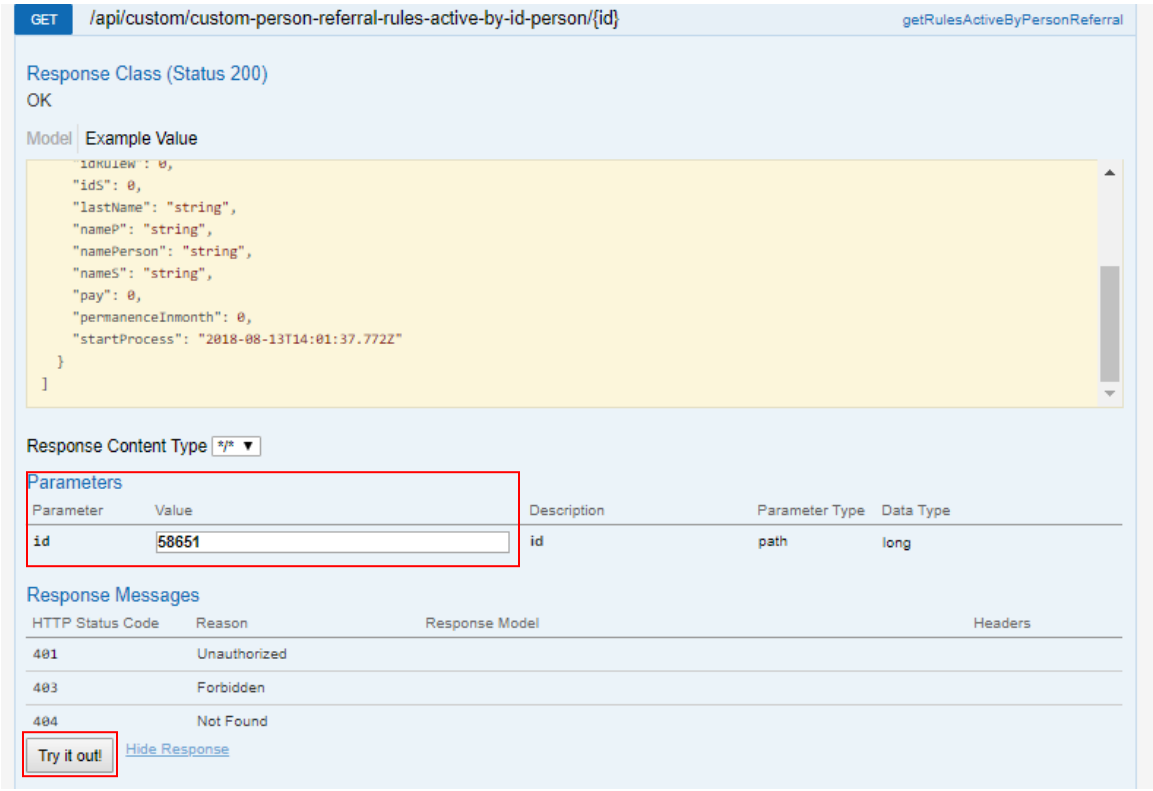


Figura 62. Solicitud a un recurso personalizado (Elaboración

Al hacer click en "Try itout" tendremos la respuesta del recurso personalizado

Figura 63. Resultado de un recurso personalizado (Elaboración propia).

5.3.13 Pruebas al módulo

Se realizaran pruebas de entradas y salidas al módulo además de simular los diferentes roles que interactúan.

Mis referenciados

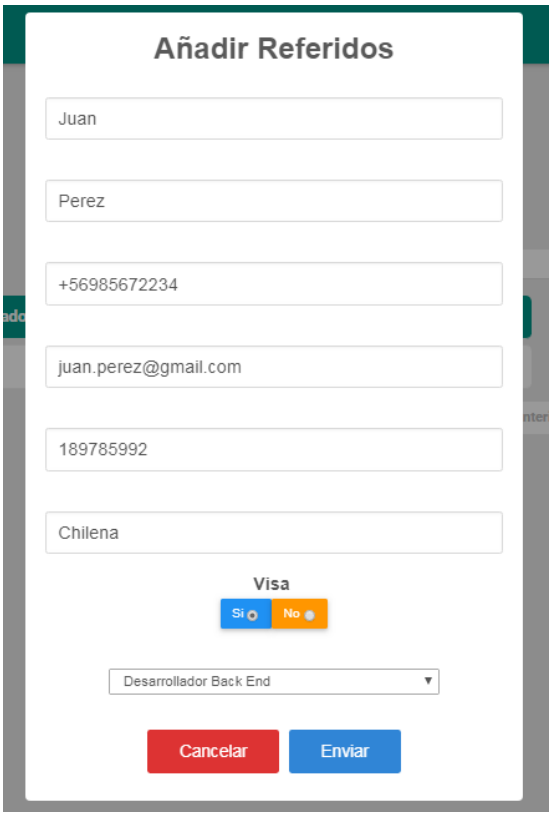


Figura 64. Añadir referido (Elaboración propia).

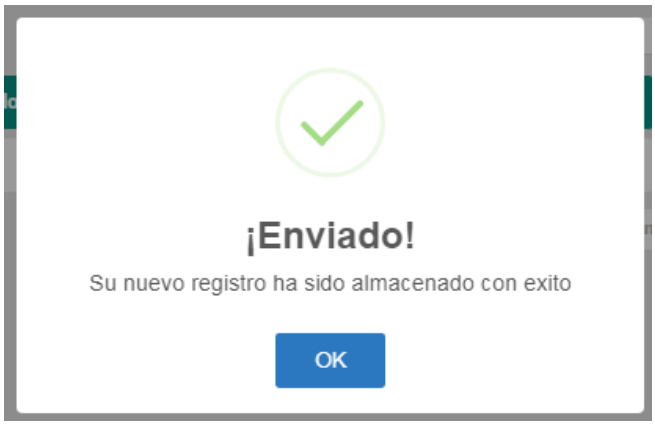


Figura 65. Referido ingresado exitosamente (Elaboración propia).

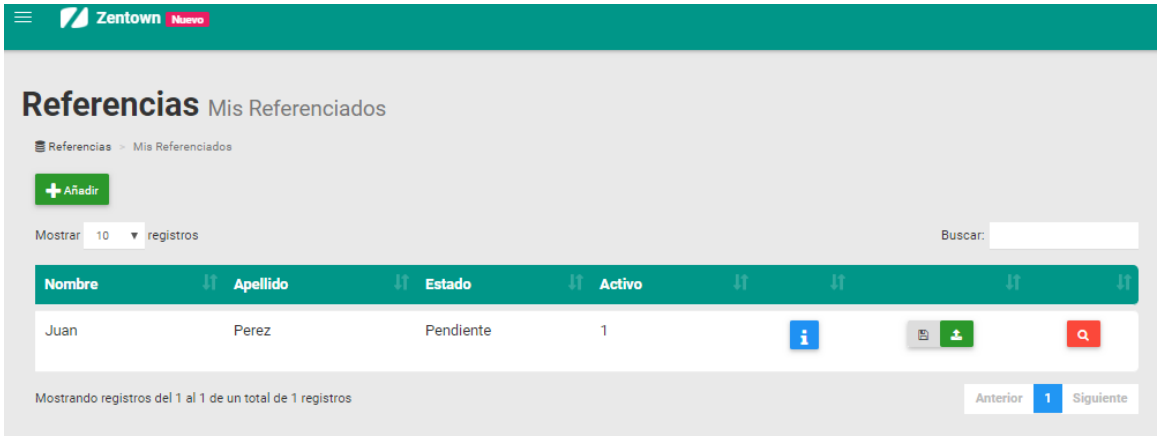


Figura 66. Interfaz mis referenciados (Elaboración propia).

Guardar curriculum

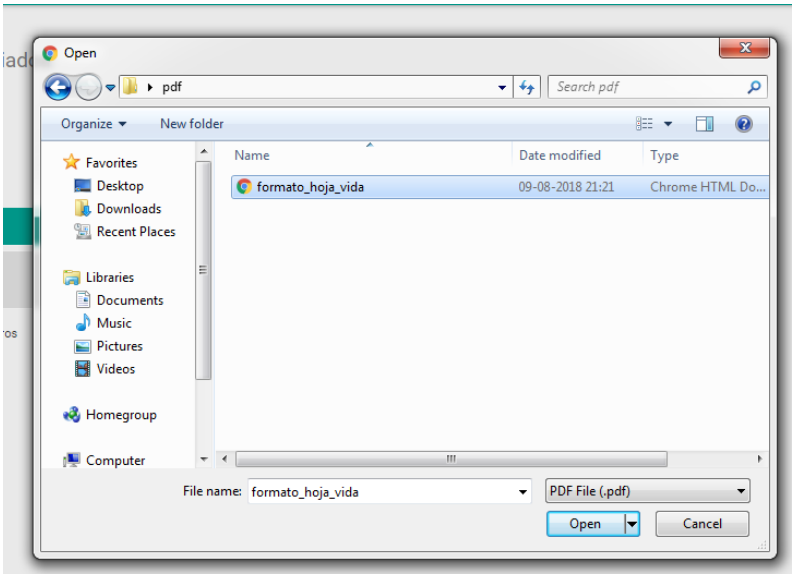


Figura 67. Acción guardar curriculum (Elaboración propia).

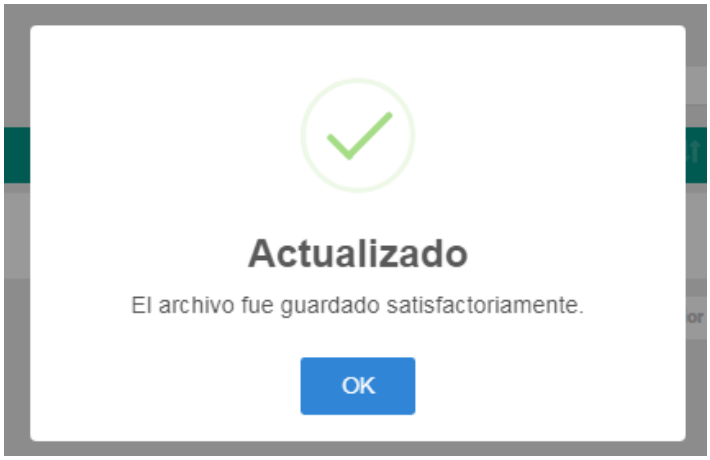


Figura 68. Curriculum guardado exitosamente (Elaboración propia).

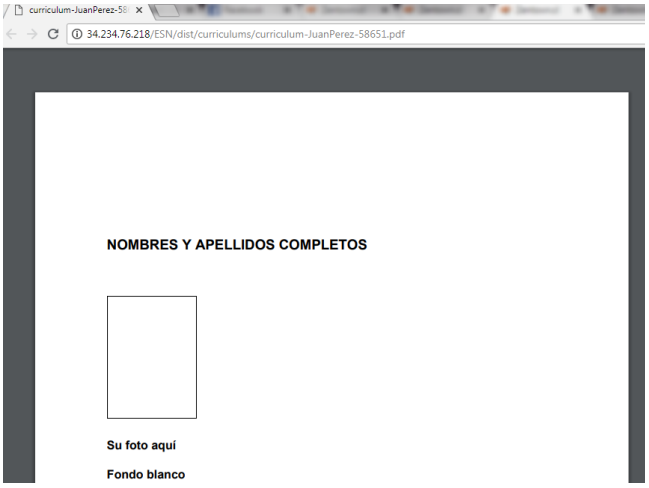


Figura 69. Visualización del curriculum (Elaboración propia).

Proceso de contratación

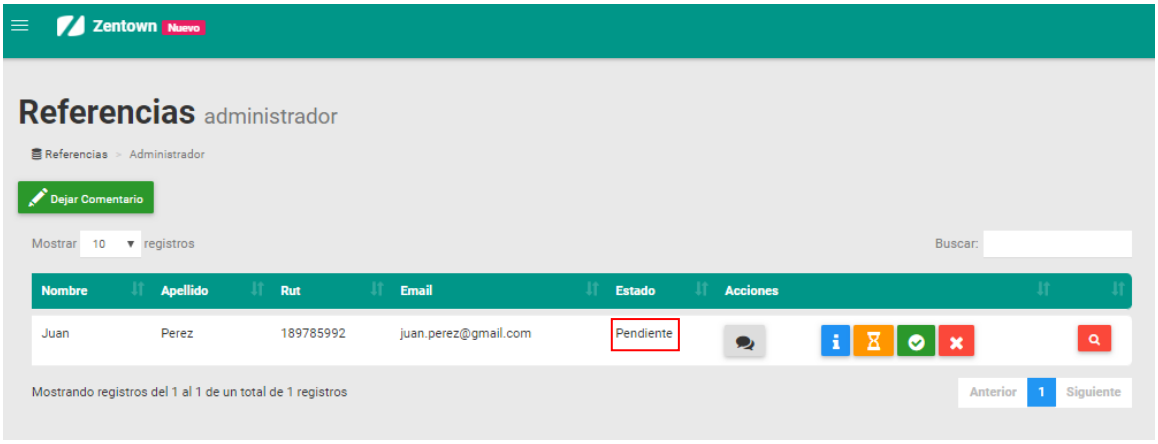


Figura 70. Interfaz administrador (Elaboración propia).

