MARCO DE TRABAJO PARA DESARROLLAR UNA APLICACIÓN WEB USANDO DOMAIN DRIVEN DESIGN Y DEVOPS

JAVIER ALEJANDRO PÉREZ FERNÁNDEZ

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MÉCANICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2019

MARCO DE TRABAJO PARA DESARROLLAR UNA APLICACIÓN WEB USANDO DOMAIN DRIVEN DESIGN Y DEVOPS

JAVIER ALEJANDRO PÉREZ FERNÁNDEZ

Trabajo de grado presentado como requisito para optar al título de Ingeniero de Sistemas

Director
FERNANDO ANTONIO ROJAS MORALES
Magíster en Ciencias Computacionales

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MÉCANICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2019

DEDICATORIA

To my family, my brother Julian and
the angels who take care of me
in heaven and earth.

AGRADECIMIENTOS

A la Universidad Industrial de Santander y la Escuela de Ingeniería de Sistemas por creer en mis habilidades y darme oportunidad de formarme en esta institución, de apoyarme para estudiar un intercambio a la Universidad de los Andes donde complementé mis estudios enfocándome en áreas de importancia profesional,

Al Banco Scotiabank Colpatria por brindarme una práctica empresarial de aprendizaje donde empecé a desarrollas mis habilidades de gestión y desarrollo de proyectos de software junto a todos los conocimientos adquiridos previamente.

A Amaris Consulting, que es el lugar donde me apoyaron con el tiempo para la realización de este proyecto y la experiencia adquirida como líder de proyectos de software, las oportunidades de poner a prueba mis habilidades comunicativas y de ingeniería, y sobre todo por esa amistad incondicional durante el tiempo que he estado con ellos. Este trabajo son fruto de esas experiencias vividas, que me han hecho crecer profesionalmente.

En especial al profesor Fernando Rojas, por aceptar la dirección de este proyecto, por su tiempo y dedicación, compromiso, consejos, persistencia y el apoyo durante la realización de este trabajo de grado.

Y agradezco principalmente a mi familia y mis mejores amigos, que me apoyaron en los momentos difíciles, y me dieron una mano de fortaleza, amabilidad y apoyo para continuar con mi proyecto de vida.

CONTENIDO

Pág.

INTRODUCCIÓN **16**

1. DESCRIPCIÓN GENERAL **19**

1.1. PLANTEAMIENTO Y JUSTIFICACIÓN DEL PROBLEMA 19

1.2. OBJETIVOS 21

1.2.1. Objetivo General. 21

1.2.2. Objetivos Específicos. 21

1.3. METODOLOGÍA 22

1.3.1. FASE 1: Exploración tecnológica. 22

1.3.2. FASE 2: Análisis del framework. 23

1.3.3. FASE 3: Desarrollo del framework. 23

1.3.4. FASE 4: Descripción del prototipo. 24

1.3.5. FASE 5: Desarrollo y despliegue del prototipo. 24

1.3.6. FASE 6: Uso del framework en el prototipo. 25

2. ESTADO DEL ARTE **26**

3. MARCO DE REFERENCIA **28**

4. DISEÑO Y DESARROLLO DEL FRAMEWORK **29**

4.1. DESCRIPCIÓN Y OBJETIVO DEL FRAMEWORK 29

4.2. CONCEPTOS DE DISEÑO 31
4.3. ¿POR QUÉ SE UTILIZA DEVOPS Y DDD? 32
4.4. PROPUESTA I: MODEL VIEW CONTROLLER (MVC) 33
4.4.1. Arquitectura de software PROPUESTA I. 33
4.4.2. Arquitectura de infraestructura PROPUESTA I. 34
4.5. PROPUESTA II: SERVICE ORIENTED ARCHITECTURE (SOA) 37
4.5.1. Empezando por el Backend. 37
4.5.2. Luego por el Frontend. 39
4.5.3. Arquitectura de software PROPUESTA II. 40
4.5.4. Arquitectura de infraestructura PROPUESTA II. 44
4.6. PROPUESTA FINAL: FRAMEWORK FOR WEB DEVELOPMENT (FWD) 46
4.6.1. Arquitectura de software FWD (Ambiente desarrollo). 46
4.6.2. Arquitectura de software FWD (Ambiente producción). 49
4.6.3. Arquitectura de infraestructura FWD (Ambiente desarrollo). 51
4.6.4. Arquitectura de infraestructura FWD (Ambiente producción). 51
4.6.5. DevOps Pipeline FWD. 55
4.6.6. Roles y responsabilidades del equipo de desarrollo. 55
4.6.7. Interacción entre ambientes FWD. 58

5. ESTRATEGIA PARA EL USO DEL FWD 61
5.1. ETAPA 1: PRERREQUISITOS Y ANÁLISIS 62
5.2. ETAPA 2: APLICACIÓN BASE 63
5.3. ETAPA 3: AMBIENTE DE DESARROLLO Y LINEAMIENTOS. 63
5.4. ETAPA 4: DATOS Y PERSISTENCIA 64

5.5. ETAPA 5A: FRONTEND 65

5.6. ETAPA 5B: BACKEND 67

5.7. ETAPA 6: INTEGRACIÓN CONTINUA 69

5.8. ETAPA 7: VALIDACIÓN DEL MVP 70

6. CONCLUSIONES 71

7. RECOMENDACIONES 74

BIBLIOGRAFÍA 76

ANEXOS 81

LISTA DE TABLAS

Pág.

Tabla 1. Stakeholders del framework 119
Tabla 2. Detalle funcionalidad 1 FWD 120
Tabla 3. Detalle funcionalidad 2 FWD 120
Tabla 4. Detalle funcionalidad 3 FWD 121
Tabla 5. Detalle funcionalidad 4 FWD 121
Tabla 6. Detalle funcionalidad 5 FWD 122
Tabla 7. Priorización de funcionalidades 123
Tabla 8. Codificación de roles para la aplicación 129
Tabla 9. Contratos de servicio para la aplicación FWD 130
Tabla 10. Stakeholders de UPost 135
Tabla 11. Detalle funcionalidad 1 UPost 136
Tabla 12. Detalle funcionalidad 2 UPost 137
Tabla 13. Detalle funcionalidad 3 UPost 137
Tabla 14. Detalle funcionalidad 4 UPost 138
Tabla 15. Detalle funcionalidad 5 UPost 138
Tabla 16. Detalle funcionalidad 6 UPost 139
Tabla 17. Detalle funcionalidad 7 UPost 139
Tabla 18. Detalle funcionalidad 8 UPost 140
Tabla 19. Detalle funcionalidad 9 UPost 140
Tabla 20. Detalle funcionalidad 10 UPost 141
Tabla 21. Detalle funcionalidad 11 UPost 141
Tabla 22. Detalle funcionalidad 12 UPost 142
Tabla 23. Priorización de funcionalidades UPost 142
Tabla 24. Contratos de servicio para la aplicación UPost 145

LISTA DE FIGURAS

Pág.

Figura 1. Metodología. 22
Figura 2. Arquitectura de software PROPUESTA I. 34
Figura 3. Arquitectura de infraestructura PROPUESTA I (Ambiente desarrollo). 35
Figura 4. Arquitectura de infraestructura PROPUESTA I (Ambiente producción). 36
Figura 5. Arquitectura N-capas Backend. 38
Figura 6. Arquitectura Flux Frontend. 39
Figura 7. Arquitectura de software PROPUESTA II (Ambiente desarrollo). 40
Figura 8. Arquitectura de software PROPUESTA II (Ambiente producción). 43
Figura 9. Arquitectura de infraestructura PROPUESTA II (Ambiente desarrollo). 44
Figura 10. Arquitectura de infraestructura PROPUESTA II (Ambiente prod.). 45
Figura 11. Arquitectura de software FWD (Ambiente desarrollo). 48
Figura 12. Arquitectura de software FWD (Ambiente producción). 50
Figura 13. Arquitectura de infraestructura FWD (Ambiente desarrollo). 53
Figura 14. Arquitectura de infraestructura FWD (Ambiente producción). 54
Figura 15. DevOps Pipeline. 56
Figura 16. Interacción entre ambientes. 59
Figura 17. Estrategia para el uso del FWD. 61
Figura 18. Ciclo de iteración DDD 86
Figura 19. Arquitectura Web cliente-servidor simple 88
Figura 20. Arquitectura de N-capas 89
Figura 21. Arquitectura Flux 92
Figura 22. Herramientas para desarrollo web 93
Figura 23. Ciclo de vida del desarrollo de software usando DevOps 100
Figura 24. Ciclo infinito de las etapas de DevOps 100
Figura 25. Visión general de DevOps 101

Figura 26. Infraestructura de máquina local (Solo ambiente desarrollo) 103
Figura 27. Infraestructura de máquina local con máquinas virtuales 104
Figura 28. Infraestructura de máquina local con conteiner 104
Figura 29. Infraestructura en la nube con máquinas virtuales 104
Figura 30. Infraestructura en la nube con conteiner 104
Figura 31. Sistema de control de versiones distribuido Git 107
Figura 32. Funciones relevantes de Git 107
Figura 33. Git Flow general 108
Figura 34. Ambiente de desarrollo e integración del repositorio central. 115
Figura 35. Ambiente de desarrollo con la herramienta de integración continua. 116
Figura 36. Integración de Terraform y DevOps pipeline final. 117
Figura 37. Diagrama conceptual FWD 127
Figura 38. Diagrama de datos FWD 128
Figura 39. Customer Journey FWD 129
Figura 40. Maqueta de Home and About 132
Figura 41. Maqueta de Login and Sign Up 132
Figura 42. Maqueta de Home (Authenticated) and User Profile 133
Figura 43. Maqueta de Edit your profile Modal and Confirm edit Modal 133
Figura 44. Maqueta de Configuration Modal and Delete user account Modal 133
Figura 45. Maqueta de Confirm delete user account Modal 134
Figura 46. Diagrama conceptual UPost 143
Figura 47. Diagrama de datos UPost 144
Figura 48. Customer Journey UPost 146
Figura 49. Landing Page y vesión móvil de UPost 147
Figura 50. Login y Sign Up de UPost 148
Figura 51. Home page con listado de posts y Home Page versión móvil UPost 149
Figura 52. Modal para cargar imagen del post y Modal de confirmación UPost 150
Figura 53. User page con posts del usuario y User page versión móvil UPost 151
Figura 54. Eliminar y editar usuario UPost 152

LISTA DE ANEXOS

Pág.

Anexo A Marco de referencia. 81

Anexo B PoC DevOps pipeline y enfoque DDD con un servidor. 114

Anexo C MVP Framework FWD. 119

Anexo D MVP Prototipo UPost. 135

RESUMEN

TÍTULO: MARCO DE TRABAJO PARA DESARROLLAR UNA APLICACIÓN WEB USANDO DOMAIN DRIVEN DESIGN Y DEVOPS[*].

AUTORES: JAVIER ALEJANDRO PÉREZ FERNÁNDEZ[**].

PALABRAS CLAVE: MARCO DE TRABAJO, APLICACIONES WEB, DOMAIN DRIVEN DESIGN, DEVOPS, AUTOMATIZACIÓN, INFRAESTRUCTURA, ARQUITECTURA DE SOFTWARE, INGENIERÍA DE SOFTWARE, DESARROLLO DE SOFTWARE.

DESCRIPCIÓN:

Con el alto crecimiento de iniciativas para el desarrollo e innovación de productos digitales y de transformación tecnológica, la amplia cantidad de herramientas disponibles y la falta de estrategias claras para usarlas de la mejor manera, se plantea una solución tecnológica diseñando un marco de trabajo (framework) mediante el cual es posible la creación de aplicaciones web, estructuradas y organizadas por dominios.

Este framework, realizado usando una metodología ágil, se enfoca en las necesidades actuales que tienen los desarrolladores y las organizaciones para el crecimiento, actualización y reutilización de las aplicaciones.

Abierto para mejoras constantes o personalizaciones y pensado para una alta disponibilidad de usuarios, integración con diferentes sistemas y automatización usando herramientas de DevOps, donde el framework mejora la calidad integrando y probando el código del equipo, crea entregas de la solución listas para desplegar a producción y obtiene una eficiencia en reducir los tiempos de ajustes, creación e instalación de la plataforma de producción. Adicionalmente, tiene en cuenta la homologación del entorno de desarrollo, incluyendo los criterios de seguridad necesarios.

Es una solución que posteriormente se podrá extender a ser reutilizada para crear aplicaciones multiplataforma y tener una herramienta que facilitará la creación y puesta en producción de ideas potenciales de negocio y emprendimiento.

[*] Trabajo de grado

[**] Facultad de ingenierías Físico-mecánicas. Escuela de ingeniería de sistemas e informática. Director MSc. Fernando Antonio Rojas Morales.

ABSTRACT

TITLE: FRAMEWORK FOR THE DEVELOPMENT OF A WEB APPLICATION BY USING DOMAIN DRIVEN DESIGN AND DEVOPS*.

AUTHOR: JAVIER ALEJANDRO PÉREZ FERNÁNDEZ***.

KEYWORDS: FRAMEWORK, WEB APPLICATIONS, DOMAIN DRIVEN DESIGN, DEVOPS, AUTOMATION, INFRASTRUCTURE, SOFTWARE ARCHITECTURE, SOFTWARE ENGINEERING, SOFTWARE DEVELOPMENT.

DESCRIPTION:

With the high growth of initiatives for the development and innovation of digital products and technological transformation, the large number of available tools and the lack of clear strategies to use them in the best way, a technological solution is proposed designing a framework through which the creation of structured web applications and organized by domains is possible.

This framework, made using an agile methodology, focused on the current needs of developers and organizations for the growth, update and reuse of applications.

Open for constant improvements or customizations and designed for high user availability, integration with different systems and automation using DevOps tools, where the framework improves quality by integrating and testing the team code, creates deliveries of the solution ready to deploy to production and it obtains an efficiency in the times of adjustments, creation and installation of the production platform. Additionally, it takes into account the homologation of the development environment, including the necessary security criteria.

The described solution can be subsequently extended and reused to create cross-platform applications because it has a tool that facilitates the creation and production of ideas for business and entrepreneurship.

* Bachelor Thesis

** Faculty of Physical-Mechanical Engineering. School of System Engineering and Informatics. Director Fernando Antonio Rojas Morales.

INTRODUCCIÓN

En el mundo del desarrollo se encuentra un amplio espectro de herramientas tecnológicas, enfoques y metodologías que ayudan a la construcción de aplicaciones web. A partir de esto, nace la iniciativa de diseñar una estrategia para aprovechar este conocimiento de la mejor manera en el proceso de desarrollo de una aplicación web moderna.

Teniendo en cuenta las ventajas que ofrece la web, se hace necesario determinar una estructura para la construcción de una aplicación, que facilite replicar el proceso de diseño en una nueva idea, para así beneficiarse de su utilidad rapidez y simpleza, a través de tabletas, smartphones o un computador, sin requerir la instalación de un aplicativo.

Con el propósito de desarrollar diferentes ideas innovadoras de productos digitales, motivar el emprendimiento y la transformación tecnológica, y de esta manera, satisfacer las necesidades globales de la industria, se busca diseñar un marco de trabajo (*Framework*) mediante el cual sea posible la creación de aplicaciones web, como una solución para mejorar la calidad y la efectividad en la gestión del tiempo.

El proyecto encuentra su origen en la iniciativa de investigar, aprender y difundir el conocimiento adquirido sobre las herramientas de software y el enfoque de diseño guiado por el dominio (*Domain Driven Design* – DDD) y DevOps (*Development & Operations*) empleados durante para la construcción del framework, con el fin de ponerlos al alcance de los estudiantes, desarrolladores o interesados en el desarrollo web.

Este trabajo es realizado tomando como base una metodología ágil, se orienta en el crecimiento, la actualización y la reutilización de las aplicaciones, está abierto

para mejoras constantes o personalizaciones, y pensado para una alta disponibilidad de usuarios e integración con diferentes sistemas.

Adicionalmente, el framework incluye la automatización del paso a producción, usando herramientas de DevOps, las cuales mejoran la calidad del software construido, integrando y probando el código del proyecto. Igualmente, se crean entregas de la solución ya listas para desplegar, lo cual reduce los tiempos de ajuste, creación e instalación de la plataforma de producción. Así mismo, el framework tiene en cuenta la homologación del entorno de desarrollo, considerando los criterios de seguridad pertinentes, y un marco de trabajo que facilita la creación y puesta en marcha de ideas potenciales de negocio y emprendimiento.

El presente documento se encuentra estructurado de la siguiente manera:
En el Capítulo 1 se presenta el planteamiento y la justificación del proyecto junto a sus objetivos generales y específicos, y la metodología propuesta para el desarrollo del proyecto a fin de cumplir con los objetivos descritos.

En el Capítulo 2 se explica el estado del arte de este proyecto, el cual está enfocado en la creación de un framework y las ventajas que éstos ofrecen.

Por su parte, el Capítulo 3 expone el marco de referencia para introducir una noción básica de los conceptos usados durante el desarrollo del libro, y es complementado en la sección de anexos donde se explica con mayor detalle cada uno de los conceptos, herramientas, enfoques y metodologías usadas durante el desarrollo del proyecto.

En el capítulo 4 se documenta la descripción, características, funcionalidades generales, conceptos de diseño y requerimientos que tendrá el framework, de manera iterativa las propuestas realizadas durante el diseño y desarrollo del framework, validando el uso de las herramientas de software para la construcción

de aplicaciones web usando DevOps y resolviendo preguntas que se plantearon durante la construcción. Lo cual aporta al perfeccionamiento del diseño del framework con el fin de verificar que se cumplan los lineamientos descritos y presentando respectivamente, las arquitecturas de software e infraestructura junto con la interacción entre ambientes de despliegue y de software para la propuesta final y los roles que se involucran en el desarrollo de un proyecto web.

Finalmente, en el capítulo 5 se muestra la metodología funcional para el uso del framework como estrategia para la implementación de nuevas ideas de negocio y de manera complementaria los roles que se verían involucrados en los proyectos a realizar.

Estos capítulos son complementados con los anexos, que incluyen temas como la explicación de los conceptos usados en el libro, la prueba de concepto para validar las herramientas de DevOps y los lineamientos de DDD. Para el caso del framework se presenta el listado de requerimientos, el *Customer Journey* y las interfaces construidas, así como las componentes que hacen parte de las especificaciones del framework y también todas las especificaciones para el prototipo UPost.

1. DESCRIPCIÓN GENERAL

1.1. PLANTEAMIENTO Y JUSTIFICACIÓN DEL PROBLEMA

Gracias a los avances de la tecnología y la ingeniería de software (Sommerville, 2010), actualmente existen herramientas que permiten hacer implementaciones en cualquier plataforma tecnológica. Una de estas plataformas es la Web (Petrie, 1998), donde los aportes de las comunidades de desarrolladores facilitan la construcción de aplicaciones. Inicialmente, muchas aplicaciones fueron diseñadas para correr únicamente en una computadora personal, ya que el desarrollo para la web presentaba mayor dificultad y las tecnologías como los smartphones, smart TV y wearables, eran todavía incipientes.

Adicionalmente, las conexiones a internet se realizaban sobre anchos de banda muy reducidos que restringían significativamente el tipo de servicios y aplicaciones que se podían ofrecer. Posteriormente, la velocidad de la Internet aumentó pasando de 50 Kbps, en la década del 2000, a conexiones de 5 Mb, 10 Mb o hasta 30 Mb en servicios para hogares disponibles en la actualidad (Leiner, 1997; Akamai, 2016). Fue solo hasta el lanzamiento del iPhone en el año 2009, que se consolidó el concepto de Smartphone (Harford, 2016) a partir del cual nació una nueva línea de pequeñas computadoras que hoy en día constituyen la tecnología más usada para conectarse a internet en el mundo (Statcounter, 2017). Colombia no ha sido ajena a esta tendencia con una cantidad cada vez mayor de usuarios que emplean smartphones como medio principal de acceso a la red, con un 21,8% del total nacional (DANE, 2016).

Las velocidades de conexión a internet y los dispositivos disponibles en la actualidad han permitido ofrecer nuevos servicios y optimizar la búsqueda de información.

Entre las ventajas que tiene el desarrollo de aplicaciones web se cuentan:

1. La disponibilidad y actualización de documentación que le da soporte a estas tecnologías gracias a la realimentación de las comunidades de desarrolladores y usuarios.
2. La existencia de múltiples lenguajes de programación que ofrecen diferentes alternativas a los desarrolladores y que se adaptan a las necesidades de la aplicación (O'Grady, 2017; TIOBE, 2017).
3. La propuesta de diferentes arquitecturas para las aplicaciones dependiendo de los requerimientos del cliente o la organización (Shaw, 1996; Bass, 2003).
4. La infraestructura y las herramientas que le permiten a una aplicación ser altamente escalable y estar disponible en la nube de forma automatizada (Hill, 19990; Bondi, 2000; Artač, 2017).
5. La capacidad de compartir información rápidamente y de disponer de aplicaciones en línea que se caracterizan por la interacción de muchos usuarios. Como se observa, las ventajas actuales que provee la Internet permiten desarrollar de una manera más rápida, sencilla, eficiente y ágil nuevas aplicaciones para diferentes plataformas (BBVAOpen4U, 2016).

Este proyecto está orientado a describir un marco de trabajo (*framework*) como puente para entrar en el desarrollo web usando primero, el enfoque de diseño guiado por el dominio (*Domain driven design - DDD*), que abarca los aspectos de modelado principalmente desde el punto de vista conceptual (Penchikala, 2008; Evans, 2003), y que sirve como referente teórico del ciclo de desarrollo y la arquitectura de software para aplicaciones empresariales, útil en el desarrollo de una aplicación web; y como segundo, el uso de DevOps (*Development & Operations*), el cual entra como una solución cuyo foco principal es automatizar la mayor parte del ciclo de desarrollo, asegurando la calidad y el despliegue de la aplicación (Ebert, 2016).

Teniendo en cuenta las ventajas que ofrece la plataforma web, se observa la necesidad de encontrar una guía para el desarrollo de aplicaciones web, donde se especifique la secuencia para la construcción de una aplicación, con el fin de que los desarrolladores puedan replicar el proceso en una nueva idea. De esta manera, el desarrollador puede beneficiarse de su utilidad de manera rápida y simple, a través de sus tabletas, smartphones o un computador, sin tener la necesidad de instalar un aplicativo. Esta propuesta busca contribuir a satisfacer la necesidad de contar con un marco de trabajo que facilite el desarrollo de las aplicaciones web tomando como premisa que 'cada nueva aplicación desarrollada funcione como un módulo más de internet', basado en un enfoque colaborativo.

1.2. OBJETIVOS

1.2.1. Objetivo General. Diseñar un marco de trabajo que permita desarrollar una aplicación web usando Domain Driven Design y DevOps.

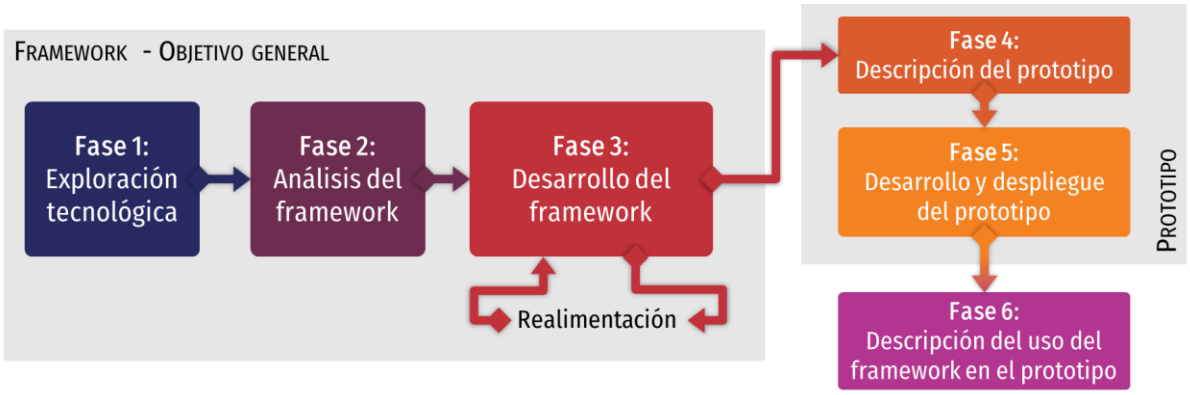
1.2.2. Objetivos Específicos.

1. Explorar sobre la implementación de Domain Driven Design y DevOps en aplicaciones web.
2. Diseñar un marco de trabajo general para el desarrollo de una aplicación web usando Domain Driven Design y DevOps.
3. Aprovisionar una infraestructura tecnológica para el desarrollo de una aplicación web.
4. Validar el marco de trabajo a través del desarrollo y despliegue de una aplicación web prototipo.
5. Describir el uso de la infraestructura tecnológica planteada durante el desarrollo del prototipo.

1.3. METODOLOGÍA

La metodología para el desarrollo del proyecto comprende seis fases, buscando obtener una infraestructura tecnológica y la estrategia de uso del framework, junto con el desarrollo de un prototipo, el cual será desplegado para evidenciar el uso efectivo del framework construido. En la Figura 1 se describe cada una de las fases y sus relaciones.

Figura 1. Metodología.



Fuente: Autor

1.3.1. FASE 1: Exploración tecnológica. Es la primera fase de la metodología, en esta se probarán las tecnologías y herramientas que se van a utilizar dentro del marco de trabajo haciendo una Prueba de Concepto (PoC), con el fin de validar si se usarán o no en el diseño del framework.

Actividades:

- A1.1. Estudiar las herramientas de software para conocer su uso y funcionamiento.

Productos:

- P1.1. Dominio de los conceptos y herramientas utilizados actualmente en el mercado.

1.3.2. FASE 2: Análisis del framework. Se realizará la investigación de los requisitos, necesidades y conceptos de diseño del framework junto al marco de referencia para desarrollo de las aplicaciones web.

Actividades:

- A2.1. Identificar las necesidades conjuntas que requieren los diferentes tipos de aplicaciones web.
- A2.2. Investigar sobre los conceptos descritos en el marco de referencia.
- A2.3. Contextualizar el framework.

Productos:

- P2.1. Requerimientos y conceptos de diseño del framework a realizar.

1.3.3. FASE 3: Desarrollo del framework. En esta fase se realizará la construcción del framework identificando el alcance y la arquitectura para realizar el desarrollo y despliegue de una aplicación web usando DDD y DevOps. También se mejorará recibiendo la realimentación de cada propuesta a realizar (de forma iterativa) hasta que se ajuste completamente a los requerimientos propuestos durante el análisis.

Actividades:

- A3.1. Describir el marco de trabajo en propuestas de forma iterativa hasta obtener una solución final.
- A3.2. Mejorar el marco de trabajo con la realimentación de cada prototipo.

Productos:

- P3.1. Diseño del framework para el desarrollo de una aplicación web.
- P3.2. Infraestructura tecnológica del ambiente de desarrollo e infraestructura del framework para el desarrollo de una aplicación web.

1.3.4. FASE 4: Descripción del prototipo. En esta fase se definirá, documentará y describirá un prototipo de una aplicación web, la cual utilice parcial o en su totalidad el marco de trabajo descrito en la “Fase 3: Desarrollo del framework”.

Actividades:

- A4.1. Describir el MVP del prototipo con las funcionalidades, requerimientos del prototipo, conjunto de herramientas (stack), interfaces de usuario y especificaciones adicionales.
- A4.2. Crear el ambiente de desarrollo e infraestructura para el prototipo con el framework final y las herramientas elegidas.

Productos:

- P4.1. Documentación del MVP con las funcionalidades, requerimientos del prototipo, interfaces de usuario y especificaciones adicionales del prototipo.
- P4.2. Ambiente de desarrollo e infraestructura del prototipo.

1.3.5. FASE 5: Desarrollo y despliegue del prototipo. Durante esta fase se realizará el desarrollo y despliegue del prototipo haciendo uso del framework con la estrategia planteada. Al finalizar, se deberá tener el prototipo de una aplicación web.

Actividades:

- A5.1. Desarrollar el prototipo siguiendo el framework propuesto.
- A5.2. Desplegar el prototipo según los lineamientos del framework.

- A5.3. Extraer datos, capturas, videos o imágenes del proceso de desarrollo y despliegue.

Productos:

- P5.1. Prototipo implementado en el ambiente de desarrollo.
- P5.2. Datos, capturas e imágenes del proceso de desarrollo.

1.3.6. FASE 6: Uso del framework en el prototipo. En esta fase se realizará la descripción del uso del framework construido en un prototipo que utiliza los lineamientos descritos, documentando el proceso secuencial para obtener el prototipo de una aplicación web específica.

Actividades:

- A6.1. Describir el proceso de desarrollo a producción del prototipo siguiendo los pasos del uso del framework.

