UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE CUENCA

CARRERA DE INGENIERÍA DE SISTEMAS

Trabajo de titulación previo
a la obtención del título
de Ingeniero de Sistemas

PROYECTO TÉCNICO:

"ANÁLISIS, DISEÑO Y DESARROLLO DE UN SISTEMA PARA LA GESTIÓN DEL SEGUIMIENTO DE PASANTÍAS Y PRÁCTICAS PRE-PROFESIONALES A TRAVÉS DE LA IMPLEMENTACIÓN DE MICROSERVICIOS EN DOCKER PARA LA UNIVERSIDAD POLITÉCNICA SALESIANA"

AUTOR:

FRANK GABRIEL MONTALVO OCHOA

TUTOR:

ING. MAURICIO SERGIO ORTIZ OCHOA

CUENCA - ECUADOR

2020

CESIÓN DE DERECHOS DE AUTOR

Yo, Frank Gabriel Montalvo Ochoa con documento de identificación N° 0302612866, manifiesto mi voluntad y cedo a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que soy autor del trabajo de titulación: **"ANÁLISIS, DISEÑO Y DESARROLLO DE UN SISTEMA PARA LA GESTIÓN DEL SEGUIMIENTO DE PASANTÍAS Y PRÁCTICAS PRE-PROFESIONALES A TRAVÉS DE LA IMPLEMENTACIÓN DE MICROSERVICIOS EN DOCKER PARA LA UNIVERSIDAD POLITÉCNICA SALESIANA"**, mismo que ha sido desarrollado para optar por el título de: *Ingeniero de Sistema*s, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En aplicación a lo determinado en la Ley de Propiedad Intelectual, en mi condición de autor me reservo los derechos morales de la obra antes citada. En concordancia, suscribo este documento en el momento que hago entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Cuenca, octubre del 2020

Frank Gabriel Montalvo Ochoa

C.I. 0302612866

CERTIFICACIÓN

Yo, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: **"ANÁLISIS, DISEÑO Y DESARROLLO DE UN SISTEMA PARA LA GESTIÓN DEL SEGUIMIENTO DE PASANTÍAS Y PRÁCTICAS PRE-PROFESIONALES A TRAVÉS DE LA IMPLEMENTACIÓN DE MICROSERVICIOS EN DOCKER PARA LA UNIVERSIDAD POLITÉCNICA SALESIANA"**, realizado por Frank Gabriel Montalvo Ochoa, obteniendo el *Proyecto Técnico*, que cumple con todos los requisitos estipulados por la Universidad Politécnica Salesiana.

Cuenca, octubre del 2020

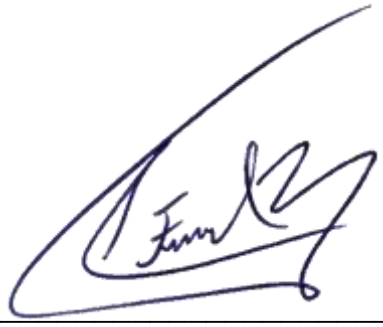
Ing. Mauricio Sergio Ortiz Ochoa

C.I. 0103667754

DECLARATORIA DE RESPONSABILIDAD

Yo, Frank Gabriel Montalvo Ochoa con documento de identificación N° 0302612866, autor del trabajo de titulación: **"ANÁLISIS, DISEÑO Y DESARROLLO DE UN SISTEMA PARA LA GESTIÓN DEL SEGUIMIENTO DE PASANTÍAS Y PRÁCTICAS PRE-PROFESIONALES A TRAVÉS DE LA IMPLEMENTACIÓN DE MICROSERVICIOS EN DOCKER PARA LA UNIVERSIDAD POLITÉCNICA SALESIANA"**, certifico que el total contenido del *Proyecto Técnico,* es de mi exclusiva responsabilidad y autoría**.**

Cuenca, octubre del 2020



Frank Gabriel Montalvo Ochoa

C.I. 0302612866

AGRADECIMIENTO

Agradezco a la Universidad Politécnica Salesiana y su cuerpo docente el cual supo darme su apoyo, guiarme e inculcarme conocimientos tanto en la parte profesional como ética. También agradezco a mi director de tesis el Ing. Mauricio Ortiz, por su apoyo y guía durante el desarrollo de este trabajo. Finalmente agradezco a todo el personal de la Secretaría técnica de tecnologías de la información con quienes he compartido momentos gratos y sin los cuales este trabajo no hubiese sido posible.

DEDICATORIA

Por siempre haber estado a mi lado y apoyarme durante todo este tiempo especialmente durante los momentos más difíciles; es por eso por lo que dedico esta tesis a mis padres **Gloria** y **Eliseo**, quienes con su esfuerzo y dedicación me motivaron a seguir adelante y no darme por vencido durante este arduo camino.

A toda mi familia por estar ahí para mí siempre que los necesitaba, especialmente a mis tías **Inés** y **Ruth**, y a mi querida abuela **Zoila**. De igual forma a todos mis amigos por brindarme su apoyo y amistad incondicional durante todo este tiempo.

ÍNDICE DE CONTENIDOS

RESUMEN 11
CAPÍTULO I INTRODUCCIÓN Y REVISIÓN DEL ESTADO DEL ARTE 12
1.1 INTRODUCCIÓN: 12
1.2 OBJETIVOS 13
1.2.1 OBJETIVO GENERAL 13
1.2.2 OBJETIVOS ESPECÍFICOS 13
1.3 JAVA 13
1.3.1 JAVA SE 14
1.3.2 JAVA EE 14
1.4 SERVIDOR DE APLICACIONES 15
1.4.1 COMPARATIVA ENTRE LOS DIFERENTES SERVIDORES DE APLICACIONES USADOS COMÚNMENTE EN LA ACTUALIDAD: 16
1.4.2 CONFIGURACIÓN DE UN SERVIDOR DE APLICACIONES 17
1.4.3 SERVIDOR DE APLICACIONES WILDFLY 17
1.5 CONTENEDORES 18
1.5.1 CONTENEDOR 18
1.5.2 CONTENEDORIZACIÓN 18
1.6 VIRTUALIZACIÓN 18
1.6.1 PROCESO DE LA VIRTUALIZACIÓN 18
1.6.2 VENTAJAS DE LA VIRTUALIZACIÓN 19
1.7 CONTENEDORES DE SOFTWARE 20
1.7.1 DOCKER 21
1.7.2 ARQUITECTURA DE DOCKER 22
1.7.3 IMÁGENES 22
1.7.4 CONTENEDORES 23
1.8 MICROSERVICIOS 23
1.8.1 KUBERNETES 23
1.8.2 KUBERNETES Y MICROSERVICIOS 24
CAPÍTULO II ELICITACIÓN Y ANÁLISIS DE LOS REQUERIMIENTOS 25
2.1 REQUERIMIENTOS DE SOFTWARE 25
2.1.1 FUENTES DE REQUISITOS 25
2.1.2 ELICITACIÓN DE REQUERIMIENTOS 26
2.2 REQUISITOS FUNCIONALES Y NO FUNCIONALES 26

2.2.1 REQUISITOS FUNCIONALES 26
2.2.2 REQUISITOS NO FUNCIONALES 26
2.3 ANÁLISIS DE REQUISITOS 26
2.3.1 CLASIFICACIÓN DE LOS REQUISITOS 26
2.3.2 MODELADO DEL SISTEMA 27
2.3.3 ARQUITECTURA DEL SISTEMA 27
2.4 DESCRIBIENDO LA REGLAS DE NEGOCIO 27
2.4.1 GESTIÓN ADMINISTRATIVA 27
2.4.2 MICROSERVICIO 28
2.4.3 CLIENTE MÓVIL 30
CAPÍTULO III DISEÑO Y DESARROLLO DEL SISTEMA 33
3.1 DISEÑO DEL SISTEMA 33
3.2 DESARROLLO DEL SISTEMA 34
3.2.1 ARQUITECTURA N-CAPAS 34
3.2.2 DISEÑO DE ESQUEMA ENTIDAD RELACIÓN 36
3.2.3 CONFIGURACIÓN DEL ENTORNO DE DESARROLLO 37
3.2.4 DESARROLLO DEL SISTEMA DE GESTIÓN DE PASANTÍAS 52
3.2.5 DESARROLLO DE LOS SERVICIOS 56
3.2.6 DESARROLLO DE LA APLICACIÓN MÓVIL 58
CAPÍTULO IV IMPLEMENTACIÓN DEL SISTEMA SOBRE UNA ARQUITECTURA DE MICROSERVICIOS 62
4.1 IMPLEMENTACIÓN SOBRE DOCKER 62
4.2 IMPLEMENTACIÓN SOBRE KUBERNETES 68
4.3 AÑADIENDO SEGURIDAD AL SERVICIO 74
4.3.1 JWT 75
4.3.2 OAUTH 76
CAPÍTULO V CONCLUSIONES Y RECOMENDACIONES 79
5.1 CONCLUSIONES 79
5.2 RECOMENDACIONES 80
5.3 TRABAJO FUTURO 80
REFERENCIAS 82

ÍNDICE DE FIGURAS

Figura 1. Buque portacontenedores. (Sarmiento, 2019) 18
Figura 2. Representación de un entorno virtualizado. (Reis, 2013) 19
Figura 3. Logotipo de Docker. (Docker, 2020) 21
Figura 4. Comparativa entre Docker y la virtualización. (Docker, 2020) 21
Figura 5. Arquitectura de Docker. (Docker, 2020) 22
Figura 6. Logotipo de Kubernetes. (Kubernetes, 2020) 23
Figura 7. Arquitectura de un Nodo. (Kubernetes, 2020) 24
Figura 8. Arquitectura del sistema de gestión de prácticas pre-profesionales. 33
Figura 9. Distribución de una arquitectura n-capas. 34
Figura 10. Esquema E-R del sistema de gestión de seguimiento de pasantías. 36
Figura 11. Configuración del Servidor de aplicaciones. 38
Figura 12. Creación del proyecto. 39
Figura 13. Estructura del proyecto en Netbeans. 39
Figura 14. Agregando submódulos al módulo padre. 40
Figura 15. Spring Initializr. 41
Figura 16. Pantalla de configuración del proyecto. 42
Figura 17. Pantalla para configuración de dependencias. 42
Figura 18. Estructura de un proyecto en Spring. 43
Figura 19. Clase que expone un servicio Rest. 43
Figura 20. Respuesta del servicio en el navegador. 44
Figura 21. Arquitectura de Flutter. (Flutter, 2020) 44
Figura 22. Jerarquía de widgets en Flutter. (Flutter, 2020) 45
Figura 23. Salida del comando flutter doctor. 45
Figura 24. Estructura de un proyecto en Flutter. 47
Figura 25. Código para crear una vista simple en Flutter. 47
Figura 26. Disposición de los widgets en la pantalla. 48
Figura 27. Inicializando un nuevo repositorio. 49
Figura 28. Configuración del remoto. 50
Figura 29. Rastreando archivos y confirmando cambios en el repositorio. 50
Figura 30. Resumen de la confirmación de cambios. 51
Figura 31. Control de versiones en Sourcetree. 52
Figura 32. Mantenimiento del tipo de práctica. 52
Figura 33. Filtros de búsqueda. 53
Figura 34. Mantenimiento de la práctica. 54
Figura 35. Creando/editando un tutor 54
Figura 36. Mantenimiento de Parámetros. 55
Figura 37. Mantenimiento de Actividades. 56
Figura 38. Listado de prácticas del estudiante. 57
Figura 39. Actividades de la práctica número tres. 57
Figura 40. Listado de prácticas en la aplicación. 59
Figura 41. Detalle de una práctica en la aplicación. 60
Figura 42. Pantalla de edición/creación de actividades. 61

Figura 43. Hola mundo desde Docker. 63
Figura 44. Dockerfile. 64
Figura 45. Docker Build. 66
Figura 46. Imágenes en Docker. 66
Figura 47. Inicialización del contenedor. 67
Figura 48. Probando el microservicio sobre Docker. 68
Figura 49. Archivo de configuración de Kubernetes 70
Figura 50. Kubeclt apply. 71
Figura 51. Pods de Kubernetes. 72
Figura 52. Minikube Dashboard. 72
Figura 53. Imagen de Docker ejecutándose sobre Kubernetes. 73
Figura 54. Endpoints del microservicio. 73
Figura 55. Probando el microservicio sobre Kubernetes. 74
Figura 56. Configuración de la seguridad en el servicio REST. 77
Figura 57. Método para inicio de sesión en Dart. 77
Figura 58. Pantalla de Inicio de sesión en la aplicación 78

ÍNDICE DE TABLAS

Tabla 1. Servidores de aplicaciones compatibles con Java. 16
Tabla 2. Contenedores de Software 20
Tabla 3. Argumentos para la creación de un nuevo proyecto. 46
Tabla 4. Hipervisores requeridos por plataforma 69
Tabla 5. Cabecera JWT. 75
Tabla 6. Contenido JWT. 76

RESUMEN

Hoy en día cuando se desarrolla software, siempre se presentan una serie de inconvenientes, ya sea durante el desarrollo o despliegue de este; estos inconvenientes por lo general tienen que ver con los SDK, bibliotecas o dependencias necesarias para el desarrollo o ejecución, puede que el software o aplicación sea incompatible con las versiones del SDK y librerías disponibles en el entorno donde se va a ejecutar, por ende, no se podrá desplegar ni ser utilizado por él o los usuarios. Para solucionar estos inconvenientes es que surgen los contenedores de Software. Los contenedores de software empaquetan todas las dependencias necesarias para ejecutar una aplicación, gracias a estos se puede construir el contenedor con todo lo necesario para su ejecución y desplegarlo en casi cualquier entorno, ya sea de desarrollo o producción.

Es gracias a esto que se puede olvidar de los problemas de compatibilidad o falta de dependencias, puesto que solo necesita el contendor y ejecutarlo para poner en marcha una o más aplicaciones, dado que los contenedores de software permiten escalar fácilmente las aplicaciones al crear y desplegar más de estos contenedores según las características y funciones requeridas, permitiendo sacar un mayor partido al hardware de los servidores. Esto sumado a los orquestadores de contendores, permite automatizar las tareas de despliegue y escalabilidad de las aplicaciones, además de ofrecer alta disponibilidad y tolerancia a fallos, contando con una infraestructura mucho más robusta, capaz de adaptarse fácilmente a la demanda de servicios solicitados a través de la red.

CAPÍTULO I
INTRODUCCIÓN Y REVISIÓN DEL ESTADO DEL ARTE

1.1 INTRODUCCIÓN:

Al hablar de servicios es inevitable asociarlos con los servidores, centros de datos y toda la infraestructura necesaria para su funcionamiento y despliegue; es por eso que al hablar de servicios uno puede imaginar que estos funcionan sobre grandes y costosos equipos de cómputo capaces de ejecutar aplicaciones en simultáneo y aunque esto pudo haber sido cierto en otros tiempos, la verdad es que en la actualidad, los avances tecnológicos que han surgido permiten sacar un mayor partido en base a los costosos de equipos.

La virtualización permite alojar dos o más sistemas operativos huéspedes sobre un mismo servidor, lo que permite aprovechar al máximo los recursos disponibles, ahorrar en consumo energético y espacio en los centros de datos. Sin embargo, la virtualización tiene ciertas desventajas como, por ejemplo: el uso de memoria y espacio en disco. Dado que cada instancia virtual es un sistema operativo completo que debe incluir todos los archivos y bibliotecas necesarias para el despliegue de una aplicación o servicio; y si existen varias instancias virtuales que utilizan los mismos archivos y bibliotecas implicaría que estos archivos existen más de una vez en espacio en disco.

Para solucionar este problema, existe la virtualización a nivel de sistema operativo, a lo que se conoce como contenedor de software; los contenedores solucionan el problema de las bibliotecas y archivos duplicados, esto permite que todos esos archivos existan una única vez en disco y sean compartidos entre los contenedores que los requieran.

A los servicios que se ejecutan sobre un contenedor de software se los conoce como microservicios, estos microservicios se ejecutan de forma independiente y aislada de los demás contenedores, por lo que, si uno tiene un mal funcionamiento, los demás pueden seguir funcionando con normalidad. Ahora, si se busca aumentar la disponibilidad y fiabilidad de los servicios expuestos, es posible complementar los contenedores con software de orquestación de contenedores, lo cual se encargará de administrar los recursos asignados y desplegar un set de réplicas de un mismo contenedor para asegurar su continua disponibilidad. (Microsoft, 2020)

Para llevar a cabo este proyecto se hará uso de Docker como contenedor de software y de Kubernetes como software de orquestación de contenedores, sobre los cuales se ejecutarán servicios para la consulta de las prácticas de un estudiante y la adición de actividades a las mismas.

1.2 OBJETIVOS

1.2.1 OBJETIVO GENERAL

Implementar una aplicación para el seguimiento a las pasantías y prácticas pre-profesionales haciendo uso de microservicios con Docker.

1.2.2 OBJETIVOS ESPECÍFICOS

- Investigar las herramientas y recursos disponibles para el diseño y desarrollo del sistema.
- Analizar y elaborar la especificación de requerimientos del sistema.
- Realizar un análisis y diseño del sistema acorde a los requerimientos solicitados.
- Diseñar e implementar el sistema para la gestión de seguimiento de pasantías acorde a una estructura adecuada.
- Desarrollar e implementar un módulo encargado de ofrecer los servicios a través de un API Rest.
- Investigar sobre la arquitectura basada en microservicios y la forma de acoplarse a los servicios API Rest.
- Desplegar microservicios para exponerlos a través de la red.
- Realizar pruebas y validar el sistema.

1.3 JAVA

Primero una breve introducción al principal lenguaje de programación que se utilizara en el desarrollo del proyecto. Java es un lenguaje de programación multiplataforma y orientado a objetos el cual fue creado por Sun Microsystems por un equipo dirigido por James Gosling. Al ser un lenguaje orientado a objetos, este permite hacer uso del paradigma de la programación orientada a objetos o POO, que ofrece la posibilidad de abstraer las características o atributos de los objetos del mundo real e incorporarlos en el software a través del uso de clases, ya que en estas se pueden definir las propiedades o atributos de dicho objeto. Java también permite hacer uso de otros conceptos de la programación orientada a objetos, como la herencia, propiedad a través de la cual somos capaces de transferir los atributos y funciones de una clase a otra. También permite separar los elementos de un software en componentes o módulos, lo cual facilita enormemente desarrollar, ya que se vuelve más entendible, fácil de mantener, modificar y añadir nuevas funciones. (Alicia Durango, 2016)

Los programas desarrollados con el lenguaje de programación Java tienen la cualidad de ser capaces de ejecutarse en cualquier plataforma, esto es posible gracias a la JVM (Java Virtual Machine) o máquina virtual de Java. Todo el código Java que se escriba para el software pasa

por el compilador de Java y luego va hacia la máquina virtual de Java antes de que pueda ser ejecutado, el cual cumple la función de traducir el código binario generado por el compilador en código máquina el cual es capaz de ser interpretado directamente por el hardware del dispositivo sobre el cual se esté desarrollando. (Díaz, 2019)

Cada sistema operativo cuenta con su propia implementación de la arquitectura de la máquina virtual de Java, otorgando a las aplicaciones escritas en Java la posibilidad de desplegarse en cualquier otro sistema o plataforma que cuente con una JVM.

1.3.1 JAVA SE

Es la edición estándar de Java, integra el kit de desarrollo de Java conocido como JDK, el cual cuenta con todas las herramientas necesarias para compilar, depurar, generar documentación, funciones para empaquetar código en archivos(.jar) y ejecutarlos, además también incluye una JVM necesaria para desplegar aplicaciones escritas en Java.

1.3.2 JAVA EE

Java EE o Java Enterprise Edition, es una plataforma para el desarrollo de aplicaciones empresariales. Está conformada por un conjunto específico de API's con las que se debe trabajar en conjunto cuando se desarrolla aplicaciones empresariales del lado del servidor. (Heffelfinger, 2017)

1.3.3 TIPOS DE EJB

EJB presenta distintos tipos de beans, entre los siguientes están:

Beans de sesión: Se encargan de mantener el control de flujo de datos o procesos de negocio. Aquí se pueden encontrar dos tipos de EJB; EJB con estado, que almacenan información sobre la sesión del usuario, y los EJB sin estado, los cuales no almacenan ningún tipo de información.

Beans de Entidad: Se encargan de abstraer los datos almacenados en el servidor y los representa a través de los beans de sesión. Estos se dividen en dos: Persistencia gestionada por **bean**, que se encarga de obtener directamente los datos desde la B.D hacia el objeto; y los de Persistencia gestionada por el **contenedor**, el cual se encarga de hacer una asociación entre los atributos o columnas de una tabla y los atributos de un objeto.

Beans dirigidos por mensajes: Se encargan de manejar los mensajes que se envían o reciben desde los componentes del software de forma asíncrona. (DeMichiel, 2001)

1.3.4 SEGURIDAD

Existen varios aspectos que se deben tener en cuenta al momento de desarrollar una aplicación empresarial, entre las siguientes están:

- **WEB:** Se define por la especificación del servlet y la versión de JEE con la que se vaya a trabajar.

- **EJB:** Se puede manejar de dos maneras, ya sea por autenticación la cual debe ser procesada antes de tener acceso a cualquier método dentro del sistema, o por autorización, la cual debe discernir a cuáles métodos tiene acceso el usuario luego del a autenticación.

La versión 8 fue la última disponible para la plataforma Java EE; en la actualidad fue reemplazado por **Jakarta EE**, el cual proporciona una completa compatibilidad con su versión predecesora. La Eclipse Foundation está a cargo del desarrollo y mantenimiento de Jakarta, luego de que en el año 2017 Oracle cediera los derechos de Java EE a Eclipse. La primera versión de Jakarta, nombrada Jakarta EE 8 fue lanzada el 10 de septiembre de 2019. (Späth, 2019)

1.4 SERVIDOR DE APLICACIONES

Está estrechamente relacionado a los sistemas distribuidos, permitiendo aplicar conceptos como alta disponibilidad, escalabilidad y mantenimiento a las aplicaciones. (Miguel, 2015)

Los elementos necesarios para desplegar una aplicación en un servidor son:

- **Cliente**: comúnmente se trata de un navegador web que interactúa con el servidor a través del protocolo HTTP o HTTPS para solicitar las páginas HTML.
- **Aplicación Cliente**: similar al anterior, salvo que estas no necesitan de un navegador para interactuar con el servidor y pueden usar cualquier otro medio o protocolo para solicitar datos.
- **Servidor web**: es el intermediario entre el servidor de aplicaciones y los clientes, es el encargado de presentar la información al cliente, usa el protocolo HTTP o HTTPS para transferir las páginas HTML hacia el cliente.
- **Servidor de aplicaciones**: es el encargado de desplegar las aplicaciones en el servidor, es el componente central del sistema.

1.4.1 COMPARATIVA ENTRE LOS DIFERENTES SERVIDORES DE APLICACIONES USADOS COMÚNMENTE EN LA ACTUALIDAD:

Tabla 1. Servidores de aplicaciones compatibles con Java. (Oracle, 2020)

Servidor	Propietario	Última versión	Compatibilidad con JEE	Licencia
Glassfish	GlassFish Community	5.1.0	JEE 8	Eclipse Public License & GNU General Public License
JBoss Enterprise Application Platform	Red Hat	7.2	JEE 8	GNU Lesser General Public License
Jetty	Eclipse Foundation	9.4.20	JEE 7	Apache License 2.0, Eclipse Public License 1.0
Lucee	Lucee Association Switzerland	5.3.2	JEE 7	LGPL v2.1
Payara	Payara	5.193	JEE 8	Common Development and Distribution License & GNU General Public License
Resin Servlet Container	Caucho Technology	4.0.62	JEE 6 Web Profile	GPLv3 or proprietary
Tamcat	Apache Software Foundation	9.0.24	JEE 8	Apache License 2.0
TomEE	Apache Software Foundation	7.1.1	JEE 6 Web Profile	Apache License 2.0
Wildfly	Red Hat(JBoss)	18.0.0.Final	JEE 8	GNU Lesser General Public License

1.4.2 CONFIGURACIÓN DE UN SERVIDOR DE APLICACIONES

Varía según el sistema operativo en el cual se va a realizar la instalación, por lo cual los pasos para configurar el servidor podrían variar de un sistema a otro. Por lo general hay algunos aspectos en el wizard del servidor que se deben tomar en cuenta a la hora de desplegar la aplicación, como puede ser la creación de un usuario administrador, configuración de seguridad, rutas y directorios a través de los cuales va a funcionar la aplicación.

1.4.3 SERVIDOR DE APLICACIONES WILDFLY

Se trata de un servidor de aplicaciones desarrollado inicialmente por **JBoss Inc**, y posteriormente mantenido por **Red Hat**. Es de código abierto y está publicado bajo la licencia LGPL por lo que cualquiera es capaz de hacer uso de este. Está escrito en Java, lo que le convierte en un software multiplataforma. (Marchioni, 2018)

Su primera versión (8.0.0) compatible con Java EE 7 fue lanzada en febrero de 2008; actualmente se encuentra en la versión 18.0.0. Final la cual es compatible con Java EE 8 además de contar con una integración casi total con el JDK 13 de Java.