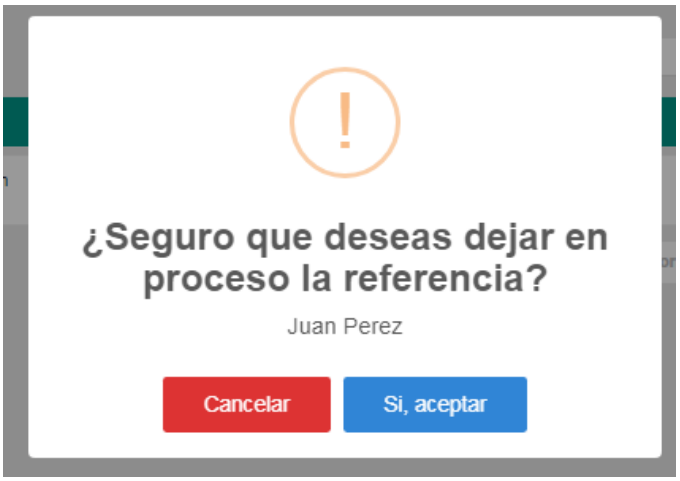


Figura 71. Cambiar estado a proceso (Elaboración propia).

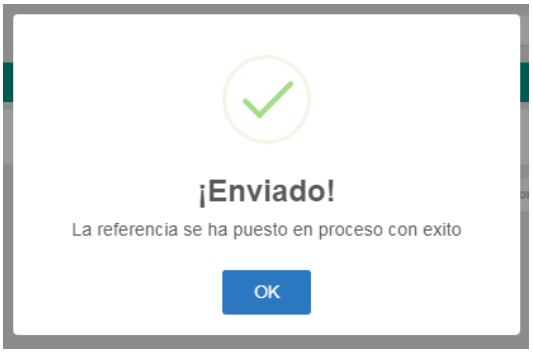


Figura 72. Confirmación del cambio(Elaboración propia).

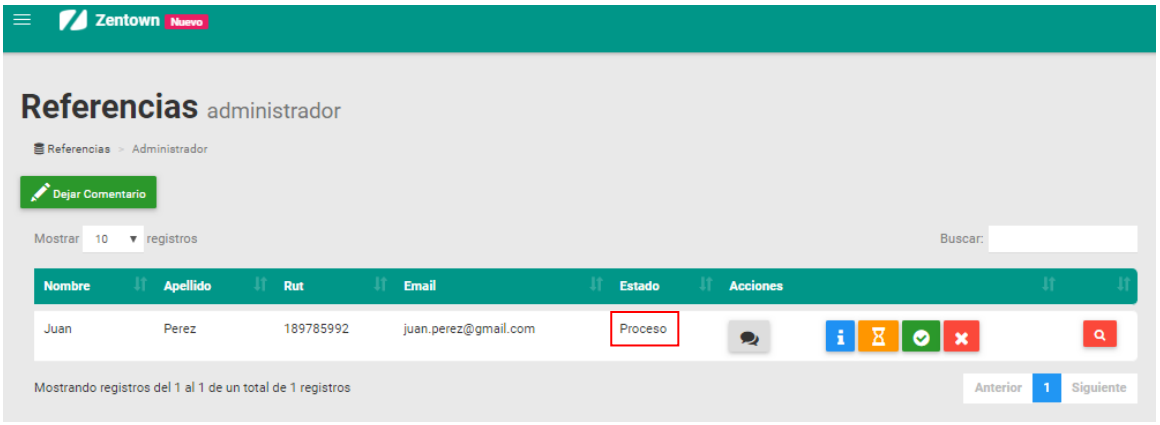


Figura 73. Solicitud en proceso(Elaboración propia).

Figura 74. Aceptar solicitud (Elaboración propia).

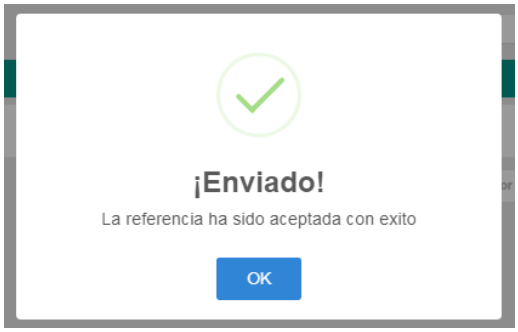


Figura 75. Confirmación del cambio (Elaboración propia).

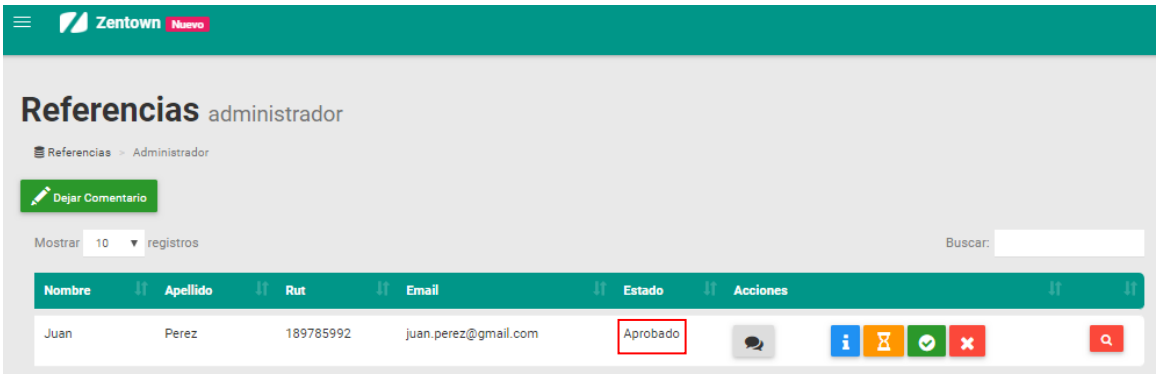


Figura 76. Solicitud aceptada (Elaboración propia).

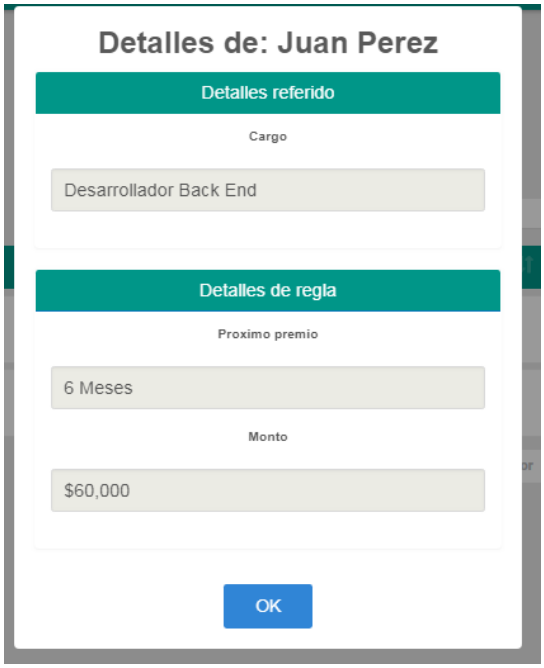


Figura 77. Detalles de la solicitud aprobada (Elaboración propia).

Figura 78. Rechazar solicitud(Elaboración propia).

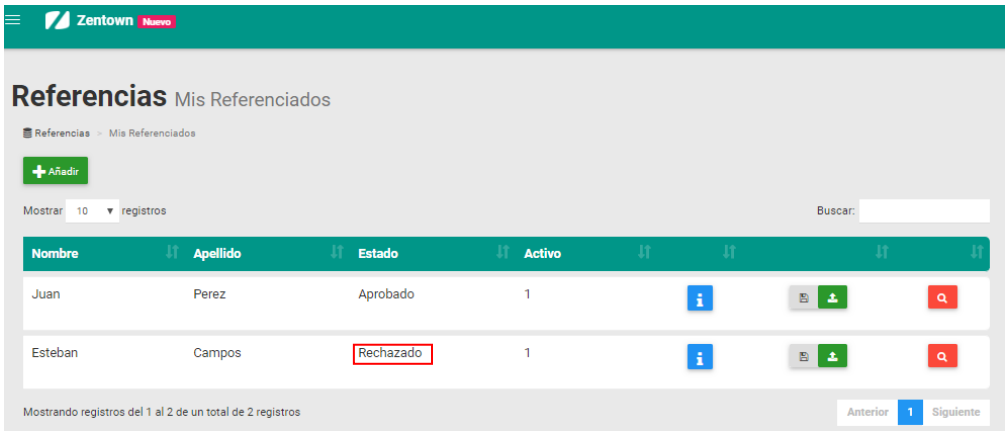


Figura 79. Solicitud rechazada (Elaboración propia).

Consultar Histórico

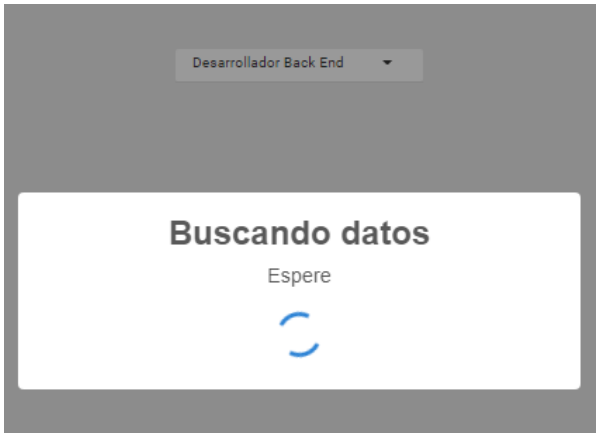


Figura 80. Mostrar grafico (Elaboración propia).

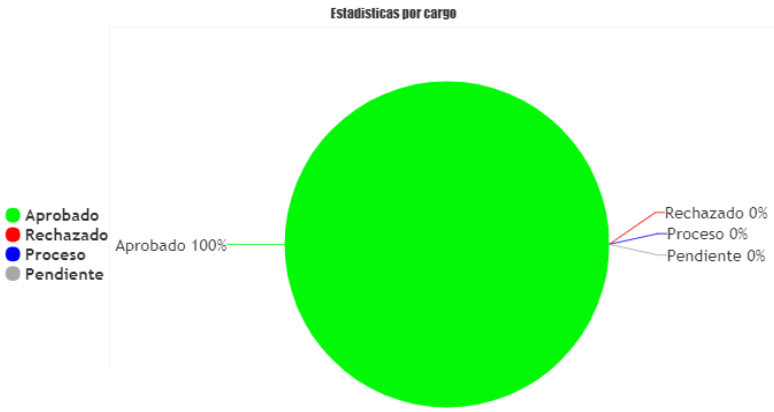


Figura 81. Gráfico de estadísticas por cargo (Elaboración propia).

Consultar histórico

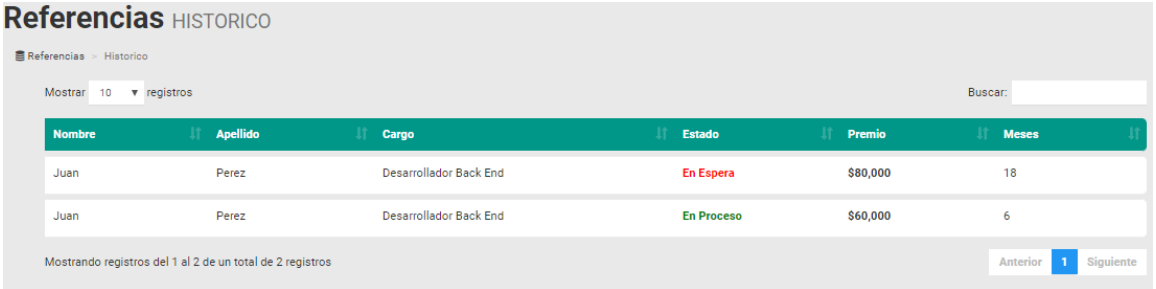


Figura 82. Interfaz histórico (Elaboración propia).

Mitigación de errores

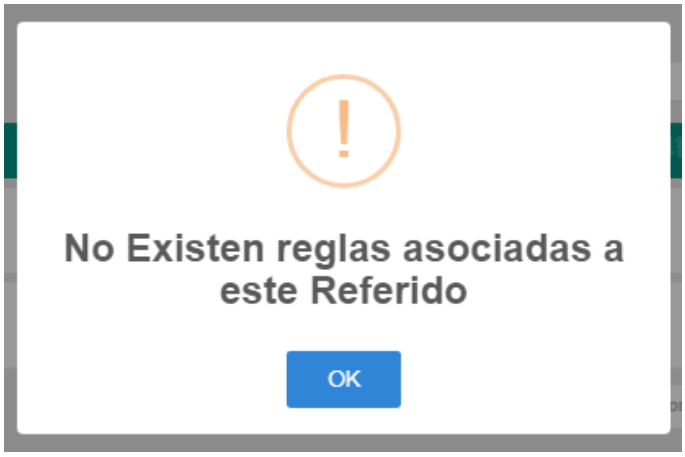


Figura 83. Error detalles solicitud histórico (Elaboración propia).