Productos:

- P6.1. Descripción del proceso de desarrollo a producción del prototipo con el uso del framework.
- P6.2. Anexos de asociados al proceso del uso del framework.

2. ESTADO DEL ARTE

Para analizar el estado del arte respecto al desarrollo de aplicaciones web, es importante identificar los diferentes actores que intervienen en el proceso, la composición interna para su operación entendida como la arquitectura y las diferentes herramientas y frameworks que se encuentran en el mercado.

Antes de entrar en las arquitecturas, hay que considerar el criterio del dueño del software, el desarrollador de software y del usuario final. Desde la visión del desarrollador, la velocidad de desarrollo, el rendimiento, la escalabilidad y las pruebas son claves para optar por un modelo arquitectónico; por su parte del lado del usuario, se observa la usabilidad, la capacidad de respuesta y la vinculación junto al potencial de trabajo fuera de línea para la decisión; y finalmente, para el dueño, prima las características de funcionalidad en el tiempo, el soporte, la seguridad y la adaptabilidad al formato móvil y de escritorio (Mobidev, 2016).

En el desarrollo de las aplicaciones, la arquitectura se convierte en el pilar central en la búsqueda de una solución concreta. En un principio la primera arquitectura que se implementó es la arquitectura clásica de cliente – servidor, donde se tiene una máquina/servidor a la cual se le realiza peticiones a través de un cliente. De ahí, se parte a la evolución de mejorar la comunicación, las posibilidades de interacción y las funcionalidades aplicables a algún contexto en específico, razón por la cual se establecieron arquitecturas más especializadas como MVC, arquitectura de N-capas, arquitectura SOA y la más reciente: la arquitectura de microservicios, la cual viene a ser una especialización de la arquitectura SOA que ofrece mayores capacidades de aislamiento conservando la alta cohesión de la aplicación de la interfaz y con otros sistemas.

Haciendo referencia a las herramientas y frameworks, es común encontrar elementos con un propósito específico, enfocados exclusivamente al Frontend o al Backend. Pueden diferenciarse un primer grupo de frameworks como herramientas y un segundo grupo de frameworks integradores que a su vez fusionan otros frameworks y herramientas. Del primer conjunto se listan:

- Para Backend: Spring Framework, PHP Laravel, Express.js, Django, Ruby on Rails, .NET Framework, entre otros.
- Para Frontend: Angular 4, React/Redux, Vue, Bootstrap, Foundation, entre otros.

Del segundo conjunto, están:

- MEAN (MongoDB, Express.js, Angular, Node.js)
- XAMPP (Apache, MariaDB, PHP, Perl)
- LAMP (Linux, Apache, MySQL, PHP)
- Configuraciones propias para máquinas o máquinas virtuales que integran Base de Datos, Servidor Web, Framework de Frontend y Framework de Backend.

Dadas las condiciones en términos de disponibilidad y función de las herramientas, este proyecto representa un avance con la unión de diferentes frameworks en medio de la dispersión de alternativas para cubrir desarrollos. El objetivo primordial es ofrecer un framework integral, realizado con buenas prácticas y distinguido por el uso de metodologías ágiles y de automatización que contribuye a implementaciones eficientes, de bajo costo, gran calidad y fácil uso.

3. MARCO DE REFERENCIA

El proyecto se basa en un conjunto de ideas y conceptos que usan una terminología específica, la cual es citada a lo largo del documento. No obstante, a continuación, se presenta una noción básica de términos tales como Framework, WebApp, Domain Driven Design y DevOps.

Marco de trabajo (*Framework*), conocido también como marco de trabajo o entorno de trabajo, para este caso se entiende como una estructura que sirve de guía para la organización y desarrollo de software, tiene como fin facilitar el orden a través de buenas prácticas, y mediante las utilidades que ofrece, permitir un trabajo rápido, sencillo y seguro.

Aplicación web (*WebApp*) se define como un conjunto de herramientas alojadas en un servidor web, cuyo uso se da por medio de un navegador. Entre sus ventajas está la practicidad en el acceso, independencia de instaladores y la premisa de tener información actualizada.

DevOps es el acrónimo utilizado para referirse a *Development* (desarrollo) y *Operations* (operaciones). Reúne prácticas orientadas a la automatización de procesos involucrados en el desarrollo de software que conducen a entregas rápidas, de calidad y con menor costo.

Diseño guiado por el domino (DDD) es un enfoque que se dirige a necesidades complejas y que facilita la identificación de la arquitectura del software y el planteamiento del modelado y diseño de la solución.

Una ampliación de los anteriores conceptos y de otros empleados en el libro se encuentran en el Anexo A.

4. DISEÑO Y DESARROLLO DEL FRAMEWORK

En este capítulo se presenta la descripción del framework a realizar y de forma iterativa las diferentes propuestas diseñadas para su construcción a medida que se ve la evolución del framework se van explicando cada uno de sus componentes hasta llegar a la versión final.

4.1. DESCRIPCIÓN Y OBJETIVO DEL FRAMEWORK

El framework desarrollado usando *Domain Driven Design* y *DevOps* es una base de una aplicación web, sirve como base para la creación de nuevas ideas y proyectos evitando hacer de nuevo el diseño, la automatización y la configuración, con el fin de enfocarse en el desarrollo, aportando valor y velocidad en la entrega del producto de software. Corresponde a una infraestructura tecnológica que podrá ser reutilizada para proyectos específicos donde se vea la necesidad de usar tecnologías web, con fines comerciales o de negocio aprovechando las ventajas de alcanzar un alto potencial de usuarios que requieran resolver una necesidad especifica.

Con esta aplicación se pretende crear y conectar los diferentes dominios *(Backend, Frontend* y *Storage & Database*) que la componen. A nivel de la aplicación base, deberá incluir una pantalla de inicio y de perfil de usuario, hacer un CRUD (*Create*, *Read*, *Update & Delete*) de usuarios, una pantalla simple de Login y Registro que utilice el método de seguridad por token JWT ampliamente utilizado en la industria como un estándar, usando las herramientas de DevOps y los lineamientos DDD.

Respecto a la infraestructura que se utilizará en el framework, se debe tener presente que se manejará en un ambiente de desarrollo controlado donde se

añadirá lo necesario en relación con paquetes, librerías y dependencias para que este framework funcione. Desde éste ambiente, se lanzará el DevOps pipeline que se encargará de subir la aplicación web al ambiente de producción en la nube, el ambiente de desarrollo se montará en una máquina virtual, la cual va a ser empaquetada con Vagrant con el fin de tener un ambiente portable y homogéneo para todos los desarrolladores o integrantes del proyecto (ver. Subsección 4.6.6).

Para el ambiente de pruebas se usará una herramienta de integración continúa llamada Travis CI, la cual cuenta con una buena documentación para usarla junto al repositorio central de GitHub. Además, gracias al Student Pack que entrega GitHub, se puede usar esta herramienta a nivel profesional inclusive con repositorios privados de GitHub.

Para el ambiente de producción se aprovecha la nube de Digital Ocean, la cual es una infraestructura para conectar los diferentes servidores con balanceadores de carga, firewall, nombre de dominio, con el fin enlazarlo posteriormente a un DNS para que sea haga pública la aplicación en Internet, y demás componentes, con el fin de que tenga un alcance global a los usuarios.

La elección de estas herramientas no se convierte en una dependencia tecnológica, por el contrario, el diseño del framework se diseña de manera general, con el fin de que distintas tecnologías puedan incluirse en la construcción de los diferentes módulos que componen las aplicaciones.

La toma de decisiones en el diseño de esta solución en cuanto a la arquitectura y desarrollo del framework se basó en la experiencia del desarrollo web en las empresas donde el autor de este proyecto ha trabajado, los estudios realizados en su universidad y cursos externos especializados en el desarrollo de aplicaciones web modernas enfocados en aprender a implementar cada uno de los componentes de una aplicación web.

Las funcionalidades descritas son características comunes a manera general que presentan este tipo de aplicaciones, con el fin de probar e investigar las posibilidades tecnológicas presentes para implementar diversas características.

4.2. CONCEPTOS DE DISEÑO

Para el diseño de un framework robusto, escalable y seguro, se reutilizan algunos de los principios aplicados en arquitectura SOA explicados en el marco de referencia y la integración de los siguientes conceptos de diseño:

- **Escalabilidad horizontal:** Hace referencia a la facilidad de ampliar la infraestructura de producción del framework, agregando nuevas máquinas. Cada una de ellas aumenta proporcionalmente la potencia de procesamiento total del framework (Ziemer, 2002).
- **Costo:** Se relaciona con que el framework no debe requerir servidores costosos de gama alta y licencias comerciales, la adquisición de recursos se realizará según necesidades y alcances durante el proyecto.
- **Abstracción de programación:** Los usuarios del framework pueden enfocarse en implementar lógica de negocios que resuelva sus problemas del mundo real, manejando un aislamiento por dominios, con una alta cohesión y un bajo acoplamiento.
- **Adaptabilidad:** Se puede adaptar para configurar diferentes herramientas bajo la misma arquitectura (Ziemer, 2002), donde permita hacer la integración de un DevOps Pipeline.
- **Compatibilidad:** Al crear una infraestructura de desarrollo se pueda usar y desplegar en diferentes maquinas con diferentes sistemas operativos.

- **Rendimiento en la red:** Tener una latencia ideal en las interacciones a través de la red o canal de comunicación (Ziemer, 2002).

El listado de funcionalidades, contratos de servicio, detalles sobre el diseño del framework, diagramas conceptuales, diagramas de datos, interfaces, entre otros. pueden encontrarse en el Anexo C MVP Framework FWD.

4.3. ¿POR QUÉ SE UTILIZA DEVOPS Y DDD?

DevOps se utilizará para facilitar el paso a producción de la aplicación evitando la manualidad en el proceso de instalación, ajuste y despliegue del producto de software. Ésta es una tarea repetitiva que consume demasiado tiempo, por lo tanto, se ve la viabilidad de utilizar este tipo de tecnologías y cultura.

Por otra parte, DDD representa los lineamientos del modelo usado para estructurar, trabajar y desarrollar una aplicación web, acompañados de metodologías ágiles que aportan en el entendimiento para llegar a una solución, sobre un dominio en específico. Así se traza las características esperadas para poder implementar cualquier tipo de idea de negocio sobre este framework.

Como se observa los conceptos DevOps y DDD son complementarios y comparten muchas características en común que aportan a los lineamientos de diseño del framework.

La prueba de concepto de DevOps con DDD se encuentra en el Anexo C PoC DevOps pipeline y enfoque DDD con un servidor. En éste Anexo se explica la exploración realizada para validar el uso de estas tecnologías y lineamientos.

4.4. PROPUESTA I: MODEL VIEW CONTROLLER (MVC)

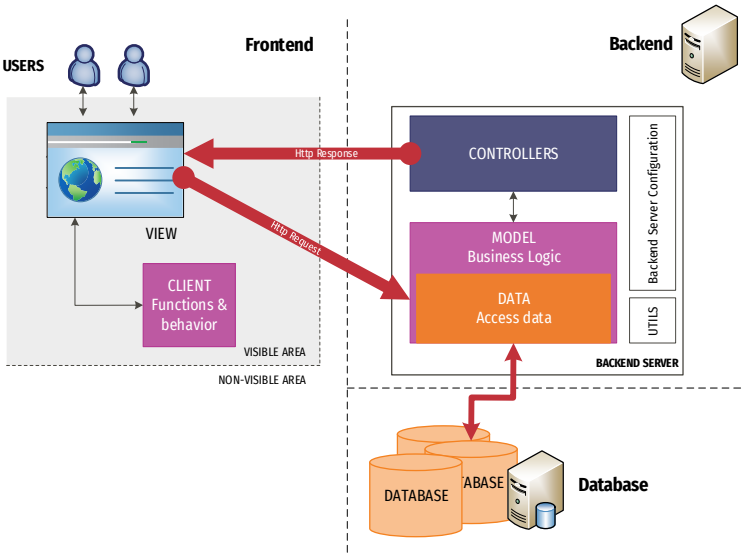
Inicialmente se propuso realizar una aplicación web usando la arquitectura MVC que ofrece Spring Framework entre su conjunto de proyectos. Se logró crear una aplicación web simple, donde se utiliza i) una organización en el proyecto de Controladores como aquellos que envían los datos y la respuesta de solicitudes a la interfaz, ii) el Modelo que tiene integrada la lógica de negocio y iii) el acceso a la base de datos, módulo donde la Vista realiza solicitudes y peticiones. Esto tiene la ventaja de que todos los recursos se encuentran centralizados en un servidor, tanto los estáticos como la base de datos (la cual también se probó de forma externa).

4.4.1. Arquitectura de software PROPUESTA I. La arquitectura para la Propuesta I mostrada en la Figura 2, utiliza un solo servidor web (Backend) y tiene todo centralizado allí. Sin embargo, se hizo la prueba conectando un servidor de base de datos externo (Database). Esta arquitectura es la misma tanto en desarrollo como en producción.

Integra las siguientes partes:

- **Model** es donde se encuentra almacenada la lógica de negocio junto al acceso a los datos **Data**. Es modulo donde la **View** realiza las solicitudes al servidor.
- **View** es la interfaz de usuario.
- **Controllers** son aquellos que apoyan a la interfaz en las solicitudes de información.
- **Utils** para utilidades comunes de la aplicación.
- **Backend Server Configuration** que es para todo lo relacionado con la gestión del servidor web.

Figura 2. Arquitectura de software PROPUESTA I.



Fuente: Autor

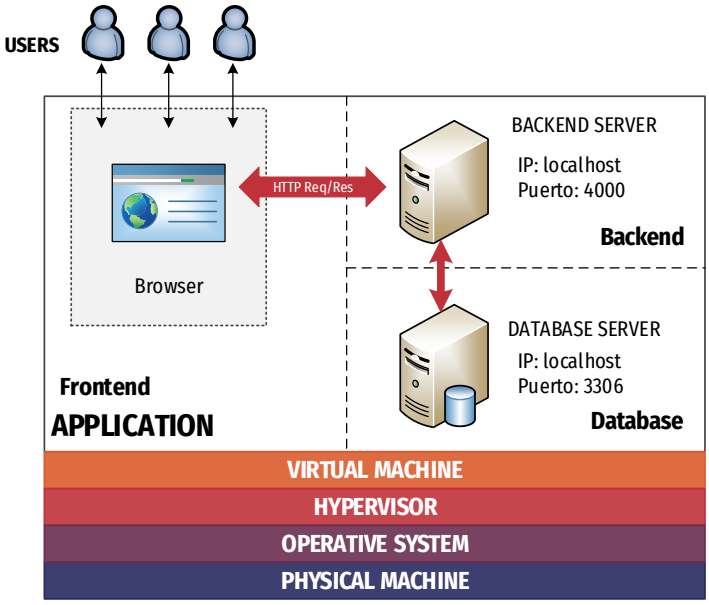
4.4.2. Arquitectura de infraestructura PROPUESTA I.

Ambiente desarrollo

Para el ambiente de desarrollo (ver Figura 3) se dispuso de una máquina virtual (guest) con las siguientes características:

- Sistema operativo Ubuntu 14.04 LTS de 64bits, por el soporte que tiene esta versión.
- Memoria RAM de 2GB.
- Se instalaron las dependencias y librerías de Spring Framework como Java JRE 8 y Java JDK 8, Linux Essentials y paquetes básicos para el funcionamiento correcto del ambiente.
- Se instaló también el motor de Bases de datos MySQL 5.7.
- La máquina virtual se ejecuta sobre el hipervisor VirutalBox en una máquina host con sistema Windows 10 Pro de 8GB de RAM y 1TB de disco duro.

Figura 3. Arquitectura de infraestructura PROPUESTA I (Ambiente desarrollo).



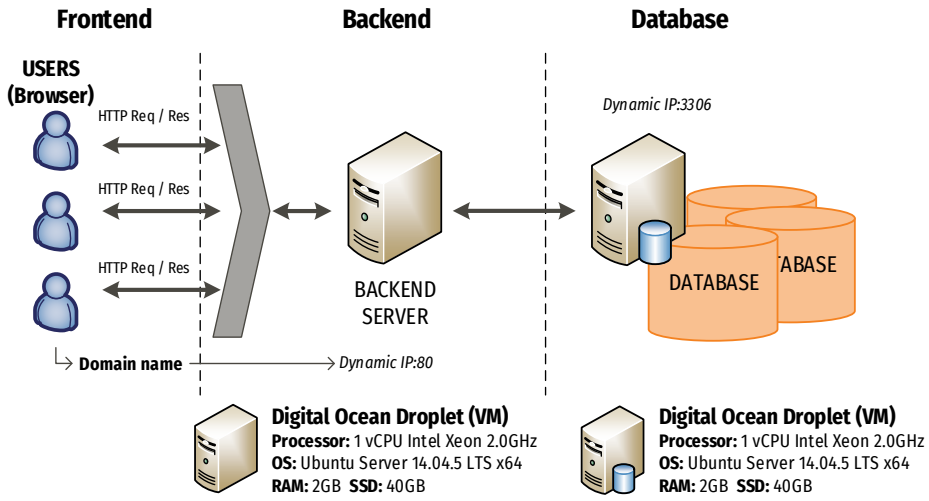
Fuente: Autor

Ambiente producción.

Para el diseño del ambiente de producción se utilizaron dos máquinas virtuales conectadas entre sí, ver Figura 4. Una corresponde al **Backend Server** que recibirá todas la solicitudes e interacciones de los usuarios y la otra al **Database Server** que contendrá los datos de la aplicación.

El **Backend Server** es aquel que tiene por dentro toda la arquitectura de software planteada y donde se gestiona la aplicación web, tiene un **nombre de dominio** (*Domain name*) para facilitar el acceso de los usuarios, y cuenta con los recursos suficientes para soportar múltiples interacciones. Para esta primera propuesta no se alcanzaron a incluir componentes adicionales de seguridad en la automatización de DevOps, como lo son la integración de una red privada con IP's privadas, Firewalls y reglas de seguridad. Finalmente, los usuarios podrán hacer uso de la aplicación a través de un navegador web desde su teléfono, tableta o computador personal gracias a que se desarrolla con diseño responsivo para diferentes pantallas.

Figura 4. Arquitectura de infraestructura PROPUESTA I (Ambiente producción).



Fuente: Autor

La arquitectura tiene varias debilidades, entre éstas el hecho de que los controladores están muy atados a las funcionalidades de la interfaz. Por este motivo, no son fáciles de reutilizar y se requiere hacer una implementación detallada por cada funcionalidad que se va a poner en la interfaz o cuando se desea obtener los datos almacenados en el servidor. Además, debido a que tiene todo integrado en un solo lugar, si se desea ampliar la capacidad y el rendimiento dependiendo de las necesidades, no es tan fácil ya que la arquitectura de software no cumple el concepto de escalabilidad horizontal.

Por lo tanto, para cumplir con el concepto de escalabilidad se requerirían de servidores de alta gama que ayuden en la gestión de las solicitudes de los usuarios, lo cual conduce a soluciones demasiado costosas, incumpliendo el otro concepto de diseño propuesto. Adicionalmente, como recomendación es mejor siempre tener un servidor de estáticos externos al igual que la base de datos. De la misma manera, es importante incluir elementos de red como proxies con el fin de ocultar la ubicación del servidor de Backend y evitar ataques de seguridad informáticos. En conclusión, por las razones anteriores se decidió plantear una segunda propuesta usando una arquitectura SOA buscando mejorar la escalabilidad y la flexibilidad.

4.5. PROPUESTA II: SERVICE ORIENTED ARCHITECTURE (SOA)

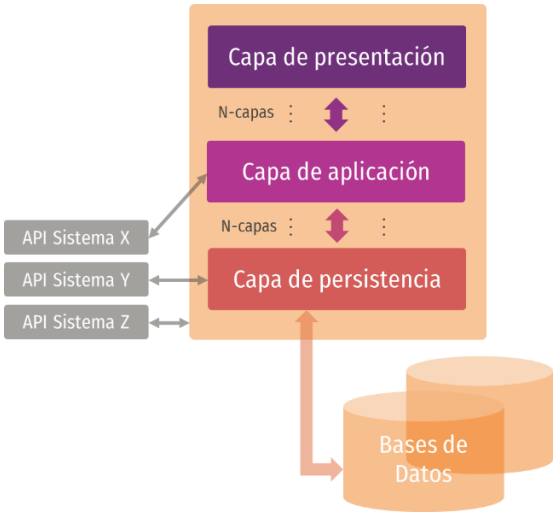
Tomando como base los resultados que arrojó la Propuesta I, se rediseño la arquitectura de software para mejorar la escalabilidad de la aplicación implementando una arquitectura SOA, la cual tiene como interfaz una capa de presentación o API (*Application Programing Interface*), la cual responde con datos o archivos binarios a través de *endpoints* (como por ejemplo: 'http://<server route>/picture', 'http://<server route>/'), y se establece un contrato de servicio el cual va a indicar como usar ese *endpoint* y que es lo que nos va a responder a través del mismo.

Esta API está soportada por la capa de aplicación, que es la encargada de realizar la lógica de negocio, ya sea ejecutar un algoritmo, ejecutar una ETL (*Extract, Transform and Load*), usar sistemas externos a través de APIs, entre otros. Es el lugar donde se realiza la funcionalidad que debe tener la aplicación todo a nivel de los datos y archivos. Las funcionalidades de la interfaz se trasladan directamente a la construcción de la interfaz, aislando módulos para un bajo acoplamiento con una alta cohesión entre ellos.

4.5.1. Empezando por el Backend. Con el fin de poner a prueba la Propuesta II, se propuso emplear una prueba de concepto de la arquitectura Flux para el Frontend (React con Javascript) y la arquitectura de N-capas para el Backend (Spring Framework con Java 8).

Entonces aparece la pregunta ¿Cómo construir un Backend usando Spring y su arquitectura de N-capas?

Figura 5. Arquitectura N-capas Backend.



Fuente: Autor

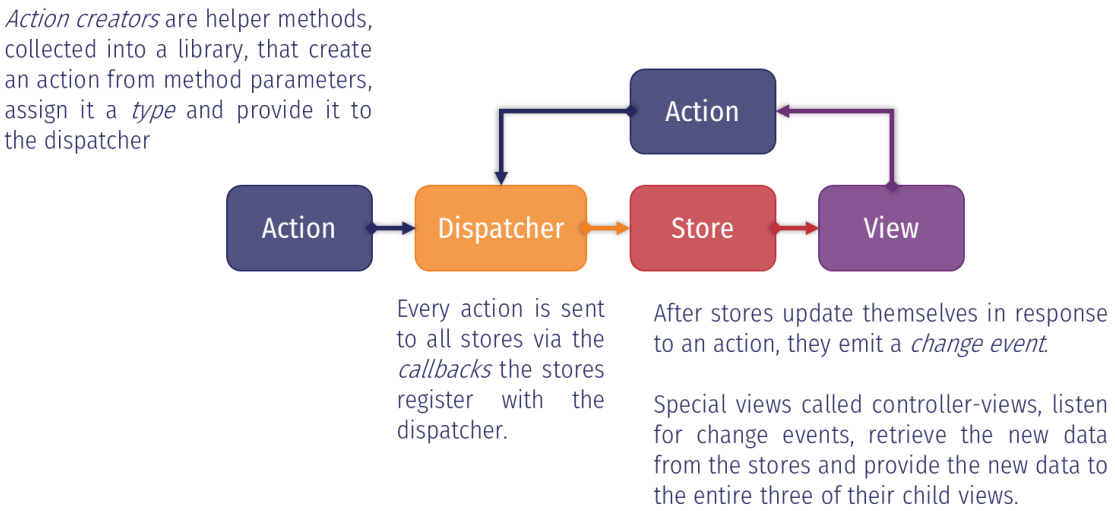
Buscando contestar la pregunta anterior se presenta la arquitectura de la Figura 5, la cual se dispone iniciando de abajo hacia arriba la construcción de cada una de las diferentes capas, empezando por la capa de persistencia (o Datos, en inglés, *Data & Persistence*), la cual se conecta mediante un conector JDBC a la base de datos. Para este experimento se utilizó el motor de Base de datos MySQL 5.7, en donde se construyó un modelo de datos simple en SQL con dos tablas (Usuarios y Roles), donde se obtuvo como resultado la conexión al servidor de Apache Tomcat (donde Spring encapsula la aplicación web) al servidor de Base de datos MySQL, realizando un mapeo objetos-relacional (*Object Relational Mapping – ORM*) con una librería llamada Hibernate, la cual integra el Spring Framework para facilitar el control de los datos como objetos de Java.

Posteriormente, se construyó la capa de aplicaciones (o de Servicios o Lógica de Negocio, en ingles *Business Application*), la cual, según el diseño de la arquitectura, es la única que tiene acceso a la capa de Datos y Controladores. Por la anterior razón, en ésta se ubica la lógica de negocio que va a tener la aplicación. En esta capa se realizan las operaciones, validaciones y algoritmos que integra la aplicación objetivo a construir.

Finalmente, se tiene la capa de Presentación (o Controladores, en ingles *Controllers*) la cual se encarga de la conexión con el cliente y de recibir o enviar los datos a través de *'endpoints'*, también contiene otro nivel de verificación de datos que serán enviados a la capa de aplicaciones para su respectivo tratamiento. Al final de esta implementación de software se tiene como resultado una API REST que permite hacer un CRUD de los datos y otras funcionalidades, así la API REST funciona como el punto de entrada a la aplicación.

4.5.2. Luego por el Frontend. Luego se construyó una página web que consume la API REST diseñada, con el objetivo de recoger y enviar datos que van a ser visualizados por el usuario. La arquitectura Flux, mostrada en la Figura 6, ayuda a organizar el estado de las diferentes vistas que visualizará el usuario final, obteniendo como resultado que Flux controla el estado de la aplicación; sin embargo, éste se puede mejorar añadiendo componentes que organicen de manera reusable los accesos al API.

Figura 6. Arquitectura Flux Frontend.



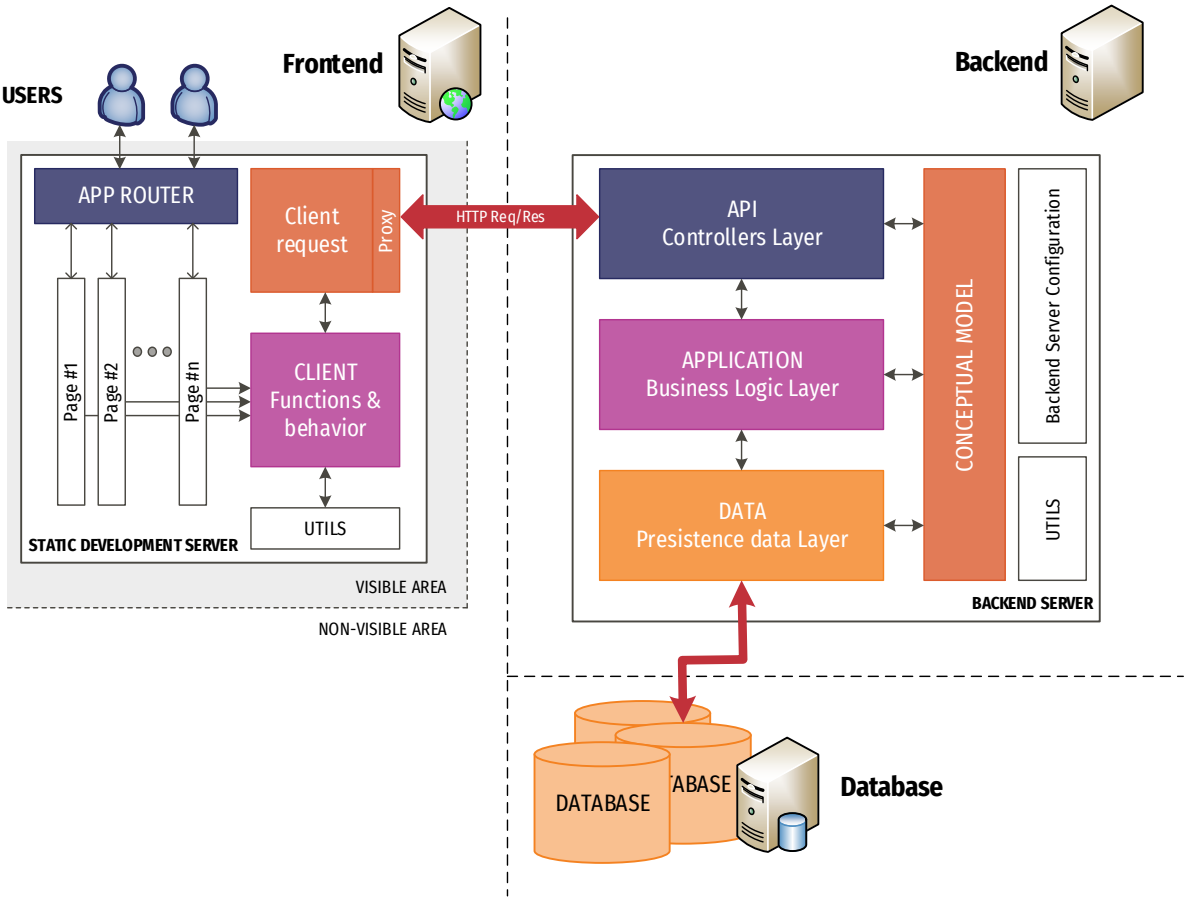
Fuente: https://facebook.github.io/flux/docs/in-depth-overview.html#structure-and-data-flow

4.5.3. Arquitectura de software PROPUESTA II.

Ambiente desarrollo.