Entre sus principales características están (Wildfly, 2020):

- **Inicio rápido**: se vale de los núcleos del procesador para iniciar sus procesos en paralelo, además asigna prioridades a los procesos, permitiendo ejecutar primero los de más alta prioridad.
- **Alto rendimiento y escalabilidad**: es capaz de tolerar una gran cantidad de concurrencia gracias a Undertow su servidor web de alto rendimiento.
- **Optimización del uso de memoria**: hace uso de índices de memoria para determinar objetos y datos duplicados almacenados en memoria y los elimina de la memoria, de esta forma reduce el consumo de memoria e incrementa la velocidad de respuesta.
- **Personalizable**: cuenta con una estructura dividida en módulos, los cuales se pueden acoplar o desacoplar según se requiera, esto permite ahorrar espacio en disco.
- **Configuración y administración**: cuenta con un archivo de configuración el cual se encuentra organizado por subsistemas el cual es sencillo de comprender y configurar según se desee.

1.5 CONTENEDORES

1.5.1 CONTENEDOR

Figura 1. Buque portacontenedores. (Sarmiento, 2019)

"Contenedor, término genérico utilizado para designar una caja que transporta mercancías, suficientemente resistente para su reutilización, habitualmente apilable y dotado de elementos para permitir la transferencia entre modos de transporte" (Sarmiento, 2019).

1.5.2 CONTENEDORIZACIÓN

Se refiere al sistema de transporte multimodal producto del uso de contenedores, así como la infraestructura necesaria para la logística del transporte. Principalmente, se basa en el hecho de apilar o acomodar mercancías en contenedores para posteriormente embarcarse para su transporte.

La contenedorización reduce los tiempos y costes en el transporte de los contendores; tuvo su origen a partir de la estandarización de los contenedores como herramientas para el transporte de mercancías, dado que un contenedor es independiente al medio de transporte que se emplee para llegar a su destino.

1.6 VIRTUALIZACIÓN

1.6.1 PROCESO DE LA VIRTUALIZACIÓN

Se trata de un proceso mediante el cual se simulan los componentes físicos de un ordenador común en un entorno virtual, el cual puede tener una o más instancias de un Sistema Operativo ejecutándose sobre el mismo. (Reis, 2013)

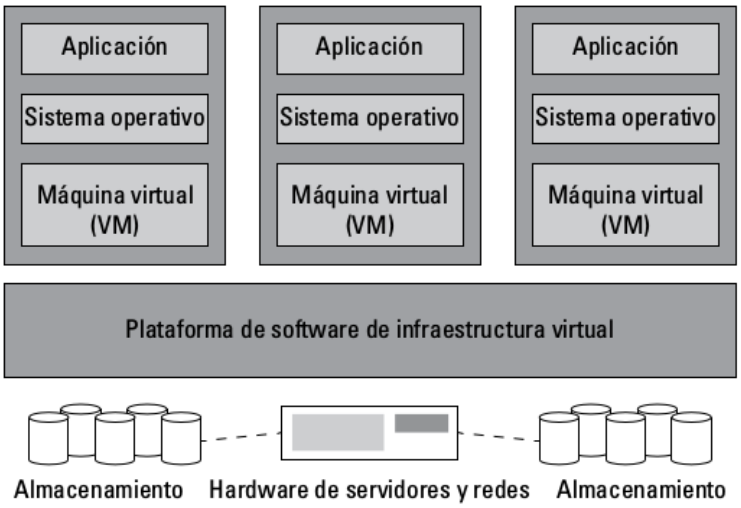


Figura 2. Representación de un entorno virtualizado. *(Reis, 2013)*

El proceso de virtualización es posible gracias a un software especializado conocido como **hipervisor**; el hipervisor es el encargado de crear y asignar recursos a los entornos virtuales, también es el encargado de gestionar la comunicación entre las instancias virtuales. La virtualización puede llevarse a cabo de dos formas:

- La primera es instalar el hipervisor directamente sobre el hardware del ordenador o servidor sobre el cual se desea desplegar los entornos virtuales, a este tipo de virtualización se la conoce como *bare metal*, hace uso de un software dedicado exclusivamente a la tarea de gestionar los recursos del servidor y las instancias de entornos virtuales que se ejecutan sobre el mismo.
- La segunda consiste en instalar el hipervisor sobre un sistema operativo anfitrión el cual hará de host para los entornos virtuales que se instalen sobre el mismo; el sistema operativo anfitrión crear varias instancias de entornos virtuales a través de su propio kernel.

1.6.2 VENTAJAS DE LA VIRTUALIZACIÓN

- Ahorran espacio al permitir que varias instancias de servidores y sistemas operativos coexistan a la vez en un mismo servidor.
- Agiliza el proceso de instalación y despliegue de nuevos servicios.
- Aprovecha al máximo los recursos de hardware del ordenador anfitrión.
- Simplifica la gestión y administración de los servidores.

(Reis, 2013)

1.7 CONTENEDORES DE SOFTWARE

Se trata del mismo principio de la logística de transporte internacional, pero con un enfoque en la informática y el desarrollo de aplicaciones. Básicamente, se trata de virtualizar varias instancias de espacios de usuario dentro de un sistema operativo anfitrión o host. Por lo general, estos entornos suponen una carga muy pequeña o casi inexistente para el sistema anfitrión a diferencia de la virtualización tradicional que exige una mayor carga en cuanto a recursos de hardware y software del sistema anfitrión (Carlos Caballero González, 2016). Los contenedores tampoco requieren gestionar los recursos de hardware del ordenador anfitrión.

La principal meta de los contenedores es agilizar el proceso de desarrollo y despliegue de software, facilitando la creación de una arquitectura cliente servidor sobre la cual desplegar una aplicación. Los contenedores también trabajan como instancias virtuales de una parte de un sistema operativo completo, ya que corren sobre el kernel de Linux (Wolf, 2015).

Ejecutar una aplicación sobre un contenedor que a su vez se ejecuta sobre un sistema Linux de base tiene sus ventajas, pero la más importante es la capacidad de *poder ejecutar la misma aplicación sobre cualquier sistema independientemente de si este cuenta con las bibliotecas necesarias o no*, puesto que la aplicación realmente no se ejecuta directamente sobre el sistema operativo anfitrión, en su lugar este se ejecuta sobre el contenedor de software además otra ventaja que aporta es la portabilidad de las aplicaciones que se ejecutan sobre estos contenedores.

A continuación, algunos de los contenedores de software que son utilizados en la actualidad:

Tabla 2. Contenedores de Software

Software	Sistema Operativo	Versión	Licencia
Docker	Windows, Linux, MacOS	19.03.5	Apache License 2.0
LXC	Linux	3.2.1	GNU GPLv2
Singularity	Linux	3.5.1	BSD Licence
Sandboxie	Windows	5.33.1	Trialware
Container Linux	Linux	2247.5.0	Apache License 2.0

1.7.1 DOCKER

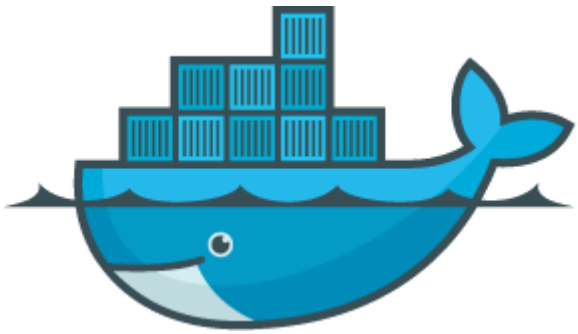


Figura 3. Logotipo de Docker. *(Docker, 2020)*

Se trata de un proyecto de código abierto el cual tiene por objetivo compilar y ejecutar aplicaciones sobre contenedores de software, separa las aplicaciones de la infraestructura y agiliza el proceso de despliegue de software, reduce de forma significativa el tiempo desde que se escribe el código de la aplicación hasta que esta se pone en producción. (Docker, 2020)

La plataforma de Docker permite empaquetar una aplicación (código fuente, o archivos binarios) en un entorno virtual conocido como contenedor. Los contenedores son bastante ligeros, esto en gran medida se debe a que no requieren de un hipervisor al contrario que las máquinas virtuales, en su lugar un contenedor se ejecuta directamente sobre el kernel del servidor anfitrión, esto permite ejecutar muchos más contenedores sobre un mismo servidor que máquinas virtuales; Docker incluso puede ejecutarse sobre instancias de máquinas virtuales dentro de un mismo servidor.

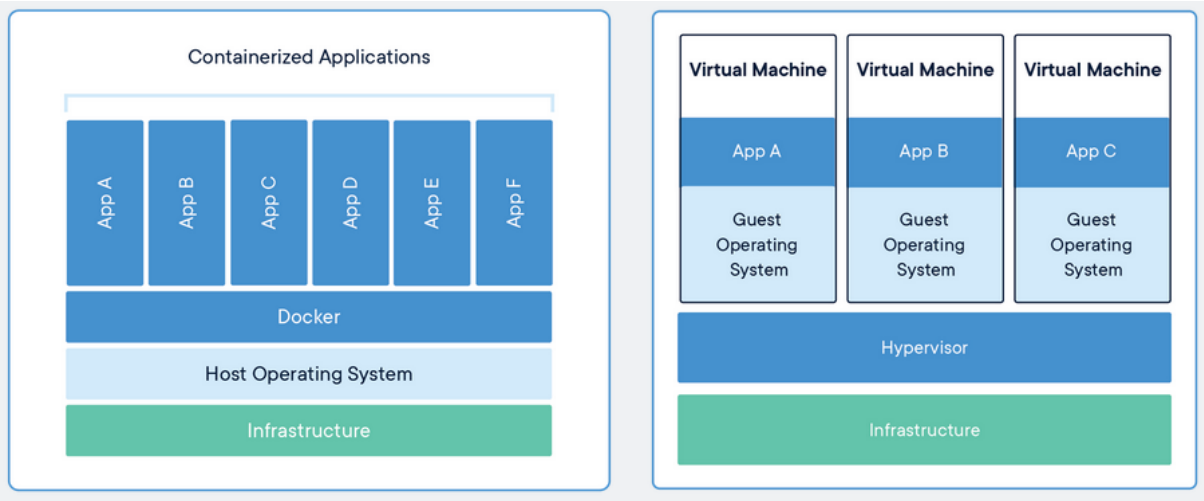


Figura 4. Comparativa entre Docker y la virtualización. *(Docker, 2020)*

El ciclo de vida de los contenedores en Docker se maneja de la siguiente forma:

- Desarrollo de la aplicación usando contenedores.
- Distribuir y probar la aplicación haciendo uso de los contenedores.
- Desplegar el contenedor a producción sin importar la plataforma con la que trabaje.

1.7.2 ARQUITECTURA DE DOCKER

Docker hace uso de una arquitectura cliente servidor; en la que tanto el cliente como el servidor pueden funcionar en la misma máquina, o en máquinas diferentes ya que estos usan un API REST para comunicarse entre sí.

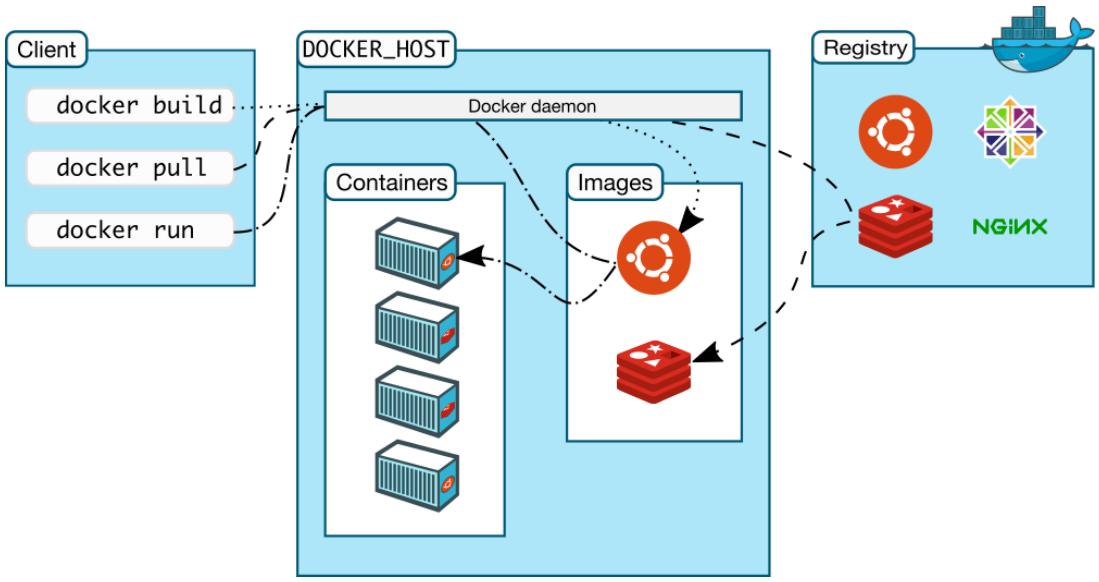


Figura 5. Arquitectura de Docker. *(Docker, 2020)*

Como se puede apreciar en la Figura 5, la arquitectura de Docker se conforma de tres partes:

- **Cliente:** Ejecuta los comandos necesarios para administrar los contenedores de docker; interactúa con el daemon de Docker.
- **Daemon:** Escucha las peticiones del cliente a través del API REST de Docker; interactúa directamente con los contenedores.
- **Repositorio:** Almacena las imágenes necesarias para desplegar contenedores, también se pueden almacenar contenedores personalizados en estos repositorios, para poder descargar una imagen es necesario ejecutar algunos comandos desde el cliente.

1.7.3 IMÁGENES

Contiene la estructura e instrucciones necesarias para crear un contenedor en Docker. Además, una sola imagen puede estar construida sobre una o más imágenes previamente creadas, por ejemplo, puede tratarse de una imagen basada en un S.O de Linux, o en algún lenguaje de programación. También es posible crear imágenes personalizadas, para esto es necesario crear un Dockerfile, el cual es un archivo de texto en el que a través de una sintaxis simple se puede definir los componentes necesarios para crear una imagen personalizada. En estos componentes se pueden incluir además otras imágenes que servirán como base para ejecutar una aplicación y también se puede incluir el código fuente de una aplicación o en su defecto incluir sus archivos binarios.

1.7.4 CONTENEDORES

Es la instancia de una imagen la cual se puede ejecutar; estos contenedores son altamente flexibles, se pueden crear, iniciar, detener, mover e incluso eliminar. También es posible orquestar dos o más de estos contenedores para que trabajen en conjunto, esto es posible debido a que internamente cada contenedor define una interfaz a través de la cual se puede comunicar con otros contenedores.

1.8 MICROSERVICIOS

Los microservicios han constituido un gran cambio para la forma en la que se construyen y despliegan los sistemas informáticos, gracias al continuo avance de nuevas tecnologías que han ido integrándose en el desarrollo y ciclo de vida de los sistemas, facilitando su creación, mantenimiento y puesta en producción. (Blokehead, 2017)

Tradicionalmente, los sistemas informáticos están construidos sobre una base monolítica de código fuente que, a pesar de contar con una sólida documentación y una estructura modularizada, puede presentar inconvenientes al momento de añadir nuevas características y funcionalidades ya que el código fuente aumenta cada vez más en tamaño, haciendo cada vez más difícil mantener el sistema y añadir nuevas características al mismo, limitando el tamaño y la escalabilidad.

De manera general se tratan de servicios o aplicaciones pequeñas e independientes los cuales operan o trabajan en conjunto. Estos servicios o aplicaciones que se ejecutan bajo su propio proceso y se enfocan en realizar una tarea (cada tarea corresponde a una funcionalidad del negocio) específica, a su vez, cada uno de estos microservicios puede estar escrito en un lenguaje de programación distinto y pueden estar usando diferentes fuentes o estructuras de datos además de distintas formas para establecer la comunicación y compartir datos entre sí. (Microsoft, 2020)

1.8.1 KUBERNETES

Figura 6. Logotipo de Kubernetes. *(Kubernetes, 2020)*

Es una plataforma de código abierto la cual facilita el despliegue y administración de aplicaciones en contenedores, además de permitir la creación de clústeres que ejecuten contenedores y administrar los mismos de forma sencilla; estos clústeres pueden estar alojados en la nube, lo que convierte a Kubernetes en la plataforma ideal para ejecutar aplicaciones en la nube de forma nativa. (Kubernetes, 2020)

Kubernetes es capaz de crear clústeres en máquinas virtuales y físicas, permitiendo construir un entorno de producción con una arquitectura basada exclusivamente en contenedores, entre las principales tareas que puede realizar Kubernetes están:

- Escalar las aplicaciones que se ejecutan sobre contenedores.
- Administrar y optimizar los recursos del host necesarios para ejecutar las aplicaciones.
- Gestionar las actualizaciones de las aplicaciones en los contenedores.
- Añadir o remover recursos para ejecutar las aplicaciones.
- Monitoreo constante de los contenedores.
- Reinicio y replicación automática de contenedores.
- Crear clúster con múltiples host y coordinarlos.

1.8.2 KUBERNETES Y MICROSERVICIOS

Al crear un microservicio sobre Kubernetes, es necesario previamente empaquetarlo en un contenedor, cada uno de estos es creado y desplegado por separado. Cada microservicio puede ser creado a partir de un Dockerfile (si se está trabajando con Docker para desplegar los contenedores), el cual cuenta con todas las API's, archivos binarios, código fuente o volúmenes de datos necesarios para el correcto funcionamiento de la aplicación, seguidamente se crea una imagen a partir de ese Dockerfile, el cual es una instancia que se puede ejecutar, con esto se habrá generado un nuevo microservicio. (Sayfan, 2019)

Posteriormente esta imagen se ejecutará sobre un *Pod* (Es la agrupación de varios contenedores) en Kubernetes, el cual a su vez se ejecutará sobre un *nodo* (Puede tratarse de una máquina virtual o física) que a su vez es administrado por *kubelet* (Se ejecuta dentro del nodo, garantiza el correcto funcionamiento de los contenedores), este se encarga de monitorear el estado de los pods. En el caso de que el pod falle, kubelet se encargará de reiniciarlo, aunque si el nodo sobre el cual se ejecuta un pod falla, este dejará de funcionar.

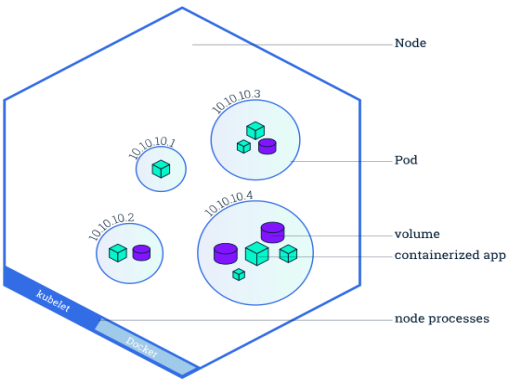


Figura 7. Arquitectura de un Nodo. *(Kubernetes, 2020)*

CAPÍTULO II
ELICITACIÓN Y ANÁLISIS DE LOS REQUERIMIENTOS

2.1 REQUERIMIENTOS DE SOFTWARE

Un requisito es "una característica que se debe exhibir por el software desarrollado o adaptado para solucionar un problema particular" (IEEE Computer Society, 2004). Por lo general, los requisitos buscan dar solución a un problema de la vida real o para automatizar un proceso o tarea que el usuario desee. También los requisitos se consideran para dar soporte a las actividades y procesos de negocio del usuario; añadiendo una nueva característica o corrigiendo el mal funcionamiento en un software existente.

Una característica importante sobre los requisitos es que estos puedan ser verificados y validados de acuerdo con la función descrita por el usuario que realizó dicho requisito; estos también deben cumplir con otros criterios como puede ser el rendimiento del proceso que se ejecuta, la prioridad con la que se ejecuta y la capacidad de medir su progreso a través del tiempo.

2.1.1 FUENTES DE REQUISITOS

Un software típico tiene varias fuentes de requisitos los cuales deben estar identificadas para poder delimitar su alcance, los principales puntos que se deben tener en cuenta en esta sección son:

- **Metas:** Son los objetivos principales que debe alcanzar el sistema.
- **Conocimiento del dominio:** Es el conocimiento que guarda relación con los procesos y reglas de negocio que debe manejar el sistema; permite a los ingenieros de software abstraer y modelar los requisitos que el usuario no puede interpretar.
- **Stakeholders:** Es la parte interesada en el desarrollo del sistema, es necesario tomar en cuenta los requisitos de todos los interesados.
- **Entorno operacional:** Son requisitos que vienen dados por el entorno en el cual opera el sistema, por ejemplo, puede ser la capacidad que tiene el sistema para integrarse con otros sistemas dentro del mismo entorno.
- **Entorno organizativo:** Vienen dados por la estructura interna de la organización, típicamente están para apoyar a las reglas del negocio.

2.1.2 ELICITACIÓN DE REQUERIMIENTOS

Para poder modelar y dar forma al sistema primero se debe saber lo que el usuario busca en una solución de software, por tal motivo, debe mantener contacto con la parte interesada en el desarrollo de la solución y averiguar qué es lo que busca en el sistema.

Primero, se debe identificar a la parte interesada, que, en este caso en particular, sería el Departamento de Secretaría Técnica de Vinculación con la Sociedad de la Universidad Politécnica Salesiana; una vez identificada a la parte interesada se procede a realizar el levantamiento de los requerimientos para el sistema, en este caso se busca un sistema que tenga la capacidad de:

- Registrar (Tutores, Estudiantes e Instituciones).
- Gestionar el seguimiento de pasantías y prácticas pre-profesionales.

2.2 REQUISITOS FUNCIONALES Y NO FUNCIONALES

2.2.1 REQUISITOS FUNCIONALES

Básicamente detallan cuál es el propósito y finalidad del software que se va a construir; describe las tareas y procesos específicos que va a ejecutar el software, como podría ser un registro para usuarios, realizar cálculos de costos o control de inventarios.

2.2.2 REQUISITOS NO FUNCIONALES

Se complementan con los requisitos funcionales, pueden ser características añadidas a los requisitos funcionales los cuales aportan mayor calidad y fiabilidad al funcionamiento del software, por ejemplo, añadir funciones de encriptación a los datos que maneja el software, añadir manejo de sesiones y niveles de autorización.

2.3 ANÁLISIS DE REQUISITOS

La forma tradicional de analizar los requerimientos consiste en detectar los diferentes conflictos que pueden darse entre los requisitos y determinar cuáles son los límites de estos, para de este modo elaborar la especificación de requerimientos. (IEEE Computer Society, 2004)

2.3.1 CLASIFICACIÓN DE LOS REQUISITOS

Los requerimientos según SWEBOK se pueden clasificar de la siguiente forma:

- Requerimientos funcionales y no funcionales.
- Requerimientos derivados de otros de más alto nivel.
- Requerimientos que están en el proceso o producto.

- Requerimientos por prioridad.
- Requerimientos por su alcance.
- Requerimientos por su volatilidad/estabilidad.

2.3.2 MODELADO DEL SISTEMA

Es un paso esencial para poder comprender el dominio del problema que se busca solucionar, es crucial para el análisis de los requisitos del sistema. Se pueden desarrollar varios tipos de modelos conceptuales como casos de uso, diagramas de actividad o diagramas BPMN. Para la gran mayoría de problemas podría resultar de gran ayuda diseñar un modelo para entender el contexto del sistema y su entorno de trabajo, algunos factores que influencian al desarrollo de un modelo son la naturaleza del problema, puesto que ciertos aspectos del sistema deben ser analizados de una forma más rigurosa, y así mismo otro de los factores que pueden influenciar al desarrollo de un modelo son los requisitos del proceso del cliente.

2.3.3 ARQUITECTURA DEL SISTEMA

Es necesario conocer e identificar los componentes del sistema encargados de satisfacer los requerimientos, es un proceso importante ya que al asignar un requisito a un componente puede hacer surgir otros requisitos previamente no contemplados para la integración de ese componente con otros componentes dentro del sistema. Además de esta forma se puede apreciar al sistema como un todo y a partir de esto podra analizar cada subsistema, y analizar sus respectivos requerimientos.

2.4 DESCRIBIENDO LA REGLAS DE NEGOCIO

2.4.1 GESTIÓN ADMINISTRATIVA

Como punto de partida el estudiante debe iniciar el trámite para empezar su pasantía o práctica pre-profesional, para lo cual requiere llenar la solicitud de la carta de aceptación que debe constar los datos del estudiante y la institución donde llevará a cabo sus prácticas, una vez concluido este proceso el estudiante deberá dirigirse a su dirección de carrera correspondiente, para que el director de la misma le asigne un docente tutor y además pueda registrar y dar inicio a la práctica del estudiante, una vez se haya aprobado el inicio del trámite y el docente designado haya aceptado, se procederá a almacenar la siguiente información dentro del sistema:

- **Registrar y validar al docente tutor**: Para registrar un docente tutor, primero hay que buscarlo en la base de datos de colaboradores de la universidad haciendo uso de su número de identificación, o sus nombres y apellidos, luego de seleccionar al docente lo registra en la tabla de responsables del sistema.
- **Registrar y validar al estudiante**: Para registrar un estudiante es necesario comprobar antes que este se encuentre inscrito o matriculado en el periodo actual, luego a través de su número de identificación o sus nombres y apellidos lo debe buscar en la base de

datos de estudiantes de la universidad y posteriormente seleccionar al estudiante y registrarlo en la tabla de inscripciones de prácticas del sistema.