Figura 84. Error ver curriculum

Si se quiere ver las reglas de un candidato en estado pendiente, proceso o rechazado el sistema no permite mostrar las reglas asociadas, ya que solo existen reglas asociadas si la solicitud es aceptada.

Manejo de error si se quiere visualizar el curriculum antes de haberlo guardado.

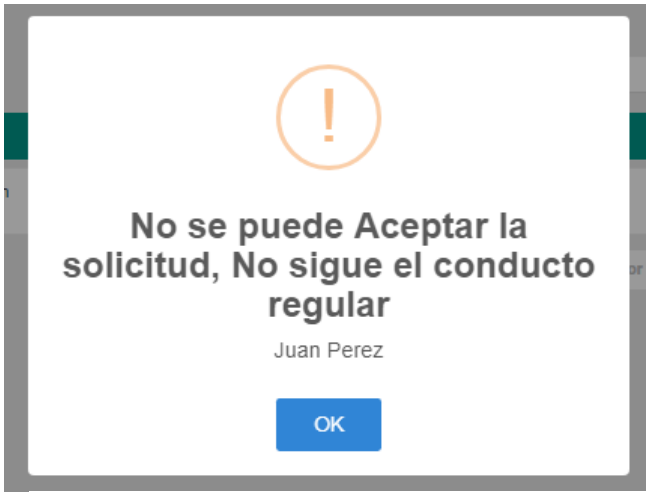


Figura 85. Error de flujo (Elaboración propia).

Figura 86. Error cargar gráfico (Elaboración propia).

Control de error para el proceso de contratación, el flujo es el siguiente: • Solicitud pendiente cambia a proceso. • Solicitud en proceso cambia aceptado o rechazado. El sistema no permite aceptar la solicitud de cambio de estado si el flujo es incorrecto.

Si no existen personas referidas con el cargo seleccionado, el sistema no permite visualizar el gráfico.

5.4 Desarrollo del módulo rendiciones

5.4.1 Objetivo general

Diseñar e implementar un módulo bajo la arquitectura orientada a microservicios, que permita subir rendiciones.

5.4.2 Objetivos específicos

- El módulo permitirá a un colaborador ingresar una nuevarendición.
- El módulo permitirá controlar el proceso de todas las rendiciones.

5.4.3 Descripción de interfaz.

Interfaz Mis rendiciones

Interfaz de usuario que listara todas las rendiciones creadas por el colaborador ingresado en el sistema, se podrá ingresar una nueva rendición adjuntando su respectiva boleta o factura.

Interfaz Rendiciones Recepción

Interfaz de usuario que listara todas las rendiciones ingresadas atreves del módulo dándole los atributos a la persona de recepción poder ver el detalle de cada una de las rendiciones para posteriormente aceptar o rechazar la rendición.

Interfaz Rendiciones Gerencia

Interfaz de usuario que listara todas las rendiciones aceptadas por la persona de recursos humanos. En gerencia se volverá a verificar que la rendiciones estén de forma correcta para posteriormente aceptar o rechazar.

Interfaz Contabilidad

Interfaz de usuario que listara el monto total de rendiciones por colaborador, la persona encargada de contabilidad podrá ver el detalle de todas las rendiciones y dar la última confirmación para que posteriormente se efectué el pago en un tiempo establecido.

5.4.4 Requerimientos funcionales

Id	Nombre	Descripción
RF-01	Ingresar rendición	Registrar una nueva rendición.
RF-02	Flujo recepción	Aceptar o rechazarlas rendiciones ingresadas.
RF-03	Flujo gerencia	Aceptar o rechazar las rendiciones aprobadas previamente por recepción.
RF-04	Flujo contabilidad	Confirmar el pago de las rendiciones aceptadas por gerencia

Tabla 7. Requisitos funcionales del módulo.

5.4.5 Requerimientos no funcionales.

Id	Nombre	Descripción
RNF01	Arquitectura	Se desarrollara bajo la arquitectura orientada a microservicios.
RNF02	Lenguaje de programación	Se utilizara principalmente Java, PHP, JavaScript.
RNF03	Gestor de Base de Datos.	Se utilizara PostgreSQL y MySQL.

Tabla 8. Requisitos no funcionales del módulo.

5.4.6 Análisis del caso de uso modulo rendiciones

En esta sección del informe se identifican los actores principales que interactuaran con el módulo además se describe el contexto del caso de uso. Por otro lado se ilustrará diagramas de secuencias para visualizar la interacción entre el sistema y el microservicio.

Actores

- Colaborador
    - Podrá crear, ingresar una nueva rendición.
- Recepción
    - Podrá aceptar o rechazar las rendiciones registradas.
- Gerencia
    - Podrá aceptar o rechazar las rendiciones aprobadas previamente por recepción.
- Contabilidad
    - Confirmara el pago de las rendiciones aprobadas por gerencia.

Descripción

En la figura 70 se aprecia como los diferentes actores pueden interactuar con las distintas funcionalidades del módulo, en donde el actor "Colaborador" realiza una interacción que le permite ingresar una nueva rendición, se puede ver como el actor "Recepción" puede aceptar o rechazar de las rendiciones, por otro lado el actor "Gerencia" puede aceptar o rechazar las solicitudes previamente aceptadas por recepción, finalmente cuando gerencia aprueba las rendiciones, "Contabilidad" podrá confirmar y efectuar el pago de forma exitosa.

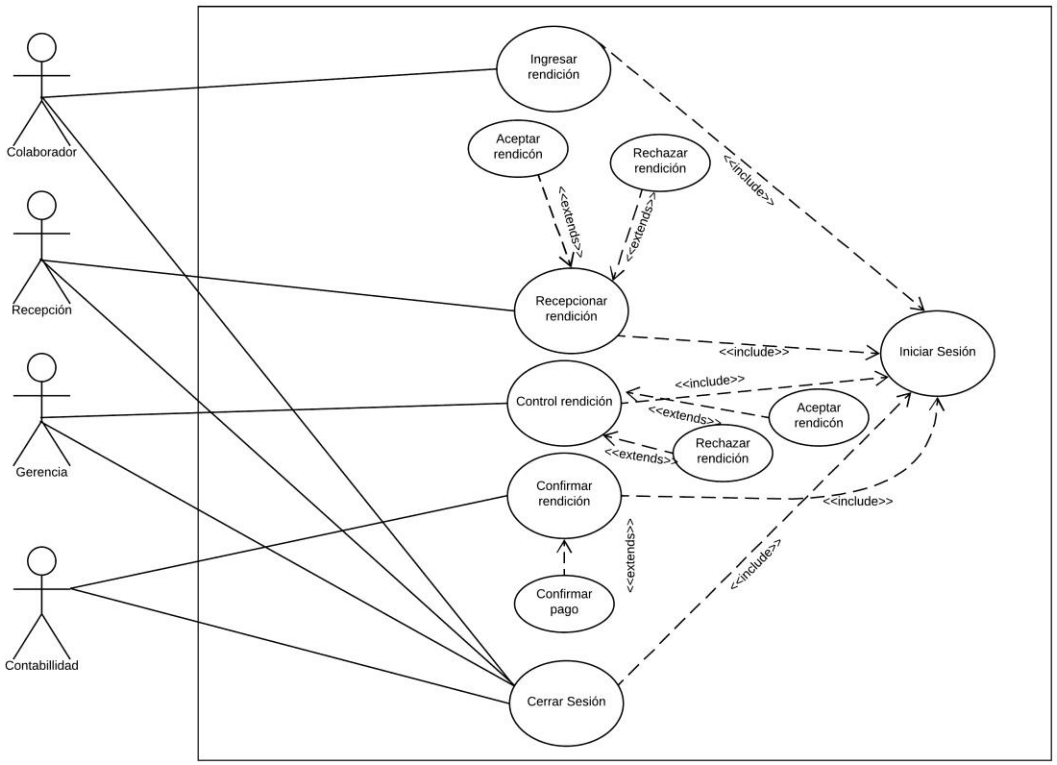


Figura 87. Diagrama de caso de uso para el módulo rendiciones (Elaboración propia).

5.4.7 Especificación de casos de uso

A continuación se detallan todos los casos de uso explicando que función cumple, su pre y su post condiciones y el flujo de eventos respectivo.

Caso de uso Ingresar rendición:

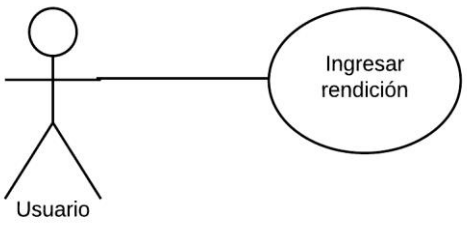


Figura 88. Caso de uso ingresar rendición al sistema (Elaboración propia).

En la tabla **9** se detallara el caso de uso Ingresar rendición.

Nombre	Ingresar rendición
ID	CU-001
Descripción	El usuario debe ingresar una nueva rendición al sistema Zentown permitiendo subir una boleta o factura de forma digital.
Actores Principales	Usuario o colaborador que ingresa rendición.
Actores Secundarios	-
Pre-Condiciones	Ingresar al sistema
Flujo principales	1) El usuario ingresa su nombre de usuario y contraseña al sistema Zentown. 2) El sistema comprueba que el usuario tiene un perfil asociado al sistema. 3) El usuario selecciona dentro del menú, mis rendiciones. 4) El usuario selecciona añadir rendición. 5) El usuario ingresa los datos de la rendición 6) El usuario ingresa la boleta o factura en formato imagen al sistema Zentown.
Post-Condiciones	Rendición registrada a la espera que sea confirmada
Flujo alternativo	2.- El sistema comprueba que el usuario no tiene un perfil asociado -Responde con mensaje de reintentar ingresar al sistema.

Tabla 9. Antecedentes del caso de uso Ingresar rendición.

Caso de uso Recepcionar rendición:

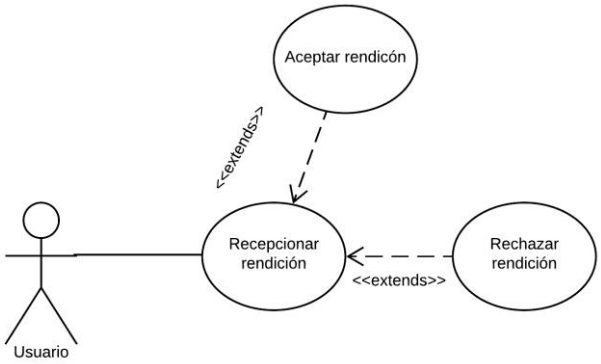


Figura 89. Caso de uso Recepcionar rendición (Elaboración propia).

En la tabla **10** se detallara el caso de uso Recepcionar rendición.

Nombre	Recepcionar rendición
ID	CU-002
Descripción	El usuario recepcionista debe aceptar o rechazar todas las rendiciones del sistema
Actores Principales	Usuario recepcionista
Actores Secundarios	-
Pre-Condiciones	Ingresar al sistema y que existan rendiciones
Flujo principales	1) Aceptar o rechazar rendiciones.
Post-Condiciones	Rendición aceptada o rechazada.
Flujo alternativo	-

Tabla 10. Antecedentes del caso de uso Recepcionar rendición.

Caso de uso Control rendición:

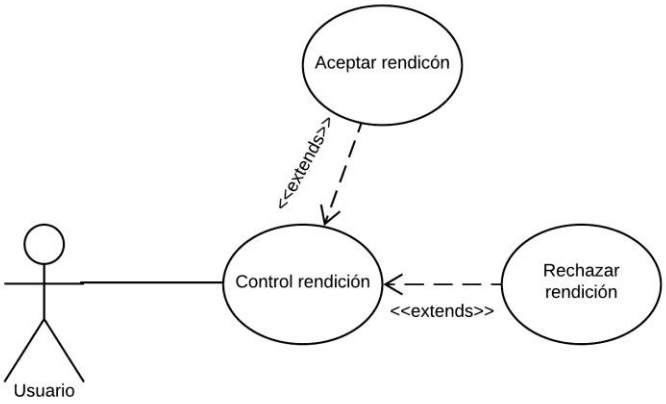


Figura 90. Caso de uso Control rendición (Elaboración propia).

En la tabla **11** se detallara el caso de uso Control rendición.

Nombre	Control rendición
ID	CU-003
Descripción	El usuario de gerencia debe aceptar o rechazar todas las rendiciones del sistema previamente aceptadas por recepción.
Actores Principales	Usuario gerencia
Actores Secundarios	-
Pre-Condiciones	Ingresar al sistema y que existan rendiciones aceptadas por recepción.
Flujo principales	1) Aceptar o rechazar rendiciones.
Post-Condiciones	Rendición aceptada o rechazada.
Flujo alternativo	-

*Tabla 11.*Antecedentes del caso de uso Control rendición.

Caso de uso Confirmar rendición:

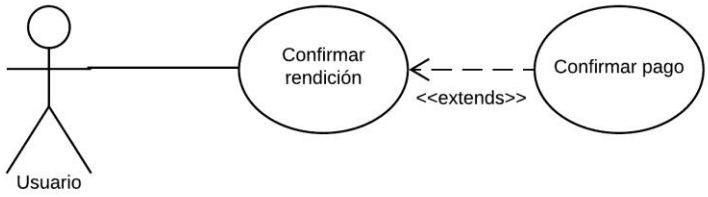


Figura 91. Caso de uso Confirmar rendición (Elaboración propia).