Para la arquitectura de software de la Propuesta II presentada en la Figura 7, se usaron las estructuras y esquemas encontrados durante las pruebas de concepto. Igualmente, se vio la necesidad de simplificar la etapa para el diseño de las interfaces, con el fin que los diseñadores puedan ir viendo las modificaciones realizadas en su entorno de desarrollo, razón por la que en esta arquitectura se incluye un servidor de estáticos embebido.

Figura 7. Arquitectura de software PROPUESTA II (Ambiente desarrollo).



Fuente: Autor

La arquitectura integra las siguientes partes:

- En la **zona visible** se tiene el *Static Development Server*, donde se encuentra el código público que solo sirve para la interacción del usuario con la aplicación y no posee datos sensibles de los usuarios o datos que son privados de la aplicación. Este código es el que renderiza el navegador para visualizar las interfaces. Esto no afecta la usabilidad de la aplicación, son niveles de seguridad al acceso de los datos del servidor.
- **Frontend App Router** tiene como funcionalidad redirigir las peticiones a las diferentes páginas de la aplicación, tiene un nivel de seguridad simple que permite bloquear ciertos sectores no públicos de la aplicación.
- **Frontend Page** son las interfaces e incluyen las interacciones que puede realizar el Usuario con la aplicación.
- **Frontend Client** hace referencia a la función que coordina y distribuye las tareas que se gestionan el Frontend, donde se crean las 'Action' para que los datos nuevos que llegan del Frontend o Backend Server y las interacciones del usuario con la interfaz, modifiquen el State de la aplicación. Así mismo, este sirve para crear funciones locales de la aplicación que no requieran el uso del Frontend o Backend Server, pero tiene la característica de que este código es público, por lo tanto, puede ser modificado.
- **Frontend Client Request y Proxy** realiza las peticiones al Frontend o Backend Server a los diferentes servicios que soportan la aplicación. Y con el proxy enmascarar la URL del servidor de Backend.
- **Frontend Utils** se define como las funciones genéricas reutilizables que pueden servir al Client o a cualquier interacción de las páginas de la aplicación.
- **Backend Server Conceptual Model** es donde se estructuran los objetos principales que intervienen en la aplicación. Son la base principal o el núcleo para el funcionamiento del Backend y son utilizados por las demás capas de la arquitectura de N-Capas.

- **Backend Server API** es el canal de comunicación e interacción con el núcleo de la aplicación, y por donde se va a consultar, modificar, actualizar, y eliminar datos inclusive en tiempo real (con sockets), en batch, u offline.
- **Backend Server Servicios** ejecuta la lógica de negocio necesaria de para que la aplicación funcione (Por ejemplo, puede ser el núcleo de contabilidad en una aplicación de Tesorería, o manejar la gestión de los datos en una red social, etc…). Esta capa es accedida únicamente por la capa de API.
- **Backend Server Datos** se encarga de extraer los datos de la base de datos y traerlos en el formato deseado a la capa de servicios (por ejemplo, si es una aplicación desarrollada en Java, esta capa realizará las consultas a la base de datos y mapeará los datos planos a objetos de Java, que son más fáciles de usar, en la capa de servicio). Esta capa es utilizada y llamada solo desde la capa de servicios.
- **Backend Server Configuration**, son las configuraciones parámetros, endpoints del API, los servicios, url y puertos donde va a operar el servidor de Frontend, entre otras configuraciones.
- **Backend Server Utils** se define como las funciones genéricas reutilizables que pueden servir a los servicios del Backend o a cualquier otra funcionalidad que las requiera.
- **Database** para la Base de datos es un servidor de Base de Datos aparte de los demás dominios con su motor de Base de Datos especificado en el stack, que contiene datos de prueba mientras se está en etapa de desarrollo.

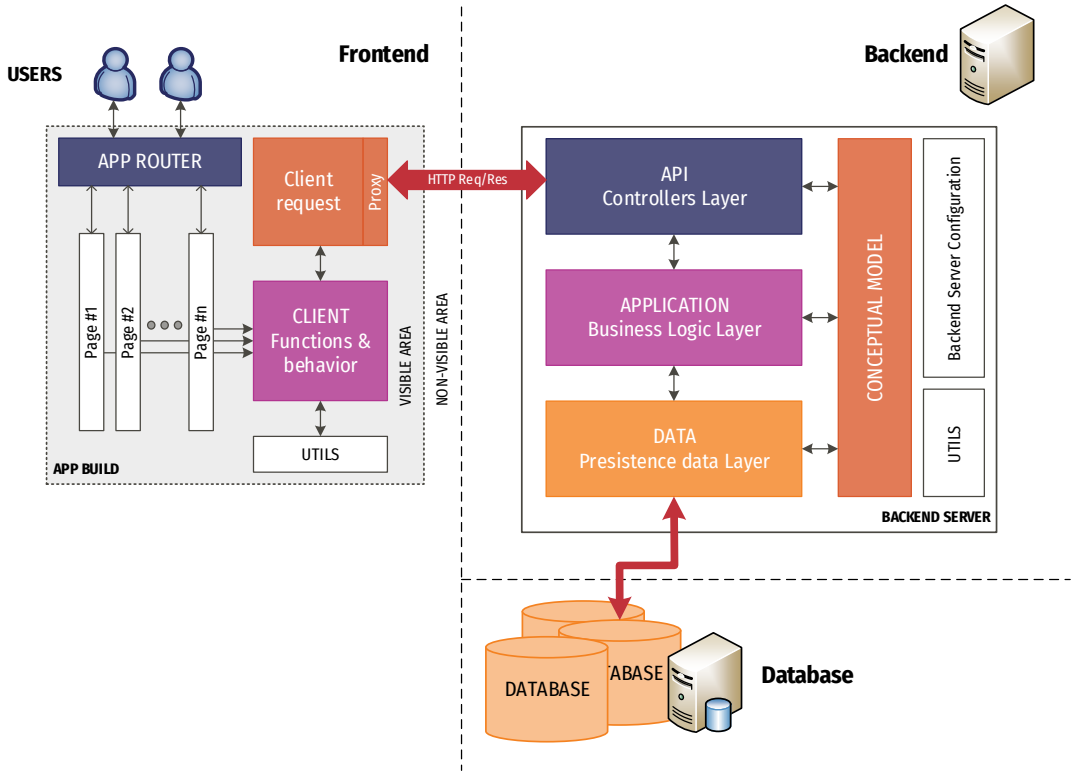
Ambiente producción.

Cuando se está en producción se utiliza los mismos componentes del ambiente de desarrollo descrito anteriormente. Sin embargo, este ambiente no requiere del *Static Development Server*, provocando un cambio en el diseño ya que ahora se crea un solo ejecutable llamado **App Build,** el cual tiene las características que se describen a continuación:

- **Unificado** En lugar de tener muchos archivos todo se consolida en uno solo
- **Minificado (*Minified*) y Comprimido** Se comprime el código hasta tener una sola línea que es la que se ejecuta
- **Desfigurado (*Uglyfied*)** Consiste en cambiar los nombres a los métodos y funciones de la aplicación con el fin de que otro desarrollador no entienda el código realizado, a simple vista.

Este ejecutable es enviado desde el servidor de *Backend* o un *Content Delivery Network*, y es accesible a través de la URL de la aplicación web, convirtiéndose en el punto de entrada para los usuarios que acceden a la aplicación. Se tiene presente que este ejecutable no contiene código sensible de la aplicación (es decir, código que solo debe ser visible por el dueño de la aplicación), el cual contiene las interacciones, las interfaces y la gestión del estado en el cliente, ver Figura 8.

Figura 8. Arquitectura de software PROPUESTA II (Ambiente producción).



Fuente: Autor

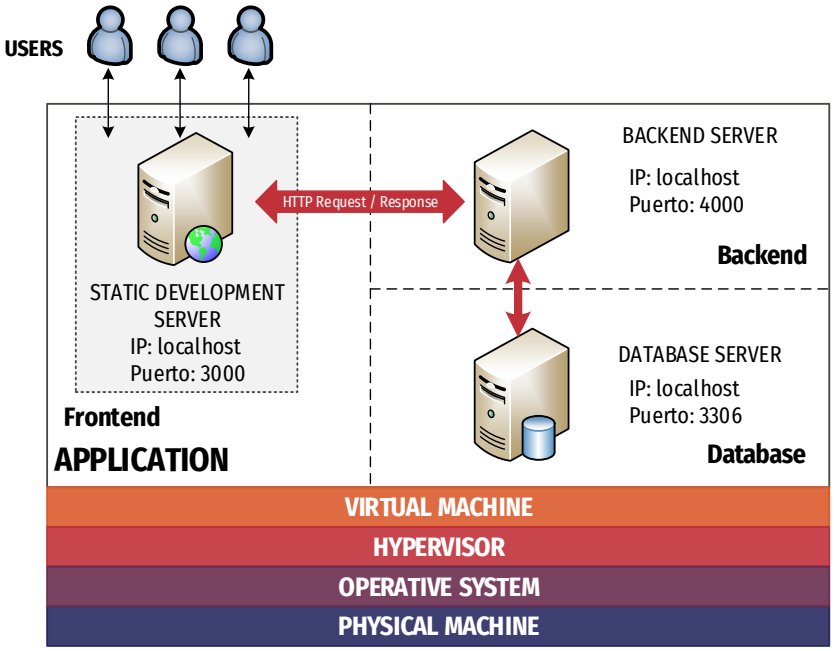
4.5.4. Arquitectura de infraestructura PROPUESTA II.

Ambiente desarrollo.

Para el ambiente de desarrollo descrito en la Figura 9 se dispuso de una máquina virtual (guest) con las mismas características de la Propuesta I, a diferencia:

- Se instalaron las dependencias y librerías de Node.js 8 con la librería React (create-react-app) y el Webpack Development Server.

Figura 9. Arquitectura de infraestructura PROPUESTA II (Ambiente desarrollo).



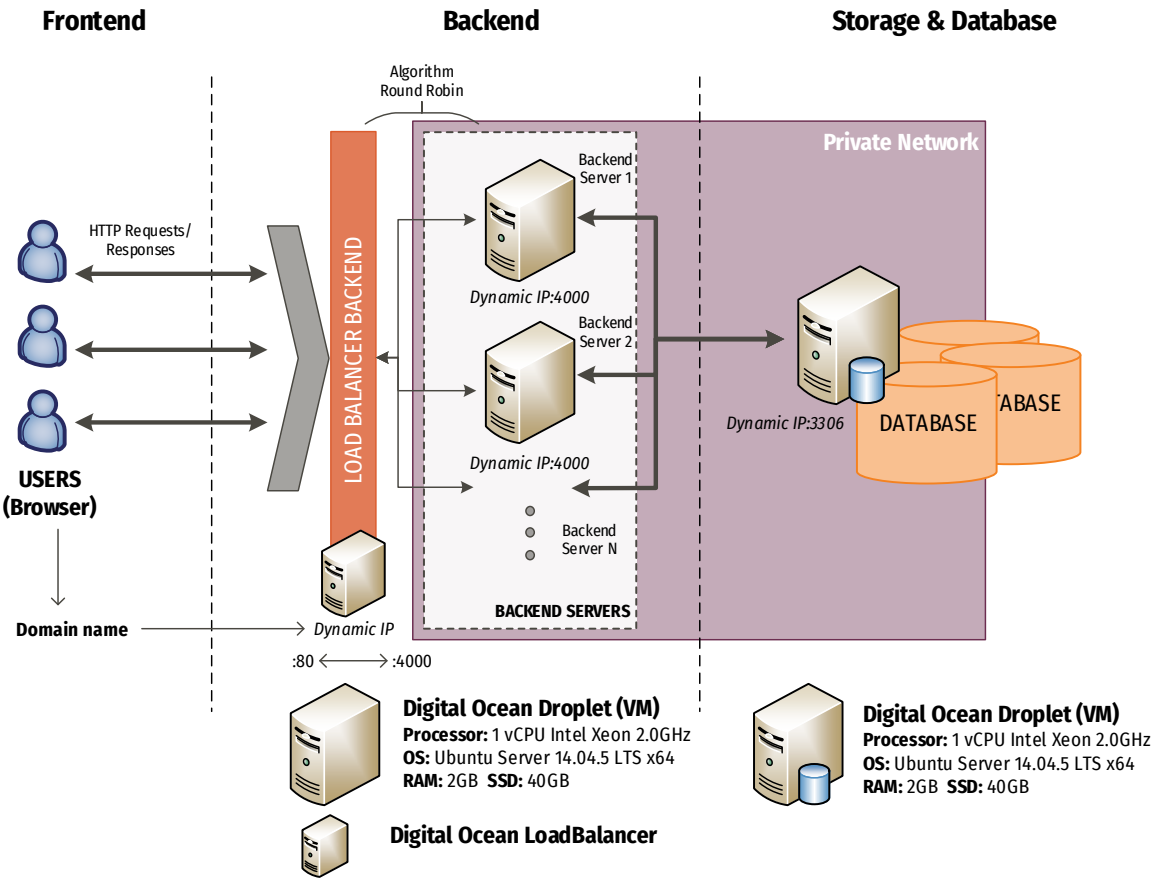
Fuente: Autor

Ambiente producción.

Para esta propuesta, se utilizó una red privada donde se tienen todos los servidores Backend y la base de datos, por seguridad del sistema, dejando como único punto

de entrada un balanceador de cargas el cual tiene un nombre de dominio (*Domain name*) para el fácil acceso de los usuarios, que distribuye el tráfico y las peticiones de los usuarios entre los diferentes servidores Backend. Se disponen 2 servidores Backend al balanceador de cargas, para probar la arquitectura con la aplicación.

Figura 10. Arquitectura de infraestructura PROPUESTA II (Ambiente prod.).



Fuente: Autor

Debido a que el diseño planteado deja toda la integridad de los datos en la base de datos como un servidor aparte, permite que se tengan múltiples servidores de Backend en paralelo atendiendo a los usuarios, sin causar problemas a la respuesta del sistema y sin generar inconsistencias. En esta propuesta no se alcanzaron a incluir componentes adicionales de seguridad en el DevOps pipeline, como Firewalls, repositorios privados y reglas de seguridad en la nube, ver Figura 10.

4.6. PROPUESTA FINAL: FRAMEWORK FOR WEB DEVELOPMENT (FWD)

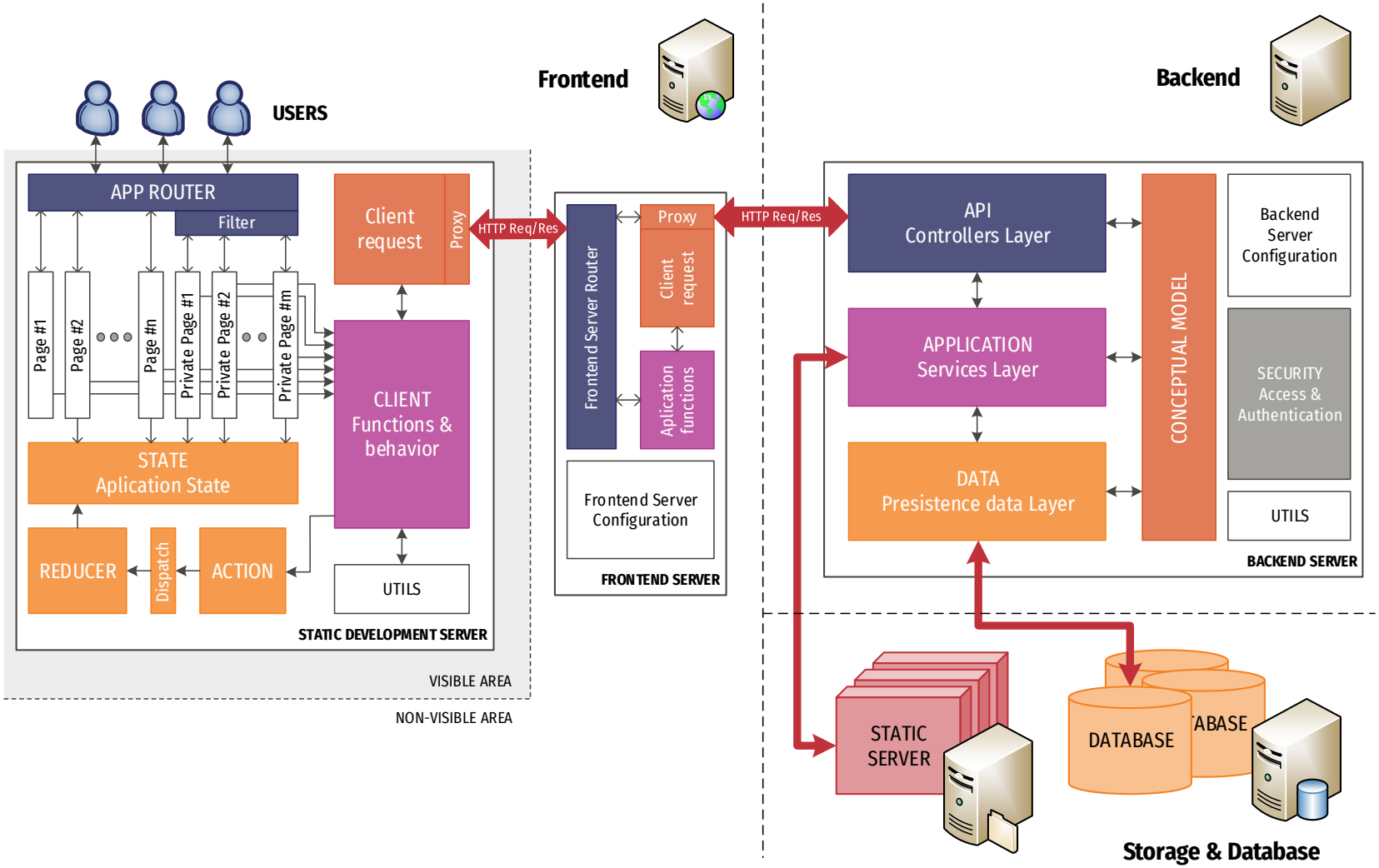
Con el objetivo de simplificar el nombre del framework se usará la sigla FWD para referirse al framework final que se va a utilizar en el prototipo. Durante la revisión del estado del arte se determinó la posibilidad de usar una arquitectura de microservicios, pero se opta por la arquitectura SOA porque es un punto en común en los diferentes frameworks para la construcción de Backend. De esta manera, se aprovechan las ventajas que puede ofrecer cada uno para que sea compatible con el diseño realizado, haciendo posible la implementación de una aplicación web que se ajusta a las necesidades del negocio con diferentes tecnologías. No obstante, se mantiene la posibilidad de usar la tecnología de microservicios para nuevos frameworks o proyectos específicos que la requieran.

4.6.1. Arquitectura de software FWD (Ambiente desarrollo). En la arquitectura de software de la propuesta FINAL mostrada en la Figura 11, se perfeccionaron componentes de la anterior propuesta, añadiendo módulos que mejoran la seguridad de la aplicación y el control de los datos. Se vio la necesidad de cambiar la arquitectura Flux por MVU (Model-View Update), la cual permite organizar el código y controlar mejor el estado de la aplicación en Frontend haciendo que la conexión con el API REST fuera controlada y reutilizable. Gracias a la experimentación con la Propuesta II se vio la necesidad de que fueran incluidos otros módulos en el diseño, integrando en la arquitectura las siguientes partes:

- **Frontend Client Request y Proxy** Se modifica la funcionalidad para enmascarar la URL del Frontend Server, protegiendo que no sea visible este dato desde el cliente en un navegador.
- **Frontend State, Reducer, Dispatch y Action** conforman la arquitectura Flux, aquí modificada usando MVU, la cual fue finalmente aplicada para la gestión del estado de la aplicación en el Frontend.

- **Frontend Server**, dedicado a manejar, controlar y redirigir el tráfico de peticiones solicitado por el usuario final, incluso tiene dentro de sus funciones la de realizar la lógica de negocio que se necesite de la aplicación ya que este servidor se encuentra en una zona protegida no visible para el usuario. Por lo tanto, los usuarios no tienen acceso al código o lógica de negocio puesta en esta zona, permitiendo la creación servicios que soportan las interacciones del cliente, ya sea utilizando o no los servicios del Backend. Esta capa generalmente se usa para que todo el tráfico, que va a recibir el Backend Server, solo provenga de un servidor autorizado en el caso que se tengan peticiones de cualquier usuario no autenticado.
- **Frontend Server Router** recibe las peticiones generadas por el cliente o el servidor de estáticos y re direcciona las peticiones a los servicios que ofrece el mismo Frontend Server o el Backend Server. Adicionalmente, permite re direccionar los sockets que ofrezca el servidor de Backend.
- **Frontend Server Configuration** son las configuraciones parámetros, endpoints del API, los servicios, url y puertos donde va a operar el servidor de Frontend, entre otras configuraciones.
- **Frontend Server Aplication functions** son funciones con lógica de negocio de la aplicación de acceso público, donde se puede aprovechar ubicar funcionalidades que no requieran de un ‘login’ de usuario, y que responden a través de endpoints configurados como una capa Servicios en el Backend.
- **Frontend Server Client Request y Proxy** son todas las peticiones que puede realizar el Client en el Frontend desplegado en el cliente. Es el que llama a los servicios que provee el Frontend Server, el cual internamente redireccionará la petición, en caso de que se necesite realizar peticiones al Backend Server (y de forma protegida con un proxy), es una función de re direccionamiento realizada por el Frontend Server Router.

Figura 11. Arquitectura de software FWD (Ambiente desarrollo).



Fuente: Autor

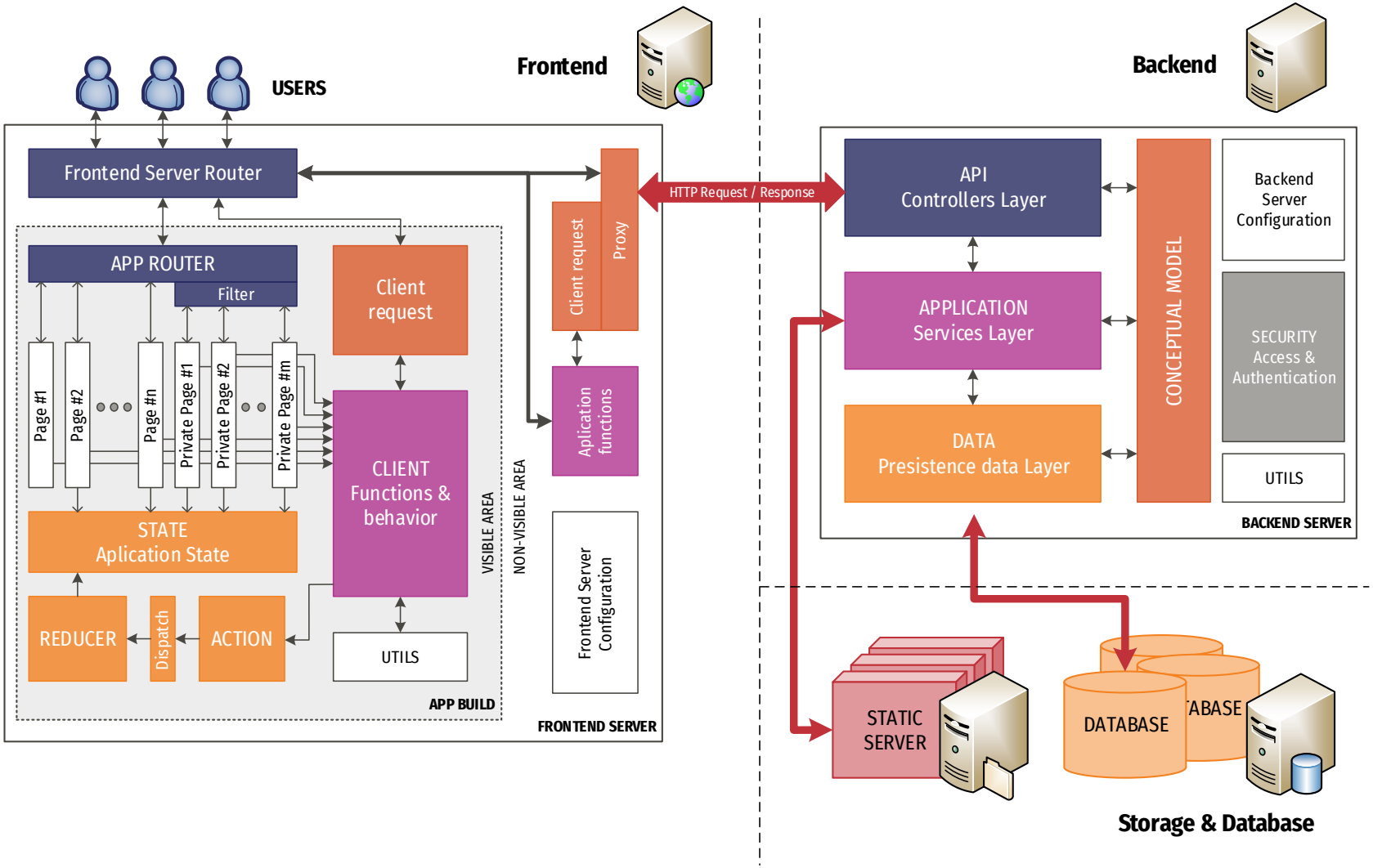
Backend Server Security Access & Authetication, es el encargado de la seguridad en la gestión de control de acceso a los usuarios en la aplicación. Este módulo se encarga de validar las credenciales de un usuario de la aplicación, usando un servicio de la aplicación que accederá a la base de datos y validará las credenciales. Si son válidas generará un token único de acceso, el cual proveerá al usuario una llave para hacer uso de todos los servicios que provee la aplicación. Además, se encarga de revisar a que a servicios tienen acceso los usuarios, mediante la validación del rol que tenga cada usuario. En caso de no ser válidos, el módulo bloqueará el acceso y el usuario no podrá hacer uso de los servicios que provea la aplicación. Así se controla quienes acceden a la aplicación y a qué tipo de servicio.

- **Storage** Para el servidor de estáticos, es un espacio reservado en la nube para guardar archivos mientras se realiza el desarrollo. Se puede utilizar también como *Content Delivery Network*.

4.6.2. Arquitectura de software FWD (Ambiente producción). En la arquitectura de software en producción, mostrada en la Figura 12, el Frontend Server tienen la función de manejar, controlar y redirigir las peticiones del usuario, tanto a las funciones internas del servidor como a las que debe enviar al Backend Server.

Además, en la zona visible para el usuario, tiene el App Build que será enviado al cliente. Así como sucede en la Propuesta II, el Static Server desaparece luego de generar el App Build, y queda solo el ejecutable optimizado dentro del Frontend Server.

Figura 12. Arquitectura de software FWD (Ambiente producción).



Fuente: Autor

Cabe aclarar, que el Backend Server funcionará de igual manera en ambiente productivo como en ambiente de desarrollo, salvo que éste apuntará a los servidores ambiente productivo (servidores como Base de datos y de estáticos). La Base de datos estará dispuesta en ambiente productivo optimizada para tal fin, lista para ser usada e inicializada para la ejecución de la aplicación. Igualmente, se contará con un servidor de estáticos independiente, donde se guardarán los archivos que se utilicen de la aplicación en ambiente productivo.