Una vez se registra al docente y al estudiante, se procede a registrar los siguientes parámetros para la funcionalidad principal del sistema, para lo cual se realizarán los siguientes pasos:

- **Registrar y validar la pasantía o práctica pre-profesional con la resolución de aprobación**: Puede registrar una práctica con o sin resolución, lo que permite al estudiante seguir avanzando en su práctica mientras esta no se presente en el consejo de su respectiva carrera para generar una resolución para su aprobación.
- **Registrar las actividades acordes a la carta de compromiso**: Permite registrar las actividades tal cual las haya definido el estudiante en su carta de compromiso con la institución donde va a realizar sus prácticas.
- **Se registrar el plan de trabajo del estudiante**: El plan deberá registrar los siguientes datos para poder ser almacenado en el sistema.
    - Actividades
    - Fecha de inicio
    - Fecha de fin
    - Observaciones
    - Documento en formato estándar

Luego de esto el tutor será capaz de dar seguimiento a las actividades previamente registradas. Además, en esta instancia también se involucra a la institución, que al igual que el tutor deberá firmar el documento de seguimiento de las actividades para dar validez al documento, luego se digitaliza y almacena en el sistema. Posteriormente se deberá firmar y registrar el informe del estudiante para dar validez al documento, digitalizarlo y almacenarlo en el sistema. Al mismo tiempo, se puede llenar el documento de la autoevaluación del estudiante, filmarlo, y subirlo al sistema.

Finalmente, el tutor será el encargado de registrar el informe final firmado, y también será el encargado de aprobar la pasantía o práctica pre-profesional del estudiante, luego este informe final es enviado a consejo de carrera para obtener la resolución final para aprobar la práctica, posteriormente la secretaría técnica de vinculación generará un certificado, y para finalizar se enviará el expediente de la práctica del estudiante a la secretaría de sede.

2.4.2 MICROSERVICIO

En primer lugar, el microservicio debe verificar que el usuario sea válido y que este tiene una sesión activa para permitir al estudiante consumir los microservicios. El microservicio debe recibir como parámetro el correo estudiantil del estudiante, a través del cual se debe verificar que el estudiante se encuentra inscrito en una práctica, pasantía u extensión, una vez realiza la

consulta y verificar que el estudiante se encuentra inscrito en una práctica el servicio consultará los detalles de esta, así como las actividades que tiene. En caso de no poseer ninguna actividad, el servicio permitirá añadir actividades a la práctica, así como editarlas.

Para registrar una actividad, el servicio recibe como parámetros:

- La práctica, pasantía o extensión a la que pertenece.
- La fecha de inicio.
- La fecha de finalización.
- El título.
- Una descripción de la actividad.

2.4.2.1 REQUISITOS FUNCIONALES

El servicio deberá ser capaz de:

- **Consultar las prácticas pre-profesionales, pasantías y extensiones de un estudiante:** Permite realizar una consulta a la base de datos a través del servicio y haciendo uso del correo institucional del estudiante que desea consultar las prácticas registradas en él.
- **Consultar las actividades de una práctica, pasantía o extensión**: Igual que el anterior, la diferencia radica únicamente en el parámetro usado para realizar la consulta al servicio, este devuelve un listado de todas las actividades asociadas a una práctica.
- **Registrar las actividades de una práctica, pasantía o extensión**: Permite crear y asignar nuevas actividades a una práctica; recibe los siguientes parámetros para su creación:
    - Fecha de inicio.
    - Fecha de finalización.
    - Título de la actividad.
    - Observaciones de la actividad.

    Es necesario primero seleccionar la práctica a la que se le asignan las actividades a través del código de esta.
- **Modificar las actividades de una práctica, pasantía o extensión**: Funciona de una forma similar al registro de nuevas actividades, la diferencia está en que para modificar una actividad requerimos de su código puesto que se trata de una actividad previamente creada en el sistema.

2.4.2.2 REQUISITOS NO FUNCIONALES

Se requiere que el servicio sea capaz de:

- **Autorizar el acceso a los servicios**: El servicio debe ser capaz de conceder acceso a él mismo a través de un token de acceso siempre y cuando se trate de un token válido por el sistema.
- **Verificar si una sesión ha expirado**: El servicio también debe comprobar la fecha de creación del token y su fecha de expedición para en función de eso conceder o denegar el acceso al servicio.

2.4.3 CLIENTE MÓVIL

La aplicación debe permitir al estudiante iniciar y mantener su sesión mientras use la aplicación, para iniciar sesión el usuario deberá ingresar:

- Su correo institucional.
- Contraseña.

Una vez iniciada la sesión, el estudiante accede a un listado con todas sus prácticas, al presionar sobre cualquier práctica de la lista la aplicación mostrará los detalles del elemento seleccionado. El estudiante podrá visualizar:

- Tipo (práctica, pasantía, extensión).
- El nombre del estudiante.
- El nombre de la carrera a la que pertenece.
- El listado de las actividades de esa práctica.

En el listado de actividades, el estudiante podrá seleccionar cualquier actividad de la lista, siempre que se haya añadido actividades previamente, entonces podrá ver los detalles de la actividad:

- Fecha de inicio.
- Fecha de finalización.
- Título.
- Descripción de la actividad.

El estudiante también será capaz de editar los datos de la actividad y guardarlos, y si la práctica no cuenta con ninguna actividad, el estudiante podrá crear actividades y guardarlas en la práctica.

2.4.3.1 REQUISITOS FUNCIONALES

La aplicación móvil deberá ser capaz de:

- **Mostrar las prácticas pre-profesionales, pasantías y extensiones de un estudiante:** La aplicación mostrará un listado de ítems el cual contiene algunos metadatos de la práctica, extensión o pasantía; estos datos incluyen:
    - Tipo de práctica.
    - Observaciones.
    - Carrera a la que pertenece la práctica.

  Al seleccionar una práctica, la aplicación mostrará información más detallada de la misma y las actividades que corresponden a dicha práctica.

- **Mostrar las actividades de una práctica, pasantía o extensión**: Está se muestra en forma de una lista expandible dentro del detalle de la práctica seleccionada; esta lista también muestra algunos metadatos en los ítems que contiene, algunos de estos son:
    - Título.
    - Observaciones.
    - Fecha de inicio.
    - Fecha de finalización.

  De igual forma que el listado de prácticas permite acceder a un detalle si selecciona alguna actividad del listado.

- **Guardar las actividades de una práctica, pasantía o extensión**: Esta pantalla contiene un formulario el cual recibe los mismos parámetros que en el servicio, además este formulario válido que todos los campos obligatorios contenga sus respectivos datos antes de enviarlos al servicio para su almacenamiento en el servidor de base de datos.

- **Editar las actividades de una práctica, pasantía o extensión**: Hace uso del mismo formulario para crear actividades, la diferencia está que primero debe escoger uno de los ítems del listado de actividades, al hacer esto en el formulario aparecerán los datos de dicha actividad listos para su modificación.

2.4.3.2 REQUISITOS NO FUNCIONALES

La aplicación deberá:

- **Permitir iniciar sesión al estudiante**: La aplicación mostrará un pequeño formulario antes del listado de actividades, en este formulario se solicitará el correo institucional del estudiante y su contraseña. En caso de que el usuario o contraseña son incorrectos

la aplicación mostrará un diálogo de error, lo mismo sucederá si se deja algún campo vacío.

- **Mantener la sesión del estudiante**: La aplicación debe ser capaz de almacenar el token que genera el servicio de autenticación, y usarlo cada vez que se realiza una petición al servicio, esto es así ya que el servicio en cada petición que se le realice requiere verificar que el usuario que la realizo existe y aún mantiene sesión en el sistema, debido a que el servicio es *stateless* y no almacena ninguna información sobre el estudiante y su estado de sesión.

- **Terminar la sesión del estudiante**: Cuando el estudiante decida terminar su sesión, la aplicación debe de eliminar el token de su registro y mostrar nuevamente la pantalla con el formulario para el inicio de sesión.

CAPÍTULO III
DISEÑO Y DESARROLLO DEL SISTEMA

3.1 DISEÑO DEL SISTEMA

Una vez se han recopilado los requerimientos del sistema, se procede a realizar un esquema general con la arquitectura que se va a desplegar, en el cual se puede apreciar cada uno de los componentes del sistema y la forma en la que estos se acoplan e interactúan entre sí para entregar sus servicios al usuario.

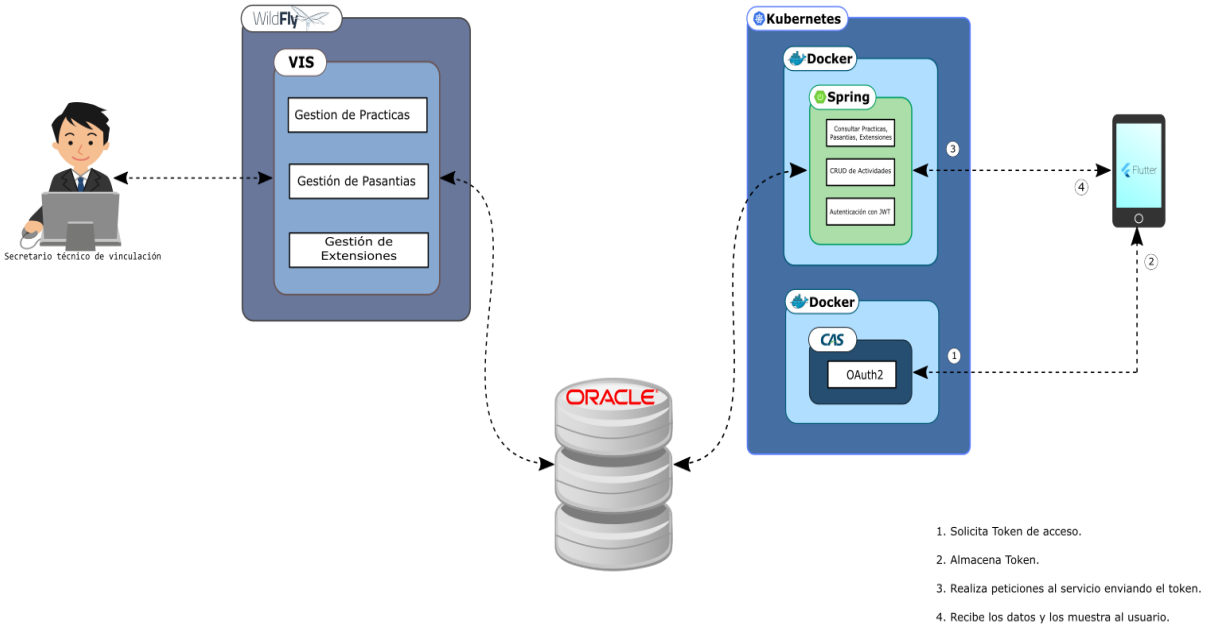


Figura 8. Arquitectura del sistema de gestión de prácticas pre-profesionales.

En la Figura 8, se aprecia los distintos componentes, elementos y usuarios que intervienen del sistema. Aquí intervienen el secretario técnico de vinculación con la sociedad, los docentes encargados de las prácticas, las instituciones donde se realizan, los tutores de las instituciones a cargo y los estudiantes pasantes de la Universidad Politécnica Salesiana.

El sistema cuenta con una parte administrativa desde la cual el secretario técnico de vinculación con la sociedad puede gestionar y dar seguimiento a las pasantías, prácticas pre-profesionales y extensiones de los estudiantes, también puede registrar las prácticas, buscar, registrar y asignar un tutor de la universidad y otro de la institución donde se va a realizar la práctica, puede buscar un estudiante por sus nombres, número de identificación, la sede, campus o carrera que estudia. Además, puede subir y revisar el estado de toda la documentación asociada a una práctica de un estudiante, puede recuperar un listado de las prácticas, pasantías y extensiones que ha realizado el estudiante en una carrera.

Por otra parte, el microservicio expondrá un método para autenticar un estudiante, luego de comprobar que el estudiante está autenticado, el servicio le permitirá acceder a su listado de

prácticas y por cada práctica puede acceder a un listado de actividades correspondientes a la práctica seleccionada, si no cuenta con actividades registradas entonces podrá registrar una nueva actividad y también será capaz modificar las actividades previamente registradas. El servicio también es responsable de comprobar si la sesión del estudiante ha expirado o no.

Finalmente, la aplicación móvil la cual permite al estudiante iniciar y mantener una sesión en su dispositivo móvil, también le permite visualizar una lista con sus prácticas, pasantías o extensiones junto con una breve descripción sobre la misma. Una vez el estudiante haya seleccionado una práctica en particular se desplegará una pantalla en la cual podrá visualizar más información pertinente a la práctica, además de una lista desplegable en la cual podrá observar todas las actividades asociadas a la práctica, también podrá añadir nuevas actividades y editar actividades al seleccionar cualquiera de estas.

3.2 DESARROLLO DEL SISTEMA

3.2.1 ARQUITECTURA N-CAPAS

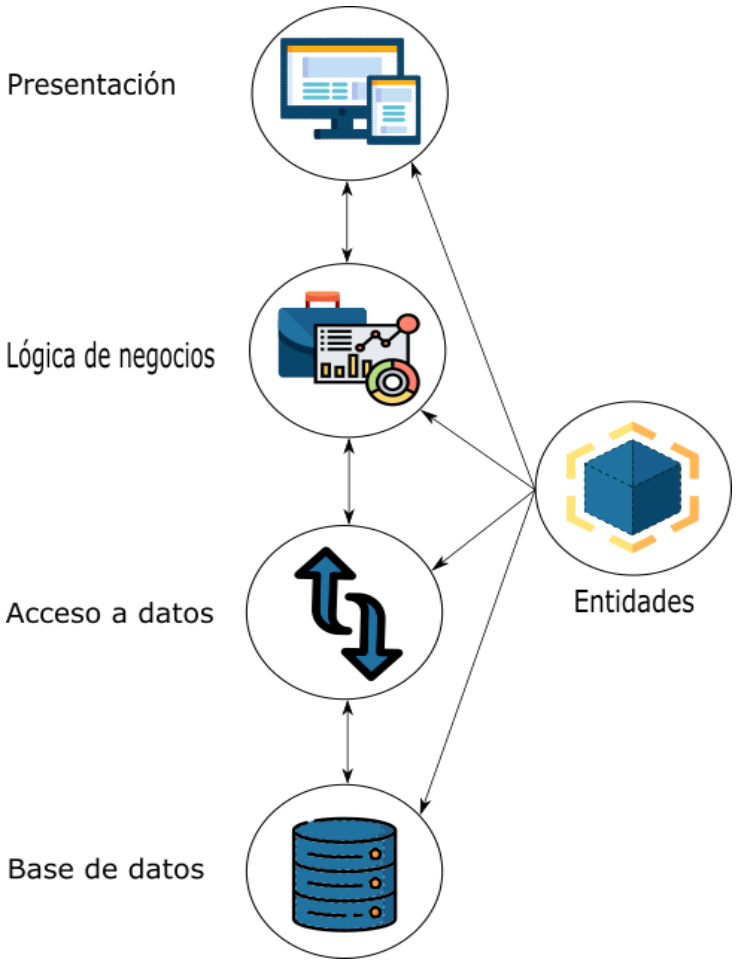


Figura 9. Distribución de una arquitectura n-capas.

En la Figura 9, se puede apreciar la distribución de una a*rquitectura n-capas*, la cual cuenta con los siguientes componentes:

- **Capa de servidor:** Cuenta con los elementos necesarios para recibir y procesar las peticiones de datos de capas superiores, también se encarga de persistir la información que se desea almacenar.
- **Acceso a datos:** En esta capa es donde encontrara a las entidades de mantenimiento, es decir, son las entidades encargadas de realizar las operaciones C.R.U.D(Crear, Leer, Actualizar y Eliminar) sobre la base de datos. También están los *objetos de acceso a datos* los cuales son los intermediarios entre el sistema y la fuente de datos que son los encargados de ejecutar sentencias sobre la fuente de datos.
- **Entidades de negocio:** Representan en su totalidad o parcialmente la estructura y organización de los datos almacenados en la fuente de datos, como pueden ser las tablas del esquema de datos en la base. Esta capa es transversal a todas las demás capas dentro del sistema.
- **Lógica de negocio:** Implementa y hace uso de sus capas predecesoras, aquí se encuentran todas las reglas necesarias para el funcionamiento del sistema, pueden tratarse de validaciones, o cálculos necesarios para el funcionamiento.
- **Presentación:** Interactúa directamente con el usuario, es la capa encargada de procesar y preparar los datos para que tengan sentido para el usuario. (Xhafa, 2007)

3.2.2 DISEÑO DE ESQUEMA ENTIDAD RELACIÓN

Figura 10. Esquema E-R del sistema de gestión de seguimiento de pasantías

En la Figura 10, se observa todas las entidades que intervienen en el sistema, sin embargo, las entidades clave para el funcionamiento son:

- **Tipo Práctica:** En esta entidad se define los tipos de prácticas y su duración en horas, entre los que puede encontrar tres tipos:
    - Práctica pre-profesional (200 horas).
    - Pasantía (200 horas).
    - Extensión (60 horas).
- **Responsable:** Registra el tutor responsable que estará a cargo de supervisar la práctica, su principal función es diferenciar si un tutor pertenece a la universidad o la institución donde el estudiante está realizando su práctica.
- **Parámetro:** Registra distintos tipos de atributos que están relacionados a una práctica, los tipos de datos que puede almacenar son fechas, texto, valores numéricos y archivos binarios. Su principal función es almacenar toda la documentación asociada con una práctica dependiendo de su tipo.
- **Inscripción Práctica:** Permite registrar a un estudiante en una práctica específica, es decir, permite asociar un estudiante a una práctica en particular.
- **Actividades:** Almacena un registro de actividades correspondientes a una práctica específica.
- **Práctica:** Es la entidad clave para el funcionamiento del sistema, aquí es donde se registrará toda la información relevante de una práctica y es el punto central donde las demás tablas se relacionan y aportan su información para poder crear una nueva práctica. Aquí se puede especificar el tipo de pasantía que se va a realizar, el tutor a cargo por parte de la universidad y otro por parte de la institución se puede especificar la documentación requerida para la práctica y almacenarla en el sistema para mantener un registro de que todo se realizó correctamente y también podrá llevar un registro de las actividades que el estudiante realizó durante el transcurso en el que se encontraba realizando su práctica.

Estas son las entidades que más peso tienen sobre las reglas y procesos de negocio del sistema, sin embargo, las demás entidades también desempeñan una función crucial dentro del proceso de gestión de seguimiento de prácticas pre-profesionales.

3.2.3 CONFIGURACIÓN DEL ENTORNO DE DESARROLLO

Posteriormente a los requisitos y las funcionalidades con las que debe cumplir el sistema para alcanzar un nivel de calidad aceptable, se debe configurar el entorno de desarrollo sobre el cual

se construirá todo el sistema, para lo cual se requiere de diferentes herramientas y entornos de desarrollo integrados (IDEs) para comenzar a trabajar.

3.2.3.1 CONFIGURACION DE NETBEANS

NetBeans es un entorno de desarrollo integrado (IDE) para el desarrollo de aplicaciones, está pensado para trabajar con Java. Para configurar un servidor de aplicaciones en netbeans se debe seguir los siguientes pasos:

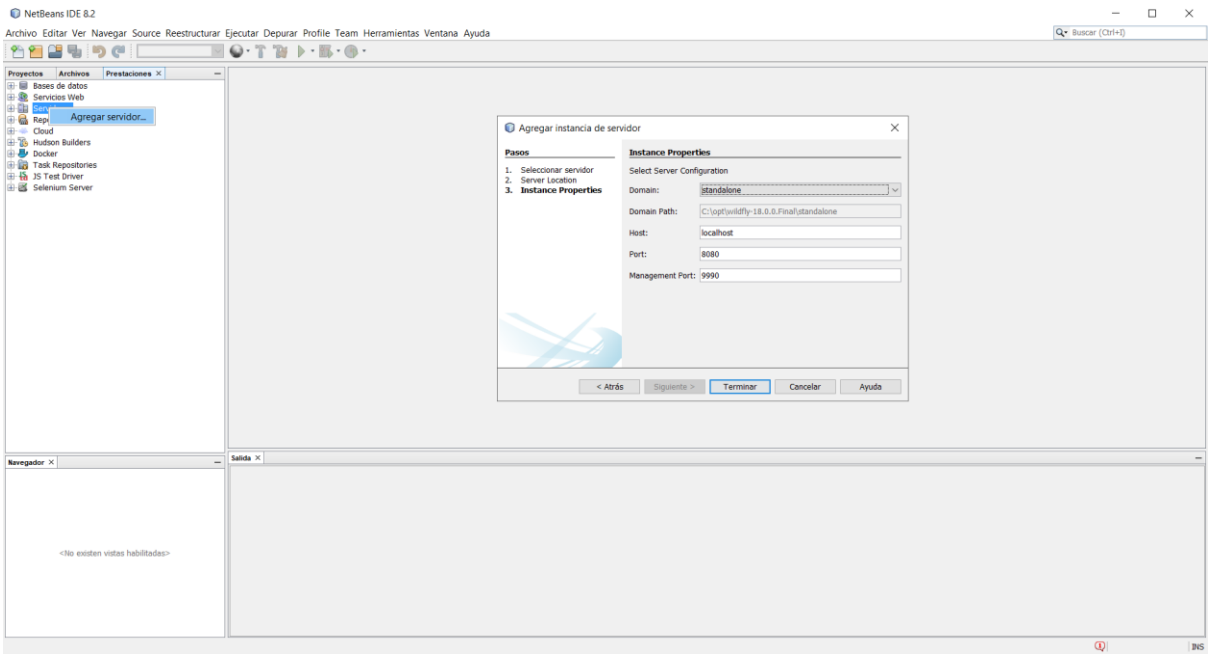


Figura 11. Configuración del Servidor de aplicaciones.

1. Dirigirse al apartado de *Prestaciones*, dar clic derecho sobre *Servidores* y dar clic sobre *Agregar servidor.*

2. Luego aparecerá una pantalla en la que permite escoger cuál instancia de servidor a agregar. En este caso Wildfly.

3. Ahora escoger la ruta donde se encuentra el servidor de aplicaciones. En este caso la ruta es C:/opt/wildfly-18.0.0.Final.

4. Finalmente se debe ingresar a configurar el host, el puerto de acceso y el puerto de administración, en este caso dejar la configuración tal cual viene por defecto.

Con el servidor de aplicaciones configurado ya se puede iniciar el proyecto para comenzar a escribir el código del sistema, para crear un nuevo proyecto es necesario realizar los siguientes pasos.

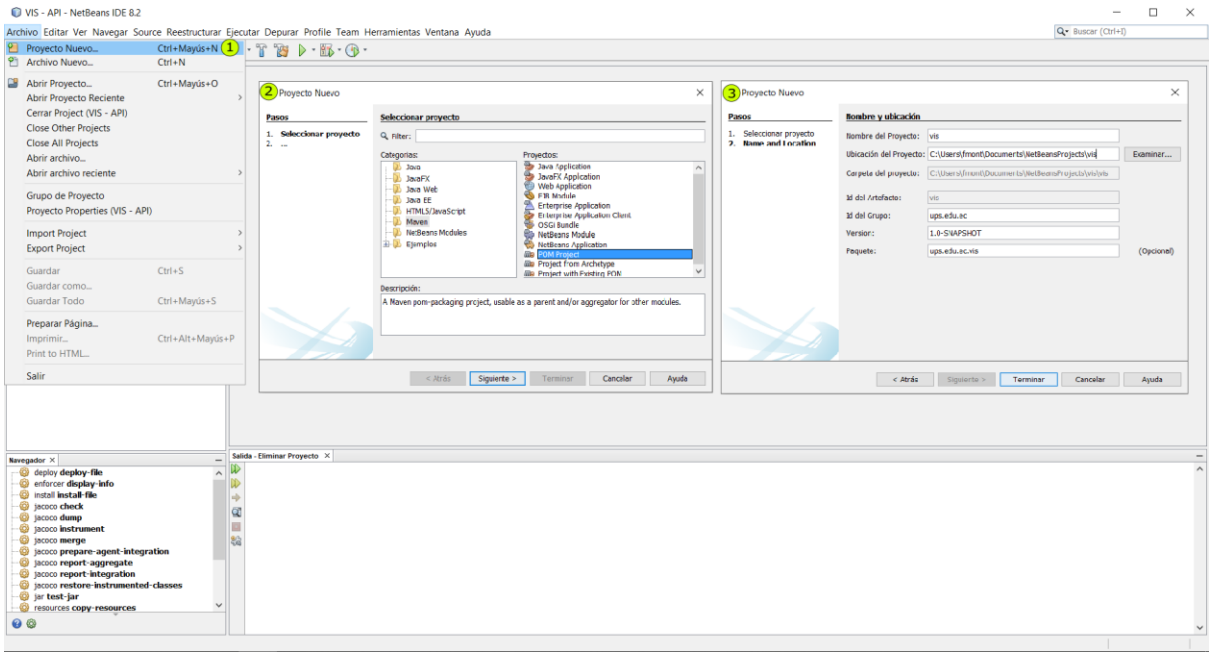


Figura 12. Creación del proyecto.