En la tabla **12** se detallara el caso de uso Confirmar rendición.

Nombre	Confirmar rendición
ID	CU-004
Descripción	El usuario de contabilidad debe confirmar el pago de las rendiciones previamente aceptadas por gerencia.
Actores Principales	Usuario contabilidad
Actores Secundarios	-
Pre-Condiciones	Ingresar al sistema y que existan rendiciones aceptadas por gerencia.
Flujo principales	1) Confirmar pago rendición.
Post-Condiciones	Rendiciones confirmadas.
Flujo alternativo	-

Tabla 12. Antecedentes del caso de uso Confirmar rendición.

5.4.8 Diagrama de secuencia.

Ingresar rendición:

En la Figura 92 se muestra la interacción que tiene el usuario o colaborador con el sistema para poder ingresar una nueva rendición, además se ilustra la interacción del sistema con el microservicio.

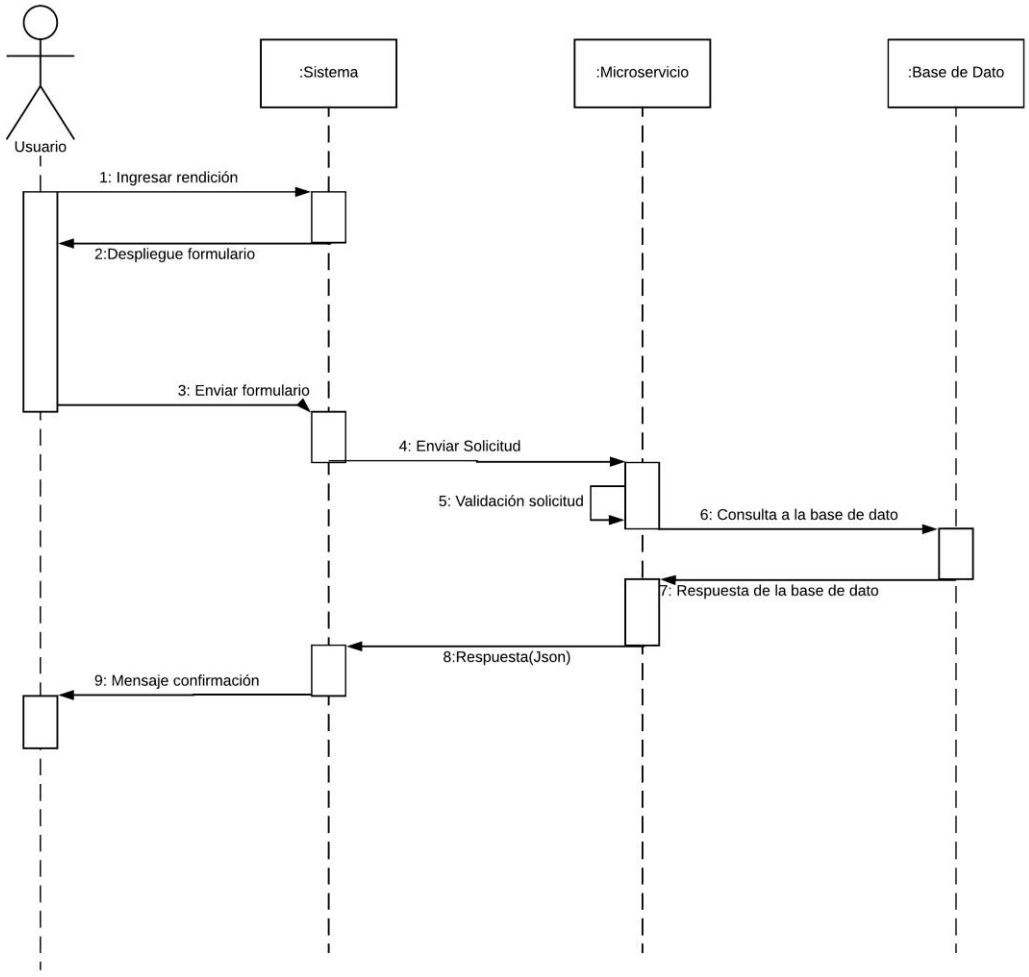


Figura 92. Diagrama de secuencia Ingresar rendición al sistema(Elaboración propia).

Recepcionar rendición

En la Figura 93 se muestra la interacción que tiene el usuario recepcionista con el sistema para poder aceptar o rechazar una rendición, además se ilustra la interacción del sistema con el microservicio.

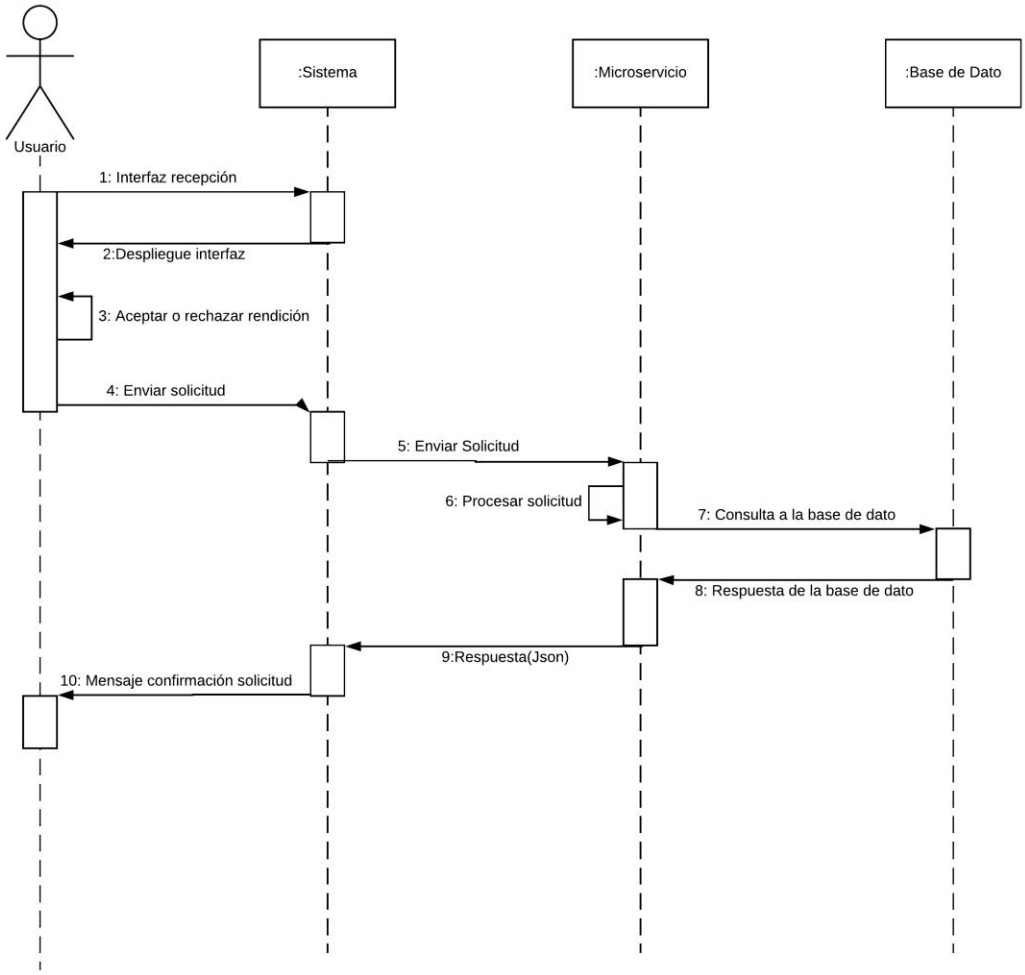


Figura 93. Diagrama de secuencia Recepcionar rendición (Elaboración propia).

Control rendición

En la Figura 94 se muestra la interacción que tiene el usuario de gerencia con el sistema para poder aceptar o rechazar las rendiciones previamente aceptadas por recepción, además se ilustra la interacción del sistema con el microservicio.

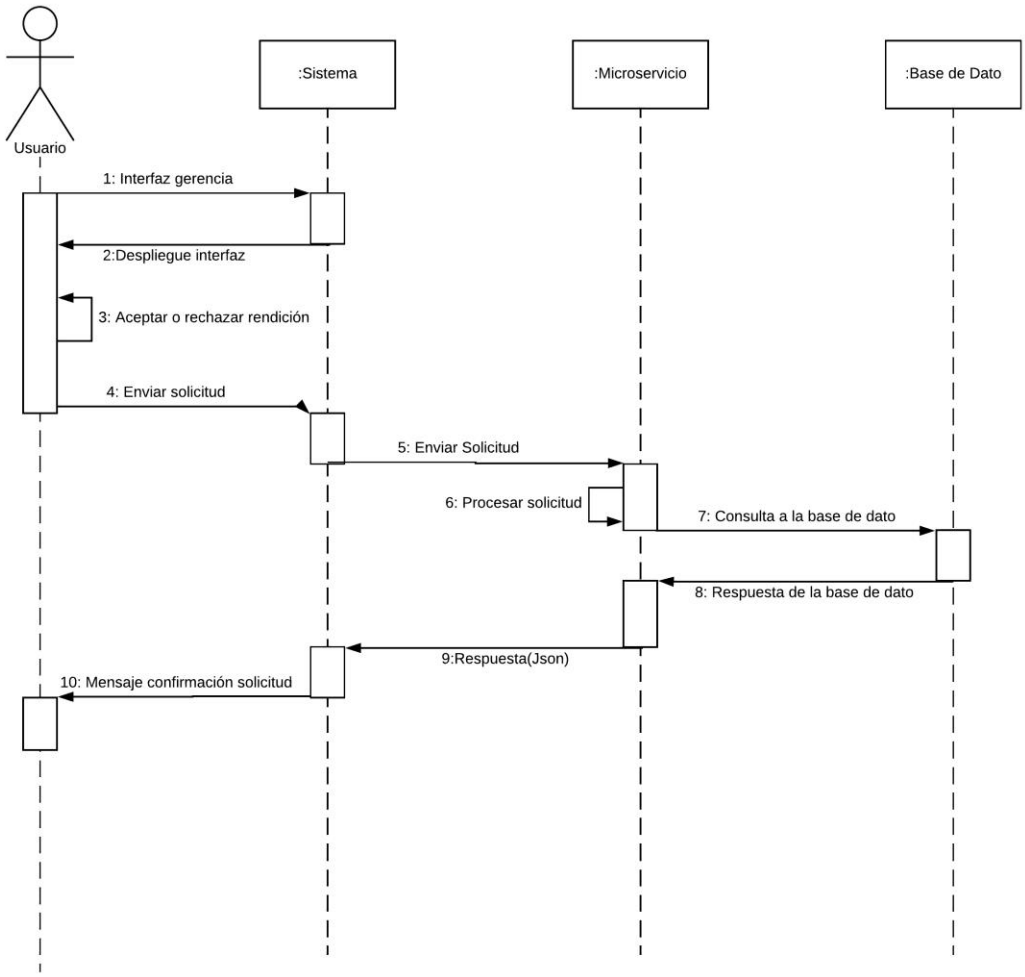


Figura 94. Diagrama de secuencia Control rendición (Elaboración propia).

Confirmar rendición

En la Figura 95 se muestra la interacción que tiene el usuario contabilidad con el sistema para poder confirmar el pago de las rendiciones previamente aceptadas por gerencia, además se ilustra la interacción del sistema con el microservicio.