4.6.3. Arquitectura de infraestructura FWD (Ambiente desarrollo). Para el ambiente de desarrollo se dispuso de una máquina virtual (guest) con las mismas características de la Propuesta II (ver Figura 13), a diferencia de esta el ambiente de la propuesta FINAL:

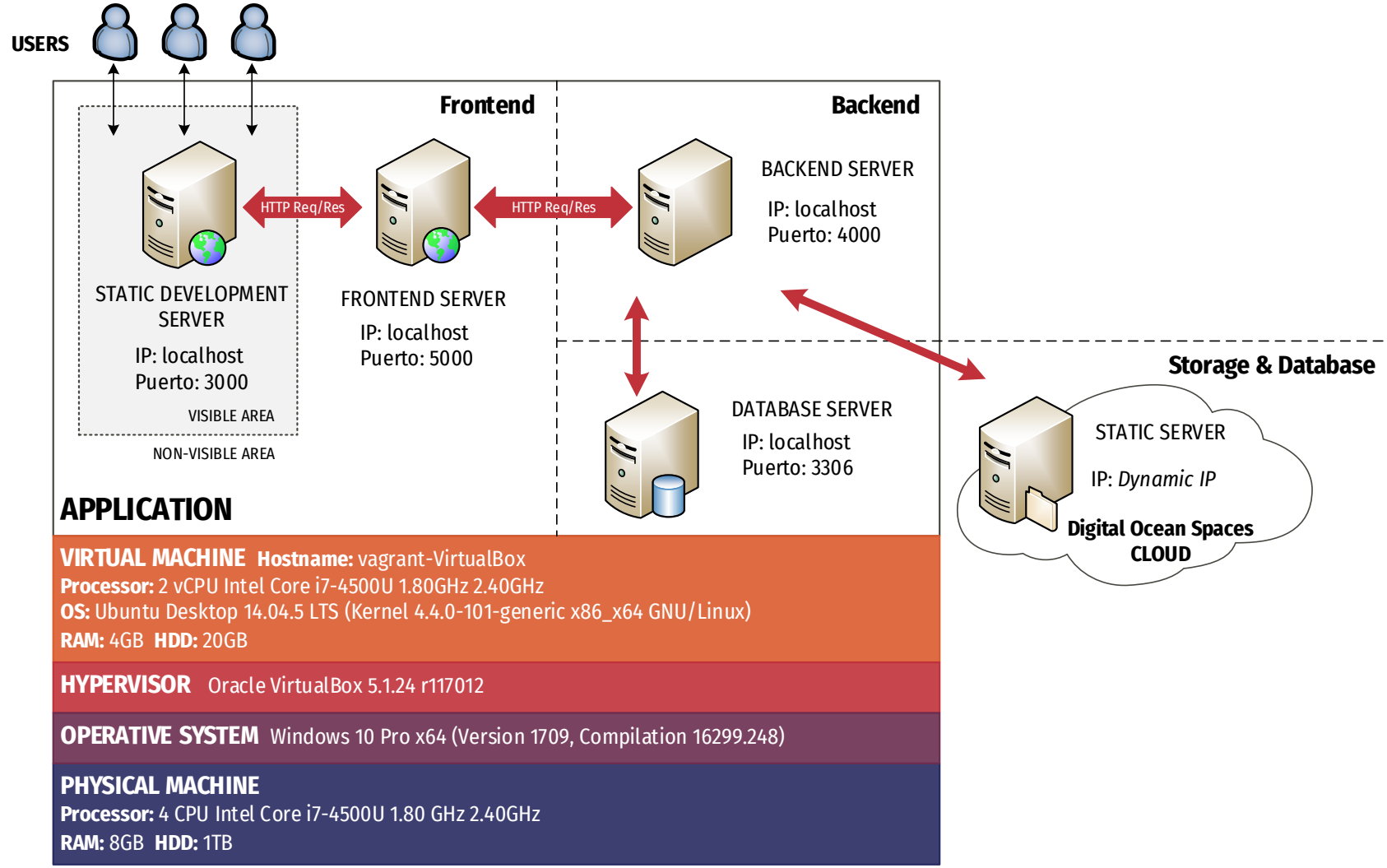
- Incluye un servidor en Node.js 8 con la librería Express para la creación del Frontend Server.
- Se conecta al ambiente de desarrollo el ambiente de desarrollo el servidor de estáticos de Digital Ocean Spaces (el cual es un Bucket de Amazon S3), para guardar archivos tales como imágenes, documentos, entre otros.

4.6.4. Arquitectura de infraestructura FWD (Ambiente producción). En la Figura 14 se presenta el ambiente de producción, el cual reutiliza la arquitectura planteada en la Propuesta II, incluyendo adicionalmente los siguiente componentes y características:

- Firewalls en cada conexión con cada servidor de manera automática. Ahora todos los servidores están protegidos contra ataques en puertos diferentes, a los que maneja la aplicación.

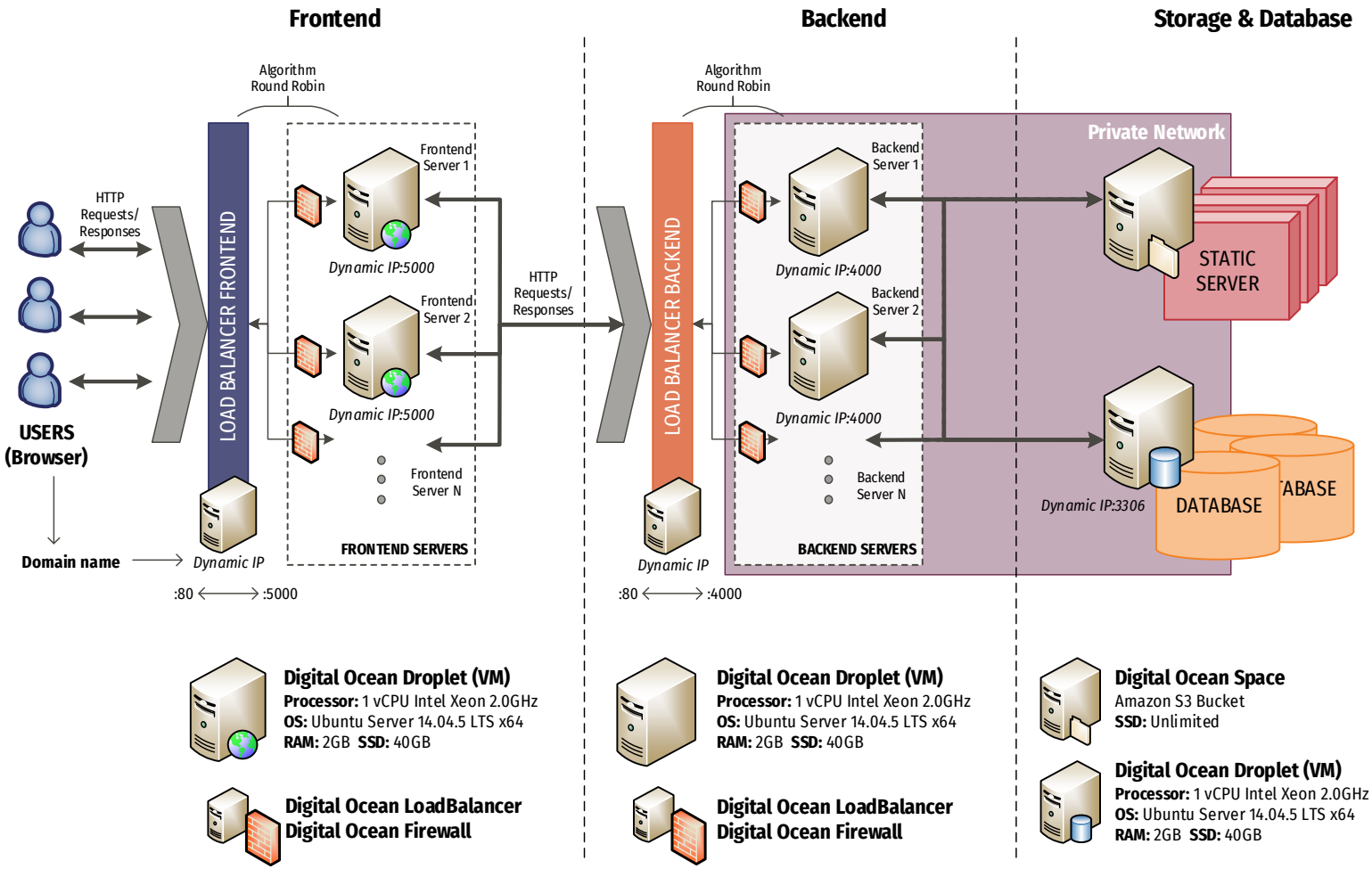
- Los balanceadores de carga tienen un mapeo de puertos, los cuales son los únicos que tienen conexión mejorando la seguridad del sistema. Adicionalmente, estos balanceadores permiten una conexión por máquina evitando ataques de carga al sistema.
- Máquinas en red privada, para impedir que se conecten usuarios externos vía SSH a los servidores. También, se añade la conexión a Digital Ocean Spaces como servidor de estáticos para los servicios de carga y eliminación de archivos.
- Se añade un nuevo balanceador de cargas para los servidores en el Frontend, que actuará como el Frontend Server, todos los servidores para realizar peticiones al Backend apuntan al balanceador de cargas del Backend.
- El hecho de aislar los dos dominios, permite que el Backend sea reutilizable para cualquier Frontend distinto al construido (aplicación web en un navegador de PC o Móvil). Por ejemplo, se puede realizar la creación de una aplicación móvil nativa que también sea un Frontend, el cual consume el Backend construido, abriendo la posibilidad de tener ahora una aplicación multiplaforma. Así mismo, es extensible a otras plataformas como Smart TV, Consolas, Wereables, entre otros.
- La arquitectura tipo granja que se propone tanto para Frontend como para el Backend permite escalar el framework para soportar más usuarios aumentando la cantidad de máquinas, cuando se requiera.
- Se modifica el punto de acceso de la aplicación, colocándole el nombre de dominio al balanceador de cargas en el Frontend para el acceso de los usuarios a la aplicación.

Figura 13. Arquitectura de infraestructura FWD (Ambiente desarrollo).



Fuente: Autor

Figura 14. Arquitectura de infraestructura FWD (Ambiente producción).



Fuente: Autor

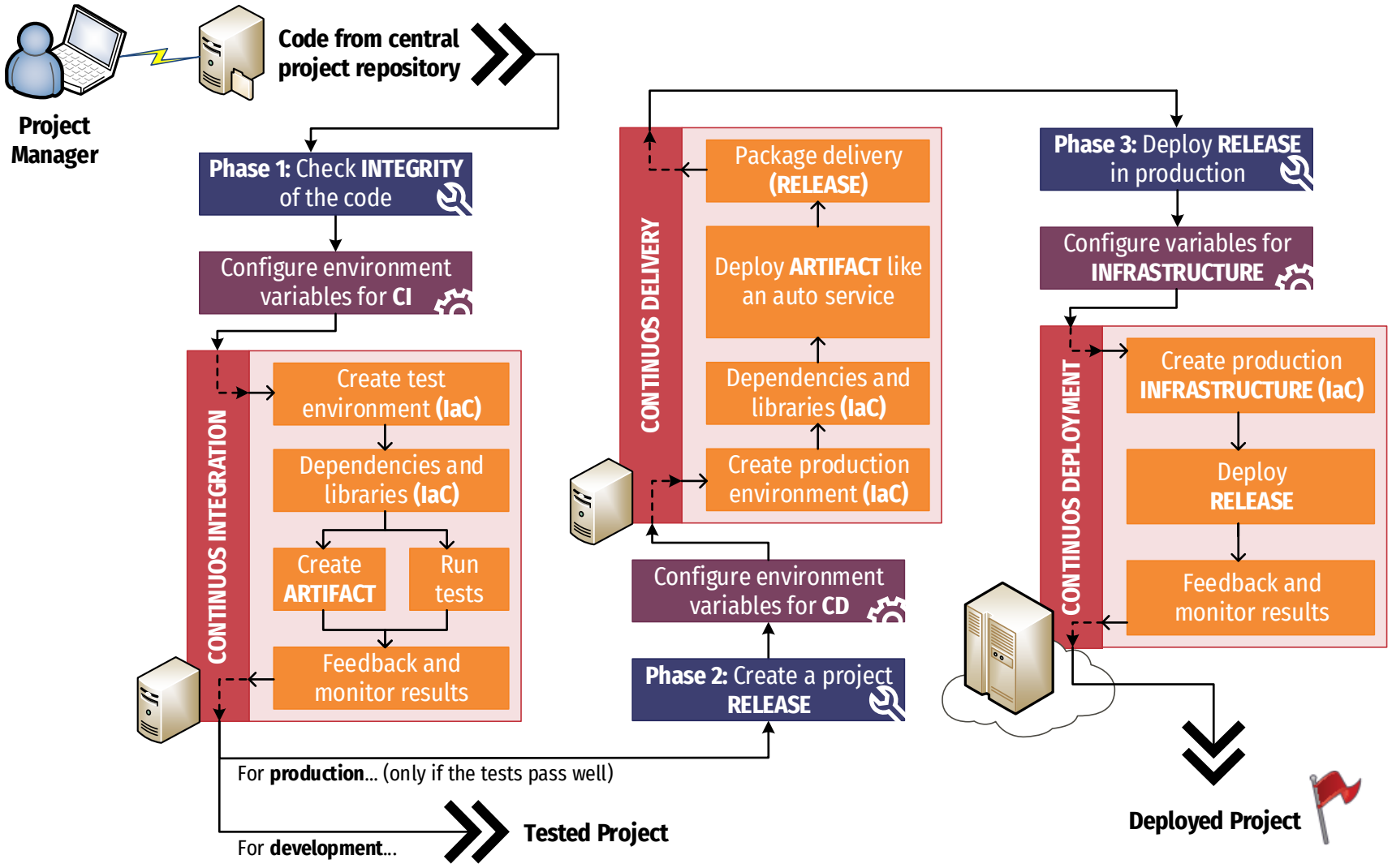
4.6.5. DevOps Pipeline FWD. En la Figura 15, se muestra un resumen del trabajo que realiza el DevOps pipeline construido y basado en la PoC realizada (ver Anexo B PoC DevOps pipeline y enfoque DDD con un servidor), es el proceso que permite entender el paso que se realiza desde la consolidación del código en integración continua hasta el paso a producción. Lo anterior conlleva un ahorro en tiempo que brinda la posibilidad de centrar los esfuerzos en mejorar el prototipo u aplicación, o proponer nuevas ideas en el desarrollo, volviendo más simple la creación de productos de software lanzados directamente a la nube.

El conjunto de herramientas usadas para la construcción del DevOps pipeline fueron i) Unix Shell para los scripts que integran y automatizan la gestión de la utilización de las diferentes herramientas de DevOps y la configuración de las variables del ambiente utilizado en cada fase durante el proceso, ii) Git y GitHub para el repositorio central de código, iii) Travis CI como ambiente en la nube para la integración continua del código, iv) Packer para la construcción del desplegable y v) Terraform que es el aprovisionador de componentes para el despliegue continuo de la infraestructura de producción en la nube de Digital Ocean, la cual es utilizada como plataforma de la infraestructura final.

4.6.6. Roles y responsabilidades del equipo de desarrollo. Para el desarrollo de los proyectos, se recomienda tener claro los siguientes roles independiente de que estén asociados a una o múltiples personas:

- **Ingeniero de Software (Developer)** Los ingenieros de software (desarrolladores) son los encargados de la implementación de la aplicación, estos se especializan en dos dominios:

Figura 15. DevOps Pipeline.



Fuente: Autor

- **Frontend** son los encargados de la aplicación para el usuario final, es la parte visual y donde se puede interactuar. Consiste en elementos de diseño cuyo comportamiento está definido por el código escrito.
- **Backend** se encargan de las operaciones que son invisibles para un usuario, pero son importantes para el equipo. Generalmente en el Backend se trabaja con la base de datos, el servidor y las aplicaciones de lado servidor.

También se consideran desarrolladores **Full stack**, como miembros del equipo que conocen de ambos dominios, Backend y Frontend. El trabajo coordinado de los desarrolladores es crucial para cada proyecto web donde cada una de las partes es importante en la aplicación.

A la par con los desarrolladores existen los **ingenieros de pruebas** (*Quality Assurance Tester*) quienes se encargan de la calidad del proyecto, verificando que todos los desarrollos realizados cumplan con los estándares esperados. Los ingenieros de pruebas son los encargados de realizar pruebas unitarias, de interacción, y de carga, entre otras.

- **Diseñador UI/UX (UI/UX designer)** Una aplicación web siempre tiene una interfaz de usuario y diseño de experiencia de usuario. El diseño de la interfaz de usuario se refiere a cómo se distribuye la superficie del sitio web, el diseño de la UX cubre la forma en que los clientes interactúan con la aplicación. En un equipo de desarrollo por lo general, una persona está a cargo del diseño de UI y UX.

- **Ingeniero de DevOps e infraestructura (DevOps engineer)** Se encargan de disponer la infraestructura para la aplicación y automatizar la integración de código de los desarrolladores, asegurar que se realice la entrega continua de funcionalidades nuevas del proyecto y los despliegues automáticos a

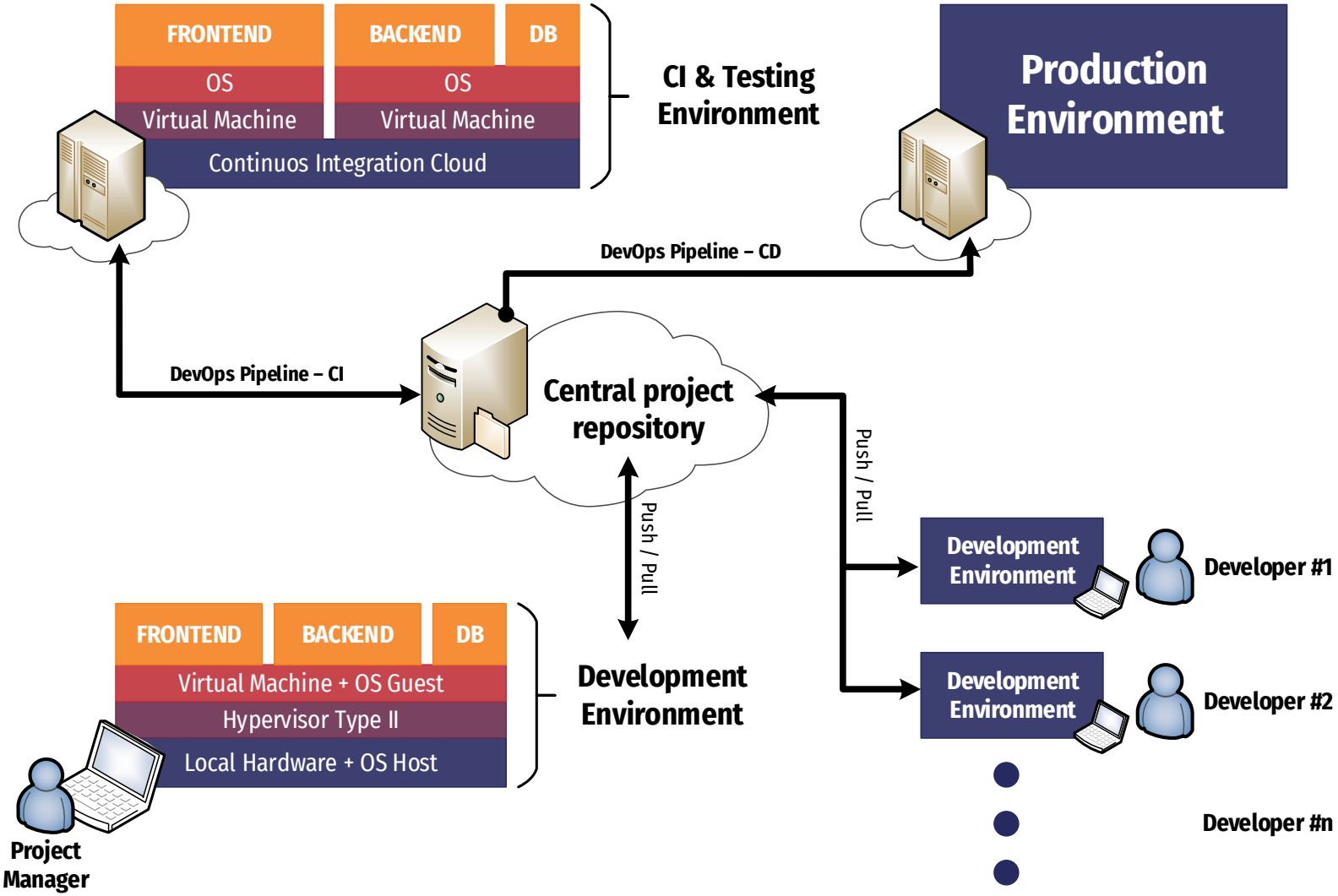
producción de una versión probada y verificada. Este monitorea el sistema completo y verifica que la aplicación esté operativa.

- **Arquitecto de TI (IT Architect)** Es el encargado del diseño de la aplicación tanto a nivel de software como de infraestructura, indicando los componentes y recursos que se necesitan para la aplicación. Se encarga de revisar la viabilidad del proyecto según los requerimientos solicitados, y también de diagramar todas las partes de la aplicación.

- **Gerente de proyecto (Project manager)** El gerente de proyecto es responsable del presupuesto, alcance, cronograma y calidad del proyecto. Conecta todos los puntos del proceso de desarrollo. Se encarga de que la comunicación en el equipo sea clara y que cada miembro del equipo conozca y comprenda los objetivos del mismo. Además, en sus funciones está la de dividir los componentes grandes del proyecto en tareas más pequeñas con el fin de segregar las funciones en el equipo asegurándose mantener su motivación y enfoque.

4.6.7. Interacción entre ambientes FWD. Todo nace de la tarea del desarrollo realizada por cada integrante del equipo de desarrollo, la cual es versionada y controlada por el repositorio central del proyecto (*Central project repository*). El gerente del proyecto (*Project manager*) es el que revisa el avance del proyecto, supervisa los conflictos que haya a nivel de desarrollo y determina los entregables de las versiones estables para el paso a producción. Se espera que el gerente del proyecto sea el único capaz de ejecutar el DevOps Pipeline, por lo cual en este rol recae dicha tarea.

Figura 16. Interacción entre ambientes.



Fuente: Autor

Al ejecutarse el DevOps Pipeline empieza el proceso de integración continua (*Continuos Integration – CI*) como se explica en la sección 4.6.5, recogiendo el código desarrollado del repositorio central (privado o público) y desplegado en un ambiente de pruebas (*Testing Environment*). Si las pruebas automáticas son superadas satisfactoriamente, el CI retorna el resultado y permite la ejecución del despliegue continuo. Este proceso es una ejecución de no retorno, aunque se puede detener su ejecución, pero no se recomienda ya que puede causar problemas que deben ser solucionados manualmente.

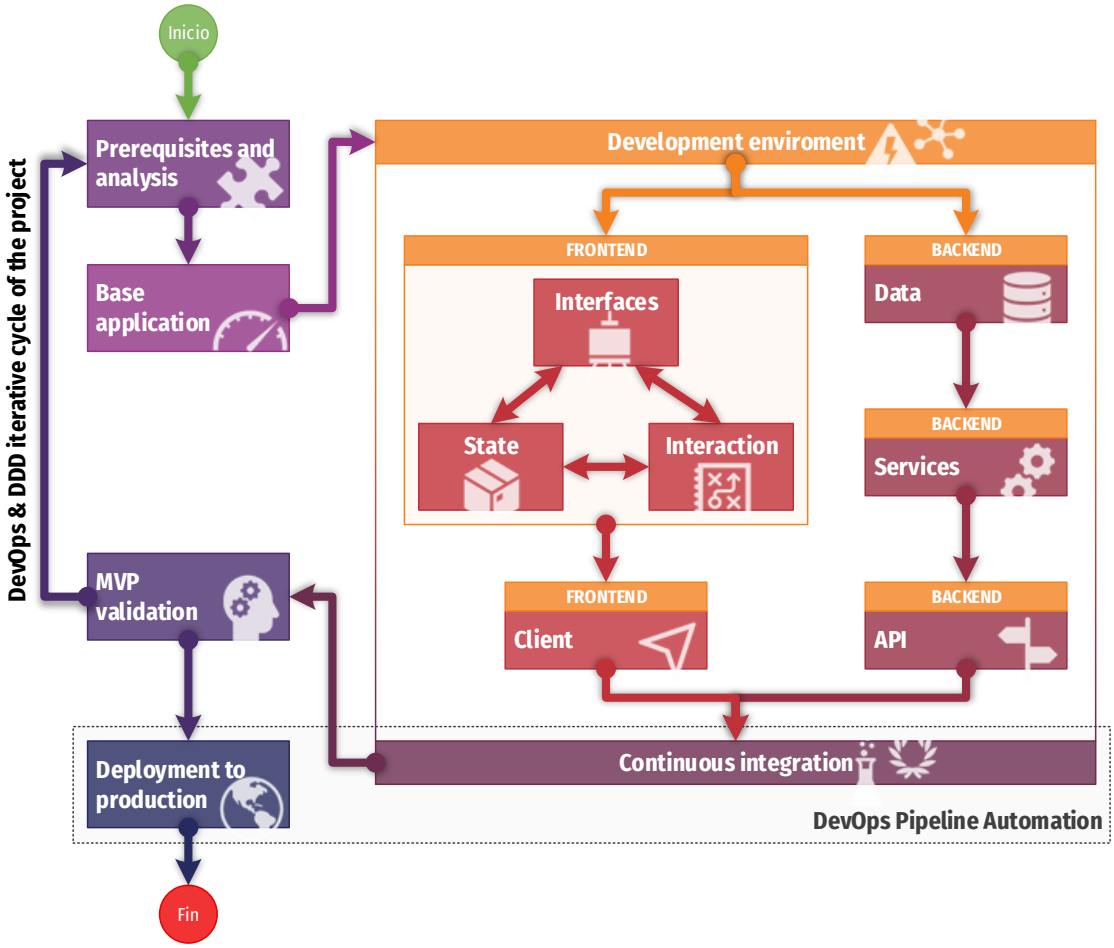
El despliegue continuo permite al código de la aplicación desarrollada ir directamente al ambiente diseñado como ambiente de producción, todo construido con infraestructura como código (*Infraestructure as a code*); adicionalmente, eñ despliegue continuo es el proceso donde interactúan la ejecución de scripts de Unix Shell, herramientas para la creación de imágenes de sistema operativo como Packer y un proveedor para la disposición de componentes de la arquitectura de producción como Terraform. Estas dos herramientas se eligieron gracias a las diferentes plataformas de la nube (*Clouds*) que las soportan, brindando un amplio espectro de elección para la disposición de la aplicación en producción.

De esta manera, se tiene como resultado final una aplicación desplegada de cara a los usuarios y una interacción ágil entre ambientes para el paso de desarrollo a producción, dando respuesta con la tecnología a los cambios que surjan desde el negocio. En la Figura 16 se esquematiza cómo interactúan los diferentes ambientes.

5. ESTRATEGIA PARA EL USO DEL FWD

La estrategia descrita en la Figura 17 está diseñada para crear proyectos de manera individual o colaborativa. El proyecto puede ser hecho por un solo desarrollador o tener varios equipos de trabajo enfocados a implementar diferentes partes de la aplicación. Se asume que se aplican metodologías ágiles usando algún Framework Agile como SCRUM, KANBAN, LEAN, entre otros, y que se emplean ciclos repetitivos desde el análisis, desarrollo, hasta producción tal como lo propone DevOps y DDD.

Figura 17. Estrategia para el uso del FWD.



Fuente: Autor

5.1. ETAPA 1: PRERREQUISITOS Y ANÁLISIS

El objetivo de este framework es el desarrollo de una aplicación web. Por lo tanto, se parte de un producto mínimo viable (*Minimum Viable Product – MVP*) compuesto por las funcionalidades que va a tener la aplicación inicialmente, extraídas a partir de la investigación realizada sobre la viabilidad de construir el producto. Este MVP en primera instancia debe contener:

1. Descripción y objetivo de la aplicación (incluido los *Stakeholders* y entidades).
2. La ruta del cliente (*Customer Journey*).
3. Interfaces gráficas por componentes y estado.
4. Lista de funcionalidades y requerimientos.
5. Contratos de servicio.
6. Modelo conceptual del dominio y modelo de datos.
7. Stack de herramientas a utilizar (*Frameworks*, Lenguajes de programación, Motor de base de datos, ORM, entre otros).

Con el fin de que la aplicación tenga un modelo conceptual normalizado y escalable es obligatorio verificar que cumpla los principios SOLID:

- Principio de responsabilidad única
- Principio de abierto para extensión/cerrado para modificación
- Principio de sustitución de Liskov
- Principio de segregación de la interfaz
- Principio de inversión de la dependencia

En caso de no cumplirlos, será necesario corregir los errores hasta que los cumpla. Como buenas prácticas y con el fin de obtener más información acerca de la aplicación a desarrollar, se sugiere responder a la siguiente lista de preguntas:

- ¿La aplicación requiere una base de datos SQL o NoSQL? ¿Requiere conectarse a servidores o recursos externos?
- ¿Cuál es la disponibilidad que deberá tener la aplicación en términos de tiempo y recursos?
- ¿Cuáles son las políticas de seguridad que deberá tener la aplicación?