1. En Netbeans dirigirse a **Archivo>Proyecto Nuevo.**

2. Seleccionar el tipo de proyecto a crear, escoger a *Maven* y luego seleccionar *POM Project* el cual permite trabajar con submódulos dentro de sí.

3. Ingresar un nombre para el proyecto, la ubicación donde se almacenarán sus archivos, el group id y el nombre de paquete.

Para crear los módulos para trabajar dentro del proyecto se tiene que seguir los pasos anteriormente mencionados, sin embargo, en el paso 2 en lugar de seleccionar *POM Project*, seleccionar *Java Application* el paso 3 es prácticamente el mismo. Estos submódulos serán los encargados de manejar las reglas de negocio del sistema, así como de almacenar las entidades necesarias para interactuar con la base de datos. También es necesario crear un *Web Application*; de igual forma que con *Java Application* el cual hará de *front-end* para el sistema en su parte administrativa. Luego de realizar todos estos pasos se tiene un proyecto similar al de la Figura 13.

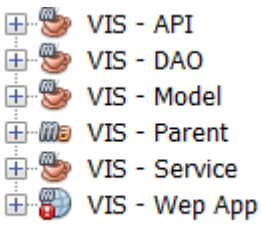


Figura 13. Estructura del proyecto en Netbeans.

Por último, añadir cada uno de los submódulos al módulo padre siguiendo los siguientes pasos.

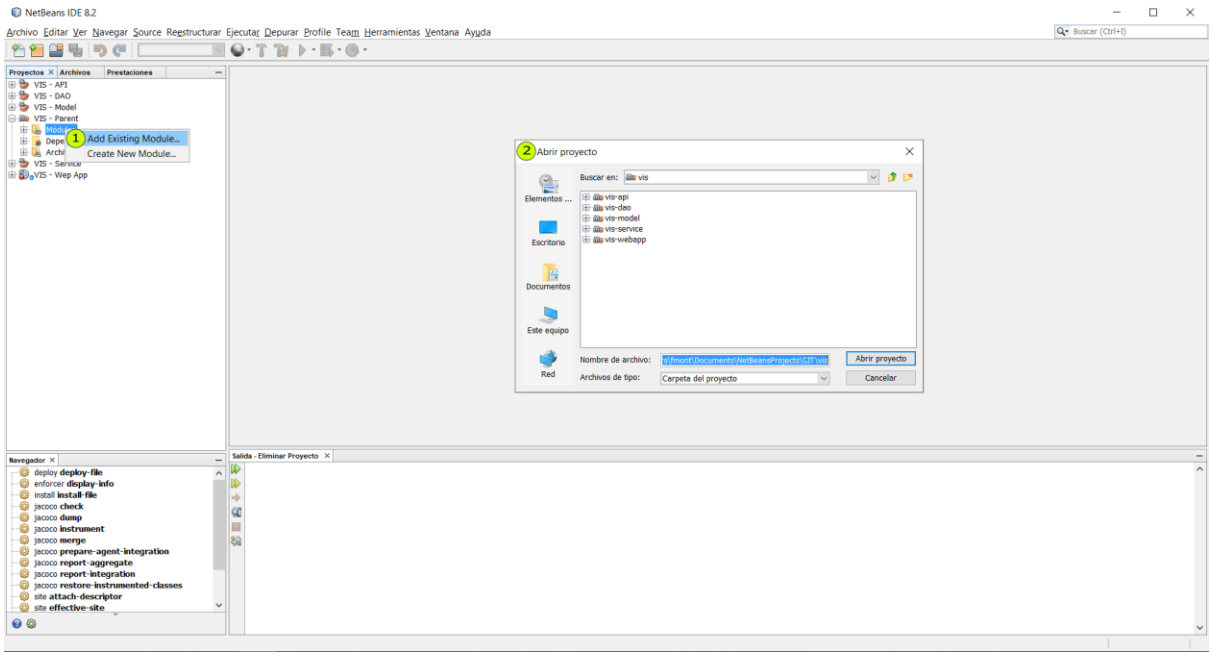


Figura 14. Agregando submódulos al módulo padre.

1. Abrir el módulo padre, y sobre la carpeta *Módulos* dar un clic derecho el cual despliega un menú contextual con dos opciones, como ya se creó los submódulos anteriormente seleccionar la opción para *Añadir un módulo existente*, sin embargo, también se tiene la opción de *Crear un nuevo módulo* si no ha creado uno aún, si escoge esta opción puede seguir los pasos anteriores mencionados.

2. En la pantalla que se muestra se listan los submódulos que se han creado, solo debe seleccionar el/el módulo deseado y añadirlo.

3.2.3.2 SRPING BOOT

Spring boot es un framework para el desarrollo ágil de aplicaciones con el lenguaje de programación Java. Es una alternativa a Java EE, puesto que ofrece una plataforma mucho más ligera y una aproximación más simple al desarrollo de aplicaciones empresariales con Java (Walls, 2016). Spring boot viene por defecto con un servidor Tomcat integrado el cual ejecuta sus aplicaciones, además no requiere desplegar archivos *WAR* (Archivo de aplicación web), también incluye un modo para configuración simple en el cual se puede escoger las librerías y dependencias con las cuales se quiere crear el proyecto, esto se puede hacer desde su página web con Spring Initializr.

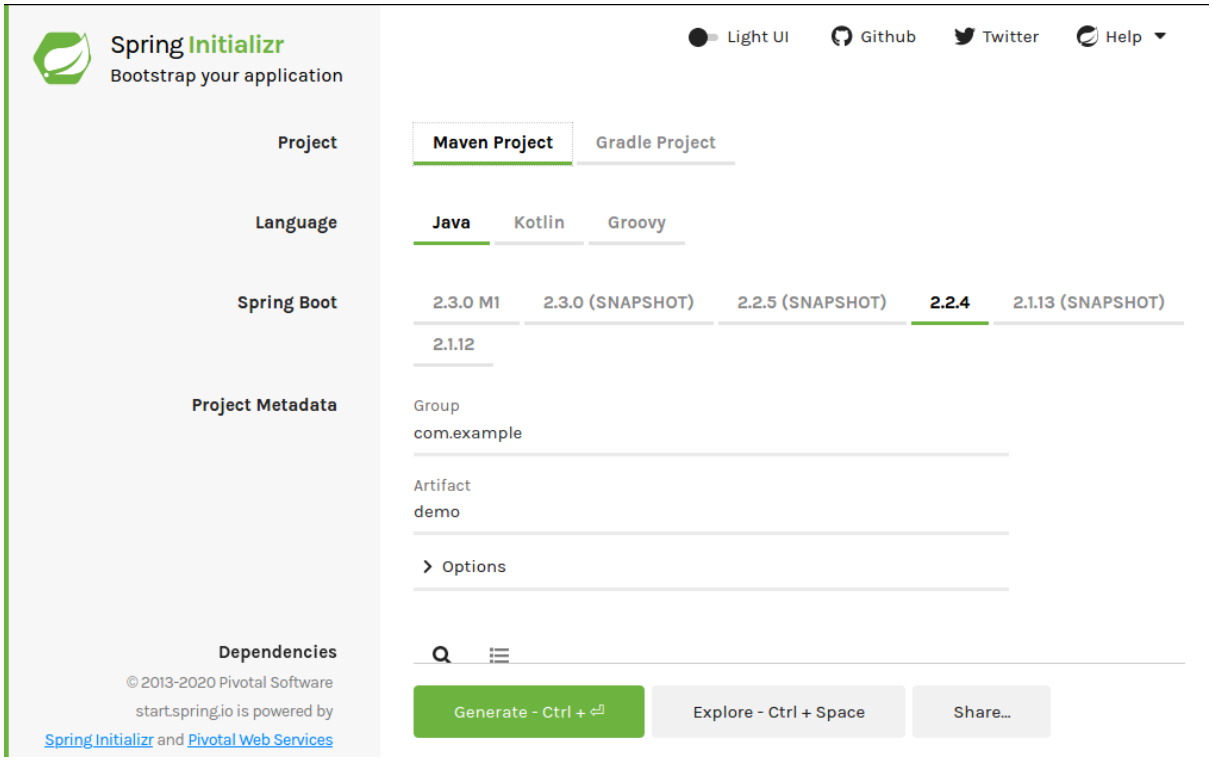


Figura 15. Spring Initializr.

Puede acceder a esta herramienta a través de start.spring.io el cual le mostrara un dashboard como el de la Figura 15 y se podrá escoger de entre varias opciones la configuración inicial de un nuevo proyecto, como el tipo de proyecto, el lenguaje de programación, la versión de Spring, el group id, artifact y las dependencias que conforman al proyecto.

Para trabajar un proyecto con Spring boot se requiere de un IDE, por fortuna se cuenta con una variedad de opciones para elegir, por ejemplo, se puede instalar el plugin de Spring boot en los siguientes IDE's

- Visual Studio Code.
- Theia.
- Eclipse.

También es posible descargar una suite de desarrollo para Spring, la cual se trata de una versión adaptada de Eclipse para trabajar con Spring boot. De igual forma que con el inicializador de Spring en línea, la suite de herramientas de desarrollo de Spring también ofrece la opción de crear un nuevo proyecto de una forma similar al inicializador en línea.

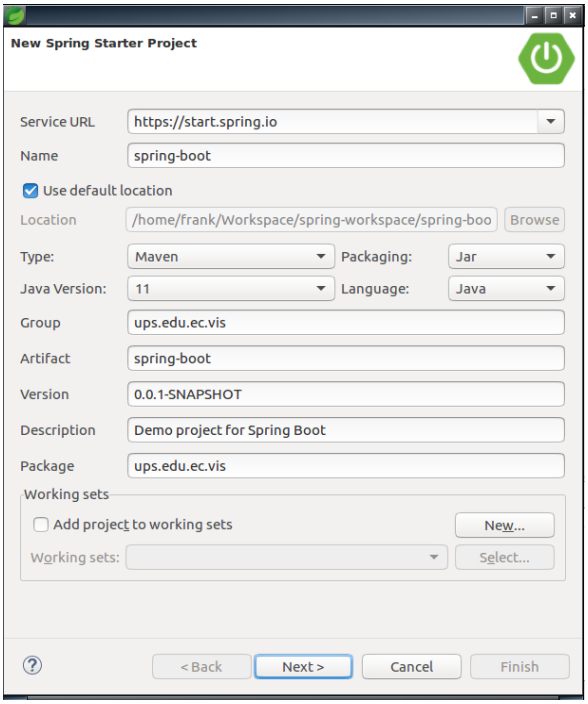


Figura 16. Pantalla de configuración del proyecto.

En la Figura 16 se muestra las opciones para la configuración del proyecto, y como se puede apreciar estas son las mismas que la herramienta en línea de Spring para la generación de proyectos, sin embargo, en la suite de Spring estas opciones se encuentran divididas en dos pantallas.

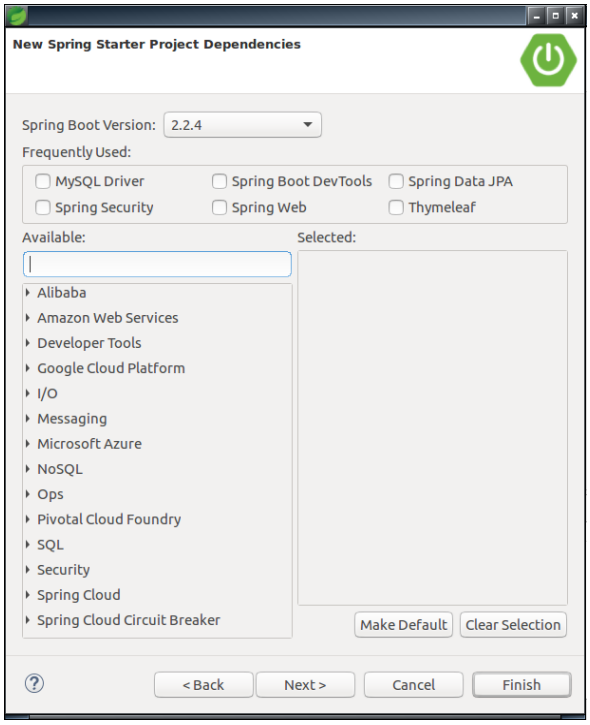


Figura 17. Pantalla para configuración de dependencias.

En la Figura 17 se puede encontrar las demás opciones para la configuración del proyecto, en particular aquí encontrara las dependencias que desee agregar al proyecto para poder trabajar con él.

Luego de escoger la configuración inicial del proyecto, las dependencias y librerías que lo conforman, tendrá un proyecto con una estructura similar al de la Figura 18.

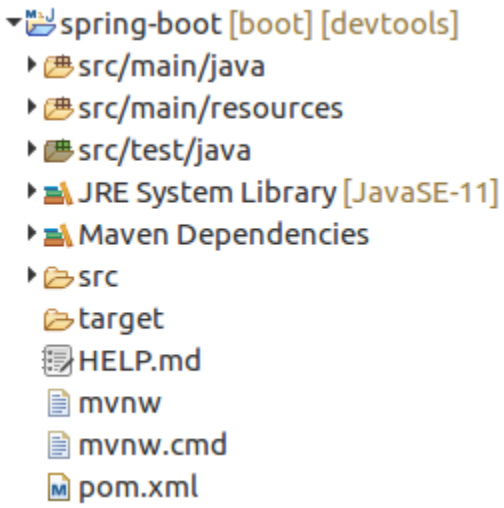


Figura 18. Estructura de un proyecto en Spring.

Ahora se puede comenzar a trabajar en el proyecto, en este ejemplo se construirá una pequeña aplicación la cual expondrá un servicio rest; para alcanzar este objetivo debe crear una clase llamada RestService tal y como se muestra en la Figura 19.

RestService.java

```
1  package ups.edu.ec.vis.ws;
2
3  import org.springframework.web.bind.annotation.GetMapping;
4  import org.springframework.web.bind.annotation.RestController;
5
6  @RestController
7  public class RestService {
8
9      @GetMapping
10     public String holaMundo() {
11         return "¡Hola Mundo!";
12     }
13
14 }
```

Figura 19. Clase que expone un servicio Rest.

En la clase se puede observar dos anotaciones *@RestController* y *@GetMapping ahora se va a* explicar para qué sirve la primera anotación:

- **RestController:** Indica que la clase funcionara como un controlador y está retorna objetos como respuesta a peticiones que se realicen a los métodos contenidos en la clase.

- **GetMapping:** Va sobre la declaración del método e indica que el mismo solo responderá a las peticiones *GET* que se realicen a través del protocolo *HTTP*, en este caso en concreto devolverá una respuesta con el mensaje "¡Hola Mundo!" tal y como se muestra en la Figura 20.

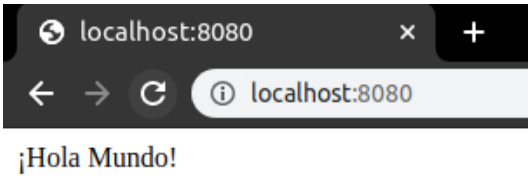


Figura 20. Respuesta del servicio en el navegador.

3.2.3.3 FLUTTER

Es un SDK enfocado al desarrollo de aplicaciones móviles para las plataformas *Android* e *iOS*. Fue creado por Google y actualmente es mantenido por el mismo y por la comunidad en conjunto. Flutter está construido sobre una especie de middleware programado en c/c++ el cual permite invocar a funciones, métodos y componentes nativos del dispositivo en el que se está ejecutando. Sobre este middleware o capa se encuentra otra encargada de la parte visual de la aplicación, esta capa está programada con Dart, el cual es un lenguaje de programación multiplataforma y orientado a objetos también desarrollado por Google. (Hosseini, 2018)

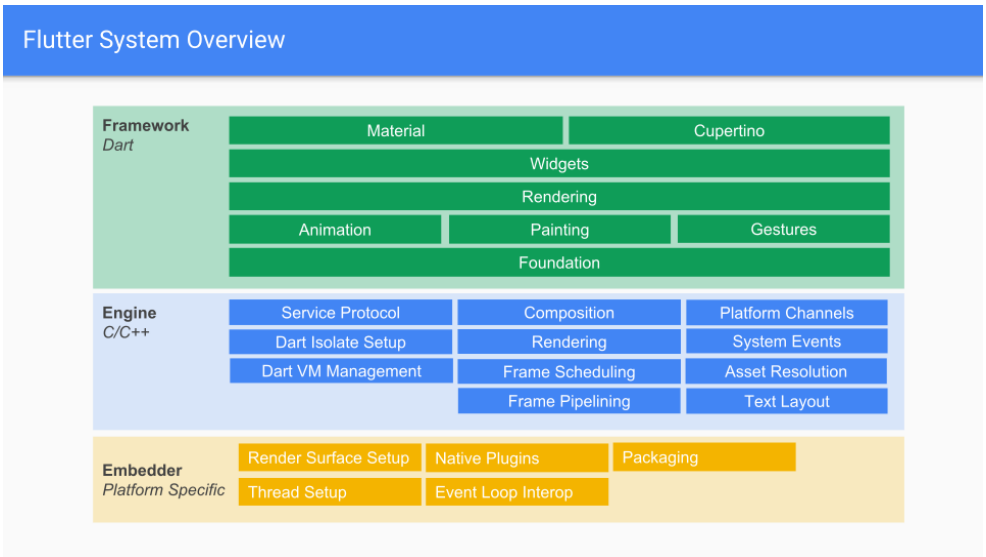


Figura 21. Arquitectura de Flutter. *(Flutter, 2020)*

Esta capa se maneja principalmente sobre *Widgets*; los widgets conforman el punto central para el diseño de la interfaz de usuario en Flutter, dada su flexibilidad. Con los widgets puede construir botones, interfaces, contenedores, etiquetas de texto, básicamente toda interfaz diseñada en Flutter está compuesta en su totalidad de varios widgets.

Además, Flutter facilita el diseño e implementación de widgets personalizados, lo que permite a los desarrolladores construir interfaces acordes a sus objetivos. También es posible manejar widgets con estados, en concreto existen dos tipos de widgets para esto:

- **Stateless Widgets:** Se usan principalmente cuando los componentes que componen la interfaz del usuario no cambian ni interactúan con otros componentes durante el ciclo de vida del widget.
- **Satateful Widgets:** Al contrario del anterior, este si maneja cambios de estado acorde a las acciones del usuario y su interacción con la aplicación.

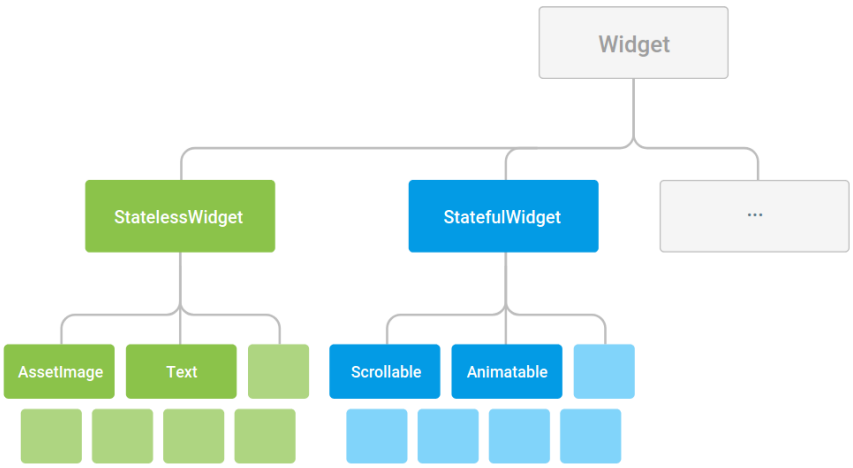


Figura 22. Jerarquía de widgets en Flutter. *(Flutter, 2020)*

Para comenzar a trabajar con Flutter es necesario descargar su SDK el cual está disponible para las plataformas de *Windows*, *Linux* y *macOS*, y puede ser descargado desde su sitio oficial en flutter-es.io, también incluye las instrucciones necesarias para su configuración en cada una de las plataformas antes mencionadas, cabe recalcar que para trabajar con Flutter es necesario contar con el SDK de *Android* para poder desplegar la aplicación en dicha plataforma. Para *iOS* es necesario instalar *Xcode* para disponer del emulador de iOS y otras herramientas necesarias para su configuración y despliegue en esta plataforma. Una vez instalado y configurado se tendra acceso a la herramienta en línea de comandos de Flutter, esta herramienta permitirá realizar varias tareas, desde generar el proyecto hasta generar el instalador de la aplicación para subirlo a la tienda de aplicaciones de la plataforma que se desee.

Posteriormente, hay una lista la herramienta de línea de comandos de Flutter el primer comando que debería ser ejecutado es `flutter doctor`.

```
frank@FGMO:~$ flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, v1.12.13+hotfix.8, on Linux, locale es_EC.UTF-8)
[✓] Android toolchain - develop for Android devices (Android SDK version 28.0.3)
[✓] Android Studio (version 3.5)
[✓] VS Code (version 1.42.0)
[✓] Connected device (1 available)

• No issues found!
```

Figura 23. Salida del comando flutter doctor.

Luego de ejecutar el comando y si no se tiene ningún problema se observa una salida como se puede apreciar en la Figura 22. Sin embargo, la primera vez que ejecute este comando se puede recibir mensajes de error por dependencias rotas o por falta de ciertas configuraciones dentro del entorno de desarrollo. Por ejemplo la primera vez que se ejecute el comando muy probablemente le muestre un mensaje en el cual le pedirá que acepte las licencias correspondientes al SDK de Android, para poder hacer esto es necesario pasar la opción `--android-licenses` al comando flutter doctor, luego de lo cual le pedirá aceptar cada una de las licencias de Android, si se encuentra trabajando en un equipo con el sistema iOS, entonces le SDK le solicitará que configure *cocoapods* y *Xcode* para poder ejecutar Flutter en este entorno.

Existen dos formas para iniciar un nuevo proyecto con Flutter, la primera es una la herramienta en línea de comandos del propio SDK para crear el proyecto, la segunda es crear el proyecto en Android Studio a través del plugin de Flutter para este IDE. Para este ejemplo se usará la línea de comandos. Primero se debe ubicar en la ruta en la que va a crear el proyecto, luego debe ejecutar el comando `flutter create nombre_aplicacion` que creará un nuevo proyecto con todas las configuraciones por defecto, si se desea personalizar un poco más el proyecto se puede pasar algunos parámetros al comando para configurar el proyecto del modo que se acople mejor a los requisitos.

Tabla 3. Argumentos para la creación de un nuevo proyecto.

Opción	Uso
`--org`	Especifica el nombre de la organización responsable del mantenimiento de la aplicación, sirve para identificar la misma.
`--description`	Proporciona una descripción para el proyecto.
`--i`	Determina el lenguaje de programación para iOS, permite escoger entre Objective-c o Swift.
`--a`	Determina el lenguaje de programación para Android, permite escoger entre Java o Kotlin.

Existen más parámetros que se pueden utilizar para configurar el proyecto, si se desea conocer qué otras opciones se pueden utilizar solo debe pasar el parámetro --*help* a flutter.

Una vez creado el proyecto con las configuraciones indicadas, el SDK de Flutter creará una serie de directorios y archivos de configuración para el proyecto tal y como se muestra en la Figura 24.

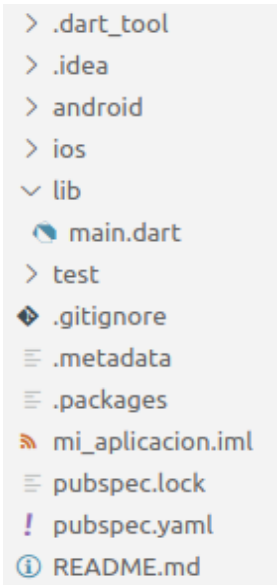


Figura 24. Estructura de un proyecto en Flutter.

Todo el código necesario para que la aplicación funcione estará en el directorio *lib*, en este directorio encontrara el archivos *main.dart* el cual es el punto de inicio de la aplicación.

```dart
class HomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Mi Aplicación'),
      ), // AppBar
      body: Center(
        child: Text('¡Hola Mundo!'),
      ), // Center
    ); // Scaffold
  }
}
```

Figura 25. Código para crear una vista simple en Flutter.

En la Figura 25 se tiene un ejemplo de código para describir una pantalla en Flutter haciendo uso de widgets. El método *build* es heredado de *Stateless Widget* o *Stateful Widget* dependiendo con cual se trabaje; este es el encargado de mostrar la vista al usuario haciendo uso de los widgets. En la anterior figura se puede apreciar algunos de estos widgets, los más importantes son:

- **Scaffold:** Aquí se define la estructura de la vista para la aplicación; cuenta con varias propiedades, sin embargo, las más importantes son *appBar* y *body* puesto que son componentes cruciales dentro de la estructura de la vista de la aplicación. Ambas propiedades son funciones de alto nivel, esto quiere decir que pueden recibir otras

funciones o incluso clases como parámetros de entrada, en este caso puntual reciben widgets como entrada.

- **AppBar:** Describe la barra de navegación de la aplicación, al igual que con el anterior widget este también cuenta con algunas propiedades para modificar su aspecto y funcionamiento.
- **Center:** Este widget sirve para determinar la ubicación de otro widget en la pantalla.
- **Text:** Permite mostrar texto en pantalla.