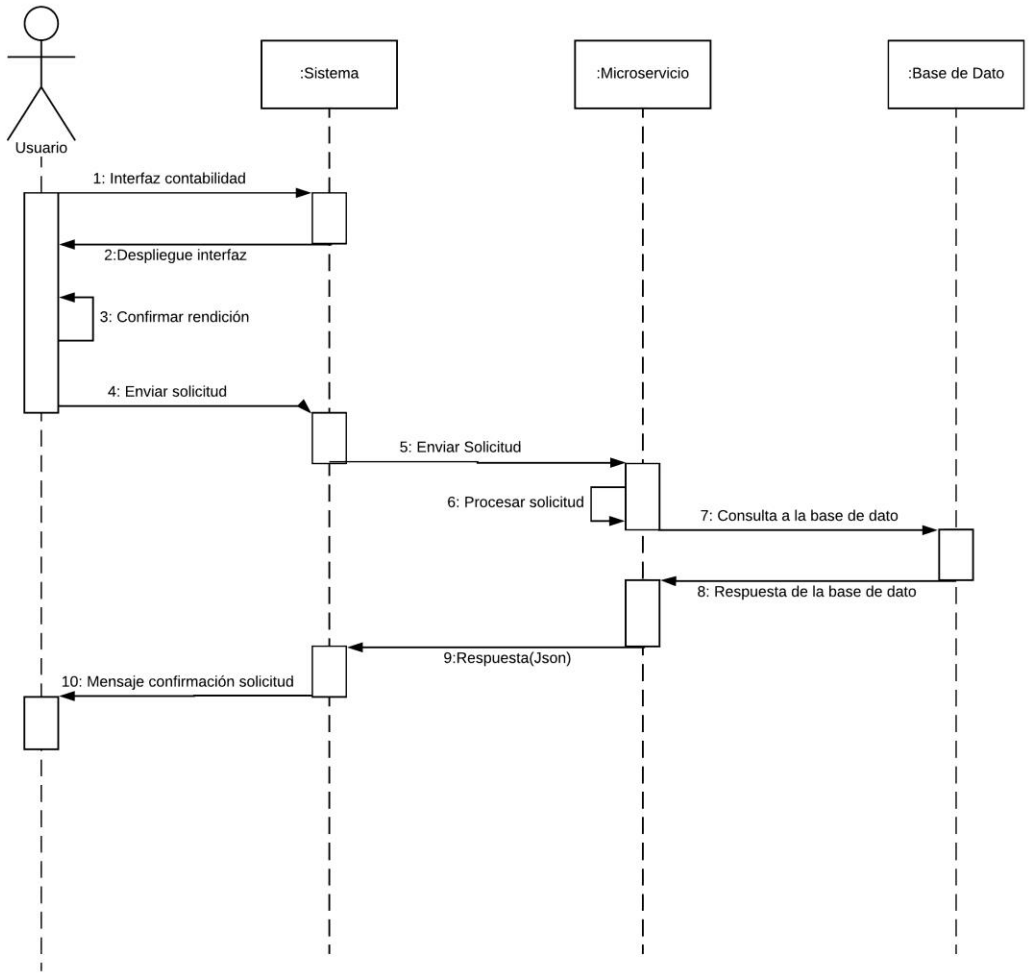


Figura 95. Diagrama de secuencia Confirmar rendición (Elaboración propia).

5.4.9 Modelamiento de datos

Diseño físico

Diseño físico con el cual se desarrolla el microservicios para el módulo rendiciones.

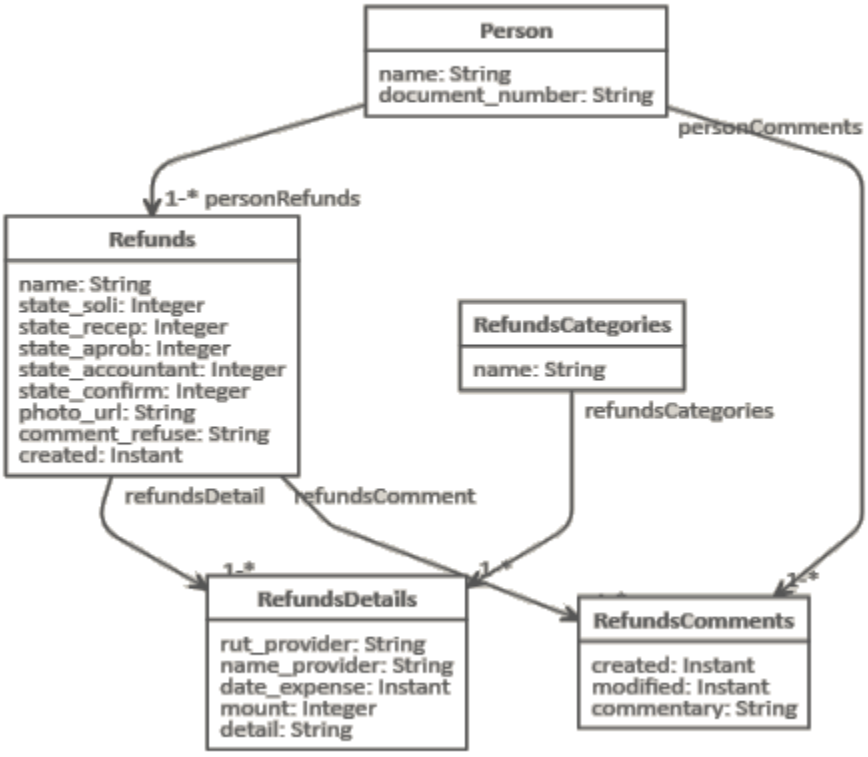
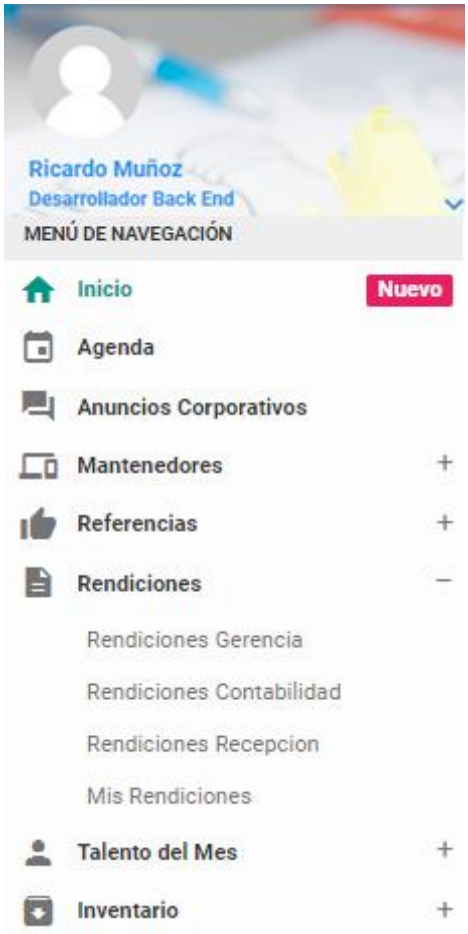


Figura 96. Diseño físico para el módulo rendiciones (Elaboración propia).

5.4.10 Ilustraciones de interfaz de usuario



Menú principal del sistema en donde se permite acceder al módulo rendiciones.

Este menú permite acceder a las vistas:

- Mis rendiciones: Ingresar nuevas rendiciones.
- Rendiciones Recepción: Aceptar o rechazar rendición.
- Rendiciones Gerencia:Aceptar o rechazar rendición.
- Rendiciones Contabilidad: Confirmar pago rendiciones.

Figura 97. Menú principal para el módulo rendiciones (Elaboración propia).

Interfaz para Mis rendiciones

Se listaran todas las rendiciones ingresadas por el usuario conectado. Ver imagen 98.a para ver la acción "Añadir".

Figura 98. Interfaz Mis rendiciones (Elaboración propia).

Añadir rendiciones

Figura 98.a.Interfaz Añadir rendición (Elaboración propia).

Interfaz para Rendiciones Recepción

Se listaran todas las rendiciones ingresadas al sistema, en donde el usuario recepcionista tendrá que aceptar o rechazar las rendiciones.

Figura 99. Interfaz Rendiciones recepción (Elaboración propia).

Interfaz para Rendiciones Gerencia

Se listaran solamente las rendiciones que hayan sido aceptadas previamente por el usuario de recepción para que posteriormente gerencia pueda aceptar o rechazar las rendiciones.

Figura 100. Interfaz Rendiciones gerencia (Elaboración propia).

Interfaz para Rendiciones Contabilidad

Se listara el monto final de todas las rendiciones aceptadas por gerencia, para que el usuario de contabilidad pueda confirmar el pago total de las rendiciones a quien corresponda.

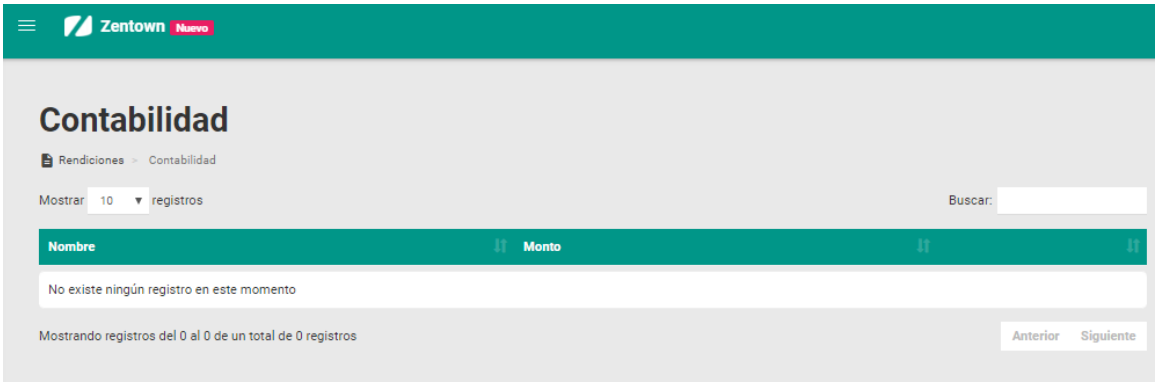


Figura 101. Interfaz Rendiciones contabilidad (Elaboración propia).

5.4.11 Ilustraciones microservicio módulo rendiciones

Se listaran imágenes de cómo está desarrollado a nivel código el microservicio, se hará referencia a los conceptos previamente descrito en el documento. Se ejemplificara solo la tabla Refunds del modelo físico.

Dominio

Clase java que representara la entidad renfunds

Refunds.java

```java
package cl.zenta.rrhhrefact.domain;


import javax.persistence.*;

/**
 * A Refunds.
 */
@Entity
@Table(name = "refunds", schema="rrhh_refunds")
@Document(indexName = "refunds")
public class Refunds implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "sequenceGenerator")
    @SequenceGenerator(name = "sequenceGenerator")
    private Long id;

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

    @Column(name = "state_soli")
    private Integer state_soli;

    @Column(name = "state_recep")
    private Integer state_recep;

    @Column(name = "state_aprob")
    private Integer state_aprob;

    @Column(name = "state_accountant")
    private Integer state_accountant;

    @Column(name = "state_confirm")
    private Integer state_confirm;
```

Figura 102. Dominio (Elaboración propia).

Dominio personalizado

CustomRefunds.java

```java
public class CustomRefunds {

    private Long id;
    /*Person*/
    private Long idPerson;
    private String namePerson;
    private String lastnamePerson;
    private String documentNumber;
    /*Refunds*/
    private String nameRefund;
    private Integer mount;
    private ZonedDateTime dateRefund;
    private Integer stateRecep;
    private Integer stateSoli;
    private Integer stateAprob;
    private Integer stateAccountant;
    private Integer stateConfirm;
    private ZonedDateTime dateAprobManagment;
    /*Detail*/
    private Long idDetails;
    private ZonedDateTime dateExpense;
    private String rutProvider;
    private String billTicket;
    private String nameProvider;
    private String detail;
    private Integer mountDetail;
    private String documentType;
    /*Categories*/
    private Long idCategories;
    private String nameCategories;
    /*Comments*/
    private Long idComments;
    private ZonedDateTime createdComment;
    private ZonedDateTime modifiedComment;
    private String commentary;
    /*Refunds*/
    private Integer active;
```

Figura 103. Dominio personalizado (Elaboración propia).

Repositorio

RefundsRepository.java

```java
package cl.zenta.rrhhrefact.repository;

import cl.zenta.rrhhrefact.domain.Refunds;


/**
 * Spring Data JPA repository for the Refunds entity.
 */
@SuppressWarnings("unused")
@Repository
public interface RefundsRepository extends JpaRepository<Refunds, Long> {

}
```

Figura 104. Repositorio (Elaboración propia).

Repositorio personalizado

CustomRefundsRepository.java

```
@SuppressWarnings("unused")
@Repository
public interface CustomRefundsRepository extends JpaRepository<Refunds, Long> {

    /*Query para obtener todas las rendiciones con detalles*/
    @Query("SELECT new cl.zenta.rrhhrefact.domain.custom.CustomRefunds("
            + "r.id, "
            + "p.id, "
            + "p.name, "
            + "p.lastname, "
            + "p.documentNumber, "
            + "r.name, "
            + "r.mount, "
            + "r.date_refund, "
            + "r.state_recep, "
            + "r.state_soli, "
            + "r.state_aprob, "
            + "r.state_accountant, "
            + "r.state_confirm, "
            + "r.dateAprobManagment, "
            + "rd.id, "
            + "rd.date_expense, "
            + "rd.rut_provider, "
            + "rd.bill_ticket, "
            + "rd.name_provider, "
            + "rd.detail, "
            + "rd.mount, "
            + "rd.documentType, "
            + "rc.id, "
            + "rc.name, "
            + "rcm.id, "
            + "rcm.created, "
            + "rcm.modified, "
            + "rcm.commentary, "
            + "r.active ) "
            + "FROM Refunds r, RefundsDetails rd, RefundsComments rcm "
            + "JOIN r.person p "
            + "JOIN rd.refund rdr "
            + "JOIN rd.refundsCategories rc "
            + "WHERE rdr=r AND rd.refund = rcm.refund ")
    List<CustomRefunds> getCustomRefundsAll();
```

Figura 105. Consulta JPA en repositorio personalizado (Elaboración propia).