Teniendo como insumo del proyecto estos requisitos, se puede avanzar a disponer de la aplicación base y los requerimientos para la ejecución del proyecto.

5.2. ETAPA 2: APLICACIÓN BASE

En esta etapa se utiliza el ambiente de desarrollo creado como prerrequisito para la construcción de la aplicación. Este ambiente debe contar con:

- Una base de sistema operativo.
- Los lenguajes de programación seleccionados para el proyecto.
- Librerías, dependencias y las herramientas a utilizar, definidos en el stack de herramientas para la aplicación.
- Recursos como servidores de Base de datos y Servidores de estáticos para guardar los datos y archivos respectivamente.

5.3. ETAPA 3: AMBIENTE DE DESARROLLO Y LINEAMIENTOS.

En la construcción de software, es necesario utilizar complementos como revisores de código, analizadores sintácticos, entre otros, los cuales facilitan el desarrollo de la aplicación siguiendo un estándar y el código debidamente organizado. Los entornos de desarrollo (*Integrated Development Environment - IDE*) tienen integradas estas herramientas y son propias para cada lenguaje de programación.

Cuando se requiere crear alguna parte de la aplicación se deben realizar las pruebas respectivas antes de su desarrollarlo, siguiendo el enfoque de desarrollo guiado por pruebas (*Test Driven Development – TDD*). Durante la fase de desarrollo se construyen pruebas unitarias, pruebas de comportamiento, pruebas de aceptación, entre otras.

Cuando un desarrollador implementa nuevas partes de la aplicación, las pruebas antes mencionadas permiten verificar la calidad, salud e integridad del software. Las pruebas son ejecutadas mediante la herramienta de integración continua y se harán en un paso previo al paso a producción. Lo ideal es que la totalidad del software cuente con sus pruebas respectivas. Se recomienda que los proyectos utilicen Git Flow, el cual es necesario para la organización del equipo y el desarrollo del proyecto. Finalmente, para el despliegue de debe hacer uso de un DevOps Pipeline para ahorrar tiempo en la configuración manual de servidores.

5.4. ETAPA 4: DATOS Y PERSISTENCIA

Este paso está orientado a crear la base de datos para la aplicación, la cual se pone en el servidor de base de datos a disposición del Backend, añadiéndose al ambiente de desarrollo como se describe a continuación:

- Hacer el diagrama del Modelo de Datos a partir del Modelo conceptual y añadirlo a la documentación.
- Documentar las tablas y las relaciones que hay entre ellas.
- Crear y configurar la Base de Datos a partir de los diagramas del Modelo de Datos en el(los) sistema(s) gestor(es) de base de datos (*Database Management System - DBMS*) del ambiente de desarrollo.
- Ejecutar el servidor de base de datos a disposición de la aplicación.

Una vez llegado a este punto se tiene lista la base de datos para que se conecte con el proyecto Backend. Se pueden utilizar diferentes motores de bases de datos, cada de los cuales se expone en una URL y puerto en específico (esto se puede configurar y depende de cada motor), los cuales serán el medio de comunicación para conectarse con el dominio Backend.

- Para el ambiente de producción es necesario crear una base de datos usando esta configuración.
- Desplegar la base de datos para el ambiente de producción.

5.5. ETAPA 5A: FRONTEND

Interfaces.

Para la interacción con el usuario se requiere crear las vistas que van a exponer las funcionalidades que ofrecerá la aplicación.

- Maquetar las vistas (*User Interface - UI*), y estructurar los componentes web que la integran.
- Construir las vistas a partir de las maquetas realizadas con los lenguajes predefinidos en las herramientas web. Se aconseja utilizar una librería que esté pensada para crear interfaces web por componentes, creando primero los componentes más básicos e integrándolos paulatinamente hasta completar las vistas.

Todas las vistas de la aplicación deben tener funciones y utilidades que le permitan a las interfaces contar con animaciones, funcionalidades, comportamiento, entre otros.

Interacción.

La interacción con el usuario hace parte del desarrollo de la aplicación y se basa en técnicas de Experiencia de Usuario (*User Experience - UX*) que permiten mejorar la usabilidad. La interacción depende del tipo de aplicación que se está construyendo. Para esto, es importante añadir la arquitectura de la información que implica diseñar donde van a estar ubicados cada uno de los componentes, sus dimensiones, etc., los cuales, en conjunto, harán parte de la calidad del producto final.

- Usar la arquitectura de la información para complementar al diseño e interacción de las interfaces de usuario.

Estado.

El estado hace parte del control de los datos en las diferentes vistas durante el ciclo de vida de la aplicación, así como también los eventos que se ejecutan cuando hay cambios en los datos o se reciben datos desde un API.

- Crear las funciones que controlen el estado de la aplicación cuando se ejecuta algún evento, tanto de interacción del usuario como de actualización de datos de la aplicación. Estas funciones tienen el objetivo de centralizar en un punto el estado y facilitar el monitoreo del ciclo de vida de la aplicación.

Cliente.

El cliente es donde se crean funciones locales de lógica de negocio, así como también la conectividad con API’s y Backend.

- Elaborar un Cliente que consuma los servicios que suministran las API externas o el Backend de la aplicación. Esto permitirá obtener y enviar datos desde el Frontend. Esta parte también incluye a los componentes de Real Time, si la aplicación los requiere.
- Construir funciones que permitan ejecutar la lógica de negocio en el cliente, la cual no requiere ser ejecutada desde el servidor de Backend.

5.6. ETAPA 5B: BACKEND

Datos.

Según la arquitectura de N-capas, el primer paso es construir la capa de persistencia, en la cual se configura el acceso a los datos almacenados en la base de datos para el proyecto Backend.

- Configurar el proyecto para conectar la base de datos a la aplicación de Backend.
- Incluir en el proyecto un Mapeo objeto-relacional (Object Relational Mapping - ORM).
- Utilizar un ORM que se pueda conectar a la base de datos y mapee los objetos al lenguaje de programación propuesto.
- Mapear cada clase y atributo del modelo conceptual para utilizar el ORM.
- Mapear las relaciones del modelo conceptual.
- Construir una clase para crear objetos de acceso a datos (*Data Access Object - DAO*) para cada clase del modelo conceptual con el fin de conectarse con su tabla correspondiente en el modelo de datos.

Servicios.

En esta capa se integran la base de datos y las diferentes aplicaciones externas que se requieran. Por tal motivo, el modelo conceptual de clases que contiene los diferentes objetos que se usarán en el proyecto debe definir la idea planteada para la aplicación, cada uno de los datos que se almacenarán en la base de datos y las funcionalidades. Los servicios implementados envían datos que son usados y tratados por el cliente.

La definición de clases está ligada con el contexto de la idea a realizar y los requerimientos planteados en la etapa de prerrequisitos y análisis. En caso de que se requiera ampliar los requerimientos se recomienda desarrollar completamente las funciones propuestas, incluyendo posteriormente las nuevas características y funcionalidades. En caso de reducción o cambio de requerimientos, se recomienda detener el ciclo e iniciar de nuevo con la fase de prerrequisitos y análisis, lo que evita pérdida de tiempo por la realización de funciones que no se van a utilizar.

- Implementar los servicios que contienen la lógica de negocio que soportará la aplicación
- En la capa de aplicación es donde se implementan los servicios, que son las funcionalidades o características de la lógica de negocio, que se puede integrar y conectar con los servicios de terceros (ya creados) que se requieran.

API.

En la capa de presentación, se crean los controladores que se exponen a través en ‘*endpoints*’ donde el cliente Frontend consume el API.

- Construir los controladores del modelo como un CRUD y funciones específicas para implementar la Arquitectura Orientada a Servicios (*Service Oriented Architecture - SOA*). Se recomienda utilizar el protocolo de integración de su preferencia. El presente framework se diseñó usando una RestFul API la cual usa formato JSON o XML como formato de comunicación.
- En caso de requerir conectividad Full dúplex, se pueden integrar también WebSockets los cuales permiten realizar aplicaciones que se actualizan en tiempo real en el cliente.

5.7. ETAPA 6: INTEGRACIÓN CONTINUA

En esta etapa se valida si las funcionalidades desarrolladas se integran perfectamente con todo el proyecto. Para esto se realiza la ejecución de las pruebas en un entorno de pruebas independiente, el cual se encarga de monitorear todo el proceso

- Subir los cambios a la rama de desarrollo para que la herramienta de integración continua valide la calidad de la aplicación.
- Si cumple con el estándar de calidad esperado, se suben los cambios a la rama de producción para que se lance la aplicación a producción con el DevOps Pipeline. En caso contrario, se deben corregir los errores y volver a subir los cambios a la rama de desarrollo.
- Monitorear que el proceso del DevOps Pipeline se ejecute correctamente.
- Validar que esté funcionando todo el proyecto en ambiente de producción.

5.8. ETAPA 7: VALIDACIÓN DEL MVP

En ésta etapa se verifica que cada funcionalidad documentada esté correctamente implementada en la aplicación web.

- Funcionalidades creadas completamente con validación manual por parte del Project Manager y el dueño del proyecto.
- Criterios de aceptación completos según la documentación realizada y las pruebas automáticas y manuales.

6. CONCLUSIONES

- Este trabajo considera el desarrollo de un framework para el diseño de aplicaciones web con automatización, buscando el ahorro de tiempo en el desarrollo de nuevas aplicaciones o en la inclusión de nuevas funcionalidades en aplicaciones existentes, garantizando de esta forma la escalabilidad, la calidad y la fiabilidad en el despliegue de producción.

- Para el desarrollo del marco de trabajo se aplica la cultura DevOps con el fin de garantizar la calidad de la aplicación y facilitar el paso a producción, teniendo en cuenta que DevOps tiene como foco la entrega constante de valor centrada en la automatización del ciclo de desarrollo. Esta automatización es lograda implementando infraestructura como código para obtener como resultado (entrega continua de valor) nuevas características de las aplicaciones. Adicionalmente, DevOps aporta una ventaja en tiempo para la verificación de la calidad del producto de software y el despliegue a producción realizándolo en un tiempo corto. La cultura de DevOps permitió mejorar iterativamente las propuestas y añadir los requerimientos solicitados durante la elaboración del framework.

- El enfoque DDD incorpora en el framework un marco ágil que permite conectar la implementación de la aplicación web, los conceptos del modelo y facilita al equipo de desarrollo mantener su atención con el núcleo de la idea del producto que se quiere desarrollar. De esta manera, el enfoque DDD estructura conceptualmente el framework, en otras palabras, provee una estructura de software y metodologías para tomar las decisiones de diseño del mismo.

- Para el desarrollo del framework, se tomaron seis conceptos de diseño, además de los conceptos que integra la arquitectura SOA, estos son: i) escalabilidad horizontal, ii) costo, iii) abstracción de programación, iv) adaptabilidad, v) compatibilidad y vi) rendimiento en la red. También se tienen en cuenta un listado de funcionalidades tales como 'login', registro de usuarios, seguridad por tokens, roles, perfiles, nivel de acceso, entre otras. Con el fin de cumplir con estos conceptos y funcionalidades se desarrollaron tres propuestas denominadas Propuesta I, Propuesta II y Propuesta FINAL.

- Para la Propuesta I se considera una arquitectura MVC, se observa que esta arquitectura no permite cumplir con los conceptos de escalabilidad horizontal y costo por lo que se desarrolla la Propuesta II, la cual consideró una arquitectura SOA, a pesar de que ésta si cumplió la mayoría de los conceptos de diseño, no se logró el concepto de abstracción de programación para un bajo acoplamiento; igualmente, la Propuesta II era deficiente en el cumplimiento de las funcionalidades de seguridad. Finalmente, con el fin de satisfacer todos los conceptos de diseño y las funcionalidades, la Propuesta FINAL del framework combina diferentes arquitecturas; así, se propone una arquitectura basada en SOA para el Backend y MVU para el Frontend. Adicionalmente, la arquitectura de infraestructura utilizada en el diseño del framework permite aumentar la cantidad de usuarios en el momento que se requiera.

- El diseño de la arquitectura de infraestructura modifica la manera en la que se diseña la arquitectura de software. Para evitar que los datos queden guardados en cada servidor − lo que provocaría un problema cuando se intente recuperar la información en el caso de tener múltiples servidores en el balanceador de carga −, en el diseño del framework se consideró un sistema de almacenamiento de datos centralizado, aparte de los servidores que están ejecutando la lógica de negocio.

- Con el fin de validar la efectividad del framework desarrollado, se implementó un prototipo de aplicación web de propósito educativo denominado UPost, cuyo objetivo principal es el de compartir imágenes dentro de una comunidad. Este desarrollo permitió utilizar todos los componentes del framework evaluando su flexibilidad a través de la implementación de diferentes módulos, como el de consumo de servicios externos. Utilizar el framework propuesto durante la construcción de UPost ayudó a la realización efectiva del prototipo consolidando que el uso del framework mejora el tiempo de entrega, disminuye la complejidad en la realización de la aplicación y facilita su lanzamiento a producción gracias al DevOps pipeline que integra.

- Cabe resaltar, que el desarrollo de este proyecto permitió la integración y aplicación de conocimientos obtenidos a través de diferentes áreas y cursos de formación tales como: matemáticas, bases de datos, ingeniería del software, programación orientada a objetos, programación web, bases de datos y análisis numérico.

7. RECOMENDACIONES

- Hacer uso de la misma arquitectura probando con diferentes conjuntos de herramientas tecnológicas, revisando los cambios a realizar para ajustarlas al framework y aprovechando al máximo las diferentes capacidades y bondades que ofrecen las herramientas en la solución de un problema en particular, tales como tareas de alto procesamiento, masiva concurrencia de usuarios y demás.

- Mejorar las automatizaciones realizadas en el DevOps pipeline, buscando obtener una aplicación que permita la gestión y construcción sencilla de diferentes Pipelines para diferentes tecnologías, más allá de las utilizadas en este proyecto.

- Explorar en la ampliación y modificación del diseño de la arquitectura tanto de software como de infraestructura, para que sea extensible y se pueda abarcar más herramientas con el fin de mejorar la seguridad, la capacidad, la escalabilidad, la usabilidad, entre otros conceptos de diseño.

- En lugar de usar arquitectura orientada a servicios, experimentar con microservicios como respuesta a mejoras en disponibilidad y escalabilidad en el servicio ofrecido por el Backend, actualizando el framework a las nuevas tendencias de desarrollo en este dominio.

- Mejorar y profundizar en el diseño de interfaces y de experiencia de usuario.

- Experimentar con el uso de diferentes metodologías ágiles en el diseño aplicaciones web para probar y medir la eficiencia de estas, en el uso del framework propuesto con un equipo real de desarrollo.

- Utilizar el framework diseñado buscando ampliarlo hacia una aplicación Multiplataforma (Multi-platform); por ejemplo, implementando aplicaciones para móviles, Smart TV, wereables u otros dispositivos, donde las aplicaciones actúan como Frontend que consuman los servicios expuestos en el Backend. Esto será posible aprovechando la arquitectura propuesta en el proyecto y complementándose con los beneficios que ofrecen los microservicios.

- Implementar una prueba de concepto empleando sistemas distribuidos de bases de datos permitiendo así verificar las ventajas que ofrecen estas tecnologías, con el fin de integrarlas al framework propuesto.

- Especializar y adicionar tecnologías diseñadas para hacer Big Data, Machine Learning, Inteligencia Artificial (p. ej. Apache Spark, Hadoop, Scala, Cognos), con el fin de crear un framework reutilizable para aplicaciones empresariales de alta capacidad, transaccionalidad, estandarización de datos, con reportes dinámicos, inteligencia artificial y algoritmos de analítica (para análisis de datos de usuarios y nuevas ideas potenciales de negocio), entre otros.

BIBLIOGRAFÍA

Akamai. Q4 2016 State of the Internet/Connectivity Report [En línea]. 2016, vol. 9 No.4., p13-14. (Recuperado en 8 de agosto de 2017) Disponible en: https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q4-2016-state-of-the-internet-connectivity-report.pdf

ARTAČ, Matej, *et al*. DevOps: introducing infrastructure-as-code. Proceedings of the 39th International Conference on Software Engineering Companion (ICSE-C '17). IEEE Press, Piscataway, NJ, USA, 2017, p. 497-498. DOI: https://doi.org/10.1109/ICSE-C.2017.162

AZAUSTRE, Carlos. ¿Qué es Flux? Entendiendo su arquitectura [En línea]. 7 de marzo de 2017 (Recuperado 3 de septiembre de 2017) Disponible en: https://carlosazaustre.es/como-funciona-flux/

BASS, Len; CLEMENTS, Paul y KAZMAN, Rick. Software Architecture in Practice. 2da edición. Boston, MA, USA: Ed. Addison Wesley. 2003. 560 páginas. ISB: 0-321-15495-9

BBVAOPEN4U. Tendencias en desarrollo de software para 2016 [En línea]. Portal BBVAOPEN4U. 1 de febrero de 2016. (Recuperado en 8 de agosto de 2017) Disponible en: https://bbvaopen4u.com/es/actualidad/tendencias-en-desarrollo-de-software-para-2016

BONDI, André B... Characteristics of scalability and their impact on performance. Proceedings of the 2nd international workshop on Software and performance (WOSP '00). ACM, New York, NY, USA, 2000, p. 195-203.

CAUM, Carl. What is infrastructure as code? [En línea]. Puppet. 10 de febrero de 2017. (Recuperado en 7 de septiembre de 2017) Disponible en: https://puppet.com/blog/what-is-infrastructure-as-code

CHRISTENSSON, Per. Framework Definition [En línea]. TechTerms. 7 de marzo de 2013. (Recuperado en 7 de septiembre de 2017) Disponible en: https://techterms.com/definition/framework.

CHRISTENSSON, Per. Web Application Definition [En línea]. TechTerms 17 de febrero 2014. (Recuperado en 7 de septiembre de 2017) Disponible en: https://techterms.com/definition/web_application

DDD Community. What is Domain-Driven Design [En línea]. 28 de marzo de 2007. (Recuperado en 15 de agosto de 2017) Disponible en: http://dddcommunity.org/learning-ddd/what_is_ddd/

Departamento Administrativo Nacional de Estadística (DANE). Boletín Técnico Comunicación informativa DANE - Indicadores básicos de tenencia y uso de tecnologías de la información y comunicación – tic en hogares y personas de 5 y más años de edad 2016 [En línea]. 2016, p. 15. (Recuperado en 8 de agosto de 2017) Disponible en: https://www.dane.gov.co/files/investigaciones/boletines/tic/bol_tic_hogares_2016.pdf

EBERT, Christof, *et al*. DevOps. IEEE Software. Mayo-junio, 2016, vol. 33, no. 3., p 94-100. DOI=http://dx.doi.org/10.1109/MS.2016.68

ELLISON, Richard. Software Testing Environments Best Practices [En línea]. Software Testing Magazine. Martinig & Associates. 20 de junio de 2016. (Recuperado en 7 de septiembre de 2017) Disponible en: http://www.softwaretestingmagazine.com/knowledge/software-testing-environments-best-practices/

EVANS, Eric J. Domain-Driven Design: Tacking Complexity in the Heart of Software. 1ra edición. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003. 592 páginas. ISBN-13: 978-0321125217

FAURA, Oscar. Integración continua (CI), entrega continua (CD) y despliegue continuo (CD) [En línea]. Blog DevOpsTI. 2017. (Recuperado en 18 de septiembre de 2017) Disponible en: https://devopsti.wordpress.com/2014/09/26/integracion-continua-ci-entrega-continua-cd-y-despliegue-continuo-cd/

FORSGREN, Nicole, *et al.* 2017 State of DevOps Report [En línea]. Puppet. (Recuperado en 7 de septiembre de 2017) Disponible en: https://puppet.com/resources/whitepaper/state-of-devops-report

HARFORD, Tim. La fascinante historia de cómo los smartphones se volvieron tan inteligentes [En línea]. BBC News. 27 de diciembre de 2016. (Recuperado en 8 de agosto de 2017) Disponible en: http://www.bbc.com/mundo/noticias-38436119

HILL, Mark D. 1990. What is scalability? En: ACM SIGARCH Computer Architecture News. Diciembre, 1990. vol. 18, no.4 p. 18-21.

IEEE Computer Society. SWEBOK V3 [En línea]. (Recuperado en 17 de agosto de 2017) Disponible en: https://www.computer.org/web/swebok/v3

Impactum. Diferencia entre plataforma web, página web, y apps [En línea]. (Recuperado en 7 de septiembre de 2017) Disponible en: https://impactum.mx/diferencia-pagina-web-plataforma-web-apps/

Intelygenz. Nuestra metodología de desarrollo: Despliegue Continuo [En línea]. 2017. (Recuperado en 22 de noviembre de 2017) Disponible en: https://www.intelygenz.es/servicios/nuestra-metodologia-de-desarrollo-despliegue-continuo/

KERSTEN, Nigel. What is DevOps? [En línea]. Puppet. 30 de enero de 2017. (Recuperado en 11 de noviembre de 2017) Disponible en: https://puppet.com/blog/what-is-devops/

LEINER, Barry M, et al. Breve historia de internet [En línea]. Internet Society. 1997. (Recuperado en 8 de agosto de 2017) Disponible en: https://www.internetsociety.org/es/breve-historia-de-internet

MOBIDEV. 3 Types Of Web Application Architecture [En línea]. Mobidev. 18 de febrero de 2016. (Recuperado en 3 de febrero de 2017) Disponible en: https://mobidev.biz/blog/3_types_of_web_application_architecture

O'GRADY, Stephen. The RedMonk Programming Language Rankings: June 2017 [En línea]. RedMonk. 8 de junio de 2017. (Recuperado en 8 de agosto de 2017) Disponible en: https://redmonk.com/sogrady/2017/06/08/language-rankings-6-17/

PENCHIKALA, Srini. Domain Driven Design and Development In Practice [En línea]. Portal InfoQ. 12 de junio de 2008. (Recuperado en 8 de agosto de 2017) Disponible en: https://www.infoq.com/articles/ddd-in-practice

PETRIE, Charles. A Brief History. En: IEEE Internet Computing. Noviembre-diciembre, 1998, vol. 2 no.6., p. 6–7.

SHAW, Mary y GARLAN, David. Software Architecture: Perspectives on an Emerging discipline. 1ra edición. USA: Ed. Prentice Hall. 1996. 242 páginas. ISBN-13: 978-0131829572

SOMMERVILLE, Ian. Software Engineering. 9na edición. USA: Addison-Wesley Publishing Company. 2010. 792 páginas. ISBN-13: 978-0137035151

Statcounter. Desktop vs Mobile vs Tablet Market Share Worldwide [En línea]. (Recuperado en 8 de agosto de 2017) Disponible en: http://gs.statcounter.com/platform-market-share/desktop-mobile-tablet.

TATCHYN, Andrly. WebApp development team structure: roles and responsabilities [En línea]. 16 de abril 2018. (Recuperado en 22 de diciembre 2018). Disponible en: https://blog.lasoft.org/web-development-team-roles-responsibilities/

TIOBE. TIOBE Index for August 2017 August Headline: The real fight is in the backyard [En línea]. (Recuperado en 8 de agosto de 2017) Disponible en: https://www.tiobe.com/tiobe-index/

ZIEMER, Sven. An Architecture for Web Applications, Essay in DIF 8914 Distributed Information Systems [En línea]. 28 de noviembre de 2002. (Recuperado en 3 de febrero 2018) Disponible en: https://pdfs.semanticscholar.org/d4e7/cfa0710ea154edc2e8ade7496efe4405d6df.pdf

ZIMMERMANN, Olaf. Web Server project roles [En linea]. 10 de enero 2004. (Recuperado en 22 de diciembre 2018) Disponible en: https://www.ibm.com/developerworks/library/ws-roles/ws-roles-pdf.pdf

ANEXOS

Anexo A Marco de referencia.

APLICACIÓN WEB

Son aquellas aplicaciones que acceden a través de un navegador a un servidor web en Internet o Intranet, tienen la ventaja de que sin importar el sistema operativo o dispositivo que las utilice funcionan casi de la misma manera. Se codifica pensando en los lenguajes soportados por los navegadores web y son fáciles de actualizar y mantener sin tener que instalar el software en la máquina de cada usuario (Christensson, 2014).

Se diferencian respecto a una página web o sitio web, donde se tiene un contenido estático y una muy baja interacción con el usuario. Por lo general, consisten en páginas informativas cuyo contenido se carga en un servidor y se muestra sin alguna otra funcionalidad, a comparación de que una aplicación web soporta gran cantidad de interacciones con el usuario, existe una comunicación activa del usuario con la información, son dinámicas y utilizan bases de datos. Existen diferentes aplicaciones como tiendas en línea, gestores de contenido, editores de documentos de texto, blogs, redes sociales, entre otras, y todo mientras está conectado en línea. Se pueden crear todo tipo de aplicaciones que se diseñan generalmente para ser usadas por varios usuarios en concurrencia. Un concepto muy utilizado actualmente en el mercado es el de plataforma web que se diferencia de una aplicación web porque se conforma de aplicaciones web, aplicaciones móviles, servidores externos, entre otros. Donde cada una de las partes se integran entre sí (Impactum, 2017).

MARCO DE TRABAJO (*FRAMEWORK*)

Un marco de trabajo (*framework*, en inglés) es esquema para el desarrollo de un producto o funcionalidad definida que utiliza un conjunto de herramientas tecnológicas, conceptos o patrones que ayudan a agilizar de manera eficiente la entrega de valor. No son específicos respecto a un lenguaje de programación, pero suelen incluir librerías y programas de software, se caracterizan por su alta cohesión y bajo acoplamiento entre componentes. Un framework puede definir la estructura para una aplicación completa o parte de ella, todo dependiendo del objetivo del framework (Christensson, 2013).

También existen frameworks genéricos de los cuales no se conoce su funcionamiento interno, esto puede ser algo negativo para los desarrolladores porque no pueden tener el gobierno completo de su aplicación para verificar dónde se encuentran ubicados los posibles errores y problemas que surgen durante el ciclo de desarrollo.

Las ventajas más significativas de usar frameworks utilizando buenas prácticas son:

- Evitar escribir código repetitivo
- Crear productos avanzados y robustos que costarían tiempo implementar.
- Desarrollar más rápido y eficiente.
- Organización de código y archivos.

Desarrollar aplicaciones web puede ser a veces engorroso por la cantidad de herramientas que existen para tal propósito, sobre todo si se trata de una persona que está empezando a desarrollar en la web, por lo tanto, es más útil seguir los lineamientos que ofrece un framework el cual facilita y aclara el desarrollo del producto.

DISEÑO GUIADO POR EL DOMINIO (*DOMAIN DRIVEN DESIGN*)

El diseño guiado por el dominio (*Domain Driven Design* – por su sigla, DDD), es un enfoque para el desarrollo de software de necesidades complejas que conecta la implementación, los conceptos del modelo y núcleo de la idea. Inserta la idea del producto en artefactos de software soportados en la arquitectura, diseño y ejecución. DDD no es una tecnología o una metodología, provee una estructura de prácticas y terminologías para tomar decisiones de diseño que enfoquen y aceleren el manejo de dominios complejos en los proyectos de software (DDD Community, 2007). El término fue por primera vez mencionado en el libro de Eric Evans “Domain-Driven Design – Tackling Complexity in the Heart of Software” (Evans, 2003). A continuación, se explicarán algunos conceptos del libro de Eric Evans de donde se enmarcará este proyecto.

Contexto y dominio (Domain)

El contexto es el entorno en el que aparece una palabra o declaración que determina su significado, y el dominio es un campo de estudio que define un conjunto de requisitos comunes, terminología y funcionalidad para cualquier programa de software.

Modelo de dominio (Domain model)

Es una representación formal de un problema en específico, con conceptos, roles, tipos de datos, individuos y reglas. En ingeniería de software es un modelo conceptual que incorpora comportamiento y datos. Éste ofrece varios beneficios, entre los cuales se encuentra:

Un modelo común del proyecto que los desarrolladores pueden utilizar para comunicar sobre los requisitos y las entidades de datos.

- El modelo debe ser por partes, extensible y fácil de mantener.
- Mejora la reutilización y las pruebas de los objetos.

Cuando no se sigue un enfoque de modelo de dominio para desarrollar aplicaciones conlleva a un modelo pesado porque acumula más y más objetos portadores de interfaces. Conduce a una lógica y reglas dispersas y duplicación en varias clases.