Existen gran cantidad de widgets dentro de Flutter, estos son solo algunos de los muchos que están disponibles en el framework. En la Figura 26 puede apreciar la disposición de cada uno de los widgets que se acaban de mencionar, esto es gracias a las herramientas para *debugging* que incluye Flutter.

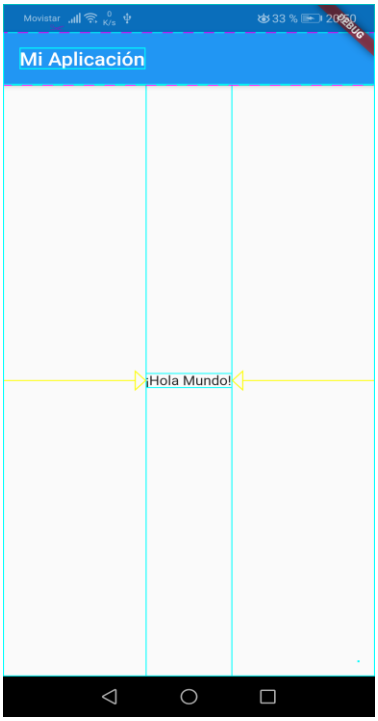


Figura 26. Disposición de los widgets en la pantalla.

3.2.3.4 GIT

Es una herramienta para el control y manejo de versiones de proyectos de software. Git puede trabajar en un entorno local o distribuido, lo que permite tener varias copias de un mismo repositorio, de esta forma si sucede algo con el servidor y este deja de estar disponible siempre se contara con una copia de todo el repositorio localmente el cual luego se puede volver a subir al servidor cuando esté disponible, además todos los cambios que se realice se guardaran localmente hasta que se suban al servidor. (Scott Chacon, 2014)

Git además permite trabajar de forma paralela sobre distintas funcionalidades del mismo software, esto es gracias a los *branches* (ramas), los cuales son capaces de guardar instantáneas del mismo proyecto en diferentes etapas y versiones.

El uso de un sistema de control de versiones es crucial para el desarrollo de este proyecto, dado que se necesita almacenar el progreso y las distintas versiones de cada componente del sistema a lo largo del tiempo que tome en ser desarrollado, por lo que es transversal a todas las tecnologías empleadas en este proyecto.

Para el control de versiones de los microservicios y la aplicación móvil se ha optado por trabajar con *Git* para el versionamiento y el control de los archivos de código fuente. Antes de iniciar con git, es necesario primero instalarlo en el sistema, para esto se puede dirigir a git-scm.com en la sección de descargas, aquí encontrara los enlaces de descarga y la forma de instalarlo en las plataformas de Windows, Linux e iOS. Cuando tenga instalado git en el sistema se puede proseguir a configurar ciertos parámetros para facilitar su uso, primero debe configurar el nombre de usuario y email para poder identificar su identidad en el repositorio sobre el cual se esté trabajando, para esto necesita ejecutar los siguientes comandos:

```
git config --global user.name “Frank Montalvo Ochoa”
```

```
git config --global user.email “fmontalvoo@est.ups.edu.ec”
```

Este par de configuraciones permite almacenar el nombre de usuario y su email de forma global, es decir, que cada repositorio con el que se trabaje en el sistema siempre estará identificado con este usuario y email. También es posible configurar el nombre de usuario y email por cada repositorio dentro del sistema de forma individual, esto se logra al ubicarse sobre el directorio del proyecto y ejecutar los mismos comandos, pero sin el parámetro *--global*.

Sin embargo, antes de poder realizar esto sobre un repositorio de forma individual es necesario que este repositorio se encuentra inicializado, de lo contrario no podrá almacenar ninguna configuración, para inicializar un nuevo repositorio ejecute el comando `git init` sobre el directorio en el que se va a trabajar, esto creará un subdirectorio dentro del directorio actual, el cual almacenará todas las configuraciones pertenecientes a dicho repositorio.

```
frank@EVA01:~/Repositorio$ git init
Inicializado repositorio Git vacío en /home/frank/Repositorio/.git/
frank@EVA01:~/Repositorio$ ls -lha
total 12K
drwxrwxr-x  3 frank frank 4,0K feb 18 14:08 .
drwxr-xr-x 31 frank frank 4,0K feb 18 14:06 ..
drwxrwxr-x  7 frank frank 4,0K feb 18 14:08 .git
frank@EVA01:~/Repositorio$ ls .git/
branches  config  description  HEAD  hooks  info  objects  refs
```

Figura 27. Inicializando un nuevo repositorio.

En la Figura 27 puede observar el directorio *.git* el cual se crea al ejecutar el comando *git init*, el punto frente al nombre del directorio indica que esta se encuentra oculta en el sistema por lo que el usuario no puede acceder a ella de forma directa, esto es debido a motivos de seguridad para que de esta forma el usuario no elimine esta carpeta accidentalmente. Dentro de este directorio se encuentran más directorios y archivos de configuración necesarios para el repositorio, por ejemplo, el archivo *config* guardará todas las configuraciones que realice con

el comando *git config*, como lo es el nombre de usuario y el email del mismo, también almacena otras configuraciones necesarias en el repositorio.

Luego de inicializar el repositorio y setear algunas configuraciones básicas ya se puede iniciar a trabajar en el repositorio, sin embargo, todos las confirmaciones y cambios que realice en este repositorio se almacenarán únicamente de manera local, para poder trabajar con más desarrolladores en el mismo repositorio es necesario crear un repositorio remoto el cual se encuentra alojado en algún servidor en la internet.

Para configurar el repositorio de modo que este pueda ser accedido por otros desarrolladores es necesario configurar un remoto en el repositorio, para esto se ejecuta el comando `git remote add [nombre] [url]` en el cual se debe especificar el nombre del remoto, comúnmente es llamado *origin* pero puede tener cualquier nombre que se desee y también se debe especificar la url del servidor en el cual se aloja el repositorio.

```
frank@EVA01:~/Repositorio$ git remote add origin https://github.com/fmontalvoo/repositorio.git
frank@EVA01:~/Repositorio$ git config --list
user.name=Frank Montalvo Ochoa
user.email=fmontalvoo@est.ups.edu.ec
alias.last=log --stat -1
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
remote.origin.url=https://github.com/fmontalvoo/repositorio.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
```

Figura 28. Configuración del remoto.

En la Figura 28 se puede encontrar el comando para configurar el repositorio remoto, además de un listado con todas las configuraciones que posee. Seguidamente puede comenzar a trabajar sobre el repositorio y todos los archivos y directorios que este posee, para dar seguimiento de los archivos que contiene el repositorio necesita ejecutar el comando `git add <<Archivo>>` este comando recibe como parámetro el nombre de archivo o los archivos que desea rastrear, también puede recibir la ruta a estos o si desea puede agregar todos los archivos contenidos en el directorio directamente ejecutando `git add .` donde el punto hace referencia al directorio actual.

```
frank@EVA01:~/Repositorio$ echo "Mi Repositorio" >> README.md
frank@EVA01:~/Repositorio$ git status -s
?? README.md
frank@EVA01:~/Repositorio$ git add README.md
frank@EVA01:~/Repositorio$ git status -s
A  README.md
frank@EVA01:~/Repositorio$ git commit -m "Primera confirmacion al repositorio."
[master (commit-raíz) 33b8a37] Primera confirmacion al repositorio.
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
```

Figura 29. Rastreando archivos y confirmando cambios en el repositorio.

La Figura 29 muestra todos los pasos necesarios para agregar archivos para su rastreo además de confirmar dichos cambios en el repositorio, primero debe crear un archivo con el nombre

README.md el cual contiene la frase “Mi Repositorio”, luego de crear este archivo ejecute `git status -s` el cual muestra de forma resumida los cambios pendientes de confirmación en el repositorio, motivo por el cual debe ejecutar `git add README.md` para iniciar el rastreo de este nuevo archivo.

En el ejemplo puede ver que al comprobar por primera vez el estado del repositorio aparecen dos signos de interrogación frente al archivo README.md, esto indica que este se trata de un archivo nuevo en el repositorio y que no estaba siendo rastreado previamente; luego de añadir el archivo para su rastreo y ejecutar nuevamente el comando para verificar el estado del repositorio puede ver que frente al nombre del repositorio tiene la letra “A” mayúscula indicando que el archivo está siendo rastreado en el repositorio y está listo para ser confirmado y almacenado en el servidor.

Finalmente debe confirmar los cambios con el comando `git commit -m “mensaje de confirmación de cambios”` es importante saber que si no pasael parámetro *-m* al comando, este ejecutara el editor de texto predeterminado en el cual debe escribir el mensaje para la confirmación de cambios en el cual puede detallar todas las modificaciones que se hayan realizado sobre el repositorio.

Cada vez que se realiza una confirmación en el repositorio git genera una suma de verificación, esto principalmente sirve para mantener la integridad del repositorio y para que git sea capaz de saber cuándo se realizan cambios sobre el repositorio, también sirve para comprobar la integridad del repositorio. Git emplea el hash *SHA-1* para generar la suma de verificación, la cual se trata de una cadena de cuarenta caracteres hexadecimales.

```
frank@EVA01:~/Repositorio$ git log --stat
commit 33b8a375b18237a247f0fb72d238282e7aea4eee (HEAD -> master)
Author: Frank Montalvo Ochoa <fmontalvoo@est.ups.edu.ec>
Date:   Tue Feb 18 15:48:09 2020 -0500

    Primera confirmacion al repositorio.

 README.md | 1 +
 1 file changed, 1 insertion(+)
```

Figura 30. Resumen de la confirmación de cambios.

En la Figura 30 se visualiza un resumen más detallado de la confirmación que se acaba de realizar, y en la parte superior se observa la suma de comprobación que se creó al realizar la confirmación de cambios en el repositorio. Por último, para cargar todos los cambios al servidor ejecute el comando `git push -u origin master` haciendo referencia al remoto al cual se va a enviar los cambios y la rama a enviar.

Para gestionar el control de versiones en la parte administrativa se hará uso del software *Sourcetree*. Este software se encuentra disponible para las plataformas de Windows e iOS y ofrece una interfaz gráfica desde la cual se puede apreciar los cambios, confirmaciones y ramas que están en el proyecto. Sourcetree no es más que una herramienta que se ejecuta sobre *Git* o *Mercurial* como sistemas gestores de control de versiones, y ofrece la posibilidad de

visualizarlos a través de una interfaz más simple y comprensiva para el desarrollador. Además, permite ejecutar todos los comandos previamente revisados en esta sección.

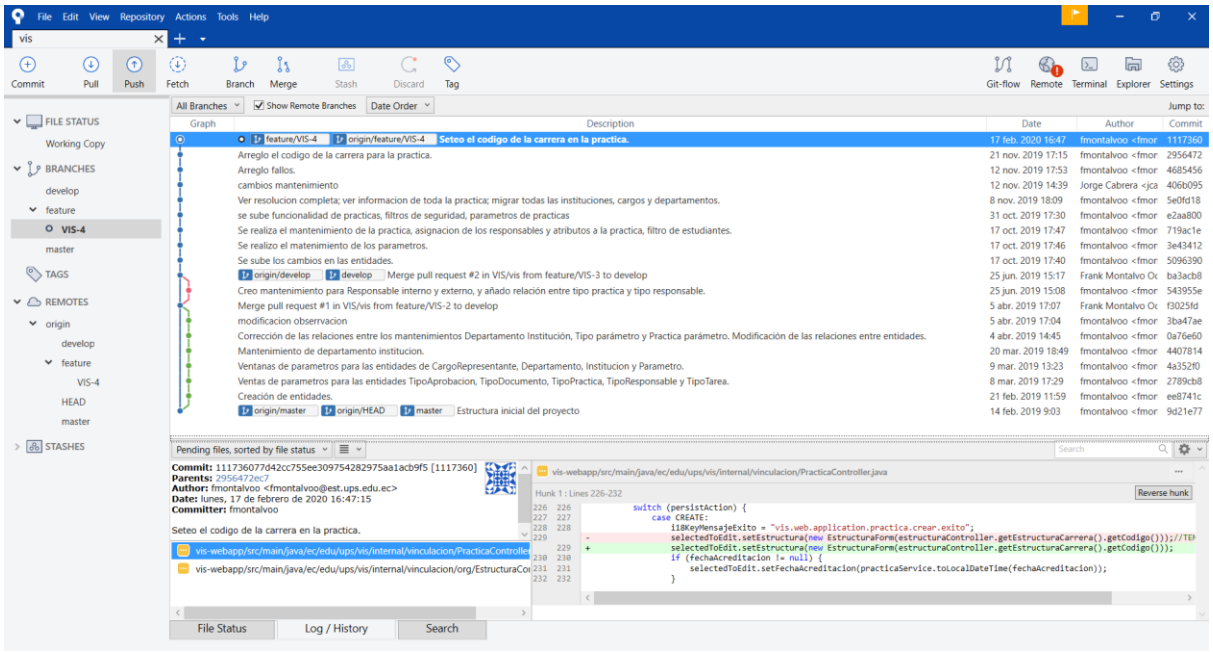


Figura 31. Control de versiones en Sourcetree.

3.2.4 DESARROLLO DEL SISTEMA DE GESTIÓN DE PASANTÍAS

Esta parte del sistema será gestionada por la secretaría técnica de vinculación con la sociedad de la universidad, aquí es donde se cumplen su función la mayoría de las reglas de negocio del sistema las cuales ya se habían analizado en el capítulo anterior.

3.2.4.1 MANTENIMIENTO DE LOS TIPO DE PRACTICAS

Para crear el tipo de práctica es necesario ingresar una descripción y el número de horas que está dura. Esta entidad es frecuentemente usada en otras funcionalidades, puesto que en función de esta se va a definir los tipos de práctica que puede realizar un estudiante, también define los tutores para agregar a la práctica, así como la documentación correspondiente a cada práctica.

La ventana de registro contará con los parámetros anteriormente mencionados, también permite editar, activar, desactivar y realizar un eliminado lógico del tipo de práctica.

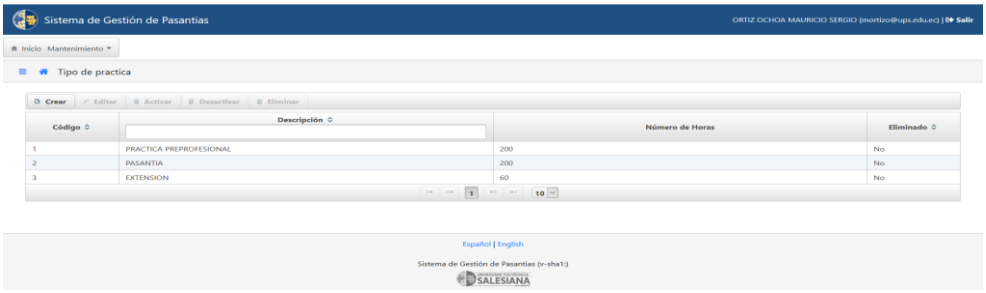


Figura 32. Mantenimiento del tipo de práctica.

En la Figura 30 también puede observar un listado de los tipos de prácticas que puede manejar el sistema y su duración.

3.2.4.2 INSCRIPCION DEL ESTUDIANTE EN LA PRACTICA

Antes de poder registrar prácticas en el sistema se necesita tener un estudiante al cual asociar dicha práctica, para poder asignar al estudiante primero se debe buscarlo en el sistema. Para llegar hasta el estudiante es necesario pasar a través de una serie de filtros, el primero filtro corresponde a la *sede*, aquí puede escoger entre las tres sedes de la universidad que existen a nivel nacional en *Quito*, *Guayaquil* y *Cuenca*, una vez que escoja la sede el siguiente filtro que se debe utilizar es el de *campus* aquí encontrara las distintos campus correspondientes a la sede que se escogió en el primer filtro, luego de escoger el *campus* puede escoger entre las diferentes carreras que son impartidas en ese campus. Finalmente debe proceder con la búsqueda del estudiante; para encontrar al estudiante necesitara sus nombres y apellidos o su número de identificación.

Con el estudiante seleccionado se observar un listado de las prácticas que tiene o registrar y asociar nuevas prácticas.

Figura 33. Filtros de búsqueda.

3.2.4.3 MANTENIMIENTO DE PRÁCTICAS

Este es el eje principal del sistema, puesto que en esta se unen las demás entidades y reglas de negocio que se habían definido previamente al momento de la elicitación de los requerimientos. Para registrar una práctica es necesario definir el tipo (práctica pre-profesional, pasantía, extensión), el tipo de aprobación, el periodo lectivo en el que se realiza la práctica, su estado (aprobado, reprobado), la resolución del consejo de carrera en caso de haber una, las fechas de inscripción, aprobación y acreditación, el total de horas que tomo la práctica y la observaciones sobre ésta en caso de haber alguna.

Cuando se haya registrado la práctica, esta automáticamente se asocia con el estudiante que escogimos con los filtros, y aparecerá en un listado de prácticas, también es posible editar las prácticas previamente asignadas al estudiante. Desde la pantalla de registro de al escoger una práctica de lista y acceder al registro de actividades, tutores y parámetros asociados a la práctica seleccionada en la pestañas situadas sobre la lista de las prácticas.

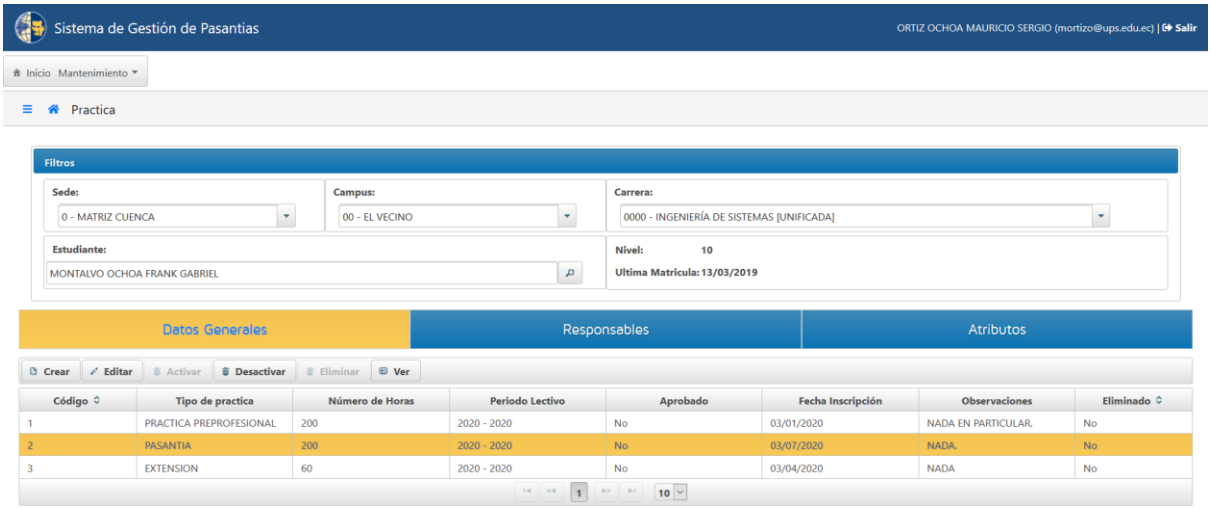


Figura 34. Mantenimiento de la práctica.

3.2.4.4 MANTENIMIENTO DE RESPONSABLES

Con la práctica registrada en el sistema es necesario asignar un responsable a la misma, esta funcionalidad del sistema permite registrar tutores pertenecientes a la misma universidad y a la institución en la que un estudiante realiza sus prácticas, para diferenciar entre estos dos tipos de tutores, el sistema manejara una entidad para definir el tipo de responsable puesto que este puede ser interno (colaborador de la universidad) o externo (empleado de la institución).

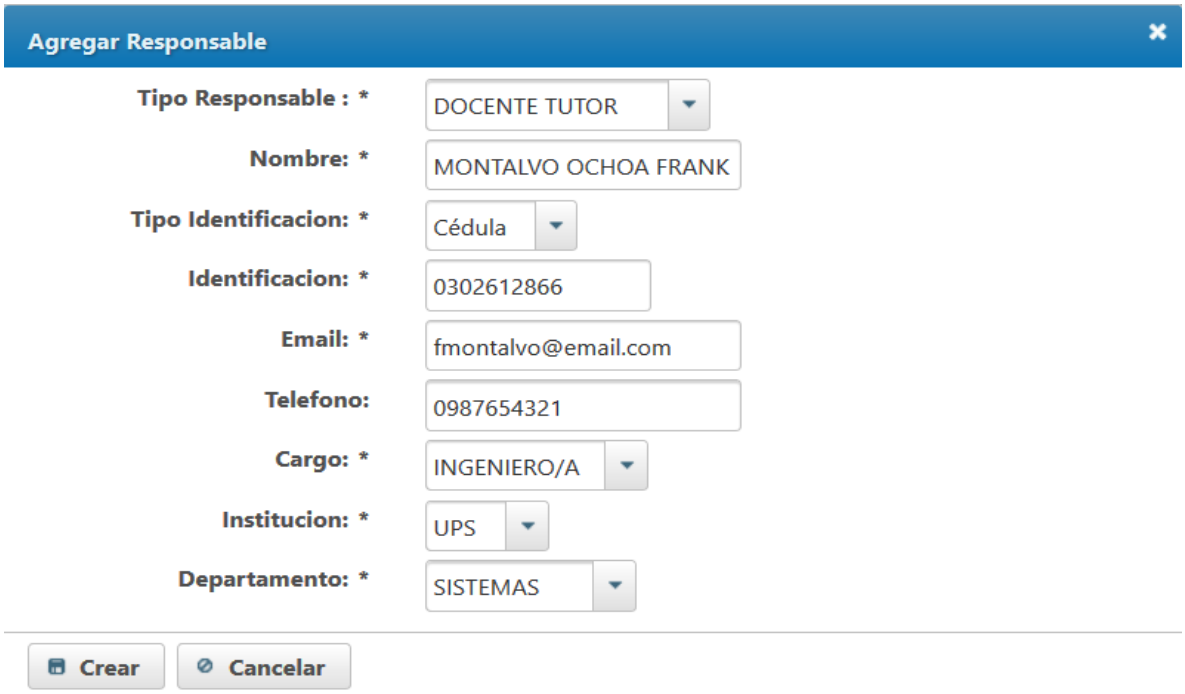


Figura 35. Creando/editando un tutor

Si el tutor a cargo es empleado de la universidad, entonces debe buscar a dicho tutor dentro de la base de empleados de la universidad y registrar sus datos en la tabla de responsables del sistema. Si el tutor encargado es empleado de la institución en la que el estudiante realiza su

práctica, entonces se debe registrar sus datos e información de contacto directamente en la tabla de responsables del sistema. Posterior a su registro el tutor podrá ser asignado a una o más prácticas de ser necesario.

Para registrar un nuevo responsable son necesarios sus nombres, apellidos, número y tipo de identificación, su email y su número de teléfono, si se trata de un colaborador de la universidad entonces estos datos se toman automáticamente de la base de datos.

3.2.4.5 MANTENIMIENTO DE PARAMETROS

Una práctica, pasantía u extensión cuenta con cierta documentación asociada que debe tener para poder ser aprobada, es por eso que en el sistema es necesario almacenar toda esta documentación por cada práctica. Sin embargo, esto no es todo lo que se puede almacenar, la tabla de parámetro puede manejar cuatro tipos de datos incluido los de tipo binario que se utiliza para almacenar archivos, de ser necesario también puede almacenar fechas, texto o datos numéricos que pueden asociarse a varias prácticas o a una en específico.

Por ejemplo, si alguien desea almacenar además de todos los datos pertinentes a una práctica el país en el que se realizó la misma, entonces puede utilizar la tabla de parámetros para almacenar esta información. En sí esta tabla almacena y agrega parámetros de forma dinámica, lo que aporta una mayor flexibilidad al sistema.

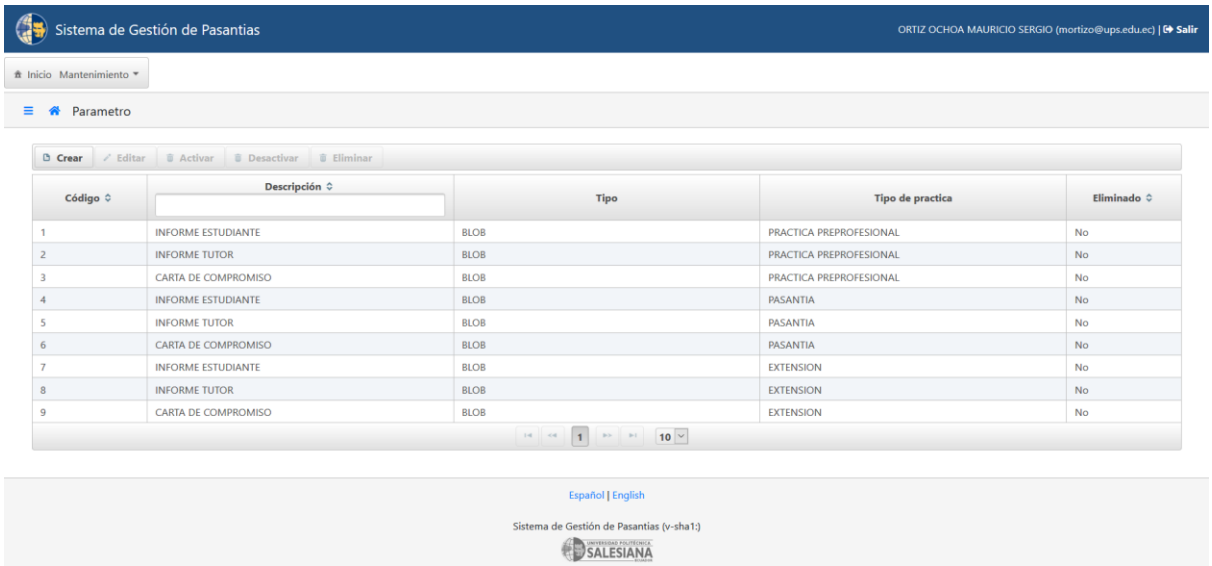


Figura 36. Mantenimiento de Parámetros.