Capa de servicio

CustomRefundsService.java

```
@Service
@Transactional
public class CustomRefundsService {

    private final Logger log = LoggerFactory.getLogger(CustomRefundsService.class);

    CustomRefundsRepository customRefundsRepository;

    RefundsRepository refundsRepository;


    public CustomRefundsService(CustomRefundsRepository customRefundsRepository,
                                RefundsRepository refundsRepository) {
        this.customRefundsRepository=customRefundsRepository;
        this.refundsRepository=refundsRepository;
    }


    /*Funcion  para los filtros*/
    public List<CustomRefundsPersonFilter> filterRefunds(){
        List<CustomRefundsPersonFilter> listrefunds = new ArrayList<CustomRefundsPersonFilter>();

                try
                {
                    listrefunds=customRefundsRepository.filterRefundByPerson();
                }
                catch (Exception e)
                {
                    log.error(e.getMessage(),e);
                }

        return listrefunds;

    }
```

Figura 106. Capa de servicio (Elaboración propia).

Recurso

RefundsResource.java

```java
package cl.zenta.rrhhrefact.web.rest;

import com.codahale.metrics.annotation.Timed;

/**
 * REST controller for managing Refunds.
 */
@RestController
@RequestMapping("/api")
public class RefundsResource {

    private final Logger log = LoggerFactory.getLogger(RefundsResource.class);

    private static final String ENTITY_NAME = "refunds";

    private final RefundsRepository refundsRepository;

    private final RefundsSearchRepository refundsSearchRepository;

    public RefundsResource(RefundsRepository refundsRepository, RefundsSearchRepository refundsSearchRepository) {
        this.refundsRepository = refundsRepository;
        this.refundsSearchRepository = refundsSearchRepository;
    }

     * POST  /refunds : Create a new refunds.
    public ResponseEntity<Refunds> createRefunds(@RequestBody Refunds refunds) throws URISyntaxException {

     * PUT  /refunds : Updates an existing refunds.
    public ResponseEntity<Refunds> updateRefunds(@RequestBody Refunds refunds) throws URISyntaxException {

     * GET  /refunds : get all the refunds.
    public List<Refunds> getAllRefunds() {

     * GET  /refunds/:id : get the "id" refunds.
    public ResponseEntity<Refunds> getRefunds(@PathVariable Long id) {

     * DELETE  /refunds/:id : delete the "id" refunds.
    public ResponseEntity<Void> deleteRefunds(@PathVariable Long id) {
}
```

Figura 107. Métodos del recurso (Elaboración propia).

Recurso personalizado

CustomRefundsResource.java

```java
package cl.zenta.rrhhrefact.web.rest.custom;

import java.util.List;

@RestController
@RequestMapping("/api/custom/")
public class CustomRefundsResource {
    private static final String ENTITY_NAME = "refunds";
    private CustomRefundsService customRefundsService;

    public CustomRefundsResource(CustomRefundsService customRefundsService) {
        this.customRefundsService=customRefundsService;
    }

    /*Para ver el monto total por persona*/
     public ResponseEntity<List<CustomPersonMount>> mountTotalPerson()

    /*Rendiciones por persona*/
     public ResponseEntity<List<CustomRefunds>> refundsByPerson(@PathVariable Long id)

     public ResponseEntity<List<CustomRefundsPersonFilter>> filterRefunds()

     public ResponseEntity<List<CustomRefunds>> getRefundsOfPerson()

     public ResponseEntity<List<CustomRefunds>> getRefundById(@PathVariable Long id)

     public ResponseEntity<Refunds> updateStatePerson(@PathVariable Long idRefund,@PathVariable Integer state)

}
```

Figura 108. Métodos de la capa de servicio (Elaboración propia).

5.4.12 Pruebas para el microservicio rendiciones

Se realizaran pruebas de entrada y salida al microservicio que contiene la logica para el módulo rendiciones.

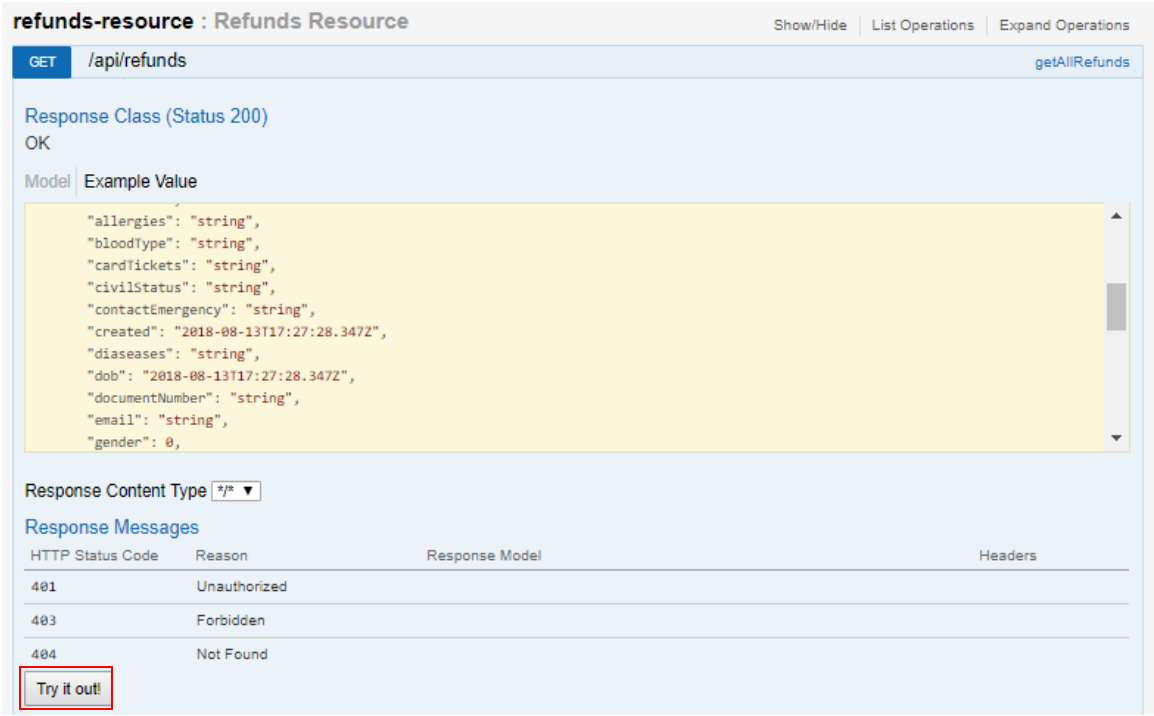


Figura 109. Recurso para obtener las rendiciones (Elaboración propia).

Al hacer click en “Try itout” tendremos la respuesta del recurso GET para la tabla Refunds.

Figura 110. Resultado de la solicitud (Elaboración propia).

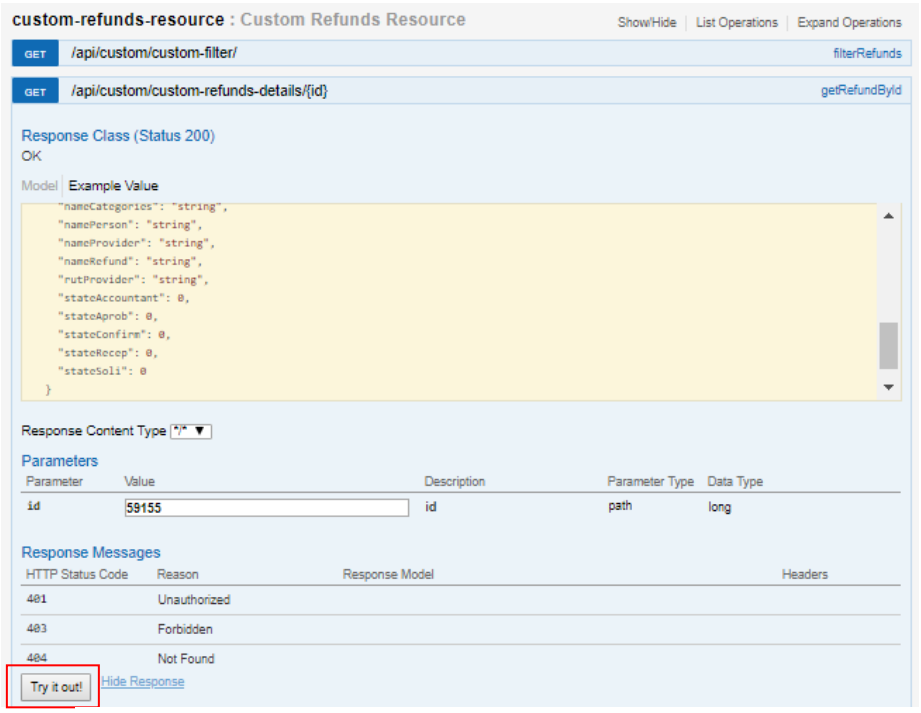


Figura 111. Recurso personalizado (Elaboración propia).

Al hacer click en "Try itout" tendremos la respuesta del recurso personalizado que consultara por el detalle de una rendición.

Figura 112. Resultado de un recurso personalizado (Elaboración propia).

5.4.13 Pruebas al módulo

Se realizaran pruebas de entradas y salidas al módulo además de simular los diferentes roles que interactúan.

Mis rendiciones

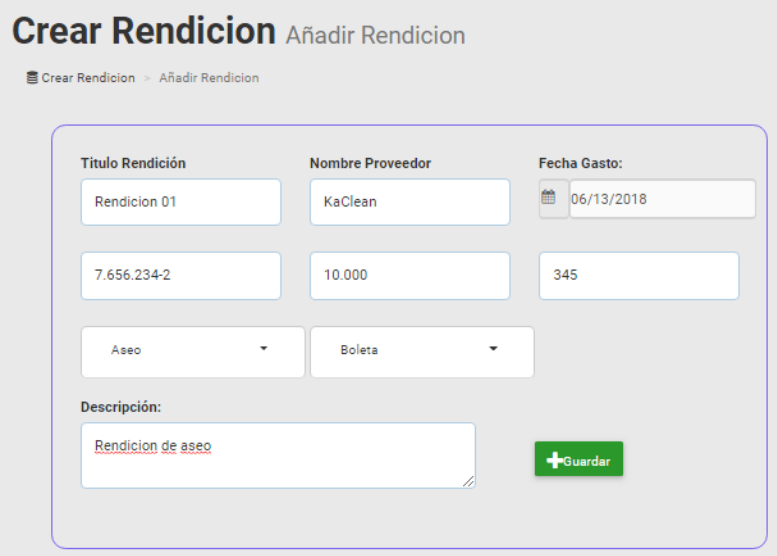


Figura 113. Añadir rendición (Elaboración propia).

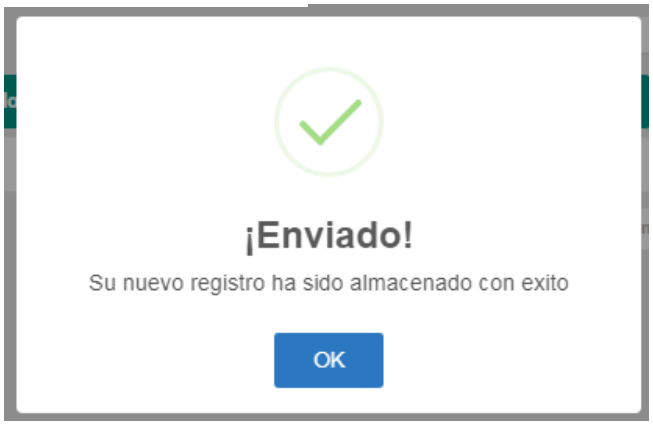


Figura 114. Rendición ingresada exitosamente.

Figura 115. Lista de rendiciones (Elaboración propia).

Subir boleta en formato imagen

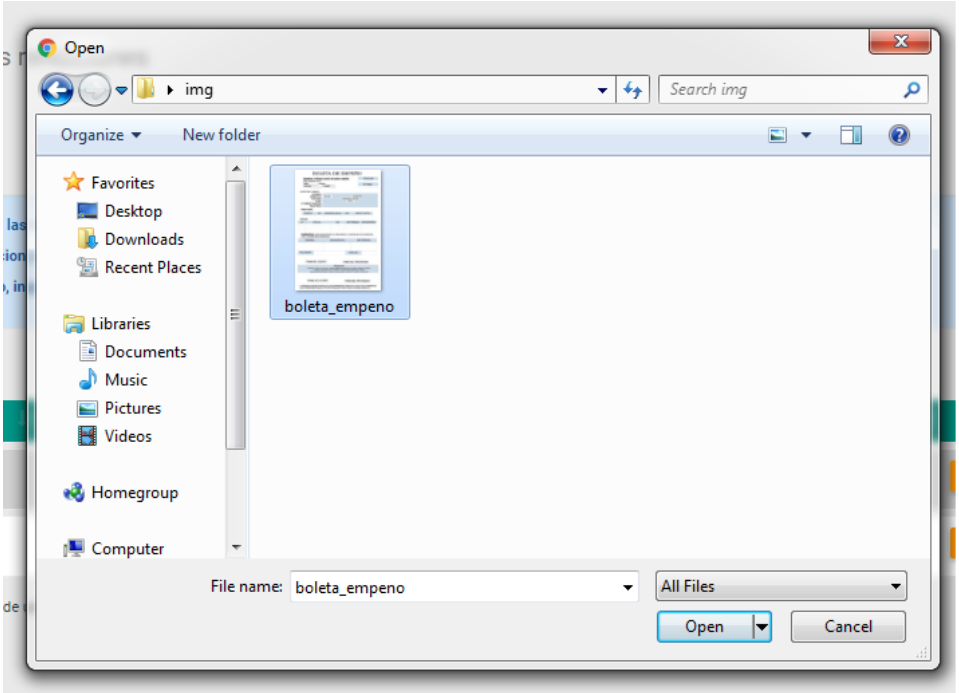


Figura 116. Seleccionar imagen (Elaboración propia).