Características de un modelo de dominio

Las siguientes son características propias de un modelo de dominio:

- Se debe enfocar en un dominio o problema en específico.
- Debe estar aislado de otros dominios.
- Debe ser reusable y evitar duplicar modelos e implementaciones.
- El modelo necesita ser diseñado para que tenga un bajo acoplamiento entre capas de la aplicación.
- El modelo debe ser abstracto y limpio para un fácil mantenimiento y versionamiento.
- El modelo de dominio debe ser independiente de los detalles de implementación de persistencia.
- Debe tener una mínima dependencia de alguna infraestructura o framework.

Para la arquitectura orientada a servicios (*Service Oriented Architecture* – por su sigla, SOA), DDD es un elemento clave porque ayuda a encapsular la lógica y las reglas en los objetos del problema.

El modelo de dominio también proporciona el idioma y el contexto con el que se puede definir el contrato de un servicio (Un contrato es la manera como se hace peticiones a un servicio y los datos con los cuales me va a responder). Un esfuerzo de SOA debe incluir el diseño y la implementación de un modelo de dominio.

Gestión de proyectos con DDD

Un proyecto con modelado de dominio típicamente incluye los siguientes pasos:

- Modelar y documentar las funcionalidades primero.
- Identificar todos los servicios que se requieran. Estos servicios pueden ser atómicos (un solo paso) u orquestados (múltiples pasos).
- Identificar y documentar el estado y el comportamiento de los objetos utilizados por los servicios identificados en el paso anterior.

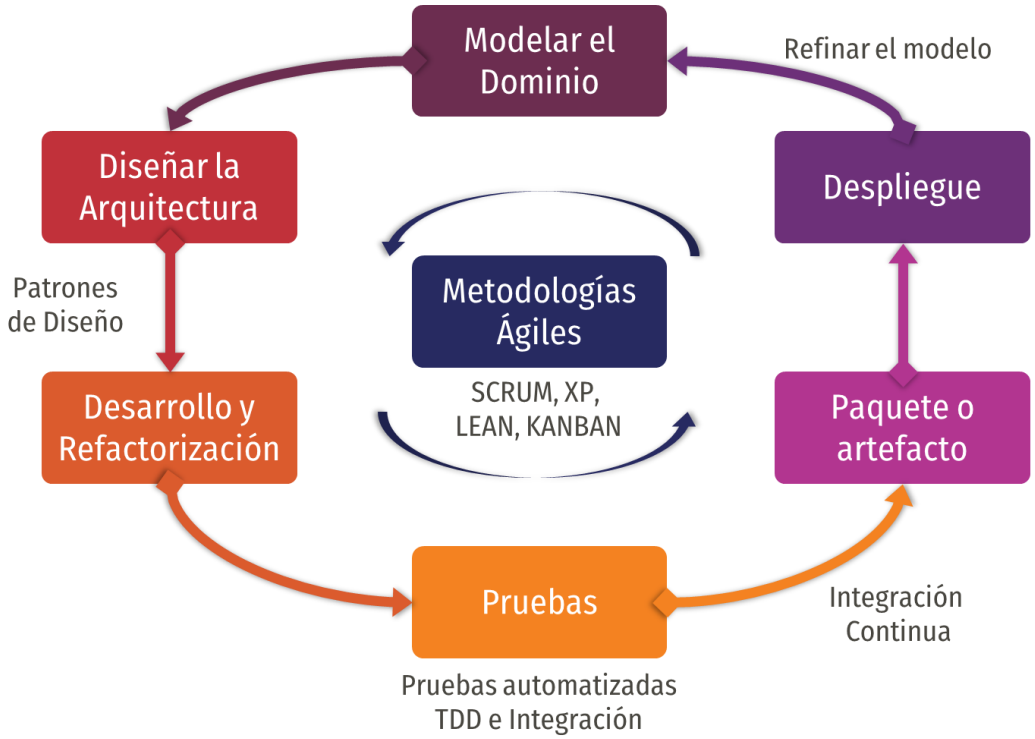
Desde un punto de vista de la gestión de proyectos, la implementación de un proyecto DDD comprende realizar las mismas fases de cualquier proyecto de desarrollo de software (ver Figura 18). Estas fases incluyen:

- Modelar el dominio.
- Diseñar la Arquitectura.
- Desarrollo y Refactorización.
- Hacer pruebas unitarias y de integración.
- Refinar y refactorizar el modelo de dominio basado en el diseño y desarrollo (utilizando Integración Continua).
- Repetir el ciclo usando el modelo de dominio actualizado.

Las metodologías ágiles de desarrollo de software se combinan con DDD porque se enfocan en la entrega constante de valor y la alineación del artefacto de software

con la idea que se quiere implementar. Además, el ciclo iterativo de DDD y los frameworks de metodologías ágiles como SCRUM, LEAN, XP o KANBAN son los mejores para administrar y gestionar un proyecto de desarrollo DDD.

Figura 18. Ciclo de iteración DDD



Fuente: Autor

ARQUITECTURA DE SOFTWARE

La arquitectura de software es una vista o abstracción donde se describe los componentes principales del sistema, como se vio en la definición de DDD, le da estructura a un proyecto de software. Según el SWEBOK son "el conjunto de estructuras necesarias para razonar acerca del sistema, que comprenden elementos de software, relaciones entre ellos y propiedades de ambos" (IEEE Computer Society, 2017) y de acuerdo con el Software Engineering Institute (SEI),

la arquitectura de software se refiere a “las estructuras de un sistema, compuestas de elementos con propiedades visibles de forma externa y las relaciones que existen entre ellos” (Bass, 2003).

El término “elementos” según el SEI puede ser variable, por lo que puede referirse a distintas entidades relacionadas con el sistema. Los elementos pueden ser:

- Entidades que existen en tiempo de ejecución (objetos, hilos).
- Entidades lógicas que existen en tiempo de desarrollo (clases, componentes).
- Entidades físicas (nodos, directorios).

Por otro lado, las relaciones entre elementos dependen de propiedades de los elementos, quedando ocultos los detalles de implementación.

La arquitectura de software comenzó a emerger como una disciplina que implicaba el estudio de estructuras y arquitecturas de software de una manera más genérica. Esto dio lugar a una serie de conceptos interesantes sobre el diseño de software en diferentes niveles de abstracción, tales como los estilos arquitectónicos y patrones de arquitectura de software.

Dentro de las arquitecturas de software comunes para el desarrollo de una aplicación se encuentran:

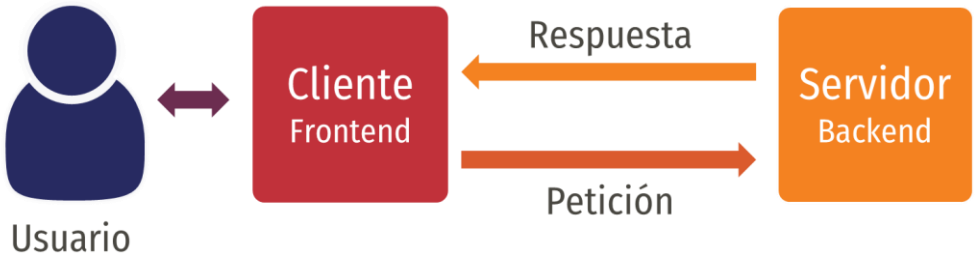
- Descomposición Modular.
- Cliente-servidor.
- Arquitectura de N-capas.
- En pipeline.
- Entre pares.
- En pizarra.

- Arquitectura Orientada a Servicios (Service Oriented Architecture - SOA).
- Arquitectura de microservicios.
- Dirigida por eventos.

Arquitectura cliente-servidor

Para desarrollar una aplicación web, se requiere definir la arquitectura de software que debe tener dicha aplicación, con el fin de entender lo que se va a realizar en un primer nivel de abstracción (ver Figura 19), la cual provee una visión muy general del proyecto y la separación de los componentes Frontend y Backend.

Figura 19. Arquitectura Web cliente-servidor simple



Fuente: Autor

Se puede diseñar una aplicación en Frontend y en el Backend respectivamente, utilizando patrones de arquitectura de software tales como Modelo Vista Controlador (Model View Controller - MVC), Modelo Vista Cualquiera (Model View Whatever - MVW, MV*), Modelo Vista Actualizar (Model View Update - MVU), entre otras.

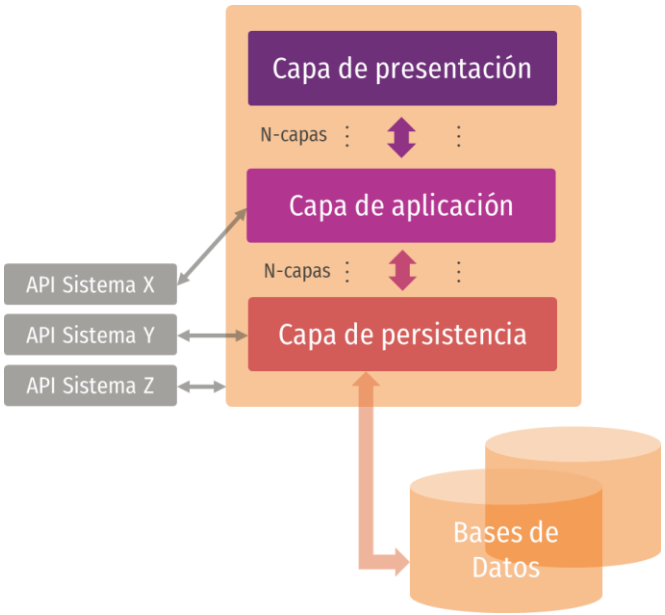
Esta característica va sujeta a las necesidades de la aplicación y experiencia del desarrollador.

Arquitectura de N-capas

Generalmente en la Arquitectura de N-capas (ver Figura 20) se manejan las tres capas descritas a continuación:

- **Capa de presentación:** Maneja interfaces de programación de aplicaciones (Application Programing Interface - API) las cuales permiten desacoplar la vista del cliente del servidor lo que se traduce en un bajo acoplamiento entre sistemas, ya que ésta es la única capa que se comunica con el Frontend y con la capa de aplicación.
- **Capa de aplicación:** Maneja la lógica de la aplicación, se comunica o se integra con otros sistemas si lo requiere la aplicación. Ésta es la única que se puede comunicar con la capa de presentación y con la capa de persistencia.
- **Capa de persistencia:** Esta capa se comunica directamente con las bases de datos y provee los datos que serán manejados en la capa de aplicación.

Figura 20. Arquitectura de N-capas



Fuente: Autor

Dependiendo de la complejidad de la aplicación se puede requerir capas adicionales, o si el desarrollador desea hacer la aplicación de forma modular puede integrar más capas fácilmente.

Arquitectura SOA

La Arquitectura Orientada a Servicios (SOA) es un estilo arquitectónico orientado a la creación de aplicaciones con base en servicios disponibles que busca la separación entre la implementación y la lógica de integración de negocio.

Entre sus ventajas se encuentra la innovación en servicios, reducción de costos, adaptación ante cambios y aumento de eficiencia en los procesos. Los servicios en SOA hacen uso de protocolos donde se describen cómo se pasan y analizan mensajes usando metadatos de descripción.

Dentro de dichos protocolos se puede encontrar: Web Services Description Language (WDSL), Simple Object Access Protocol (SOAP)., Universal Description, Discovery and Integration (UDDI), Representational state transfer (REST), entre otros.

En la industria no existe uniformidad en cuanto cómo hacer uso de SOA. Sin embargo, se pueden encontrar principios tales como:

- **Contrato de servicios estandarizados:** los servicios se especifican de acuerdo con lo descrito en documentos para mantener un acuerdo de comunicación.
- **Acoplamiento débil de sistemas:** el conocimiento entre servicios es lo único necesario para establecer una relación que conduce a una mínima dependencia.

- **Abstracción de servicios:** la lógica de los servicios se reserva ante externos.
- **Reutilización de servicios:** se logra mediante la división de la lógica en servicios.
- **Autonomía de servicios:** en vista del diseño y ejecución, cada servicio posee control sobre su lógica.
- **Servicios sin-estado:** el consumo de recursos se lleva al mínimo, retrasando inclusive la gestión de la información sólo cuando sea necesario.
- **Descubrimiento de servicios:** se crean los servicios con metadatos para ser encontrados y hacer uso eficiente de los mismos.
- **Composición de servicios:** los servicios se componen por partes de manera eficaz, sin limitaciones en tamaño y complejidad.
- **Granularidad de servicios:** una funcionalidad específica no compartida implementada en cada servicio de la capa de SOA. Responde a una necesidad única del usuario.
- **La normalización de servicios:** la descomposición de servicios se dirige a un nivel de forma normal con el fin de disminuir la redundancia.
- **Optimización de servicios:** se prioriza la selección de servicios de alta calidad sobre los de baja calidad.
- **Relevancia de servicios:** se refiere a la importancia que tiene cada servicio.
- **Encapsulación de servicios:** organizar cada servicio como una unidad funcional y administrable.
- **Transparencia de ubicación de servicios:** apunta hacia la característica por parte del consumidor de servicios para invocar un servicio independientemente de la ubicación en red.

Dentro de los beneficios con los que puede contar una organización al adoptar SOA se encuentran facilidades para la integración de sistemas y aplicaciones para mejorar la capacidad de respuesta con sistemas externos, posibilidades de

desarrollo de aplicaciones flexibles con independencia de plataformas y lenguajes de programación, reutilizables y adaptables, así como reducción de costos, optimización de recursos y mitigación del riesgo de migración.

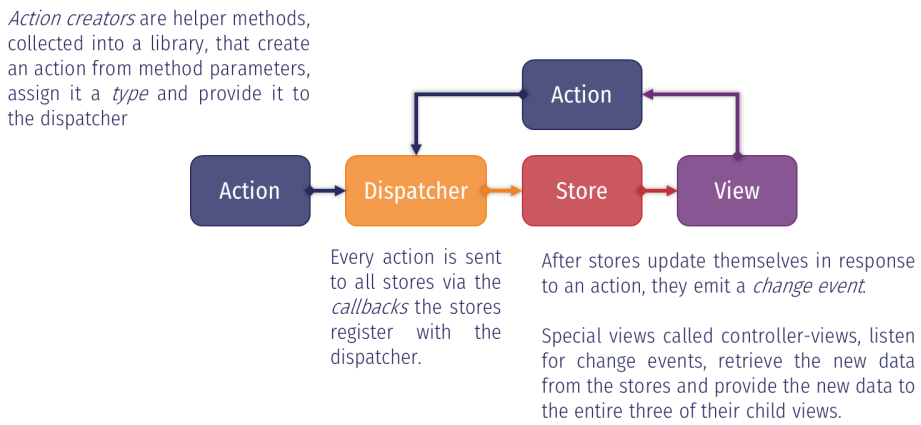
Arquitectura Flux

Flux es una arquitectura para el manejo y el flujo de los datos en una aplicación web, particularmente en el Frontend. Propone un flujo de datos unidireccional, que parte desde la vista por medio de acciones hasta llegar a un Store donde se actualizará la vista de nuevo (Azaustre, 2017), como se muestra en la Figura 21.

Flux se compone por los siguientes actores:

- **Vista:** Formada por los componentes web.
- **Store:** Guarda el estado o datos de la aplicación.
- **Acciones:** Objeto con una intención de realizar algo con datos necesarios si es necesario.
- **Dispatcher:** Mediador entre la Store o Stores y las acciones. Sirve para desacoplar la Store de la vista.

Figura 21. Arquitectura Flux

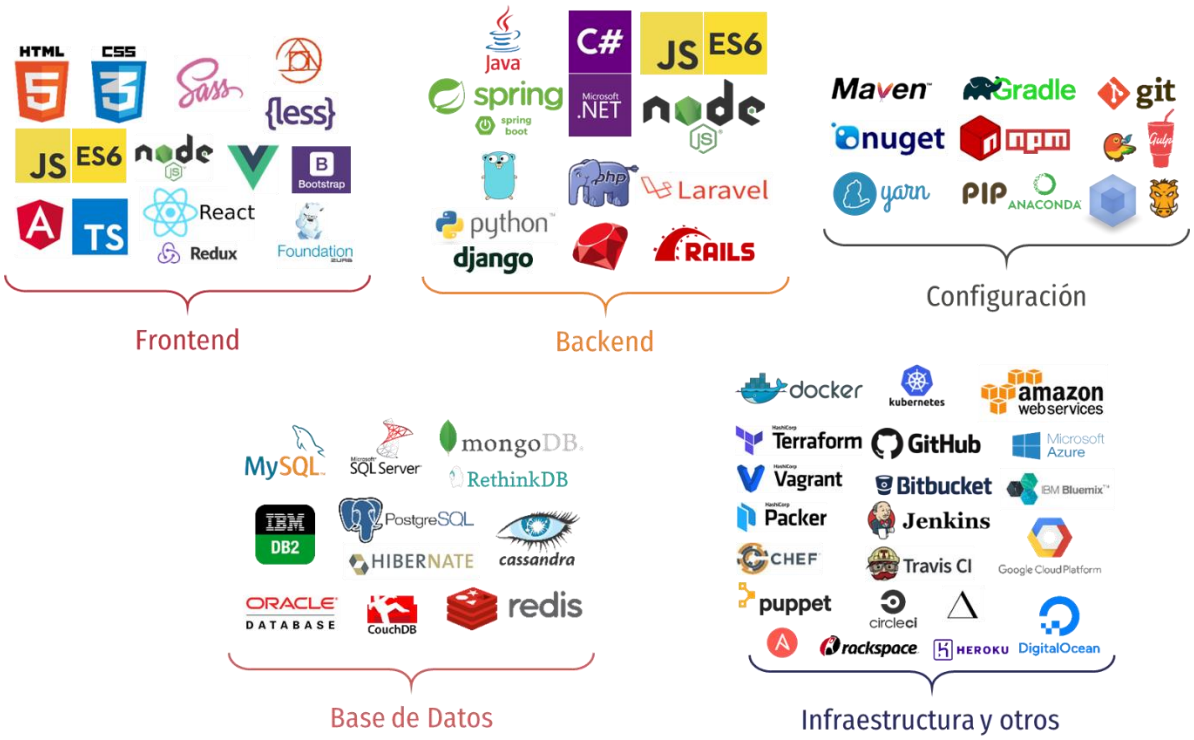


Fuente: https://facebook.github.io/flux/docs/in-depth-overview.html#structure-and-data-flow

Herramientas para el desarrollo web

Disponibilidad de herramientas para el desarrollo web (ver Figura 22):

Figura 22. Herramientas para desarrollo web



Fuente: Autor

- **Para el Frontend:** HTML5, CSS3, Bootstrap, Foundation, Sass, Less, PostCSS, Node JS, ES6, JavaScript ES6, TypeScript, Angularjs, React, Redux, Vue.js, entre otras.
- **Para el Backend:** Golang, Python, Django, JavaScript ES6, Node JS, PHP, Laravel, Java, Spring Framework, C#, .NET Core, Ruby, Ruby on Rails, entre otras.
- **Para Bases de Datos:** Hibernate, Oracle Database, Microsoft SQL Server, IBM DB2, MySQL, PostgreSQL, MongoDB, Cassandra DB, RethinkDB, CouchDB, Redis, entre otras.

- **Para la Configuración:** Git, Bower, Grunt, Gulp, Maven, Gradle, Browserify, Webpack, PIP, Anaconda, NPM, Yarn, NuGet, entre otras.
- **Para la Infraestructura y otros:** Docker, Kubernetes, Vagrant, Packer, Terraform, Ansible, Puppet, Chef, Jenkins, Travis CI, Circle CI, GitHub, Bitbucket, AWS, Google Cloud Platform, Microsoft Azure, DigitalOcean, Heroku, IBM Bluemix, Zeit now.sh, Rackspace, entre otras.

Ambientes de Despliegue de software

Un ambiente es una máquina, computadora, servidor, móvil o embebido, donde un programa o un componente de software se despliega y se ejecuta. Existen diferentes ambientes de despliegue y en el desarrollo de una aplicación siempre se inicia por el ambiente de desarrollo y finaliza en el ambiente de producción. Normalmente se propone utilizar 4 ambientes:

- **Desarrollo (Development):** Es el ambiente donde un desarrollador crea o implementa la aplicación. Es una estación de trabajo individual, puede ser un computador, un smartphone, un sistema embebido, entre otros. Incluyen herramientas de desarrollo como un compilador, entorno de desarrollo integrado (*Integrated Development Environment - IDE*), versiones diferentes o adicionales de bibliotecas y software de soporte.
- **Pruebas (Testing):** En este ambiente se ejecutan las pruebas para verificar la integridad del software, comportamiento, seguridad, entre otras. Normalmente se encuentra en un servidor y se ejecutan de manera automatizada. Si el desarrollador aprueba la aplicación se envía a uno de los ambientes de ensayo, pero si las pruebas fallan se devuelve al ambiente de desarrollo.
- **Puesta en escena o Ensayo (Staging):** Es un ambiente donde se hacen lanzamientos de la aplicación y se verifica el funcionamiento haciendo

pruebas finales antes de enviarla a interactuar con los usuarios. Es como un espejo del ambiente de producción. Se encuentra habitualmente en un servidor.

- **Producción (Production):** Es el ambiente en que un usuario interactúa directamente con la aplicación. Se encuentra en un servidor, centro de cómputo o en la nube.

Se tiene en cuenta que el software es desplegado en cada ambiente siguiendo este orden. También existen ambientes para verificar la calidad del software, como lo es el ambiente de aseguramiento de la calidad (*Quality Assurance - QA*), para experimentar como el ambiente “caja de arena” (*Sandbox*) o para recuperación de desastres (*Disaster Recovery*) para brindar una solución en caso de tener problemas en producción. Otra forma de utilizar ambientes para una aplicación es desarrollo, pruebas, QA y producción (Ellison, 2016).

DEVOPS

DevOps significa Desarrollo (*Development*) y Operaciones (*Operations*), es un conjunto de prácticas y valores culturales que ha demostrado ayudar a las organizaciones a mejorar sus ciclos de lanzamiento de software, calidad, seguridad y la capacidad de obtener retroalimentación rápida sobre el desarrollo de productos (Ellison, 2016). Tiene como foco principal realizar una mejora constante de entrega de valor, lo cual conlleva a automatizar la mayor parte del ciclo de desarrollo desde implementar una nueva característica y enviarla directamente a ambiente de producción.

El equipo de desarrollo crea e implementa nuevas soluciones o aplicaciones y el equipo de operaciones soporta las aplicaciones que están en ambiente de producción, la idea de DevOps es que varias áreas trabajen hacia un mismo

objetivo, por ejemplo, los equipos de control de calidad y de seguridad se integren más con el de desarrollo y operaciones e intervenir durante todo el ciclo de vida de la aplicación.

"*DevOps is a culture or professional movement*" – **Adam Jacob, CTO de Chef**

"*Currently, DevOps is more like a philosophical movement, not yet a precise collection of practices, descriptive or prescriptive*" – **Gene Kim, Founder of Tripware**

"*DevOps es un paso necesario en la evolución del desarrollo de software que de la mano con metodologías ágiles brinda herramientas que permiten a cualquier negocio ganar una dinámica desafiante al cambio.*" – **Fernando Lozano Martínez, Director Transformación Tecnológica, Colpatria del grupo Scotiabank**

¿De dónde viene el nombre DevOps?

En 2009, John Allspaw y Paul Hammond de Flikr entregaron una presentación titulada "*10+ Deploys per Day: Dev and Ops Cooperation at Flikr*" (en español, +10 Despliegues por día: cooperación de Dev y Ops en Flikr) en la conferencia *Velocity* de *O'Reilly*. Esta presentación alineó cómo Flikr había integrado el desarrollo y funciones de operaciones con el fin de simplificar el proceso de lanzamiento y gastos operativos. El título pronto se acortó a #devops en Twitter e inspiró a Patrick DuBois para comenzar la serie devopsdays de conferencias.

¿Qué herramientas utilizan DevOps?

Algunas de las herramientas más empleadas para el desarrollo e implementación en DevOps son:

Docker, Kubernetes, Jenkins, Bamboo, Travis CI, Circle CI, CVS, Git, GitLab, GitHub, Microsoft Team Services, DigitalOcean, Rackspace, Heroku, Microsoft Azure, Amazon Web Services, IBM Bluemix, Vagrant, Ansible, Puppet, Chef, XUnit, JUnit, Selenium, Cucumber, entre otras.

¿Quiénes están haciendo DevOps?

Google, Amazon, Netflix, Spotify, Twitter, Facebook, IBM, CA, SAP, HP, Microsoft, Red Hat, Open Suse, entre muchas otras empresas.

Antecedentes de DevOps

Tradicionalmente el software fue construido por el equipo de Desarrollo, luego otro equipo QA realizaba las pruebas y quienes hacían el soporte y mantenimiento era el equipo de operaciones. Cada uno de estos equipos se centraba en diferentes tareas. Desarrollo estaba enfocado a escribir código e implementar funcionalidades, QA se encargaba de que no hubiera errores en el código y Operaciones se encargaba de que las plataformas se mantuvieran disponibles.

Al trabajar con los equipos por separado se obtuvo como resultado que los lanzamientos a producción eran a menudo lentos, y sólo un pequeño número de despliegues grandes por año se hicieron. La división de la responsabilidad a menudo llevó a señalar con los implicados cuando surgían problemas.

A través de la década de 2000, el movimiento de desarrollo Agile hizo grandes avances en la mejora del proceso por el cual se desarrolló el software. El movimiento rompió las barreras entre los usuarios de negocios, analistas de negocios, desarrollo y control de calidad para desarrollar software de despliegue listo mucho más regular y eficientemente.

Uno de los elementos clave de cualquier proceso ágil es la importancia puesta en el valor creado por cualquier esfuerzo de desarrollo. Crucialmente, ese valor sólo se ve una vez que el software está en producción y en uso por los usuarios finales. Esto conduce a la creación de un flujo constante de piezas de funcionalidad listas para la producción que están a la espera de ser desplegadas. Las pruebas automatizadas y las herramientas de integración continua hicieron que el proceso de validación del software fuera mucho más corto.

El movimiento de DevOps evolucionó para adoptar los mismos tipos de pensamiento en la gestión operativa de los sistemas. Esto incluye no sólo los avances tecnológicos como la virtualización, los sistemas en la nube y el despliegue, la automatización que simplifican el proceso y la repetibilidad del despliegue, sino también los elementos humanos como una mayor integración entre los equipos de desarrollo y operaciones mucho antes en el proceso de desarrollo.

Culture, Automation, Measurement & Sharing

John Willis y Damon Edwards acuñaron el término CAMS: Cultura (*Culture*), Automatización (*Automation*), Medición (*Measurement*) y Compartir (*Sharing*). Son los 4 pilares que en los que se basa DevOps y es un buen concepto para entenderlo desde la perspectiva de alguien que lo practica o un líder de equipo (Kersten, 2017).

Culture: La cultura de DevOps se trata de:

- Comunicación y responsabilidad compartida
- Aceptar el fracaso
- Alineación multifuncional
- Empatía

Frente a la fuerte dimensión cultural y organizacional para resolver el conflicto entre los incentivos para los equipos de desarrollo y de operaciones, DevOps requiere de la comprensión activa y consciente de la responsabilidad compartida colectiva como parte orgánica de la cultura, a la par que aplica principios ágiles a la gestión de infraestructuras que incluye la evaluación objetiva de los fracasos.

Automation: Los enfoques de DevOps implican un alto grado de automatización y dependen de representar una infraestructura de manera similar a un código. El acercamiento entre el desarrollo de la aplicación y el despliegue de la infraestructura permite una capacidad de entrega del software más rápida y con un menor número de errores. La automatización y la infraestructura de autoservicio permiten minimizar el tiempo de ciclo, y con ello realizar una experimentación más rápida y más agilidad para manejar cambios en la dirección, así como mayor fiabilidad y calidad.

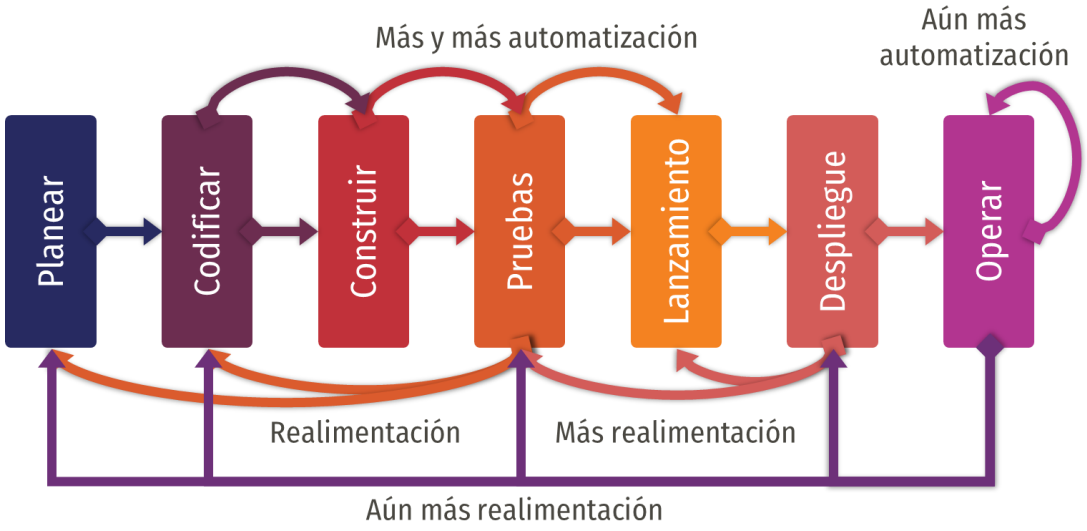
Measurement: Los procesos automatizados permiten mediciones mucho más fiables que conducen a la identificación de cuellos de botella para así poder trabajar en su eliminación o mitigación. Usar DevOps mejora todo el ciclo de vida de entrega de software. Con la declaración de métricas bien entendidas que surgen mediante sistemas automatizados, se contribuye a reducir la fricción entre equipos cuando se investigan problemas.