3.2.4.6 REGISTRO DE ACTIVIDADES

Cada práctica registrada en el sistema cuenta con un cierto número de actividades asociadas a la misma. Estas actividades requieren de ciertos parámetros para ser almacenados en el sistema, como lo son, la fecha de inicio y finalización de la actividad, el título de esta y las observaciones pertinentes a la actividad.

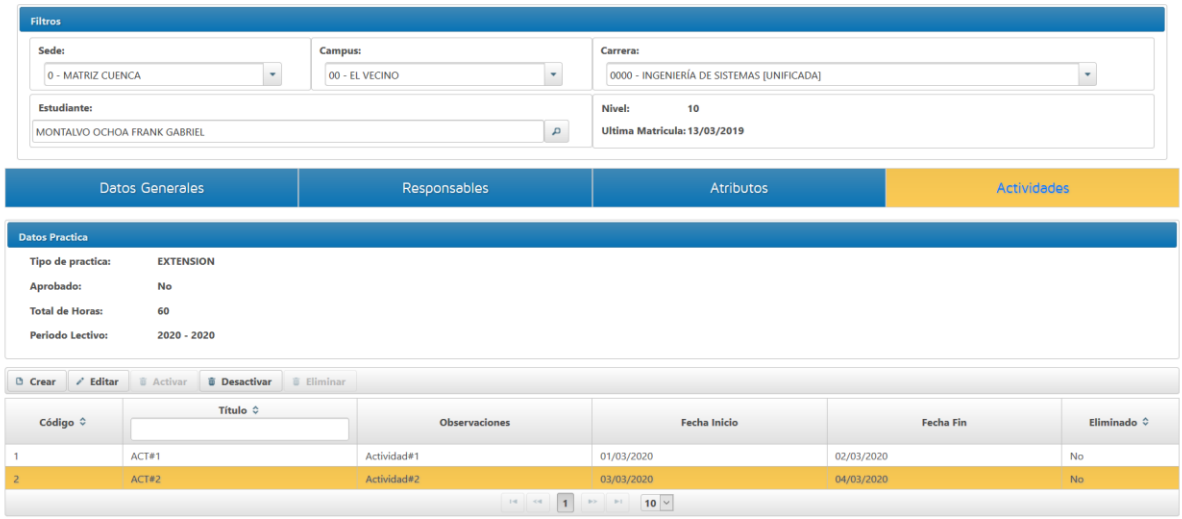


Figura 37. Mantenimiento de Actividades.

3.2.5 DESARROLLO DE LOS SERVICIOS

En el desarrollo de los servicios que se consumen desde la aplicación móvil se utilizara Spring boot el cual es un framework para trabajar sobre el lenguaje de programación Java como se ya se había mencionado. El primero de los servicios que se expondra a través de Spring serán para listar las prácticas de un estudiante, el otro servicio expondrá funciones y operaciones de mantenimientos para las actividades de la práctica.

3.2.5.1 LECTURA DE PRACTICAS

Para recuperar todas las prácticas es necesario pasar al servicio como parámetro el correo electrónico del estudiante, puesto que este sirve identificador único para cada estudiante. Al enviar el correo del estudiante el servidor realizará una serie de consultas para recuperar todas las prácticas, extensiones o pasantías asociadas con el estudiante.

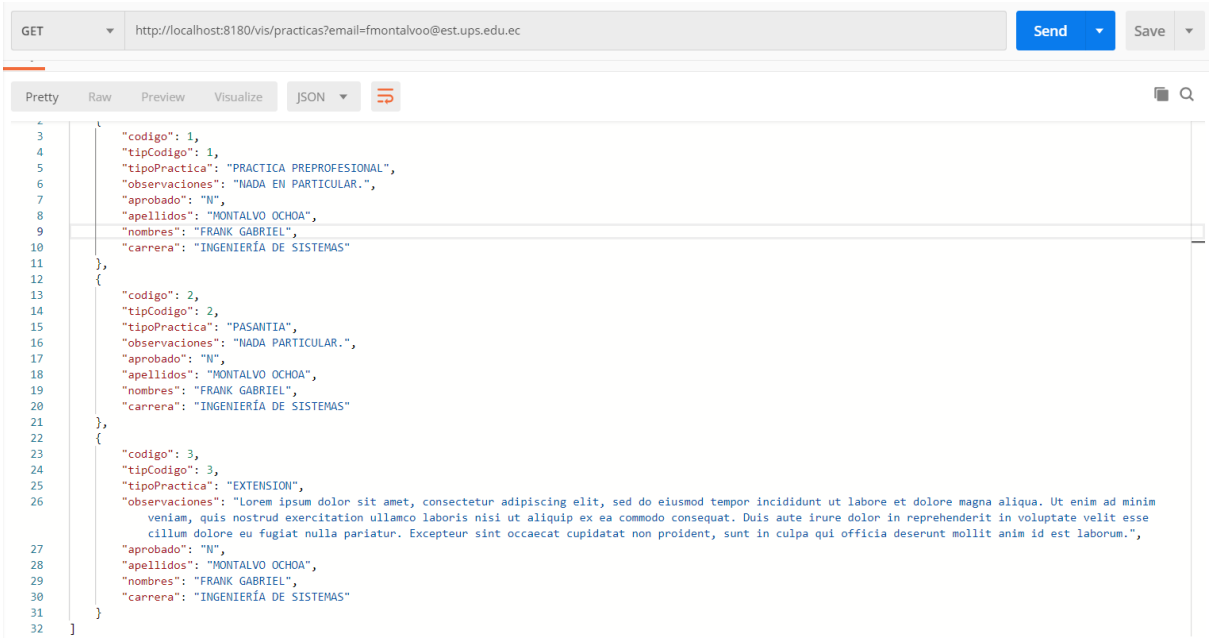


Figura 38. Listado de prácticas del estudiante.

Como se muestra en la Figura 38, el servicio devuelve una respuesta abreviada ya que no es necesario para el cliente recibir toda la información perteneciente a cada práctica, y esto también es por motivos de seguridad puesto que no se puede exponer más información de la necesaria para que el cliente trabaje.

3.2.5.2 MANTENIMIENTO DE ACTIVIDADES

Con las actividades puede realizar más operaciones de mantenimiento como crear, leer, actualizar y listar, pero no se podrán eliminar las actividades registradas, esto es por motivos de seguridad. Al igual que con las prácticas para listar todas las actividades también requerimos de un parámetro, en este caso sería el código de la práctica puesto que cada actividad corresponde únicamente a una práctica en específico.

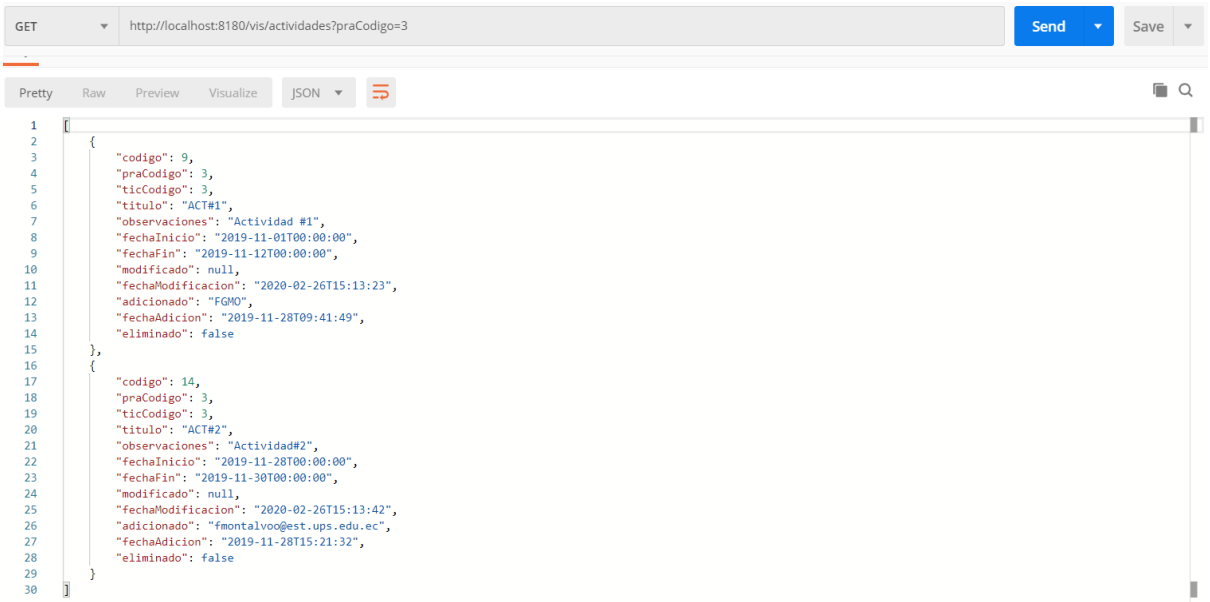


Figura 39. Actividades de la práctica número tres.

Además de recuperar un listado con las actividades correspondientes a una práctica, puede obtener cada una de estas actividades de forma individual a través de su código ya que este es único para cada actividad; al recuperar una actividad puede editar sus datos a través de un cliente *REST* como *Postman*. También agregar nuevas actividades a una práctica existente a través del mismo cliente que se había utilizado para su edición, para esto es necesario que seleccione la práctica a la que se quiere añadir más actividades, esto también se hace a través del código de la práctica ya que es un identificador único para cada práctica, pasantía o extensión que se encuentra en el sistema.

3.2.6 DESARROLLO DE LA APLICACIÓN MÓVIL

Para crear la aplicación móvil se utilizará el framework de Flutter ya que este permite mantener una misma base de código para Android e iOS, lo que facilita en gran medida el desarrollo de la aplicación y evita el trabajo de desarrollar una versión de la aplicación para cada plataforma en específico. Esta aplicación móvil tiene por objetivo reducir la carga de trabajo del tutor encargado, ya que el estudiante sería el encargado de cargar las actividades correspondientes a cada actividad llevada a cabo durante la duración de dicha práctica.

La aplicación servirá como un cliente HTTP la cual consumirá los servicios del API REST que previamente se habían construido sobre Spring boot, es a través de estos que el estudiante tendrá acceso a sus práctica y actividades registrada en la base de datos.

La aplicación móvil requiere que el estudiante se identifique con sus datos porque para consultar las prácticas de dicho estudiante necesita su correo electrónico institucional, con esto se puede acceder a un listado de prácticas, pasantías y extensiones en caso de que el estudiante cuente con alguna es de estas o las tres. Si al iniciar la aplicación no se muestra ningún listado con las prácticas, muy posiblemente el estudiante aún no tenga registrada ninguna de estas, en cuyo caso deberá realizar todo el proceso necesario para iniciar una; por el contrario si el estudiante ya cuenta con al menos una de las tres este podrá visualizar dicha práctica en una lista en la cual le muestra el tipo de práctica, las observaciones en caso de tener alguna y la carrera con la que se registró esa práctica tal como se muestra en la Figura 40.

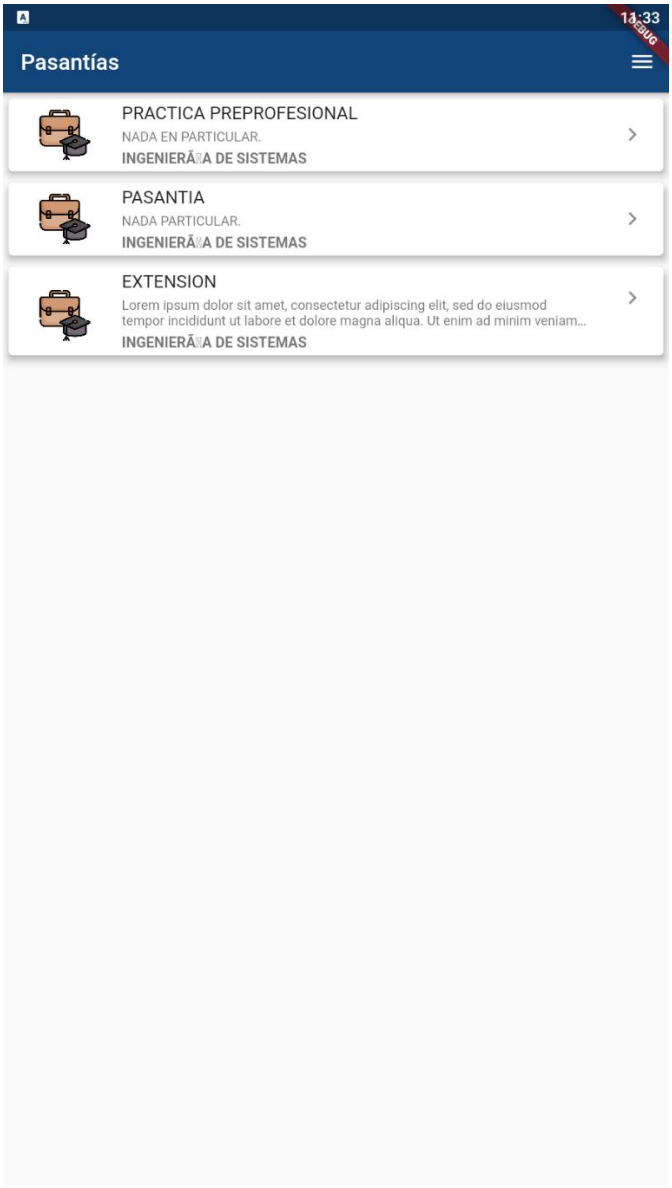


Figura 40. Listado de prácticas en la aplicación.

En la figura 40 se observa que el estudiante cuenta con los tres tipos de práctica que permite utilizar el sistema, en esta lista puede seleccionar cualquier ítem y este mostrará otra pantalla donde se pueden visualizar los datos de la práctica con más detalle, esta indica primero el tipo de práctica que es, el estudiante a la que corresponde y las observaciones que esta tiene en caso de tener alguna registrada, un poco más abajo encontrara una lista desplegable en la cual encontrara todas las actividades correspondientes a esa práctica en caso de que tenga actividades registradas, de no ser así en la parte inferior derecha hallara un botón flotante el cual permite añadir actividades a la práctica.

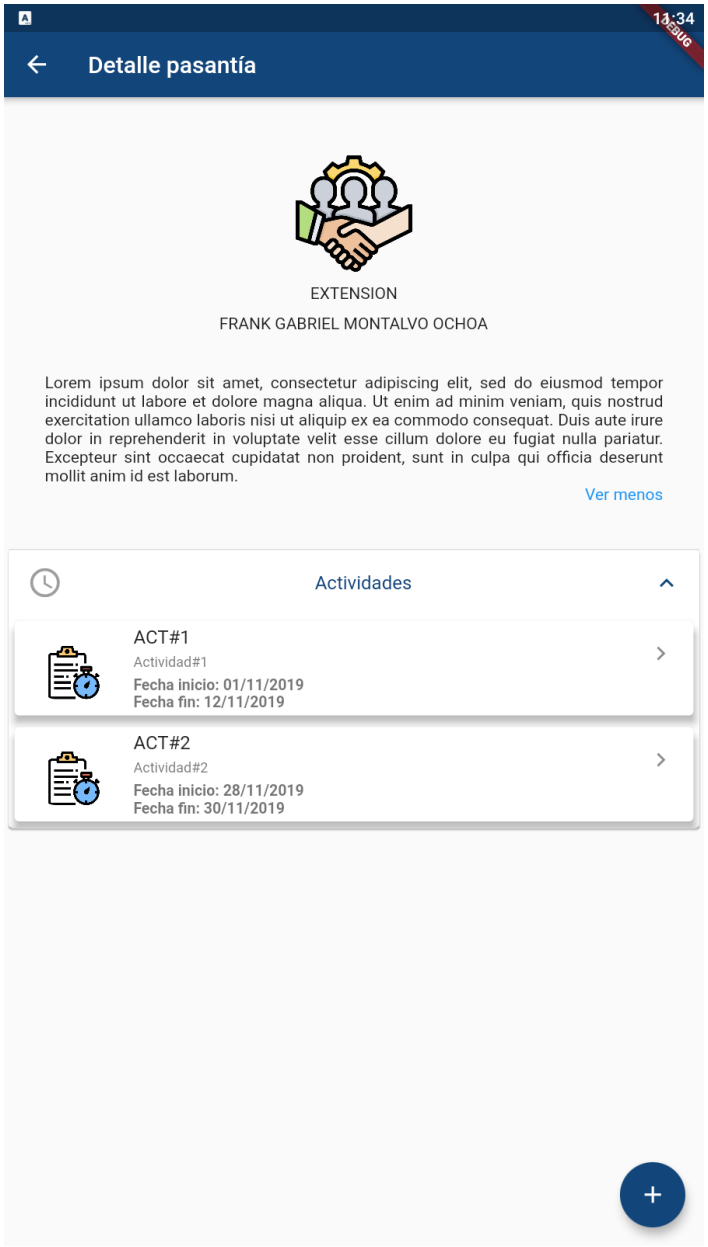


Figura 41. Detalle de una práctica en la aplicación.

La Figura 41, muestra el detalle de la práctica seleccionada con todos los datos que se habían mencionado, por ahora debe concentrarse en el listado de las actividades dado que aquí es donde puede seleccionar una actividad específica para su edición, o añadir nuevas actividades a la misma. Ya sea que seleccione una actividad de la lista o presione el botón para añadir una nueva actividad la aplicación mostrará la misma pantalla para ambos casos ya que los datos de la práctica no varían según si se está editando una ya existente o se está agregando una nueva actividad.

En la Figura 42, puede ver cómo es la pantalla de editar/crear actividades en una práctica, está solo maneja cuatro campos dentro de un formulario que el estudiante puede modificar, el primer dato es la fecha de inicio de la actividad, el siguiente es la fecha de finalización de la actividad luego está el título de esta y finalmente está el campo de observaciones en el cual se puede escribir cualquier información adicional relevante para el desarrollo de dicha actividad.

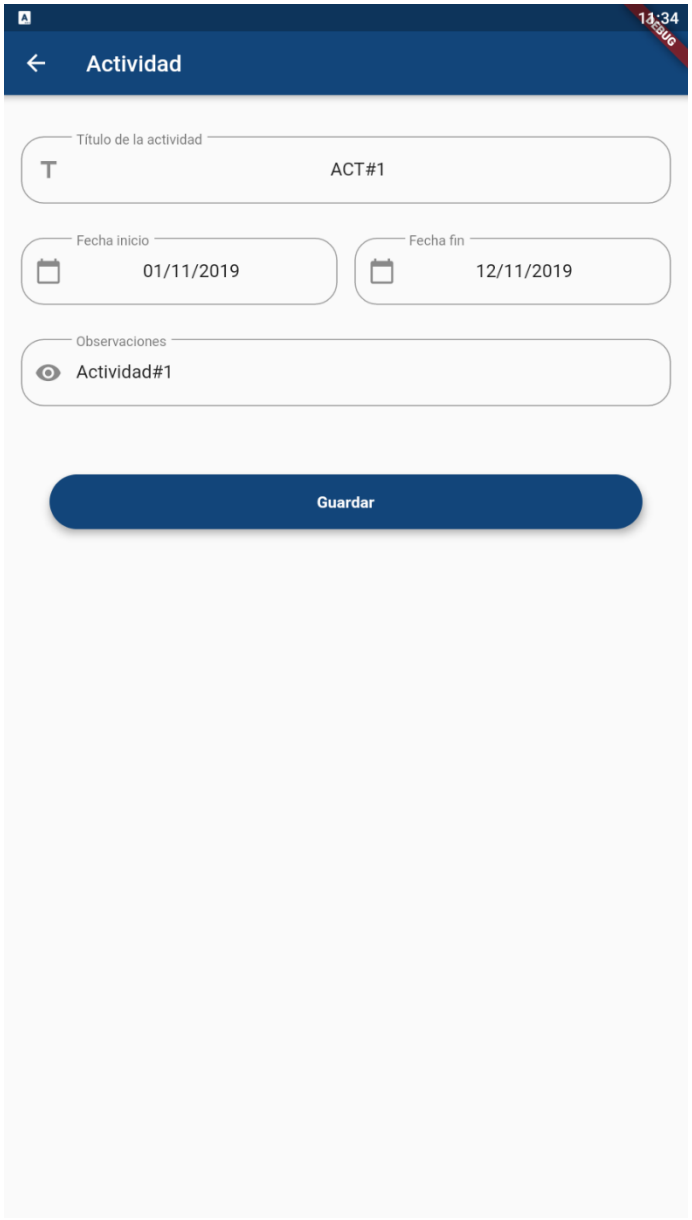


Figura 42. Pantalla de edición/creación de actividades.

CAPÍTULO IV
IMPLEMENTACIÓN DEL SISTEMA SOBRE UNA ARQUITECTURA DE MICROSERVICIOS

Con todo el sistema completamente terminado solo resta montar los servicios sobre *Docker* e implementar mecanismos de seguridad sobre estos para que no pueda ser accesible por cualquier persona.

4.1 IMPLEMENTACIÓN SOBRE DOCKER

En primer lugar, se necesita instalar Docker en el entorno de desarrollo, para esto se utiliza la documentación de Docker donde puede encontrar una serie de instrucciones específicas para cada sistema operativo que se debe seguir para instalar Docker. Si se trabaja con sistemas basados en Linux entonces tendrá diferentes opciones de instalación para escoger dependiendo de la plataforma sobre la que se vaya a trabajar. En este caso se trabajará con Windows, para el cual hay dos opciones de instalación, la primera es a través de *Docker Desktop* el cual requiere de las siguientes características para su ejecución sobre Windows (Docker, 2020):

- Windows 10 64-bit: Pro, Enterprise, o Education
- Hyper-V y los contenedores de Windows deben estar habilitados
- Procesador de 64 bits con traducción de direcciones de segundo nivel
- 4GB de memoria RAM
- Habilitar la virtualización desde la *BIOS* del sistema

Si no se cuenta con estas características en el equipo, no será capaz de instalar *Docker Desktop* como entorno de trabajo, en este caso aún puede instalar Docker mediante *Docker Toolbox*; para su instalación debe cumplir con las siguientes características:

- Windows 7, 8, 8.1 o 10 de 64 bits
- Habilitar la virtualización desde la *BIOS* del sistema
- Instalar *VirtualBox*

Sin embargo, no es necesario haber instalado VirtualBox previamente, ya que las Docker Toolbox lo instalará de ser necesario, puesto que esta herramienta crea una instancia virtual de una máquina con un sistema operativo Linux sobre el cual se ejecutará el *daemon* de Docker.

Con todo listo ahora se puede comprobar que la instalación se haya realizado de forma correcta al ejecutar alguno de los siguientes comandos:

Como primer paso para verificar la instalación se podría ejecutar el comando `docker --version` el cual a su salida devolverá la versión de Docker instalada en el equipo en caso de

estar todo bien. Por el contrario, si algún error se presentó durante la instalación es probable que el sistema operativo no reconozca el comando como válido.

Si al consultar la versión de Docker no existe ningún error quiere decir que se encuentra correctamente instalado. Ahora, para comprobar que se encuentre funcionando correctamente debe ejecutar el siguiente comando, tal y como se muestra en la Figura 43.

```
fmont@FGMO MINGW64 /c/Program Files/Docker Toolbox
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Figura 43. Hola mundo desde Docker.

Una vez comprobado que Docker está listo y funcionando correctamente puede proseguir a crear los microservicios. Para esto es necesario crear un archivo de configuración el cual se encargará de empacar los archivos y directorios que contienen el código fuente o archivos binarios necesarios para que la aplicación se ejecute e incluso procederá a descargar otras imágenes necesarias para la ejecución de la aplicación; estas imágenes pueden por ejemplo contener el SDK de Java o cualquier otro lenguaje de programación. También es posible descargar imágenes que contengan un sistema operativo como *Ubuntu,* por ejemplo, o descargar una imagen que ya contiene una aplicación preparada y lista para su despliegue ya sea para desarrollo o en un entorno de producción.

Para crear una imagen, es necesario antes crear un archivo llamado *Dockerfile* y debe llamarse así específicamente y no de ninguna otra manera, ya que los comandos del cliente de Docker no serán capaces de reconocer al archivo y por lo tanto tampoco será capaz de crear la imagen de la aplicación. En este archivo, se describirán las instrucciones necesarias para la construcción de la nueva imagen.