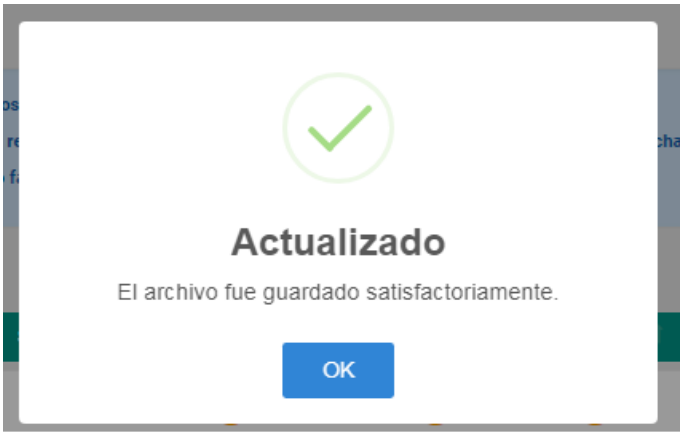


Figura 117. Imagen guardada (Elaboración propia).

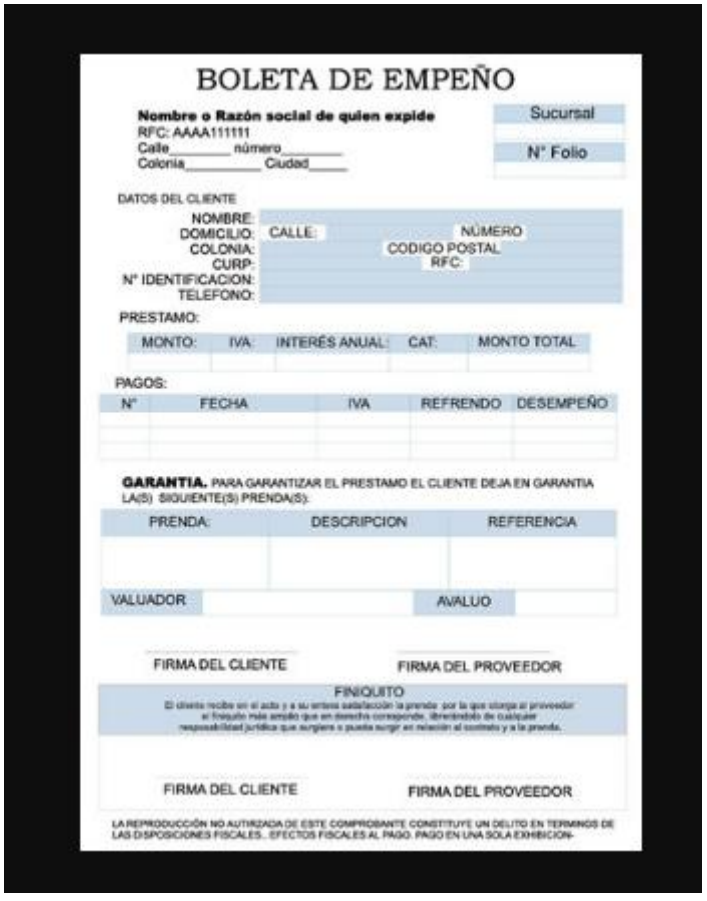


Figura 118. Visualización de imagen (Elaboración propia).

Rendiciones recepción

Figura 119. Vista recepción (Elaboración propia).

Figura 120. Recepcionar rendición.

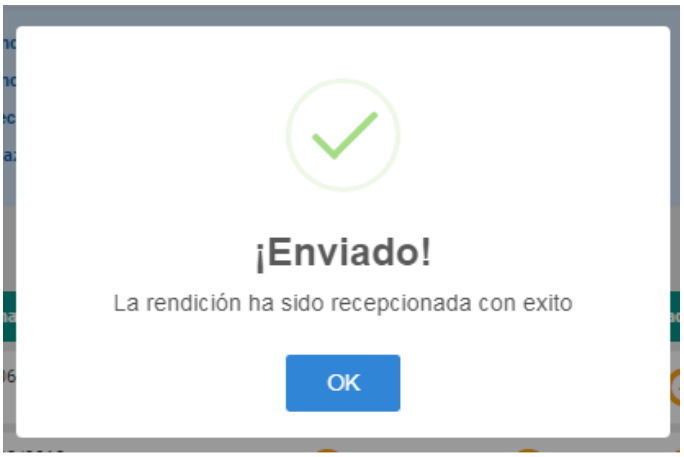


Figura 121. Rendición recepcionada.

Figura 122. Vista recepción con el estado recepción (Elaboración propia).

Rendiciones gerencia

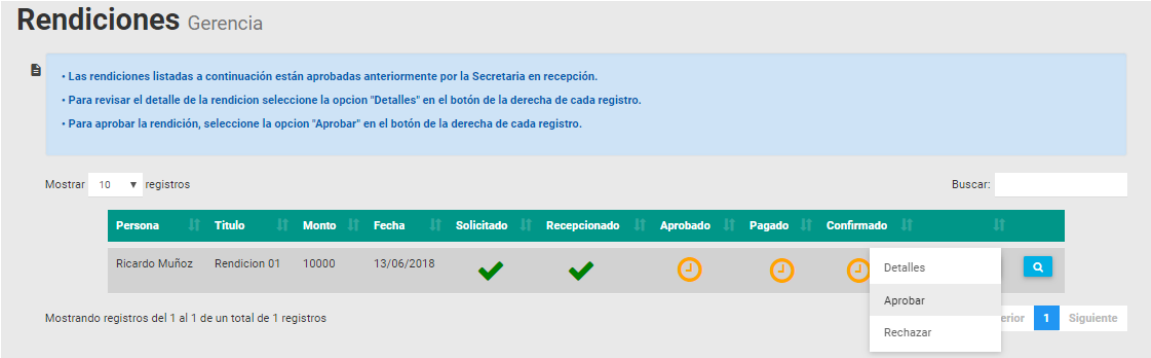


Figura 123. Vista gerencia (Elaboración propia).

Figura 124. Aceptar rendición (Elaboración propia).

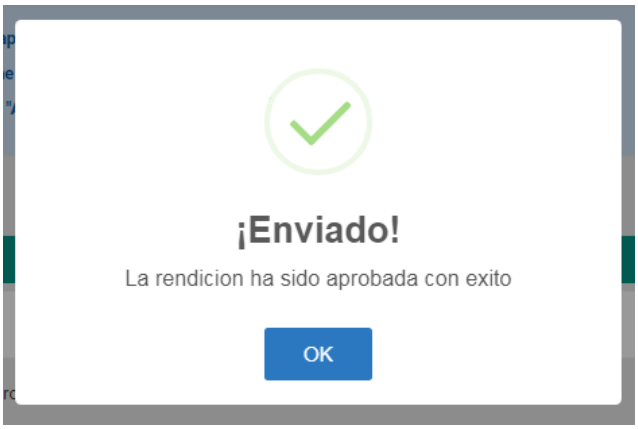


Figura 125. Rendición aceptada por gerencia (Elaboración propia).

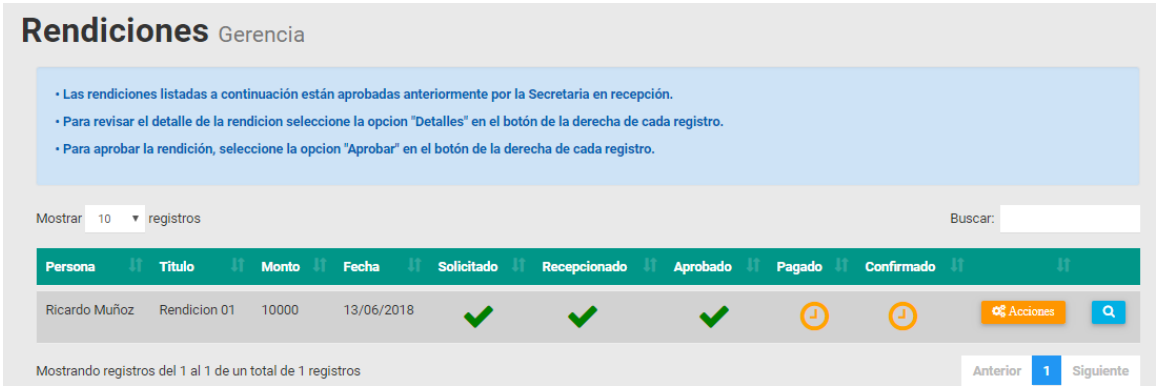


Figura 126. Vista gerencia con el estado aprobado (Elaboración propia).

Rendiciones contabilidad

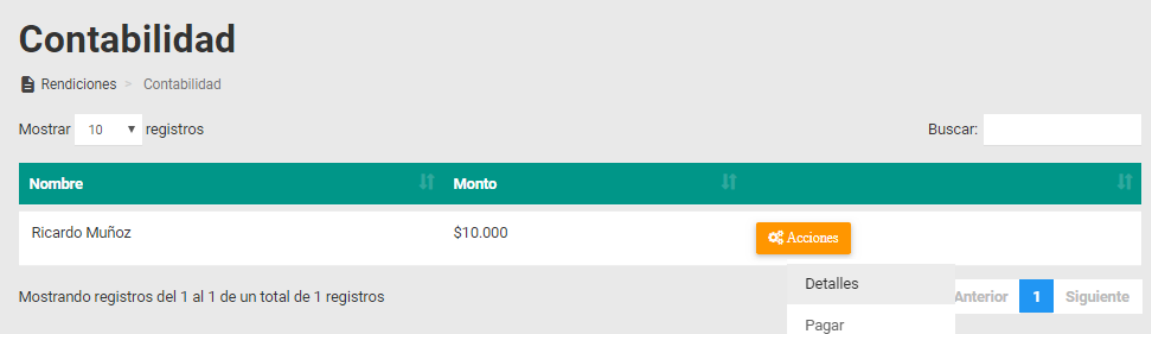


Figura 127. Vista contabilidad (Elaboración propia).

Figura 128. Detalles de la rendición para confirmar el pago (Elaboración propia).

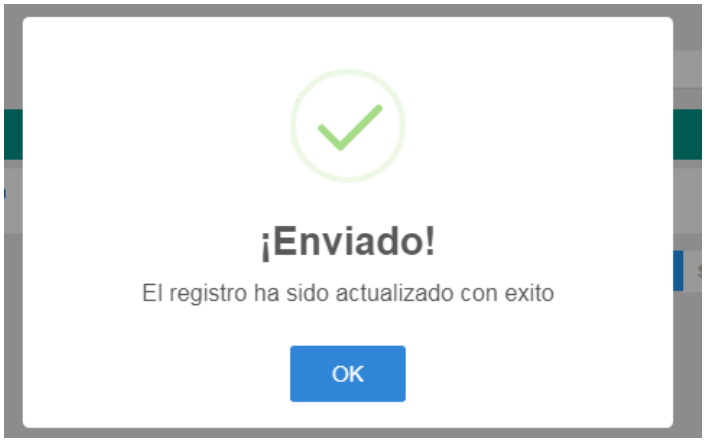


Figura 129. Confirmación del pago (Elaboración propia).

Se puede observar el proceso completado que tiene como objetivo el módulo rendiciones.

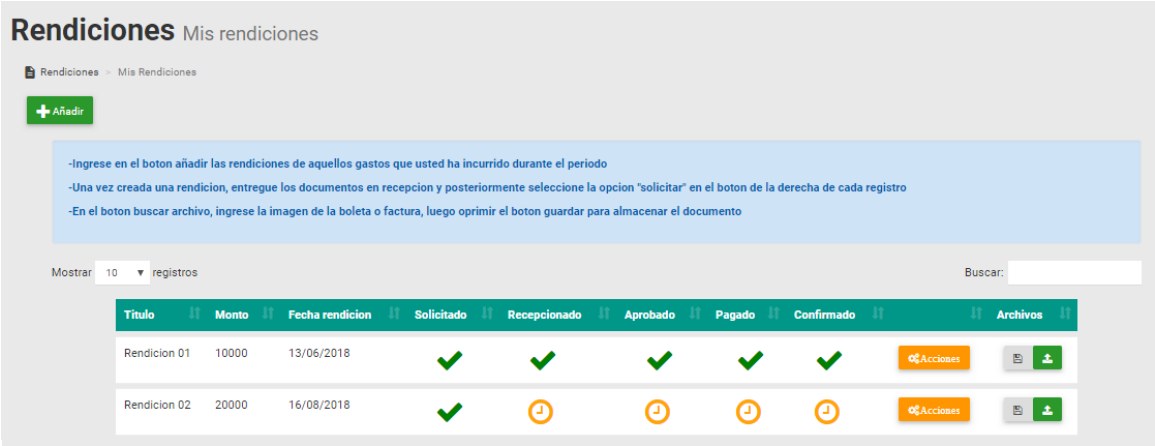


Figura 130. Mis rendiciones (Elaboración propia).