Sharing: Entre las ventajas de compartir código, herramientas y procesos se encuentra la eficiencia para los diferentes equipos de tecnología en el uso de estas, facilitando un ambiente donde se genera una comprensión más rápida entre las personas y con ello reforzando la empatía en las interacciones al interior de la empresa. Compartir entre equipos es un principio clave de DevOps. Ese intercambio se lleva a cabo en muchos formatos y lugares, y aprende, en experimentos y sus resultados

Ciclo de vida del desarrollo de software usando DevOps

Bajo un modelo de DevOps (ver Figura 23 y Figura 24), los equipos de desarrollo y operaciones ya no están aislados. A veces, los dos equipos se fusionan en uno solo, donde los ingenieros trabajan en todo el ciclo de vida de la aplicación, desde el desarrollo y las pruebas hasta la implementación y las operaciones, y desarrollan una variedad de habilidades no limitadas a una única función.

Figura 23. Ciclo de vida del desarrollo de software usando DevOps



Fuente: Autor

Figura 24. Ciclo infinito de las etapas de DevOps

Fuente: http://www.suse.com/c/can-devops-support-business-agility/

Proyecto con DevOps

Todo proyecto que use DevOps debe tener:

- Pruebas (*Testing*)
- Control de Versiones (*Version Control System* – por su sigla, VCS)
- Integración Continua (*Continuous Integration* – por su sigla, CI)
- Despliegue Continuo (*Continuous Deployment* – por su sigla, CD)

DevOps se asocia a estrategias de transformación digital.

Figura 25. Visión general de DevOps

Fuente: Autor

Para el desarrollo y despliegue de la aplicación web es necesario proveer ambientes de despliegue homogéneos con el fin de evitar errores en dependencias tanto de sistema operativo como de librerías y utilidades (ver Figura 25).

Integración continua

Modelo basado en compilar, y ejecutar pruebas automáticas de todo el proyecto en la menor brevedad. Permite la identificación temprana de errores durante la fase de desarrollo.

Este proceso inicia con una descarga de las últimas fuentes con los cambios implementados desde el control de versiones a un directorio de trabajo, se compila dichas fuentes, se instala en el entorno de desarrollo, se realiza pruebas automáticas para así generar informes.

Despliegue continuo

El despliegue continuo busca automatizar y mejorar el desarrollo de entregas parciales mediante pipelines que se definen como una línea de procesos continua que conecta cada cambio en el repositorio de código fuente con un despliegue en producción (Intelygenz, 2017).

La entrega continua se refiere a la disponibilidad de los cambios para ser desplegado en cualquier momento.

El despliegue continuo se diferencia de la entrega continua en que mientras el primero se realiza de forma automática cuando se han cubierto todos los criterios definidos para la entrada en Producción, el segundo requiere de una aprobación manual antes de implantarse en Producción (Faura, 2017).

Infraestructura como código

Se refiere a la práctica de utilizar scripts o pedazos de código para configurar cómo se quiere la infraestructura del proyecto en vez de realizarlo de forma manual. La infraestructura como código (*Infrastructure as a Code* – por su sigla, IaC) trata la configuración de la infraestructura exactamente como un lenguaje de programación. Las aplicaciones pueden contener scripts que crean y organizan sus propias máquinas. La IaC es el marco del que ha surgido DevOps. La frontera entre el

código que ejecuta la aplicación y el que configura su infraestructura es cada vez más cercana, lo cual implica que los desarrolladores tengan habilidades adicionales para elaborar el diseño de su infraestructura y los scripts que la soporten (Caum, 2017).

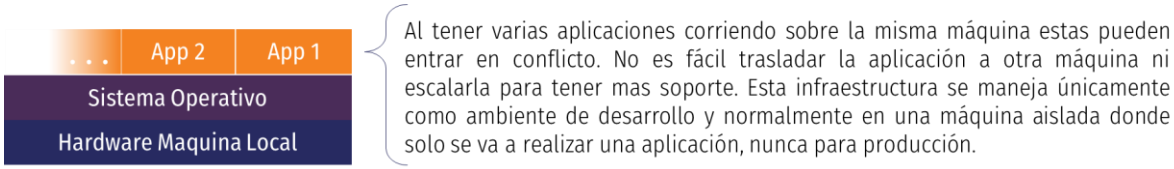
¿Cuáles son las ventajas de la infraestructura como código?

La infraestructura como código permite que las máquinas sean gestionadas de manera programada, lo que elimina la necesidad de realizar configuraciones manuales (y actualizaciones) de componentes individuales de hardware. Esto hace que la infraestructura sea muy flexible, escalable y replicable. Un solo operario puede implementar y gestionar una máquina (o tantas como se requiera) usando el mismo conjunto de código.

Diagramas de infraestructura para una aplicación

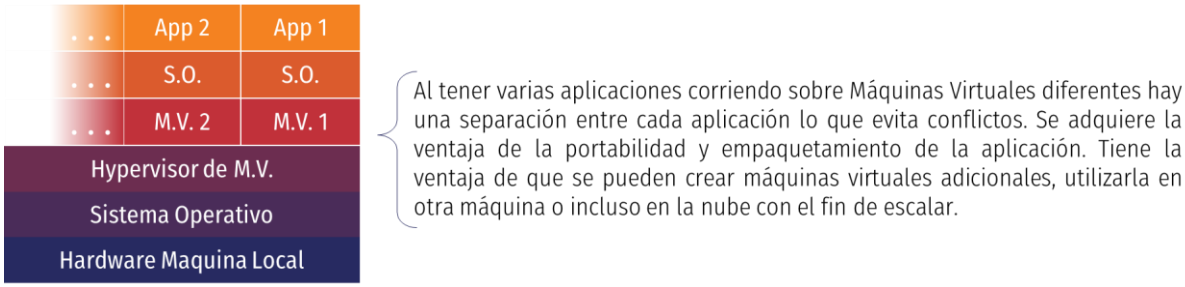
La idea de la IaC es crear scripts para desplegar aplicaciones en cualquiera de las siguientes infraestructuras:

Figura 26. Infraestructura de máquina local (Solo ambiente desarrollo)



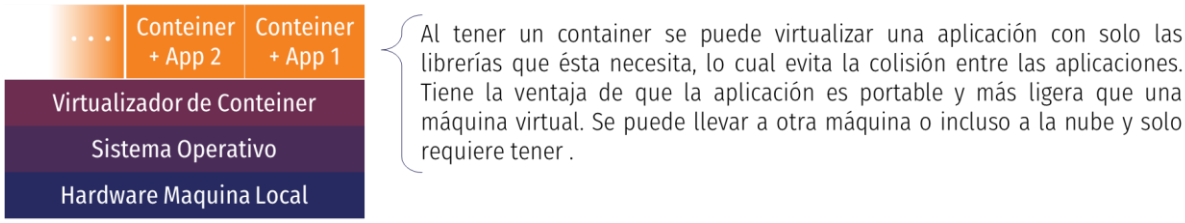
Fuente: Autor

Figura 27. Infraestructura de máquina local con máquinas virtuales



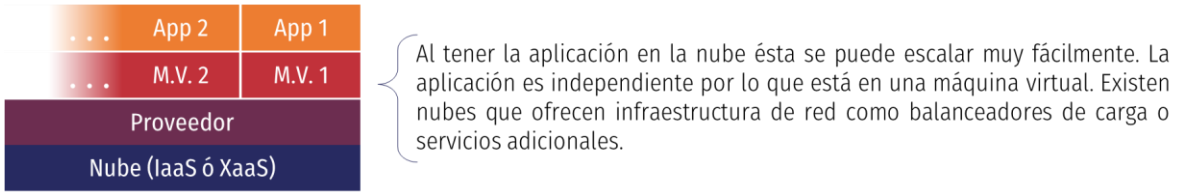
Fuente: Autor

Figura 28. Infraestructura de máquina local con conteiner



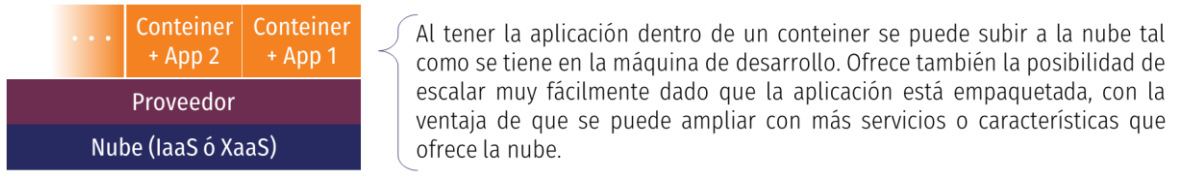
Fuente: Autor

Figura 29. Infraestructura en la nube con máquinas virtuales



Fuente: Autor

Figura 30. Infraestructura en la nube con conteiner



Fuente: Autor

Los diagramas de las Figura 26 a la Figura 30 pueden cambiar ligeramente o hacerse más grandes, dependiendo de las necesidades o características que se quieran tener en la aplicación. Los diagramas aquí descritos son la plantilla de una infraestructura para una aplicación. Al tener la infraestructura como código es posible desplegar la aplicación tanto de forma local como en la nube, básicamente con el mismo script, automatizando de esta manera el proceso de despliegue de la aplicación.

Siendo más específico el Frontend, Backend, Base de Datos y demás componentes de una aplicación web, se pueden tener dentro de la misma máquina, o si se desea, en máquinas separadas para cada componente, todo depende de las necesidades del proyecto.

DEVOPS EN UN PROYECTO DE SOFTWARE

Uso de metodologías ágiles

Entrando en materia, DevOps igual que DDD, se contextualiza mediante el uso de metodologías ágiles, siguiendo algún framework ágil como lo son SCRUM, KANBAN, LEAN los cuales permiten organizar la ruta de trabajo a seguir, describir y organizar el equipo de trabajo y asignando las funcionalidades que se van a desarrollar.

Pruebas

Se puede utilizar los lineamientos de desarrollo dirigido por pruebas (*Test driven development* - TDD) o el que sea determinado por el equipo de trabajo. El objetivo con la construcción de pruebas es tener una base para verificar la integridad del

software, lo cual resulta positivo porque se alinea con el ciclo DDD. El equipo debe tener presente que son importantes las pruebas, pues al hacer trabajo colaborativo puede que lo que desarrolle una persona afecte el comportamiento de lo que desarrolló otra y viceversa. Por este motivo se debe tener una alta cobertura del código con pruebas y de esta manera asegurar la calidad del software cuando esté en la etapa de pruebas e integración. Normalmente se espera tener una cobertura de código del 90%, y para esto existen diferentes herramientas de pruebas que verifican cuánta cobertura de código se tiene.

VCS y trabajo colaborativo

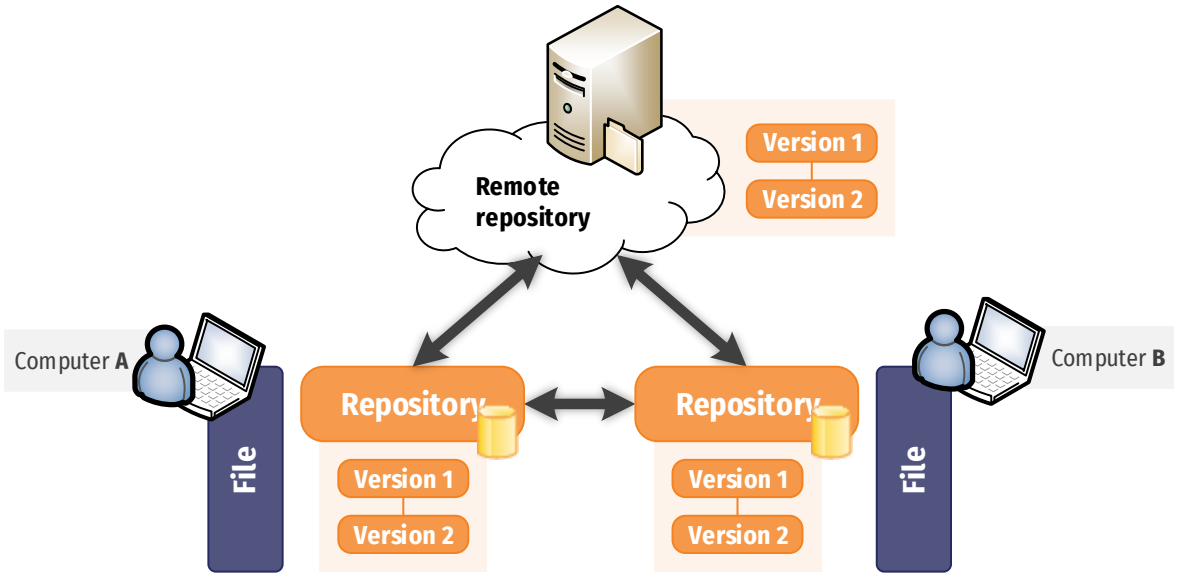
El trabajo colaborativo debe estar soportado en herramientas que permitan este fin. Como se ha visto últimamente, los sistemas de control de versiones (*Version Control System* – VCS) tienen la capacidad de versionar y hacer un gobierno del código y el desarrollo que se hace para un producto de software. Además, puede recibir interacciones de múltiples personas lo cual da paso a trabajar de manera colaborativa.

Todas las personas pueden subir sus fragmentos de código e ir trabajando en paralelo. Estas herramientas también organizan el producto de software según la etapa y el ambiente que estaba trabajando, permite tener un histórico de todos los momentos mientras se estuvo construyendo, devolverse en la historia o avanzar hasta un punto específico que se desea consultar, es decir, tener el estado del proyecto en cualquier momento.

La herramienta más utilizada en el mundo actualmente es Git (ver Figura 31 y Figura 32), porque es un sistema distribuido, no centralizado en el cual todas las personas que trabajan en el repositorio del proyecto manejan la información en su totalidad y por lo tanto pueden actuar como cliente o servidor en cualquier momento, se adapta

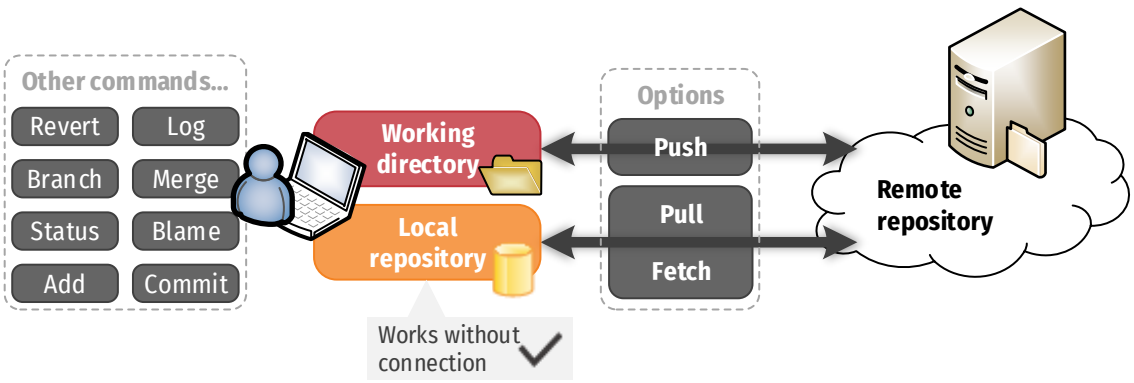
a las necesidades del proyecto y se integra fácilmente permitiendo de esta manera la capacidad de trabajar de manera colaborativa y sin tener la necesidad de estar todo el tiempo conectado a un servidor central.

Figura 31. Sistema de control de versiones distribuido Git



Fuente: Autor

Figura 32. Funciones relevantes de Git

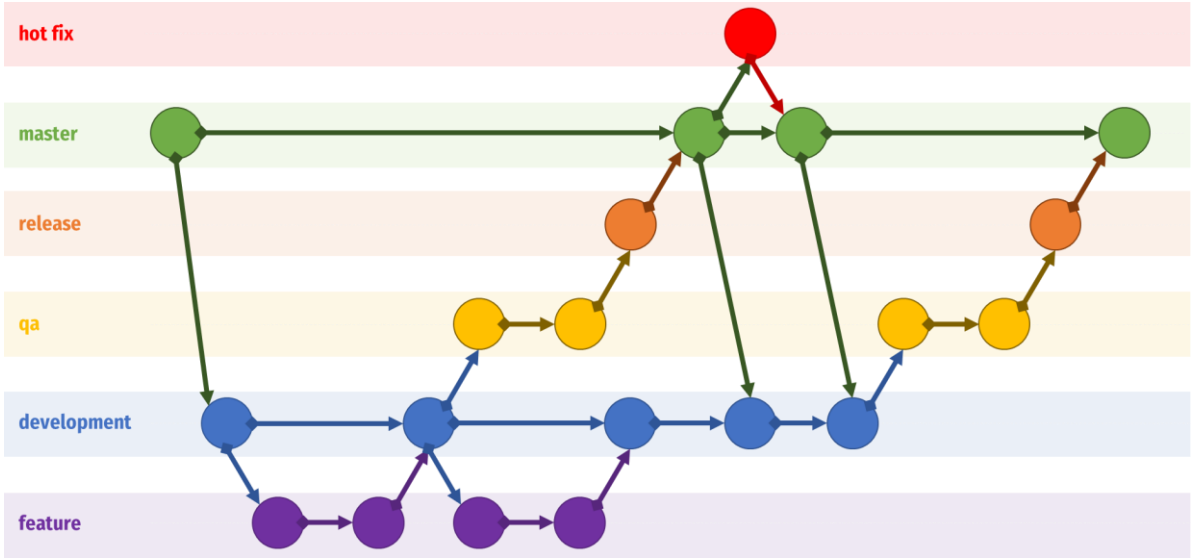


Fuente: Autor

Un proyecto con Git se trabaja siguiendo un Git Flow. Es la manera general en la que se organiza el proyecto para que los equipos de trabajo realicen aportes al

mismo de forma paralela y además tener espacios en el desarrollo del proyecto verificar la calidad del software y hacer entregas de valor. A continuación, se describirá un Git Flow (ver Figura 33) general que se utiliza en el desarrollo de un proyecto. Este diseño no es obligatorio y puede variar dependiendo de la organización planteada por el líder de equipo o proyecto:

Figura 33. Git Flow general



Fuente: Autor

Como se puede observar en la gráfica, se tienen ramadas dedicadas a partes del ciclo de desarrollo, además permite diferenciar el código que se ejecuta en los ambientes y apreciar el histórico de cómo avanza el proyecto. Las ramas tienen las siguientes funcionalidades:

1. **Master:** es la rama donde generalmente se hacen entregas de valor yse versionan los lanzamientos que se han hecho del producto de software. El código de esta rama se envía al ambiente de producción que va a ser usado por el usuario final.

2. **Release:** es la rama donde se ubica el producto de software para probarlo como si ya lo fuera a utilizar el usuario final, pero es antes de enviarlo a ambiente de producción, generalmente se dispone de un ambiente de release para este fin.

3. **QA:** en esta rama se realizan las pruebas de calidad del software, se toma el código de esta rama y se prueba en un ambiente aislado para QA.

4. **Feature:** es la rama donde se implementan características específicas del producto de software, según las metodologías ágiles generalmente aquí se desarrolla una historia de usuario.

5. **Development:** es la rama donde se acumulan las características implementadas de las ramas ‘feature’ y dan paso para hacer lanzamientos y enviar a la rama ‘master’ en producción.

6. **Hot Fix:** esta rama tiene la función de hacer arreglos rápidos al producto de software contenido en la rama ‘master’, se utiliza en caso de emergencia generalmente por un error crítico del producto que necesita ser arreglado rápidamente.

La función de estas ramas y el Git Flow es tener un gobierno completo sobre el código y el desarrollo del producto de software. Esta separación permite integrar de manera controlada los diferentes ambientes y las herramientas de integración y despliegue continuo.

Uso de las herramientas de integración continua

La herramienta de integración continua recoge el código ubicado en el repositorio central del proyecto dependiendo de la rama donde se esté desarrollando. Luego, dispone de un ambiente de integración encargado de verificar las pruebas de código e integración.

Terminadas las pruebas y si fueron satisfactorias, comparte el estado de integración del código y se pasa a la siguiente etapa que corresponde al despliegue del producto mediante las herramientas de despliegue continuo.

Despliegue continuo e Infraestructura como código

Las herramientas de despliegue continuo se encargan de proveer las dependencias del software, disponer el código en el ambiente y ejecutarlo, todo de manera automatizada. Las tareas para disponer recursos como:

- Dependencias
- Código desarrollado para la aplicación
- Librerías
- Herramientas y Frameworks
- Entre otras

Están a cargo de la infraestructura como código (*Infrastructure as a Code* - IaC). Esta es una tarea de análisis y construcción detallada para cada producto de software o aplicación, porque hay que tener presente todo lo que se utilizó durante el ambiente de desarrollo.

Es importante extraer, listar y detallar cada recurso para que se encuentren disponibles, de manera muy similar y ojalá homogéneos, estos recursos en los demás ambientes de despliegue sobre todo teniendo cuidado con el ambiente de producción. Esta condición permitirá probar de una manera más confiable el código del producto y encontrar errores con precisión. La IaC llega como respuesta para dar soporte a las herramientas de integración y despliegue continuo, ya que éstas la van a utilizar dentro de sus scripts de configuración.

Al disponer de la IaC se puede crear productos de software de manera escalable puesto que las herramientas de despliegue continuo permiten describir dicha infraestructura. Además, actualmente los productos de software se despliegan en producción directamente en la nube, muchas de estas permiten crear infraestructuras grandes y escalables, lo cual ofrece una ventaja adicional para la operación de la aplicación.

DevOps Pipeline

A partir de ahora llega el concepto de DevOps Pipeline, para mostrar a un nivel de detalle más alto y describiendo las tareas que tienen todos en conjunto:

- El sistema de control de versiones
- Las herramientas de integración y despliegue continuo
- La infraestructura como código

todo contextualizado en las metodologías ágiles, y en lo que se enmarca la cultura de DevOps

PRODUCTO MINIMO VIABLE

Para la identificación del Producto Mínimo Viable (*Minimum Viable Product – MVP*) de la aplicación o prototipo a desarrollar se debe seguir un proceso de análisis y documentación que se separa en 7 etapas:

1. **Historias de usuario y proto-personas:** Se narra una historia que contextualiza por qué una posible persona o proto-persona, haría uso de este producto.

2. **Customer Journey:** Se diseña el flujo de acciones, etapas o elementos por los que atraviesa un usuario dentro de la aplicación.

3. **Descripción del prototipo:** Objetivo principal del prototipo junto a una breve descripción de la necesidad que va a satisfacer o solucionar, los stakeholders involucrados.

4. **Lista de funcionalidades:** Se formaliza la lista de funcionalidades que se encuentran en la historia que se escribieron sobre las proto-personas y se prioriza calificando con los siguientes niveles:
    a. **Must:** Lo que sí o sí debe tener el producto.
    b. **Should:** Lo que debería tener el producto.
    c. **Nice to have:** Lo que sería bien que tuviera el producto, pero no es prioridad. Siempre que se piense en funcionalidades se debe pensar a nivel de desarrollo el esfuerzo que implican.

5. **Diagramación:** Se diagrama el modelo conceptual y de datos a partir de las funcionalidades priorizadas satisfacerlas con un modelo conceptual que soporte ese prototipo. Es muy importante que el modelo conceptual siga los principios SOLID, para que implementar nuevas funcionalidades en las

nuevas iteraciones del ciclo DDD, sea más sencillo y no requiera una refactorización del modelo.

6. **Contratos de servicio:** Se genera un contrato entre las partes Frontend y Backend donde se define el protocolo con el que se va a enviar y recibir datos por cada uno de los servicios implementados.

7. **Stack:** Conjunto de herramientas que utilizará el prototipo, que van a satisfacer las necesidades de la aplicación.

8. **Interfaces:** Se diseñan las interfaces básicas de usuario que tendrá la aplicación web.

Anexo B PoC DevOps pipeline y enfoque DDD con un servidor.

La idea u objetivo de la prueba de concepto es desplegar un servidor web simple con una página web utilizando herramientas de DevOps, este servidor se debería poder volver a desplegar cumpliendo con el ciclo de DevOps y DDD.

Según el marco de referencia en DevOps se crea un Pipeline el cual contiene la automatización de integración y despliegue automático a producción de un servidor web. Así surge una nueva pregunta ¿Cómo se crea un DevOps Pipeline? ¿Cuál es la secuencia para su uso? Explorando el código fuente de proyectos implementados en DevOps y observando la experiencia de otros desarrolladores en el área, un proyecto que utilice DevOps se consolida desde la etapa de diseño y configuración inicial. Un ingeniero de DevOps crea e implementa las configuraciones necesarias para la utilización de herramientas de integración continua y despliegue continuo.

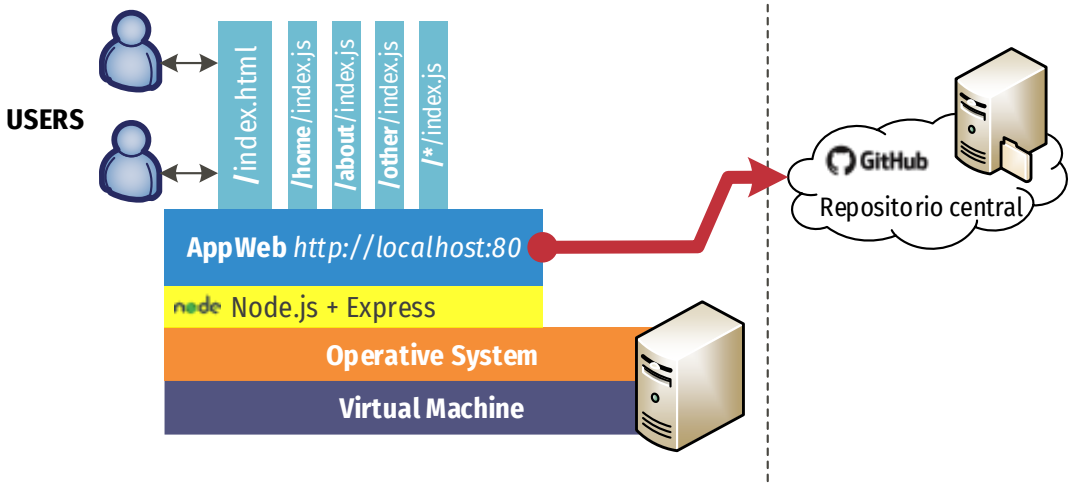
Para el ambiente de desarrollo incluye un servidor web de Node.js utilizando el lenguaje de programación JavaScript por la facilidad con que permite crear un servidor web, el cual contenía toda la lógica que tiene una aplicación web simple. Se expone en el URI *localhost* por el puerto 80 que carga una página HTML, la cual se invoca a través del protocolo HTTP en método GET al '*endpoint*' "/" desde un navegador web, respondiendo con el texto "Hola Mundo" (ver Figura 34). Se realiza sin componentes adicionales solo para probar que el servidor está activo y es accesible la aplicación.

Con el fin de cumplir los lineamientos de DevOps todos los desarrolladores deben tener ambientes de desarrollo homogéneos. Por tal motivo, se integró el uso de una herramienta llamada Vagrant. Ésta tiene la capacidad de crear imágenes de Máquinas Virtuales, que pueden ser reutilizadas cuantas veces se requiera. Mediante esta herramienta se creó una imagen (*image box*) de la máquina virtual base con todos los programas, librerías y dependencias descritos anteriormente. De

esta manera, cualquier desarrollador que desee hacer parte del proyecto utilizaría esta imagen para crear una máquina virtual, la cual funcionará como su ambiente de desarrollo.