```
1  FROM openjdk:11-jdk AS builder
2  WORKDIR target/dependency
3  ARG APPJAR=target/*.jar
4  COPY ${APPJAR} app.jar
5  RUN jar -xf ./app.jar
6
7  FROM openjdk:11-jre
8  VOLUME /tmp
9  ARG DEPENDENCY=target/dependency
10
11 COPY --from=builder ${DEPENDENCY}/BOOT-INF/lib /app/lib
12 COPY --from=builder ${DEPENDENCY}/META-INF /app/META-INF
13 COPY --from=builder ${DEPENDENCY}/BOOT-INF/classes /app
14
15 ENV TZ=ECT
16
17 ENTRYPOINT ["java","-cp","app:app/lib/*","ec.edu.ups.vis.MicroservicioTesisApplication"]
```

Figura 44. Dockerfile.

La Figura 44 muestra las instrucciones necesarias para colocar la aplicación desarrollada en Spring boot en un contenedor de Docker, en este caso en particular se trata de una imagen multicapa la cual consta de dos partes para su ejecución, a continuación, se describirán las funciones que realizará cada línea del archivo, de la línea 1 a la 5:

- **FROM:** Descarga una imagen base con el openjdk en su versión 11, además se define un alias para esta imagen.
- **WORKDIR:** Crea un directorio para almacenar los archivos y subdirectorios de la aplicación.
- **ARG:** Argumento que hace referencia a un directorio externo al contenedor, más concretamente hace referencia al directorio donde se encuentra el o los archivos de ejecución de la aplicación.
- **COPY:** Sirve para copiar uno o más archivos de un directorio a otro, en este caso está haciendo referencia a un archivo ejecutable con la extensión *.jar* alojado en el equipo y lo está copiando al directorio de trabajo que se creó con la segunda línea.
- **RUN:** Ejecuta un comando de Java que está incluido en el JDK que se descargó con la primera línea del archivo, este está descomprimiendo el archivo .jar que se encuentra en el directorio de trabajo del contenedor y lo está separando en carpetas individuales.

Ahora desde la línea 7 a la 17 se describen las instrucciones de la siguiente imagen, encargada de ejecutar la aplicación con el JRE de Java.

- **FROM:** Exactamente igual que el anterior con la excepción que este no tiene un alias y en lugar del JDK descargara el JRE.

- **VOLUME:** Crea un directorio dentro del contenedor para persistir archivos e información.

- **ARG:** Igual al anterior con la diferencia que este apunta a un directorio dentro del contenedor.

- **COPY:** Igual al anterior con la diferencia que esta copia los archivos desde el primer contenedor a un directorio con el mismo nombre en el segundo contenedor, es por ello que este usa como referencia el alias del primer contenedor. La primera línea copia los archivos desde el *lib* del primer contenedor al nuevo, la segunda línea copia el directorio *META-INF* y la tercera línea copia los archivos compilados de java, es decir las clases compiladas del proyecto.

- **ENV:** Hace referencia a las variables de entorno, debe recordar que cada contenedor de Docker se ejecuta sobre el kernel de Linux, lo que le convierte en una mini instancia virtual de ese sistema operativo. En este caso apunta a la variable de entorno que controla la zona horaria del sistema, aquí se puede definir el uso horario con el cual se ejecutará el contenedor.

- **ENTRYPOINT:** Ejecuta la clase principal(main) de la aplicación a través del JRE que había descargado. Aquí se separan los parámetros, primero este java el cual ejecuta los archivos compilados de por el JDK, el segundo comando es *-cp* el cual indica la ubicación de las bibliotecas de *maven* necesarias para la ejecución de la aplicación, el siguiente argumento indica la ruta a los archivos de la aplicación y a las bibliotecas y el último de los argumentos indica el nombre de los paquetes y la clase que contiene el método main desde el cual se ejecuta toda la aplicación.

Para crear una imagen es necesario ejecutar el comando `docker build Dockerfile` también es posible pasarle como parámetro la ruta donde se encuentre el archivo Dockerfile, también se puede pasar el argumento *-t* el cual le asignará un tag con el nombre y número de versión en caso de especificar una, de no ser así se crea una imagen con el nombre que se le indique y como versión le colocará *latest*.

```
fmont@FGMO MINGW64 ~/Workspace/spring-workspace/microservicio_tesis_security (ms_security)
$ docker build -t ms --build-arg APPJAR=target/microservicio_tesis_security-0.0.1-SNAPSHOT.jar .
Sending build context to Docker daemon  46.42MB
Step 1/13 : FROM openjdk:11-jdk AS builder
 ---> d29dd615eaf4
Step 2/13 : WORKDIR target/dependency
 ---> Using cache
 ---> 7bfd68fbfbe6
Step 3/13 : ARG APPJAR=target/*.jar
 ---> Using cache
 ---> 34d7fe79d8fe
Step 4/13 : COPY ${APPJAR} app.jar
 ---> Using cache
 ---> e9181f1148e4
Step 5/13 : RUN jar -xf ./app.jar
 ---> Using cache
 ---> d9c52e93934d
Step 6/13 : FROM openjdk:11-jre
 ---> 6ae9b19d913d
Step 7/13 : VOLUME /tmp
 ---> Using cache
 ---> cf1574cde111
Step 8/13 : ARG DEPENDENCY=target/dependency
 ---> Using cache
 ---> 8ab557a5ba86
Step 9/13 : COPY --from=builder ${DEPENDENCY}/BOOT-INF/lib /app/lib
 ---> Using cache
 ---> 545ad7715200
Step 10/13 : COPY --from=builder ${DEPENDENCY}/META-INF /app/META-INF
 ---> Using cache
 ---> 20430a0e727c
Step 11/13 : COPY --from=builder ${DEPENDENCY}/BOOT-INF/classes /app
 ---> Using cache
 ---> 06c1a82a6e64
Step 12/13 : ENV TZ=ECT
 ---> Using cache
 ---> 6bae7e173b18
Step 13/13 : ENTRYPOINT ["java","-cp","app:app/lib/*","ec.edu.ups.vis.MicroservicioTesisApplication"]
 ---> Using cache
 ---> 4dddb2a4b2be
Successfully built 4dddb2a4b2be
Successfully tagged ms:latest
```

Figura 45. Docker Build.

La Figura 45 muestra cómo el cliente de Docker ejecuta línea a línea las instrucciones contenidas en el Dockerfile, y al finalizar su ejecución se habrá creado una imagen que contiene y ejecuta la aplicación sobre Docker. En este punto, ya no se trata de un servicio común, ahora es un microservicio que se encuentra alojado en un contenedor. Para visualizar el contenedor ejecute el comando `docker images` como se muestra en la figura 46.

```
fmont@FGMO MINGW64 ~/Workspace/spring-workspace/microservicio_tesis_security (ms_security)
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ms                  latest              4dddb2a4b2be        4 minutes ago       331MB
<none>              <none>              d9c52e93934d        6 minutes ago       719MB
openjdk             11-jre              6ae9b19d913d        6 days ago          285MB
openjdk             11-jdk              d29dd615eaf4        6 days ago          627MB
```

Figura 46. Imágenes en Docker.

Sin embargo, en la Figura 46 se puede observar que además de la imagen que tiene el nombre de *ms* (abreviación para microservicio), existen otras tres imágenes. La imagen que no posee ni un nombre de repositorio ni una etiqueta de versión es la que se había ejecutado en la primera parte del Dockerfile, es decir, esta fue la imagen encargada de cargar los archivos de la aplicación y compilarlos con el JDK que se descargó para esta imagen.

Las dos últimas imágenes son el JRE y JDK de openjdk, y aquí puede observar una de las grandes ventajas que aporta Docker, ya que cosas como el lenguaje de programación, API's o bibliotecas existirán una única vez en el espacio de almacenamiento del disco y serán compartidos entre los contenedores que lo requieran, permitiendo ahorrar espacio en disco, es decir, que si existiera otra imagen además de este microservicio que use el JDK del openjdk,

entonces este no se volverá a descargar desde el repositorio de Docker si no que usará el que ya se encuentra almacenado en el equipo.

Con la imagen preparada, es momento de ponerla en marcha para comprobar que la aplicación sigue trabajando de la misma manera sobre el contenedor que en el entorno de desarrollo, para esto se debe ejecutar el comando que se muestra en la Figura 47.

```
fmont@FGMO MINGW64 ~/Workspace/spring-workspace/microservicio_tesis_security (ms_security)
$ docker run -p 8182:8180 ms

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.2.1.RELEASE)
```

Figura 47. Inicialización del contenedor.

En el comando se especifica el puerto sobre el cual el servicio va a recibir peticiones desde fuera del contenedor en este caso será a través del puerto *8182* seguido del puerto *8180 a través de*l cual el servicio recibe peticiones desde dentro del contenedor o la red interna que manejan los contenedores en Docker; el modo en el que Docker trabaja con estos puertos es similar al proceso de *nateo* siendo así, se podría decir que el puerto *8182* es público y accesible a cualquiera que tenga acceso a este puerto desde otra red, mientras que el puerto *8180* es privado y únicamente accesible desde la intranet de los contenedores y el contenedor mismo. Luego de asignar los puertos para trabajar con el contenedor se debe indicar el nombre de este para saber qué es ese en específico el que se va a ejecutar.

Para probar que el servicio sigue funcionando normalmente puede usar el cliente *Postman* para realizar una petición al servidor sobre el cual se está ejecutando el *daemon* de Docker y por ende la aplicación, para esto es necesario indicar el nombre de dominio o dirección IP de dicho servidor, seguido del número puerto y la ruta para acceder al servicio.

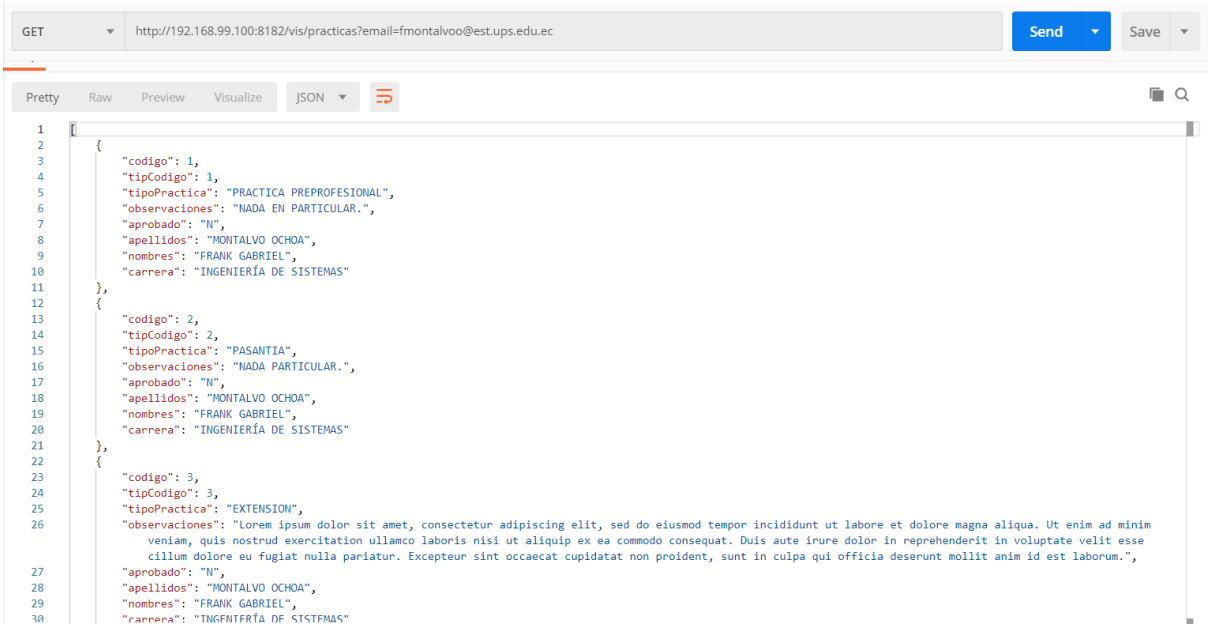


Figura 48. Probando el microservicio sobre Docker.

Como se muestra en la Figura 48, el servicio sigue funcionando de la misma forma que funcionaba sobre el entorno de desarrollo con Spring boot (véase Figura 36), la única diferencia que existe entre la primera con respecto a esta es en la dirección IP y el número de puerto usado.

Finalmente, para tener acceso a esta imagen desde cualquier otro equipo con Docker instalado es necesario acceder al repositorio de imágenes. Este funciona de la misma forma que un repositorio de Git, ya que él daemon de Docker permite descargar una imagen desde el repositorio con `docker pull <<imagen>>` o por el contrario subir una imagen ya creada a un repositorio de imágenes a través del comando `docker push ruta/<<imagen>>` donde la ruta hace referencia al repositorio del usuario al cual se va a subir la imagen.

Antes de poder subir la imagen en algún repositorio es necesario iniciar sesión en el mismo a través del daemon de Docker, para esto se debe ejecutar el comando `docker login [OPTIONS] [SERVER]` lo que mostrará un mensaje pidiendo el nombre de usuario y contraseña del repositorio, también está la opción de especificar el nombre de usuario y contraseña como argumentos del comando, e incluso es posible especificar el servidor de imágenes al cual se desea subir la imagen.

4.2 IMPLEMENTACIÓN SOBRE KUBERNETES

Al igual que con Docker, antes de trabajar con Kubernetes se debe asegurar que se encuentre instalado en el entorno de desarrollo. Para esto, ir a la documentación de Kubernetes donde se encuentra las instrucciones con todos los pasos necesarios para su instalación en cada plataforma. En este caso, se instalará una versión para desarrollo de Kubernetes llamada Minikube el cual requiere de uno de los siguientes hipervisores dependiendo la plataforma sobre la cual se instale.

Tabla 4. Hipervisores requeridos por plataforma

Hipervisor	Sistema Operativo
VirtualBox, Hyper-V	Windows
VirtualBox, VMware Fusion, HyperKit	MacOS
VirtualBox, KVM	Linux

En caso de no poder instalar el hipervisor en el entorno, ya sea porque la arquitectura del sistema no soporta virtualización o no se cuente con el espacio suficiente para hospedar otro sistema en el entorno, Minikube ofrece la posibilidad de trabajar sin este, sin embargo, es necesario instalar Docker.

Cabe mencionar que, para instalar Minikube en el entorno es necesario instalar *kubeclt* el cual viene a ser el daemon de Kubernetes. *Kubeclt* es una herramienta en la línea de comandos similar a la de Docker, y esta permite ejecutar una serie de comandos necesarios para crear y desplegar clústeres en Kubernetes. De forma similar que Docker, Kubernetes permite instalarlo de dos formas, la primera está más enfocada a un entorno de producción y consiste en instalar Kubernetes directamente sobre un servidor el cual funcionará como un nodo; la otra forma es instalar Kubernetes sobre una instancia de una máquina virtual el cual hará de nodo, esta forma es utilizada exclusivamente para desarrollo.

Cuando todo esté instalado, se ejecuta el comando `kubectl` o `minikube` dependiendo si se encuentra en un entorno de producción o uno de desarrollo, al ejecutar uno de estos comandos debería devolver una salida con cada una de las opciones y parámetros que se pueden utilizar como argumentos, si este devuelve un mensaje de error es posible que no se haya instalado de forma correcta. Para desplegar la aplicación sobre Kubernetes se necesita crear un archivo con extensión *.yaml* el cual contiene todas las instrucciones necesarias para crear un objeto de Kubernetes con las configuraciones que detalladas en el mismo.

```yaml
apiVersion: v1
kind: Namespace
metadata:
    name: vis

---
apiVersion: v1
kind: Service
metadata:
    name: vis
    namespace: vis
spec:
    type: NodePort
    ports:
        - name: http
          port: 8180
          targetPort: 8180
    selector:
        app: vis

---

apiVersion: apps/v1
kind: Deployment
metadata:
    name: vis
    namespace: vis
spec:
    replicas: 3
    selector:
        matchLabels:
            app: vis
    template:
        metadata:
            labels:
                app: vis
        spec:
            containers:
                - name: vis
                  image: fgmo/ms-tesis:1.0.0
                  ports:
                      - name: http
                        containerPort: 8180
```

Figura 49. Archivo de configuración de Kubernetes

La Figura 49 se puede ver todas las instrucciones necesarias para crear y desplegar un objeto de Kubernetes con una aplicación, ahora se describirá de manera general que es lo que está haciendo este archivo de configuración.

La primera parte que corresponde de la línea 1 a la 5 indica el nombre del objeto en Kubernetes esto lo hace para diferenciarlo de los otros objetos que ya están configurados; la siguiente sección que inicia de la línea 7 a la 20 define el servicio en Kubernetes, el cual se encarga de balancear los pods con la aplicación. Algunas instrucciones importantes que hay en esta sección son:

- **Metadata:** aquí se define el nombre de la instancia y se hace referencia al namespace que se creó anteriormente, con esto se indica a Kubernetes que la instancia corresponde al mismo dominio.

- **Spec:** aquí es donde se especifica el estado y las configuraciones o especificaciones que va a manejar el objeto.

- **Type:** indica a Kubernetes la interfaz a través de la cual se accede al servicio que ofrece la aplicación.

- **Ports:** define los puertos que se manejan internamente en el objeto (*targetPort*) y el puerto por el cual permitirá el tráfico de red (*port*) y también indica el protocolo de red con el que trabaja.

Finalmente, la sección que va desde la línea 23 a la 43 crea y despliega la aplicación sobre los pods, es la parte encargada de definir el set de réplicas para mantener el servicio disponible aun si una de las réplicas deja de funcionar; las instrucciones más importantes que se pueden encontrar en esta sección son:

- **Réplicas:** indica el número de réplicas o instancias de la aplicación que va a crear.

- **Selector:** especifica la etiqueta a través de la cual Kubernetes buscará otros pod con la misma; gracias a esta Kubernetes puede eliminar o agregar nuevas instancias de una aplicación.

- **Template:** es la plantilla base a partir de la cual Kubernetes crea los pods, este también tiene un *spec* donde agregar todas las configuraciones que desee añadir al pod.

- **Containers:** aquí es donde se crean los contenedores con la o las aplicaciones que requiera que Kubernetes genere el clúster.

- **Name:** asigna un nombre al contenedor.

- **Image:** indica el nombre y la versión de la imagen que se desea desplegar en el clúster. Aquí se agrega la imagen que se creó con Docker anteriormente, al indicar su nombre Kuberenetes se encargará de descargarla, incluirla en un pod y ejecutarla para su uso.

- **ContainerPort:** debe ser el mismo que el *targetPort* para permitir el paso de las peticiones desde el servicio al pod.

Cuando vaya a crear un objeto de Kubernetes con una aplicación debe ejecutar `kubectl apply -f <<archivo.yaml>>` y al igual que con Docker, Kubernetes comenzará a ejecutar las instrucciones en orden. Al ejecutar el comando vera una salida como la de la Figura 50.

```
fmont@FGMO MINGW64 ~/Workspace/spring-workspace/microservicio_tesis_security (ms_security)
$ kubectl apply -f vis-ms.yaml
namespace/vis created
service/vis created
deployment.apps/vis created
```

Figura 50. Kubeclt apply.

Como se puede apreciar Kubernetes ha creado las instancias sin ningún inconveniente; si desea comprobar el estado de los pods ejecute el comando de la Figura 51.