Al rechazar la segunda rendición en la interfaz de recepción, se logra observar que los demás estados de la solicitud se rechazaron de forma automática para los demás procesos.

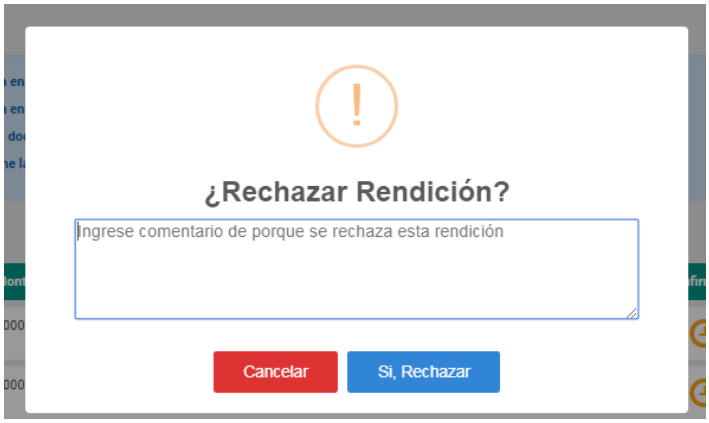


Figura 131. Rechazar rendición (Elaboración propia).

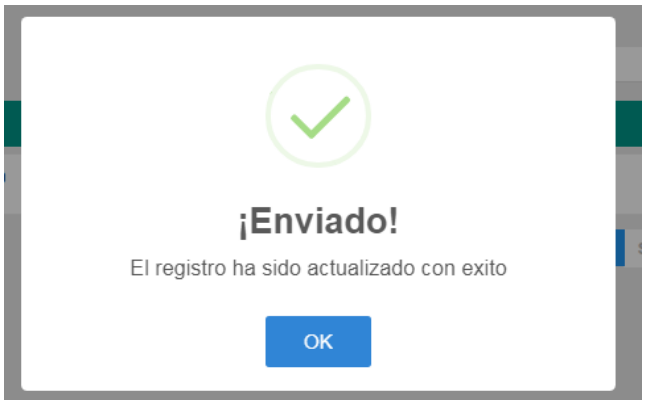


Figura 132. Confirmación de la solicitud.

Figura 133. Rendición rechazada (Elaboración propia).

Manejo de errores.

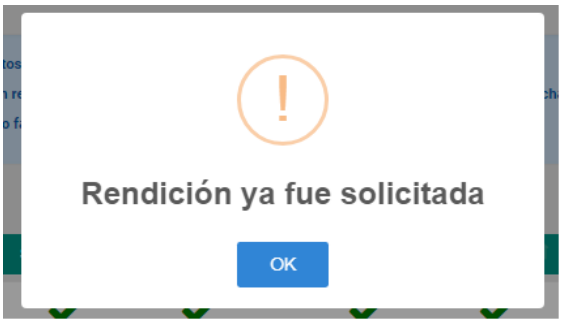


Figura 134. Solicitud no disponible (Elaboración propia).

Si no se sigue el conducto regular de una rendición se manejara con el mensaje de que la rendición ya fue solicitada. Unos de los casos puede ser que se quiera aceptar una rendición que ya fue rechazada.

Figura 136. Ver imagen (Elaboración propia).

Si se quiere visualizar la imagen y el sistema no la encuentra, se controlara con este mensaje.

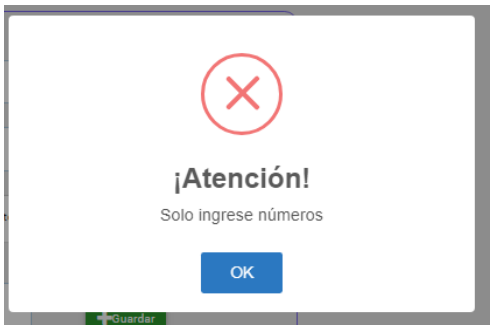


Figura 138. Campos incorrectos (Elaboración propia).

Control de error al momento de ingresar monto de la rendición, solo se acepta números.

Figura 135. Ver imagen(Elaboración propia).

Si se quiere guardar la imagen sin antes de haberla subido, este mensaje controlara ese error.

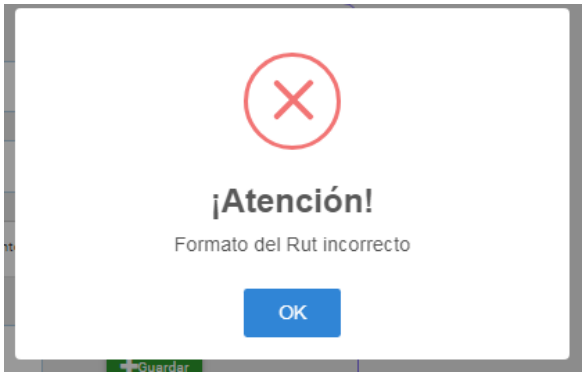


Figura 137. Campos incorrectos (Elaboración propia).

Control de error al momento de ingresar un rut con formato incorrecto.

Conclusión

Al desarrollar módulos bajo la arquitectura orientada a microservicios se logra evidenciar la eficiencia de los puntos principales que propone la arquitectura, que son el desarrollo independiente de los microservicios, el desacoplamiento de los componentes, la fragmentación del código fuente y la escalabilidad modular. Todas estas cualidades permiten que una aplicación esté apta para integraciones, tenga mayor tolerancia a fallos y mayor claridad en el código fuente, ya que cada microservicio se encuentra aislado del otro, y esto evita tener un solo proyecto con todo el código fuente. Como cada microservicio implementa su propia lógica de negocio, a los desarrolladores se les facilita el trabajo de revisar extensos archivos.

El uso de contenedores para alojar microservicios permite mantener distintas imágenes de un mismo microservicio, esto ocurre en el caso de que suceda algún tipo de incidencia, solo es necesario detener el contenedor actual y crear otro con la misma imagen del microservicio, esto permite tener un mayor control de la aplicación y no se debe ejecutar toda la aplicación nuevamente. Las herramientas de automatización basadas en contenedores automatizan el despliegue de cada contenedor, permitiendo dar aviso si existe algún problema en un contenedor. Si el microservicio alojado en un contenedor excede su capacidad de memoria, las herramientas de automatización tienen la capacidad de detener y poner en marcha los contenedores de forma automática. Se tiene que destacar la importancia del uso de contenedores y herramientas de automatización, ya que sin estos componentes no se estaría cumpliendo con la filosofía de la arquitectura basada en microservicios.

Por lo descrito anteriormente se puede concluir que la arquitectura orientada a microservicios es un nuevo paradigma de desarrollo, que plantea un cambio total en el desarrollo de aplicaciones, ya que actualmente estas aplicaciones se crean como un paquete gigante, una única unidad cohesiva de componentes, y el desarrollo basado en microservicios rompe estos esquemas permitiendo desacoplar los componentes internos ayudando a la escalabilidad y al entendimiento del código fuente disminuyendo tiempos de análisis, reparaciones del sistema o integraciones nuevas.

Empresas internacionales tales como Uber, Netflix y Amazon, han migrado a desarrollos basado en microservicios por sus niveles de concurrencia de usuario, ya que por la alta demanda que tienen sus servidores, las aplicaciones monolíticas no dan abasto a las solicitudes de usuario. Al migrar al desarrollo basado en microservicio han podido responder a la alta demanda de solicitudes, además de alivianar y mayor control en muchos de sus procesos.

Posibles tareas a realizar

- Difundir el concepto de microservicio y el desarrollo distribuido para la comunidad informática de la Universidad del Bío-Bío, realizando talleres y/o charlas para dar a conocer las nuevas tecnologías emergentes del mundo informático.
- Adaptar de forma eficiente la orquestación de varios microservicios, permitiendo orquestar e integrar microservicios de forma automática.
- Desarrollar microservicios en distintos lenguajes como C#, NodeJS, Phyton.

Bibliografía

[1] IBM Corp. (2004). Service-orientedarchitecture. Recuperado de http://cic.puj.edu.co/wiki/lib/exe/fetch.php?media=materias:soa-ibmvision.pdf.

[2] M. Josuttis, N. (2007). SOA in Practice [e-book]. Recuperado de https://www.safaribooksonline.com/library/view/soa-in-practice/9780596529550/ch01s04.html.

[3] Blancarte, O. (2014, 21 julio). Patrón de diseño Modelo Vista Controlador (MVC). Recuperado 17 mayo, 2018, de https://www.oscarblancarteblog.com/2014/07/21/patron-de-diseno-modelo-vista-controlador-mvc/.

[4] International Journal of ComputerApplications. (2014, 12 agosto). Implementation of Model-View-ControllerArchitecturePatternfor Business IntelligenceArchitecture. Recuperado 17 mayo, 2018, de https://pdfs.semanticscholar.org/41a3/3127ac74f126d6ae13a49431983fbda6f7cf.pdf.

[5]Eaves, J. (2014, 29 agosto). MicroserviceTrade-Offs. Recuperado de http://rea.tech/micro-services-what-even-are-they/.

[6] Lainez, J. R. L. F. (2014, 24 octubre). Desarrollo de Software ÁGIL: Extreme Programming y Scrum [e-book]. Recuperado de https://books.google.cl/books?hl=es&lr=&id=M4fJCgAAQBAJ&oi=fnd&pg=PA6&dq=desarrollo+agil+scrum&ots=1GYHRwWZnq&sig=Y_hRqZ1xhMK5abyFqzNCK67FCnE#v=onepage&q=desarrollo%20agil%20scrum&f=false.

[7] Clemson, T. (2014, 18 noviembre). TestingStrategies in a MicroserviceArchitecture. Recuperado de https://martinfowler.com/articles/microservice-testing/.

[8] Newman, S. (2015, febrero). BuildingMicroservices [e-book]. Recuperado de https://www.nginx.com/wp-content/uploads/2015/01/Building_Microservices_Nginx.pdf.

[9] Fowler, M. (2015, 1 julio). MicroserviceTrade-Offs. Recuperado de https://martinfowler.com/articles/microservice-trade-offs.html.

[10] Fowler, M. (2015, 1 julio). MicroservicesResourceGuide. Recuperado de https://martinfowler.com/microservices/.

[11] Red Hat Inc. (2016). CONSIGA UNA ARQUITECTURA DE MICROSERVICIOS EXITOSA. Recuperado de https://www.redhat.com/cms/managed-files/mi-microservices-architecture-design-whitepaper-inc0336100lw-201602-a4-es.pdf.

[12] Uit De Bulten, G. (2017, 12 junio). EMBRACING MICROSERVICES: HOW TO KEEP UP IN THE DIGITAL AGE. Recuperado de https://www.accenture-insights.nl/en-us/articles/6-advantages-microservices-based-architecture.

[13] Burns, B., Hightower, K., &Beda, J. (2017, septiembre). Kubernetes Up&Running [e-book]. Recuperado 1 julio, 2018, de https://mesosphere.com/wp-content/uploads/2017/09/running-kubernetes_oreilly-ebook.pdf.

[14] Oodles Technologies. (2018, 2 enero). Monolithic Vs MicroserviceArchitecture. Recuperado de https://www.oodlestechnologies.com/blogs/Monolithic-Vs-Microservice-Architecture.

[15] Microsoft Azure. (2018, 21 mayo). ServiceFabric and containers. Recuperado 25 julio, 2018, de https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-containers-overview.

[16] SaMSolutions. (s.f.). Microservices vs. Monolithic: Real Business Examples. Recuperado de https://www.sam-solutions.com/blog/microservices-vs-monolithic-real-business-examples/.

[17] IBM Knowledge Center. (s.f.). Service-orientedarchitecture (SOA). Recuperado de https://www.ibm.com/support/knowledgecenter/en/SSMQ79_9.1.1.1/com.ibm.egl.pg.doc/topics/pegl_serv_overview.html.

[18] Red Hat Inc. (s.f.). ¿What are microservices? Recuperado 25 junio, 2018, de https://www.redhat.com/es/topics/microservices/what-are-microservices.

[19] Red Hat Inc. (s.f.). ¿Qué es un contenedor de Linux? Recuperado 12mayo, 2018, de https://www.redhat.com/es/topics/containers/whats-a-linux-container.

[20] Red Hat Inc. (s.f.). ¿Qué es DOCKER? Recuperado 25 junio, 2018, de https://www.redhat.com/es/topics/containers/what-is-docker.

[21] Red Hat Inc. (s.f.). ADVANTAGES OF USING DOCKER. Recuperado 25 mayo, 2018, de https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/7.0_release_notes/sect-red_hat_enterprise_linux-7.0_release_notes-linux_containers_with_docker_format-advantages_of_using_docker.

[22] Red Hat Inc. (s.f.). GET STARTED ORCHESTRATING CONTAINERS WITH KUBERNETES. Recuperado 25 mayo, 2018, de https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html/getting_started_with_kubernetes/get_started_orchestrating_containers_with_kubernetes.

[23] Kubernetes. (s.f.). WhatisKubernetes? Recuperado 1 julio, 2018, de https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/.

.

.