```
fmont@FGMO MINGW64 ~/Workspace/spring-workspace/microservicio_tesis_security (ms_security)
$ kubectl get pods -n vis
NAME                   READY   STATUS    RESTARTS   AGE
vis-85479c99cf-6kgcm   1/1     Running   0          20s
vis-85479c99cf-csmtx   1/1     Running   0          20s
vis-85479c99cf-csxlk   1/1     Running   0          20s
```

Figura 51. Pods de Kubernetes.

De esta forma, se ha verificado que el clúster está funcionando, esto desde el cliente en línea de comandos de Kubernetes. Si lo desea, también puede monitorear y administrar el estado del clúster desde el dashboard que integra Minikube, para esto ejecute el comando `minikube dashboard` que ejecutara una ventana en el navegador desde donde podra controlar el nodo.

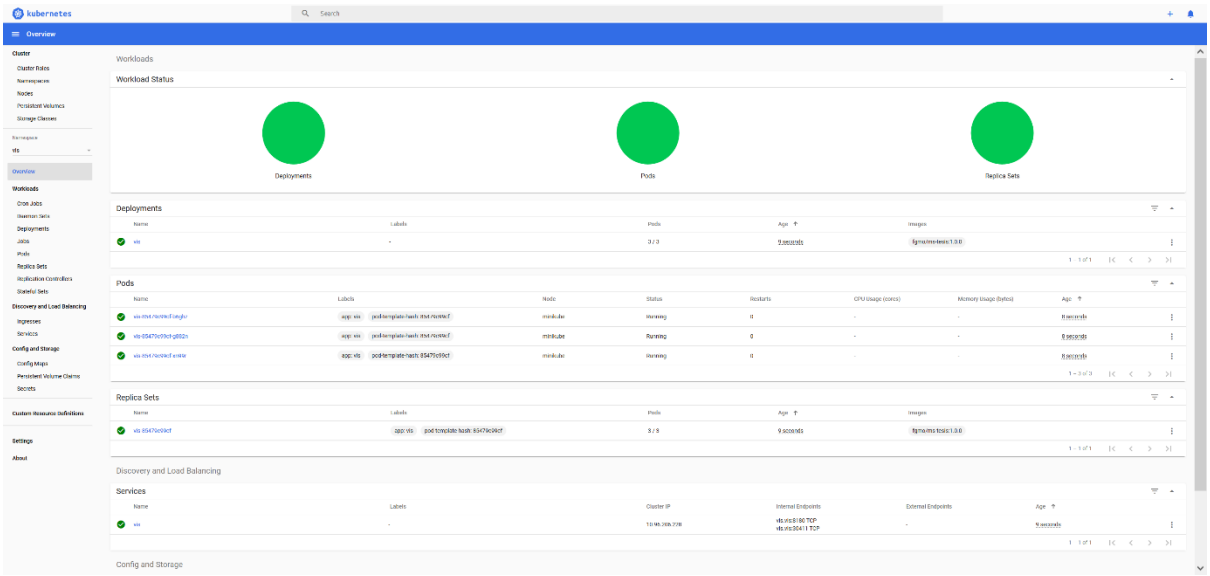


Figura 52. Minikube Dashboard.

La Figura 52, muestra el estado del clúster a través del dashboard, desde aquí puede realizar casi las mismas acciones que con el daemon de Kubernetes (kubectl). También podra acceder a los *logs* de los pods, donde podra observar la imagen de Docker ejecutándose.

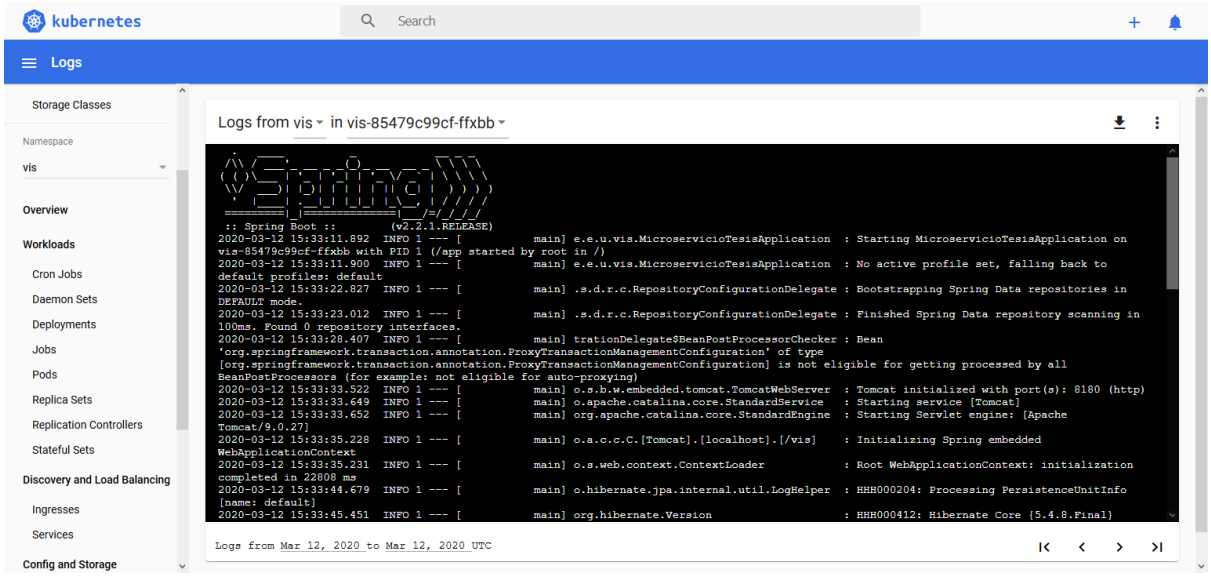


Figura 53. Imagen de Docker ejecutándose sobre Kubernetes.

En la Figura 53 se puede observar una salida similar al ejecutar la imagen directamente con `Docker run`. Ahora, para poder consumir los servicios de la aplicación se debe ingresar a el nodo a través de su dirección IP y el número de puerto que Kubernetes haya asignado, para saber cuál es el puerto, debe abrir la sección de *Services* en el dashboard donde indica el puerto que maneja internamente el contenedor y el puerto con el cual puede realizar peticiones al mismo.

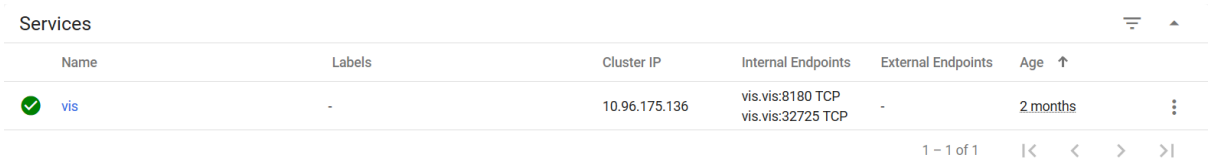


Figura 54. Endpoints del microservicio.

Ahora que conoce el puerto de acceso, ya puede realizar peticiones al servicio, para esto vuelva a hacer uso de *Postman* para realizar las peticiones, en la Figura 55, se realiza una consulta de las prácticas de un estudiante a través de su correo institucional, como puede observar la dirección IP y el número de puerto son distintos a de las figuras 48 y 38.

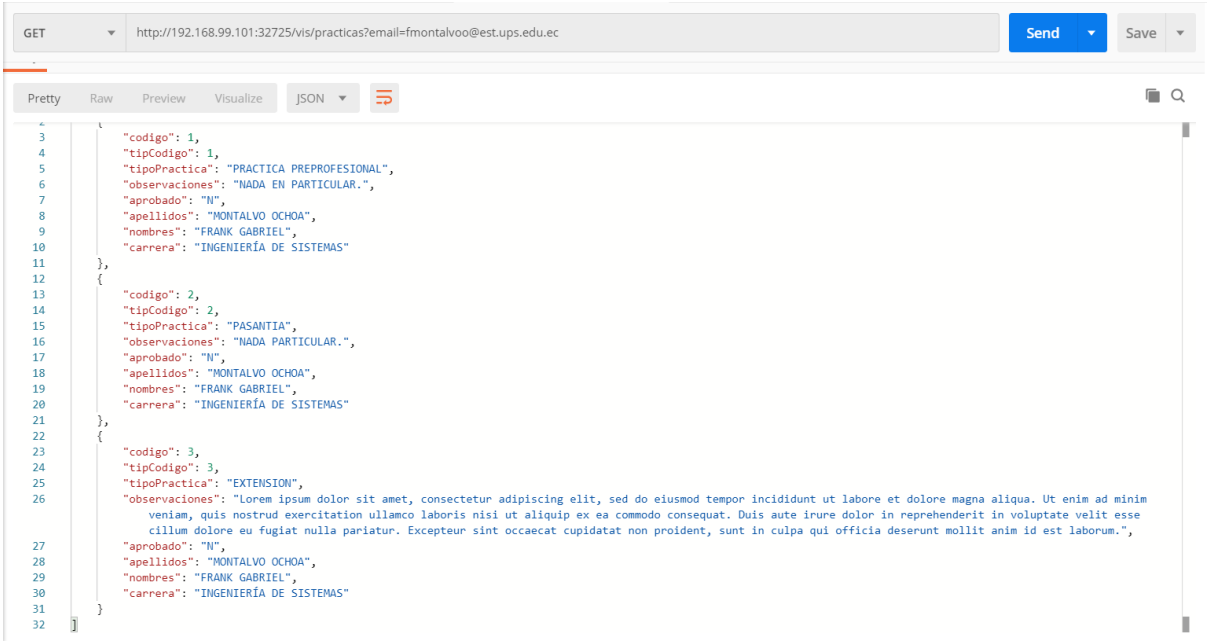


Figura 55. Probando el microservicio sobre Kubernetes.

4.3 AÑADIENDO SEGURIDAD AL SERVICIO

Una vez desplegado el servicio sobre un contenedor de Docker, que a su vez está corriendo sobre Kubernetes; el cual se encarga de gestionar y automatizar el proceso de despliegue de aplicaciones sobre contenedores, y también se ha comprobado el correcto funcionamiento de los mismos a través de peticiones de un cliente HTTP, es momento de incorporar la seguridad a esta peticiones, para garantizar o por lo menos tratar de evitar que cualquier individuo acceda a estos servicios sin ningún tipo de restricción. Como se propuso en el esquema con el diseño general del sistema (véase Figura 8), el servicio tendrá un módulo asociado el cual trabaja con *CAS (Central Authentication Service)* que no es más que un protocolo para la autenticación del usuario mediante sus credenciales como puede ser su nombre de usuario y contraseña. El CAS proporcionará acceso a los estudiantes de la Universidad con su correo institucional y contraseña conferidas por la misma Universidad y al hacer uso de los servicios de autenticación de esta se puede integrar la aplicación de una forma relativamente sencilla en conjunto con los demás sistemas que opera la Universidad.

Sin embargo, el sistema ya cuenta con un cierto nivel de seguridad dado su diseño e implementación, debido a que la parte del sistema que proporciona el API REST para la comunicación con la aplicación móvil es un sistema aislado, separado casi en su totalidad de la parte administrativa, lo único en común entre estos sistema es la información que manejan y su fuente de datos, incluso a nivel de diseño existen ciertas restricciones que hacen imposible realizar las mismas acciones desde la aplicación que la parte administrativa del sistema; acciones como crear o modificar pasantías, agregar tutores, asignar estudiantes a la misma son acciones que únicamente las podrá realizar el personal administrativo autorizado de la universidad, el estudiante únicamente podrá consultar sus prácticas desde la aplicación y podrá crear, modificar y asignar tareas a las mismas. Sin embargo, no podrá eliminar las actividades creadas esta acción será posible únicamente desde la parte administrativa. Al limitar las

acciones del estudiante sobre el sistema se asegura la fiabilidad e integridad de los datos que almacena el sistema.

Otra medida de seguridad implementada a nivel de programación se puede encontrar en los objetos que usa, ya que estos son utilizados como objetos para responder a las peticiones HTTP del cliente y no realizan un mapeo directo de las tablas en la base de datos, si no que estos solo se asocian con ciertos campos y no con todos, lo que permite enviar al usuario solo la información que este requiere, descartando la información irrelevante para el mismo. Además, con esto se protege la información delicada que no se desea que se envié a través de la internet, como, por ejemplo, el número de resolución de la práctica.

Además de todo lo antes expuesto, el sistema y sus servicios pueden ser accedidos por cualquier cliente con una conexión a la red y con un cliente HTTP como lo puede ser un navegador web, por lo que es necesario el controlar quien puede y quién no hacer peticiones al servicio y es por esto que antes de realizar peticiones al servicio se accede primero a un proveedor de identidad para autenticar al usuario; este permite generar un token de acceso el cual el cliente usará cada vez que desee realizar una consulta al servicio, que por su parte se encarga de validar el token en cada petición para verificar que se trate de una sesión válida y le conceda acceso a los servicios del sistema.

4.3.1 JWT

JSON Web Token es un estándar para crear tokens de acceso, los cuales serán utilizados por los usuarios de un determinado servicio para identificarse y tener un nivel de acceso o rol al mismo (Peyrott, 2018). Un JWT tiene una estructura que normalmente cuenta de tres partes:

- **Header:** La cabecera del token contiene información sobre el mismo, y en esta se pueden encontrar los siguientes parámetros:

Tabla 5. Cabecera JWT.

Parámetro	Descripción
`alg`	Algoritmo empleado para encriptar el token.
`typ`	Identifica el tipo de token, por lo general tiene “jwt” por valor y puede variar en muy raras ocasiones.
`cty`	Indica el contenido del token, si por alguna razón el contenido del token es otro token JWT, este campo indicará que es necesario realizar otro proceso para desencriptar el contenido.

- **Payload:** Contiene información referente al usuario y al token en sí, algunos de los parámetros que se encuentran en esta parte son:

Tabla 6. Contenido JWT.

Parámetro	Descripción
`iss`	Identifica el ente proveedor del token
`sub`	Indica a quién pertenece el token
`aud`	Identifica a los destinatarios del token
`exp`	Indica la fecha y hora a partir de la cual el token deja de ser válido para su uso
`nbf`	Indica la fecha y hora a partir de la cual el token es válido para su uso
`iat`	Indica la fecha y hora exacta de la emisión del token
`jti`	Identificador único para el token, sirve para diferenciarlo de otros tokens con contenido similar

- **Signature:** Sirve para comprobar que el token y su contenido no ha sido alterado durante su trayecto, si el token tiene una firma de clave privada se puede corroborar que el remitente es quien dice ser.

4.3.2 OAUTH

Es un estándar que describe la forma en la que los sitios web o aplicaciones deben conceder acceso y autorizar sus usuarios al hacer uso de un protocolo en el cual se definen las reglas los medios o mecanismos de autorización y acceso a los recursos de un servidor. Con OAuth un usuario puede acceder a uno o más recursos haciendo uso de una llave de acceso o token el cual es solicitado por la API en cada petición que se realiza a esta. Esta API a su vez verifica que el token proporcionado es válido, con lo cual verifica que se trata de un cliente confiable y prosigue a responder al mismo con la información que había solicitado.

Ahora, para poder obtener un token válido es necesario contar con un proveedor de identidad, el cual solicita al usuario sus credenciales, en este caso en concreto el *email* y la *contraseña* para verificar que este usuario realmente existe y tiene autorización para acceder a los recursos del servidor. Sin embargo, esta no es la única forma de autorizar el acceso de un usuario a los recursos de un servidor, también está la autenticación a través de terceros, la cual solicita el token de acceso a otro servicio como Google.

Para implementar OAuth en los servicios y la aplicación móvil es necesario agregar ciertas bibliotecas para trabajar con esta, en Spring Boot se debe usar *Spring Security* el cual ya incluye OAuth. Una vez instalada esta dependencia en el proyecto debe de configurar la seguridad en

el servicio, por lo cual se debe crear una clase en Java que se encargara de esto, dentro de esta clase se puede especificar cómo va a conceder el acceso a los servicios y manejar la sesión de usuario, en este caso se va a trabajar con *JWT* por lo tanto en la configuración se debe indicar que cualquier solicitud realizada a él API REST debe incluir un token JWT para conceder acceso al servicio, entonces la clase con la configuración de seguridad queda como en la Figura 56.

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

        @Override
        protected void configure(HttpSecurity http) throws Exception {
                http.authorizeRequests().anyRequest().authenticated()
                .and().oauth2ResourceServer().jwt();
        }

}
```

Figura 56. Configuración de la seguridad en el servicio REST.

Si ahora realiza una petición al servicio, este devolverá un código de *error 401* el cual indica que no tiene autorización para acceder al servicio, con esto se garantiza que al servicio únicamente podrán acceder aquellos usuarios que cuenten con una llave de acceso u token.

Ahora que el servicio ya cuenta con un cierto nivel de seguridad, es hora de realizar la configuración correspondiente a la aplicación móvil, para esto primero se añade la dependencia *OAuth2* a través de la cual puede solicitar el token JWT al proveedor de identidad, el cual se debe utilizar para realizar peticiones al servicio. Para esto se debe crear un método para el inicio de sesión en Flutter; este método solicitará al estudiante su correo institucional y su contraseña para verificarlos contra el proveedor de identidad, si las credenciales proporcionadas son correctas entonces el proveedor devolverá un token el cual se almacenara en el dispositivo móvil y lo que se enviará en cada solicitud realizada por la aplicación al servicio.

```
Future login(String username, String password) async {
  final authorizationEndpoint = Uri.parse(Constans.TOKEN);
  var response;
  try {
    var client = await oauth2.resourceOwnerPasswordGrant(
        authorizationEndpoint, username, password,
        basicAuth: false,
        scopes: ['openid'],
        identifier: Constans.CLIENT_ID,
        secret: null);
    response = client.credentials;
    saveToken(response.accessToken);
  } catch (e) {
    response = 'invalid_grant';
  }
  return response;
}
```

Figura 57. Método para inicio de sesión en Dart.

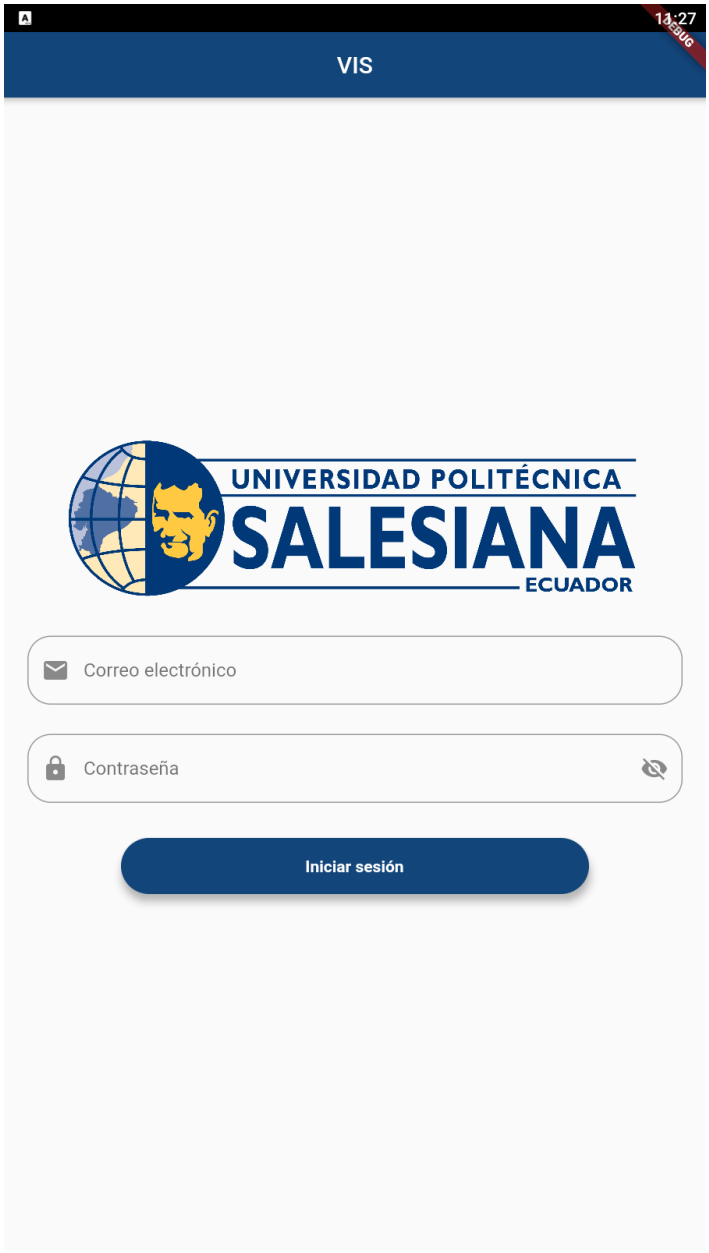


Figura 58. Pantalla de Inicio de sesión en la aplicación

CAPÍTULO V
CONCLUSIONES Y RECOMENDACIONES

5.1 CONCLUSIONES

Durante el desarrollo de este proyecto, Java ha jugado un papel muy importante para su creación, ya que la aplicación para la gestión administrativa del sistema fue escrita en su mayoría principalmente haciendo uso de este lenguaje y del framework de Java para desarrollar aplicaciones empresariales (JEE por sus siglas en inglés). También se hizo uso del marco de trabajo Spring Boot el cual también utiliza el lenguaje de programación Java para trabajar con él. A través de este framework se creó el API REST del sistema el cual ofrece los servicios para consultar las prácticas y un C.R.U.D de actividades a través de los que un estudiante puede consultar, agregar y modificar actividades a estas.

Spring Boot permitió la creación del API REST en un tiempo relativamente corto, ya que este cuenta con implementaciones simplificadas de los métodos en Java que facilitan la creación, lo que permite un desarrollo más ágil comparado con Java EE, además de que la configuración de la seguridad en esta es más simple y fácil de comprender para un desarrollador.

Otra herramienta que jugó un papel importante en el desarrollo de este proyecto fue Flutter, gracias a este construyó una aplicación móvil que sirve de cliente para los servicios REST desarrollado con Spring. Además, el rendimiento y funcionamiento de la aplicación es igual en Android e IOs haciendo posible el mantenimiento de la aplicación con una base única de código lo que permite reducir enormemente los tiempos de desarrollo para ambas plataformas, y también posibilita depurar errores y agregar nuevas funciones.

El uso de Docker como contenedor de software ha demostrado agilizar el proceso de despliegue de aplicaciones y servicios en entornos de prueba y/o producción, además de ofrecer la posibilidad de crear aplicaciones portables las cuales pueden ejecutarse en cualquier otro equipo o servidor que cuente con una instalación de Docker. Por otro lado, los repositorios de imágenes de Docker permiten almacenar, obtener y mantener diferentes versiones de aplicaciones que se ejecutan sobre los contenedores de Docker, esto es posible siempre y cuando se cuente con una conexión a internet. También es posible crear un repositorio de imágenes local para aquellos que deseen trabajar con un repositorio local en lugar de los que ofrece Docker a través de internet.

Otro aspecto importante es la integración de Docker con Kubernetes como software de gestión y orquestación de contenedores logrando automatizar el despliegue de aplicaciones que se ejecutan sobre contenedores Docker, además de que permite la administración y asignación de recursos del sistema sobre los contenedores que se encuentren corriendo sobre ese nodo.

Kubernetes también garantiza una mayor estabilidad y la continuidad del funcionamiento de la aplicación así como su disponibilidad hacia los usuarios, puesto que Kubernetes trae integrado herramientas para el balance de carga entre los contenedores con lo cual se puede evitar los cuellos de botella y sobrecargar de trabajo a uno de los contenedores; otras herramientas

importantes que integra Kubernetes es la gestión de réplicas y el monitoreo del estado del nodo y los contenedores que administra, gracias a estas herramientas es que Kubernetes puede asegurar la continuidad de los servicios, ya que si uno de los nodos o contenedores llegase a fallar este se encargaría de desplegar uno nuevo en reemplazo al que esté averiado, además de que este es un proceso es transparente al usuario y no requiere de la intervención del administrador. Sin embargo, esto no aplica si el nodo que se estropeó es el nodo maestro, en cuyo caso el sistema puede dejar de funcionar; pese a esto Kubernetes ha demostrado ser una herramienta muy útil para la administración y despliegue de aplicaciones en entornos de trabajo.

5.2 RECOMENDACIONES

Para trabajar de una forma más ágil con Kubernetes se recomienda utilizar Minikube como alternativa a una implementación completa, dado que a través de esta herramienta se puede trabajar de una forma más simple y sencilla sin la necesidad contar con un servidor o infraestructura de red. Minikube se puede instalar en un ordenador común desde el cual se pueden realizar todas las pruebas y arreglos necesarios previos a colocarlos en producción. Además, de esta forma se puede asegurar de que todo funcione de forma correcta antes de ponerlo en marcha o en producción.

Al trabajar con un API REST, también es recomendable asegurar el canal de comunicación entre el cliente y el servidor a través de protocolos de cifrado como lo es SSL, el cual permite un canal de comunicación seguro entre el cliente y el servidor lo que lo hace menos susceptible a ataques; esto en conjunto con JWT proporciona una mayor seguridad al sistema. Sin embargo, esto es más recomendable si el sistema va a ser desplegado en producción, ya que ahí es donde realmente es necesario mantener seguro el canal de información para de este modo evitar ataques y pérdida de datos e información delicada.

5.3 TRABAJO FUTURO

Integración continua: Es un aspecto importante para tener en cuenta, puesto que permite la detección de fallos en el software antes de su puesta en producción. Básicamente se trata de realizar pruebas unitarias a todos los métodos y funciones del software para asegurar su correcto funcionamiento.

Entrega continua: Ofrece la posibilidad de que el software este siempre listo para su despliegue a producción lo más pronto posible, permitiendo desarrollar y liberar más a menudo versiones nuevas de software. Esto permite ahorrar tiempo y costo de producción de software.

Despliegue continuo: Es el paso que sigue a los dos anteriores, y el punto culminante en el ciclo de desarrollo, permite lanzar software a producción y que este se accesible por el usuario. Todos estos pasos se pueden integrar en este proyecto y gracias al uso de contenedores de software y Kubernetes para la orquestación de estos, ofrece la posibilidad de poner a disposición del usuario nuevas versiones del software y al mismo tiempo ser escalable, permitir alta disponibilidad y tolerancia a fallos.

Clúster: La adición de más nodos en una red permite la posibilidad de trabajar con un cluster en Kubernetes, en este proyecto se trabajó con un solo nodo que hacia el papel de master y worker a la vez. Agregar más nodos ofrece la posibilidad de probar otras características que incluye Kubernetes, como el control de varios nodos en la misma red, escalar de forma horizontal al agregar más nodos según se demande, tener un alta disponibilidad y gran tolerancia a fallos y un mejor tiempo de respuesta a las solicitudes realizadas por los clientes.

Automatizar la creación de clusters: Existen varias herramientas que permiten el despliegue de clusters en producción de forma estable y segura; implementar estas herramientas facilitaría en gran medida el trabajo de poner en producción nuevas aplicaciones, además de permitir escalar a las aplicaciones según se requiera en función de la demanda existente a través de la red.

REFERENCIAS

Alicia Durango, Á. A. (2016). *Curso de Programación con Java: 2ª Edición.* IT Campus Academy.

Bernstein, D. (2014). Containers and Cloud: From LXC to Docker to Kubernetes. *IEEE Cloud Computing*, 81-84.

Blokehead, T. (2017). *Cómo construir Microservicios : Los diez principales trucos para modelar, integrar y desplegar microservicios.* Babelcube Inc.

Brewer, E. A. (2015). Kubernetes and the path to cloud native. *Proceedings of the Sixth ACM Symposium on Cloud Computing*, 167.

Bukhary Ikhwan Ismail, E. M. (2015). Evaluation of Docker as Edge computing platform. *IEEE Conference on Open Systems (ICOS)*, 130-135.

Carlos Caballero González, J. A. (2016). *UF1465 - Computadores para bases de datos.* Ediciones Paraninfo, S.A.

David Jaramillo, D. V. (2016). Leveraging microservices architecture by using Docker technology. *SoutheastCon 2016*, 1-5.

DeMichiel, L. G. (2001). *CiteSeerX.* Obtenido de CiteSeerX: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.68.1148&rep=rep1&type=pdf

Díaz, C. A. (2019). *Programacion en JAVA I: El entorno de programación – Sintaxis – Elementos – Estructuras de control.* RedUsers.

Docker. (12 de Marzo de 2020). *Dcoker Docs*. Obtenido de Docker Docs: https://docs.docker.com/

Docker. (12 de Marzo de 2020). *Docker*. Obtenido de Docker: https://www.docker.com/resources/what-container

Flutter. (12 de Marzo de 2020). *Flutter Docs*. Obtenido de Flutter Docs: https://flutter-es.io/docs/resources/technical-overview

Heffelfinger, D. R. (2017). *Java EE 8 Application Development: Develop Enterprise applications using the latest versions of CDI, JAX-RS, JSON-B, JPA, Security, and more.* Packt Publishing Ltd.

Hosseini, P. (2018). *Flutter: For Absolute Beginners.* Fatemeh Tajik.

IEEE Computer Society. (2004). *Guide to the software engineering body of knowledge.* IEEE Computer Society.

Jangla, K. (2018). *Accelerating Development Velocity Using Docker.* Apress.

Joy, A. M. (2015). Performance comparison between Linux containers and virtual machines. *International Conference on Advances in Computer Engineering and Applications*, 342-346.

Kubernetes. (12 de Marzo de 2020). *Kubernetes Docs*. Obtenido de Kubernetes Docs: https://kubernetes.io/es/docs/home/

Marchioni, F. (2018). *WildFly Administration Guide: Learn how to configure the newest WildFly application server.* ITBuzzPress.

Merkel, D. (2014). Docker: lightweight Linux containers for consistent development and deployment. *Linux Journal*.

Microsoft. (12 de Marzo de 2020). *Microsoft Docs*. Obtenido de Microsoft Docs: https://docs.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices

Miguel, J. T. (2015). *MF0493_3 - Implantación de aplicaciones web en entorno internet, intranet y extranet.* Ediciones Paraninfo, S.A.

Oracle. (12 de Marzo de 2020). *Oracle*. Obtenido de Oracle: https://www.oracle.com/technetwork/java/javaee/overview/compatibility-jsp-136984.html

Peyrott, S. (2018). *JWT Handbook.* Auth0 Inc.

Prateek Sharma Lucas Chaufournier, P. J. (2016). Containers and Virtual Machines at Scale: A Comparative Study. *Proceedings of the 17th International Middleware Conference*, 1-13.

Reis, D. (2013). *Seguridad para la nube y la virtualización.* Hoboken: John Wiley & Sons, Inc.

Sarmiento, A. E. (2019). *Logística de transporte de mercancias en contenedores marítimos.* Ediciones de la U.

Sayfan, G. (2019). *Hands-On Microservices with Kubernetes: Build, deploy, and manage scalable microservices on Kubernetes.* Packt Publishing Ltd.

Scott Chacon, B. S. (2014). *Pro Git.* Apress.

Späth, P. (2019). *Beginning Jakarta EE: Enterprise Edition for Java: From Novice to Professional.* Apress.

Vohra, D. (2016). *Kubernetes Microservices with Docker.* Apress.

Walls, C. (2016). *Spring Boot in action.* Manning Publications Co.

Wes Felter, A. F. (2015). An updated performance comparison of virtual machines and Linux containers. *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 171-172.

Wolf, G. (2015). *Fundamentos de sistemas operativos.* Gunnar Wolf.

Xhafa, F. (2007). *Aplicaciones Distribuidas con Java.* Delta Publicaciones.