Para el control de código de la aplicación, versionado, y en caso de tener un equipo de desarrollo surge la necesidad de tener un sistema de control de versiones. El más utilizado actualmente en la industria es Git (también se pueden utilizar otros como VCS), el cual controla los cambios del código desarrollado, se puede organizar por ramas (usando la estrategia del Git Flow) guardando la traza de quién lo realizó. Generalmente se crea un repositorio central en la nube, donde se usó GitHub con un repositorio público (ver Figura 34).

Figura 34. Ambiente de desarrollo e integración del repositorio central.

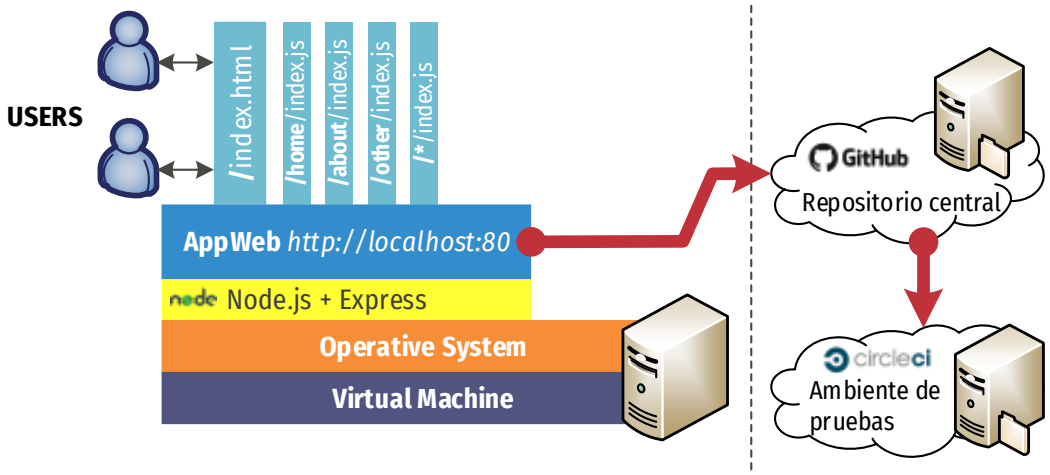


Fuente: Autor

Durante la ejecución del proyecto diferentes desarrolladores van realizando haciendo aportes e implementando nuevas características de la aplicación. Surge la necesidad de: ¿Cómo verificar la calidad e integridad a nivel de toda la aplicación? Esto se resolvió de dos maneras en conjunto, primero creando pruebas unitarias por cada desarrollador, para este caso como solo se tenía un desarrollador se aplicó

la metodología de TDD, con el fin de hacer pruebas unitarias a medida que se estuviera desarrollando.

Figura 35. Ambiente de desarrollo con la herramienta de integración continua.



Fuente: Autor

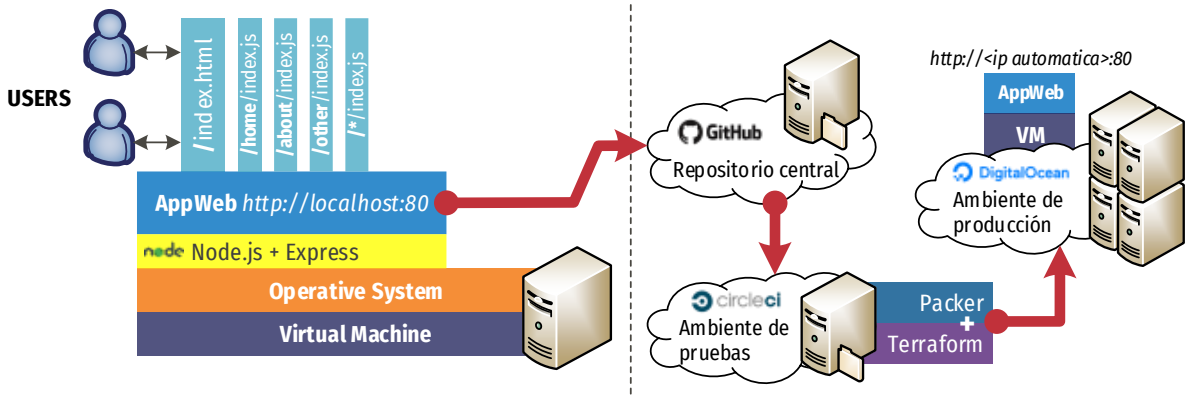
DevOps tiene como objetivo principal la automatización de la mayor parte del ciclo de desarrollo de software; por esta razón, como segunda parte de la solución a la calidad del proyecto se decide utilizar una herramienta de integración continua, que de manera automatizada recogiera el código depositado en el repositorio central y ejecutara las pruebas creadas por los desarrolladores en un ambiente pruebas (ver Figura 35).

En este caso, se utilizó la herramienta Circle CI, la cual requería la configuración de un script llamado circle.yml que describe el ambiente de pruebas donde se ejecutará el proyecto. Esta es una herramienta en la nube, y se conecta al repositorio central de GitHub extrayendo la última versión del proyecto. Adicionalmente, Circle CI tiene una interfaz que permite monitorear: la creación del ambiente, la ejecución las pruebas y el resultado final de ejecución, el cual es un indicador que muestra la integridad del proyecto.

Hasta el momento el proyecto solo se puede tener en el ambiente de desarrollo y pruebas del equipo. Ahora con el motivo de publicar el proyecto en internet y completar el objetivo de automatizar el proceso ¿Cómo desplegar la aplicación de manera automática del ambiente de desarrollo a un ambiente de producción para que lo utilicen los usuarios finales? Para esto se identificaron otras herramientas que vienen a complementar el proceso. Packer, Terrafrom y la nube Digital Ocean brindan de forma combinada una solución a este problema, (se identificaron otras que cumplen objetivos similares tales como Chef, Puppet, Ansible o Jenkins).

A través de Packer se realizó la creación de una entrega (o *release,* en inglés*)* de la aplicación con su infraestructura, es decir, con el código del proyecto, las librerías y dependencias empaquetado en una imagen de sistema operativo para instalar en una máquina virtual que se podrá desplegar en cualquier nube de tipo IaaS o PaaS. Packer es capaz de desplegar en diferentes nubes como AWS, Azure, Google Cloud Platform o Digital Ocean, entre otros, para esta prueba se usa Digital Ocean y la imagen es almacenada en la misma nube.

Figura 36. Integración de Terraform y DevOps pipeline final.



Fuente: Autor

Por otra parte, Terraform es una herramienta que permite describir la infraestructura (*Infrastructure as a code, IaC*) que se usará en la nube para la aplicación, también permite integrarse con diferentes plataformas en la nube, y tiene la posibilidad de recoger un release generado por Packer y ejecutarlo en la nube, integración que facilitó de manera automática el despliegue a producción en Digital Ocean. Se añadieron reglas de Firewall y se desplegó el release en una sola máquina, arrojando como resultado satisfactorio porque se pudo acceder a la IP del servidor y obtener respuesta del mismo visualizando la aplicación web de ejemplo, en un navegador web (ver Figura 36).

Anexo C MVP Framework FWD.

DESCRIPCIÓN DEL FRAMEWORK

La descripción y detalles del framework a construir se encuentran en subcapítulo 4.1 Descripción y objetivo del framework.

STAKEHOLDERS DEL FRAMEWORK

Los siguientes son los roles implicados en la aplicación web.

Tabla 1. Stakeholders del framework

Stakeholder	Descripción
FWD	Aplicación web con un Login de Usuarios y operaciones CRUD para los usuarios, con DDD y DevOps.
Usuario	Persona que va a usar el framework
Desarrollador	Persona que realiza el desarrollo del framework

Lista de funcionalidades

A continuación, se describe una lista completa de las funcionalidades extraídas de las historias de usuario. Al final en una tabla se encuentra un resumen con todas las funcionalidades y su respectiva priorización.

Tabla 2. Detalle funcionalidad 1 FWD

Funcionalidad:	*Usuario se registra en la página.*	**id:**	**F-01**
Stakeholders:	Usuario, Framework		
Descripción:	El usuario ingresa a la aplicación y se le solicita un Nombre de usuario, Correo Institucional y una contraseña a través de un formulario.		
Restricciones:	- El usuario debe tener un correo institucional. - No puede haber dos usuarios con el mismo correo o nombre de usuario.		
Entrada:	Ingresa al módulo de registro de la aplicación, y envía su correo institucional y una contraseña.		
Salida:	El usuario es registrado en la base de datos, se inicia su sesión y es redirigido al Inicio de la aplicación.		
Salida negativa:	- El usuario no se puede registrar porque ya se encuentra registrado, y es notificado con el error. - El usuario ingresó un correo o contraseña de forma errónea, y es notificado con el error. - La aplicación presenta fallas, o no se encuentra en línea, en el dispositivo del cliente.		

Tabla 3. Detalle funcionalidad 2 FWD

Funcionalidad:	*Usuario inicia sesión en la página.*	**id:**	**F-02**
Stakeholders:	Usuario, Framework		
Descripción:	El usuario ingresa a la aplicación y se le solicitan los datos de Correo Institucional y la contraseña.		
Restricciones:	- El usuario debe estar registrado en la aplicación. - El usuario debe digitar correctamente los datos solicitados.		
Entrada:	Ingresa al módulo de inicio de sesión, y envía su correo institucional y una contraseña.		

Salida:	El usuario se valida en la base de datos, se inicia su sesión y es redirigido al Inicio de la aplicación.
Salida negativa:	- El usuario no puede iniciar sesión porque digitó mal sus datos de registro. - La aplicación presenta fallas, o no se encuentra en línea en el dispositivo del cliente.

Tabla 4. Detalle funcionalidad 3 FWD

Funcionalidad:	*Usuario consulta su perfil.*	**id:**	**F-03**
Stakeholders:	Usuario, Framework		
Descripción:	El usuario inicia sesión en la aplicación, ingresa su perfil de usuario donde puede visualizar sus datos personales y las fotos que ha compartido.		
Restricciones:	- El usuario debe haber iniciado sesión en la aplicación.		
Entrada:	Ingresar a su perfil personal.		
Salida:	Se muestran los datos personales almacenados en la base de datos, y se listan las imágenes que ha compartido.		
Salida negativa:	- El usuario no puede visualizar su perfil si no ha iniciado sesión. - La aplicación presenta fallas y no está en línea.		

Tabla 5. Detalle funcionalidad 4 FWD

Funcionalidad:	*Usuario modifica sus datos personales.*	**id:**	**F-04**
Stakeholders:	Usuario, Framework		
Descripción:	El usuario inicia sesión en la aplicación, ingresa al módulo para la edición de datos personales.		
Restricciones:	- El usuario debe haber iniciado sesión en la aplicación.		
Entrada:	Ingresar al módulo de edición de datos personales. Modificar sus datos y guardar los cambios.		

Salida:	Se actualizan los datos en la base de datos y se redirige al perfil de usuario para validar que estén correctos.
Salida negativa:	- El usuario modificó los datos y hay un error, por duplicidad de información o no la envío con los formatos especificados. - El usuario no puede cambiar los datos sin haber iniciado sesión. - La aplicación presenta fallas y no está en línea.

Tabla 6. Detalle funcionalidad 5 FWD

Funcionalidad:	*Usuario elimina su perfil y datos personales.*	**id:**	**F-05**
Stakeholders:	Usuario, Framework		
Descripción:	El usuario inicia sesión en la aplicación, ingresa al módulo para eliminar el perfil de usuario, debe digitar la contraseña y aceptar que va a eliminar su usuario. Posteriormente el usuario es eliminado de la base datos con todos los datos que ha compartido.		
Restricciones:	- El usuario debe haber iniciado sesión en la aplicación.		
Entrada:	Ingresar a su perfil personal.		
Salida:	Se muestran los datos personales almacenados en la base de datos, y se listan las imágenes que ha compartido.		
Salida negativa:	- El usuario no puede visualizar su perfil si no ha iniciado sesión. - La aplicación presenta fallas y no está en línea.		

Consolidado final con su respectiva priorización. Las categorías son: *Must*, *Should* y *Nice to have*.

Tabla 7. Priorización de funcionalidades

Código	Funcionalidad	Priorización
F-01	*Usuario se registra en la página.*	**Must**
F-02	*Usuario inicia sesión en la página.*	**Must**
F-03	*Usuario consulta su perfil.*	**Must**
F-04	*Usuario modifica sus datos personales.*	**Should**
F-05	*Usuario elimina su perfil y datos personales.*	**Should**

Solo se van a desarrollar las funcionalidades que fueron clasificadas con la categoría '*Must*' y '*Should*' durante el ciclo de desarrollo. Todos los pasos siguientes se harán con base en estas funcionalidades seleccionadas. Es posible que el modelo conceptual, la base de datos, los contratos de servicio e interfaces de usuario se rediseñen para que las demás funcionalidades se implementen en ciclos posteriores de desarrollo.

STACK

Según los diagramas y la descripción se propone utilizar, tanto en ambiente de desarrollo como en producción, el siguiente stack de tecnologías:

Backend

Java JDK 8, Maven y Spring Framework.

Java es un lenguaje que ha sido ampliamente utilizado para el desarrollo de aplicaciones, tiene un buen soporte, documentación, una amplia cantidad de librerías, y es multipropósito. Maven es el gestor de dependencias de Java y

Spring Framework, es utilizado para facilitar la creación de aplicaciones web y empresariales. Además, fue elegido como requerimiento solicitado por el director de proyecto utilizar esta tecnología en el proyecto.

Frontend

JSON, HTML5, CSS3, JavaScript ES7 y React/Redux con Node.js

JavaScript es simple, flexible y rápido, es el lenguaje dominante para el Frontend porque es el único que funciona en los navegadores y además se adapta a las necesidades como animaciones, desplazamiento y comportamiento en la visualización de información. Está acompañado de la librería de React, la cual brinda la posibilidad de crear una interfaz en Web por componentes y está soportada por el lenguaje JSX, que es un lenguaje que combina JavaScript y HTML al mismo tiempo. Para los estilos se utilizará CSS que va integrado en el HTML.

Base de datos

DBMS MySQL Server 5.7

Es un motor de bases de datos relacional, utiliza lenguaje SQL para realizar acciones y manejar los datos para la aplicación.

Se utilizarán ambientes virtualizados con todos los programas y dependencias instaladas que requiere la aplicación. Además, según los diagramas y la descripción se propone utilizar, tanto en ambiente de desarrollo como en producción, el siguiente stack de tecnologías:

Infraestructura

- Cloud PaaS DigitalOcean
- Oracle Virtual Box

- Vagrant
- Travis CI
- Packer
- Terraform
- Sistema Operativo Windows Desktop 10 Pro x64
- Sistema Operativo Ubuntu Desktop 14.04.5 LTS x64
- Sistema Operativo Ubuntu Server 14.04.5 LTS x64

DigitalOcean es una nube de plataforma como servicio (*Platform as a Service - PaaS*) que ofrece máquinas virtuales llamadas droplets, firewalls, balanceadores de carga, manejo de snapshots, redes privadas y públicas, entre otras. Una plataforma donde se puede desplegar un producto completo con seguridad y la confianza que ofrece la nube, sin tener la necesidad de comprar todos los dispositivos y servidores físicos dedicados.

Git/GitHub como repositorio central de código para el proyecto del framework

Travis CI es una herramienta de integración continua. Se encarga de capturar automáticamente el código del repositorio principal del proyecto, disponer un ambiente de pruebas y verificar ejecutar las pruebas unitarias y de integración. Puede ser empleada con una gran mayoría de lenguajes utilizados en el mercado como lo son: Java, JavaScript con Node.js, Android, C, C#, C++, Erlang, Go, Groovy, Objective-C, PHP, Python, R, Ruby, Swift, Visual Basic, entre otros.

¿La aplicación requiere una base de datos tipo SQL o NoSQL? ¿Requiere conectarse a servidores o recursos externos?

Para el desarrollo de la aplicación requiere una base de datos SQL, es simple sencillo y es viable para esta pequeña aplicación solo con el objetivo de probar el framework. Necesitará conectarse con un servicio para guardar archivos en la nube.

¿Cuál es la disponibilidad que deberá tener la aplicación, en términos de tiempo y recursos?

Debe estar disponible para los usuarios en todas las horas del día, 7 días a la semana, y requerirá de 3 máquinas virtuales con 2 balanceadores de carga.
El código de la aplicación se encontrará disponible con propósito educativo.

¿Cuál es la seguridad que deberá tener la aplicación?

La aplicación va a manejar el uso de autenticación de usuarios mediante alguna estrategia como JWT o Auth0.

Los servidores van a tener acceso por el puerto 22 de SSH, pero va a estar restringido porque la autenticación se hará mediante el par de llaves SSH pública y privada, con algoritmo RSA de 4096 bits. La disponibilidad del API en el servidor de Backend va a estar en una red privada para que el servidor de Frontend solo se conecte por esa IP y puerto de comunicación, asegurando la autenticación. Componentes tales como Real-Time, Sockets, etc., son módulos opcionales como ocurre en el caso de conectarse a API’s de otros sistemas.

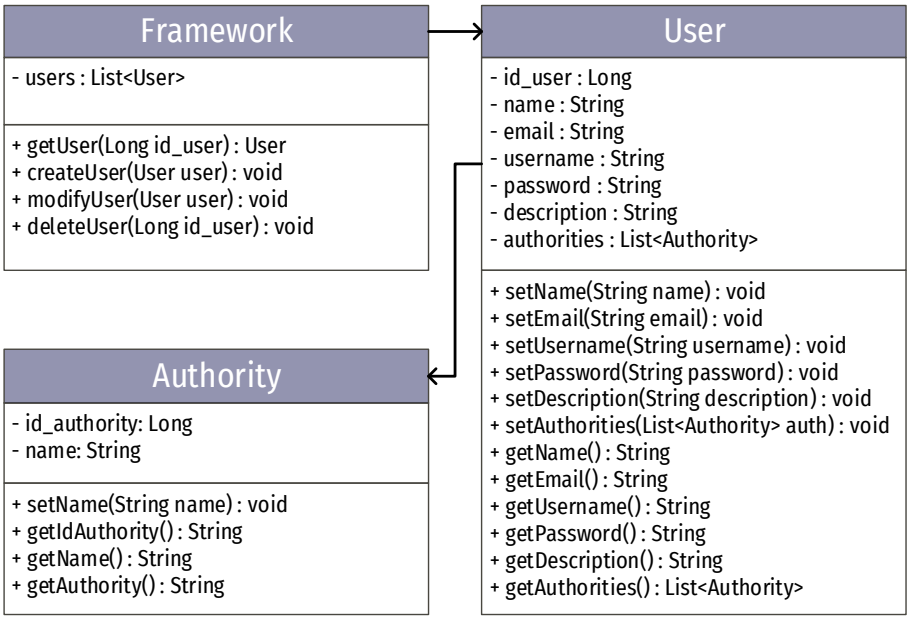
Con el fin de tener un buen control, acceso y uso de los datos se requiere contar siempre con una base de datos en un motor de base de datos y, aún más, teniendo en cuenta que la cantidad de datos puede crecer rápidamente.

DIAGRAMACIÓN

Diagrama conceptual

El siguiente diagrama de la Figura 37, satisface las necesidades y funcionalidades identificadas que tiene el framework. Incluye a los stakeholders involucrados y además cumple con todos los principios SOLID.

Figura 37. Diagrama conceptual FWD



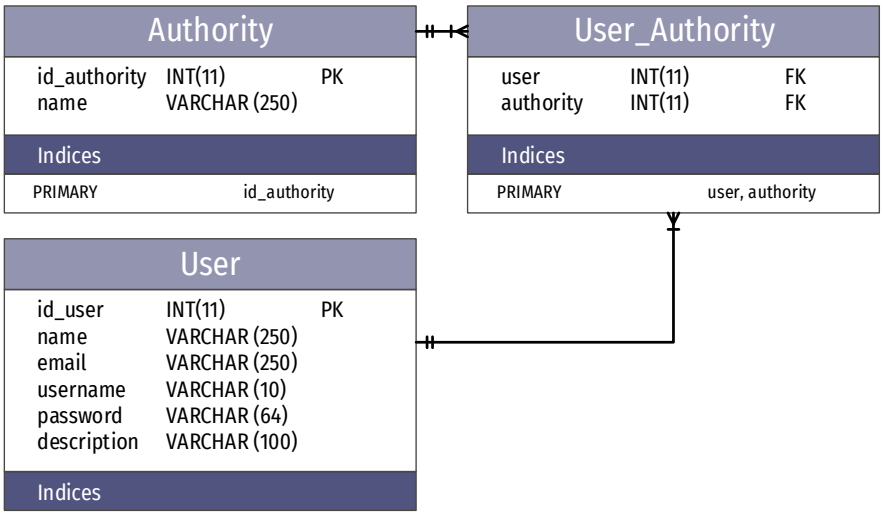
Fuente: Autor

Al desarrollar el framework la clase Framework del modelo desaparece (además no se tiene en cuenta en el modelo de datos) porque a cambio de esta, la clase es reemplazada por las 3 capas presentadas en la arquitectura de N-capas: Persistencia, Aplicación y Presentación, convirtiéndose en un conjunto de clases, transformando sus atributos en datos de la base de datos sus métodos en DAO's, servicios en la capa de aplicación y controladores que al final estos proporcionan un API que utilizará en Frontend para representar la vista del framework.

Diagrama de datos

Como se va a construir un framework que va a servir de base para futuras aplicaciones se requiere el almacenamiento de los datos básicos de los usuarios. Por este motivo se propone el siguiente diagrama del modelo de datos creado a partir del modelo conceptual (ver Figura 38).

Figura 38. Diagrama de datos FWD



Fuente: Autor

En caso de que se requiera hacer muchos cambios al modelo de datos sería también viable utilizar una Base de datos NoSQL. Pero para este MVP no se van a realizar cambios en el modelo de datos planteado para ampliar más funcionalidades, por este motivo una Base de datos SQL, funcionará perfectamente.

CONTRATOS DE SERVICIO

Los contratos de servicios indican las funcionalidades y datos que se van a utilizar en la aplicación, definen el protocolo con el que se envían y reciben datos. Los contratos de servicio ofrecen una descripción para permitir que posteriormente se

desarrollen por aparte el Frontend y el Backend, ya que con estas especificaciones ambos dominios se construirán e interactuarán mediante este intercambio de datos. Es importante resaltar que esta será el servicio ofrecido por la aplicación a los diferentes stakeholders. Los contratos de servicios van a operar o funcionar sobre la capa de API en el Backend Server y en el Frontend Server. Los siguientes son los roles para la autenticación de la aplicación para aumentar el nivel de seguridad de acceso a los datos, son descritos a continuación:

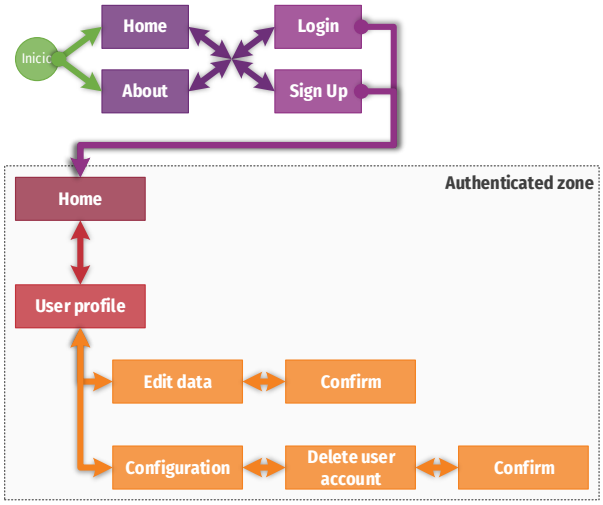
Tabla 8. Codificación de roles para la aplicación

Roles de la aplicación	Codificación
User	USER
Admin	ADMIN
Frontend Request	FRONTEND_REQUESTS

CUSTOMER JOURNEY

Estas serán las interfaces por donde un usuario se moverá a través de la aplicación base que va a tener el framework (ver Figura 39).

Figura 39. Customer Journey FWD



Fuente: Autor

Tabla 9. Contratos de servicio para la aplicación FWD

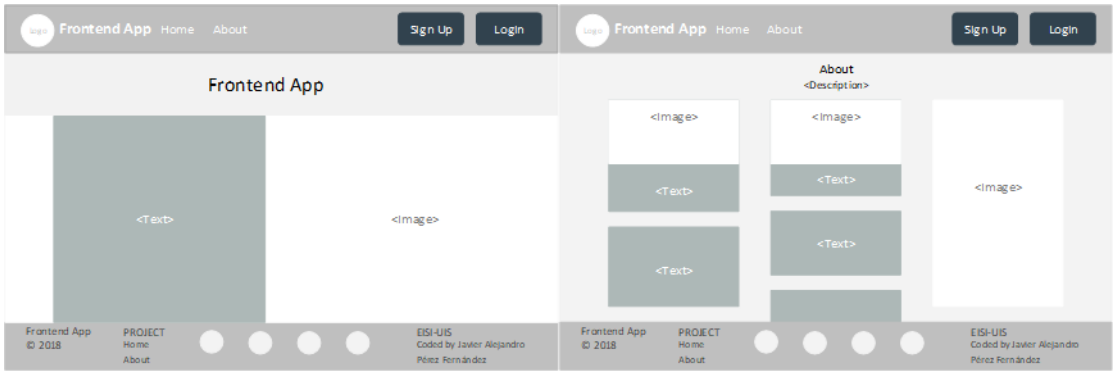
Contratos de servicio									
Server	**Endpoint**	**Subendpoint**	**Método REST**	**Descripción**	**Roles Autorizados**	**Entradas**	**Formato de Entrada**	**Salidas**	**Formato de Salida**
Backend Server (Versión API : /v1)	**/**	**/**	**GET**	Home del Backend Server	Sin autenticación	HTTP Request	N/A	Home Page	N/A
	/auth	**/auth/login**	**POST**	Genera el token de seguridad, para acceso al API de la aplicación	Sin autenticación	En el body del HTTP Request: Username y Password	x-www-form-urlencoded	En el body del HTTP Response: { access_token, expiration_in }	JSON
		/auth/refresh	**GET**	Vuelve a generar un nuevo token de seguridad, para ampliar el service time out	Sin autenticación	En el header del HTTP Request: Access Token	application/json	En el body del HTTP Response: { access_token, expiration_in }	JSON
	/user		**GET**	Buscar usuario por name o username y retorna el listado de User encontrados	USER	En el body del HTTP Request: name o username	application/json	En el body del HTTP Response: Array JSON de User o Error no encontró resultados	JSON
			POST	Crea un nuevo User	FRONTEND _REQUESTS	En el body del HTTP Request: User (a crear)	application/json	En el body del HTTP Response: User (si se creó el user) o Error si no se pudo crear el user	JSON
			PATCH	Modifica los datos de un User existente	USER	En el body del HTTP Request: User (modificado)	application/json	En el body del HTTP Response: User (si se modificó el user) o Error si no se pudo modificar el user	JSON
			DELETE	Elimina un User existente	USER	En el body del HTTP Request: User (a eliminar)	application/json	En el body del HTTP Response: User (si se eliminó el user) o Error si no se pudo eliminar el user	JSON
		/user/{id}	**GET**	Encuentra el user por su ID de la base de datos y retorna el objeto User con todos sus datos correspondientes	ADMIN	En el header del HTTP Request: Access Token En la url del HTTP Request: id	application/json	En el body del HTTP Response: User: { idUser, name, email, username, password, description, authorities: [ ] } o Error de consulta	JSON
		/user/username	**GET**	Valida si ya existe el username en la aplicacion	FRONTEND _REQUESTS	En el header del HTTP Request: Access Token En la url del HTTP Request: Username	application/json	En el body del HTTP Response: Username o Error de consulta	JSON

Server	Endpoint	Subendpoint	Método REST	Descripción	Roles Autorizados	Entradas	Formato de Entrada	Salidas	Formato de Salida
Backend Server (Versión API : /v1)	**/users**		GET	Obtiene la lista completa de los user que se encuentren en la aplicación	ADMIN	HTTP Request	application/json	En el body del HTTP Response: Array JSON de User	JSON
	/whoiam		GET	Obtiene los datos del User actual, validando la identidad mediante el token de seguridad	USER	HTTP Request	application/json	En el body del HTTP Response: User: { idUser, name, email, username, password, description, authorities: [ ] } o Error de validación de usuario	JSON
Frontend Server (/api)	**/**		GET	Application build	Sin autenticación, todos son permitidos	HTTP Request	N/A	Web Application Home Page	N/A
	/auth	**/auth***		Solicitud HTTP con proxy a Backend Server	Sin autenticación, todos son permitidos	HTTP Request	N/A	Solicitud HTTP con proxy a Backend Server	N/A
	/v[0-9]+	**/v[0-9]+/***		Solicitud HTTP con proxy a Backend Server	Sin autenticación, todos son permitidos	HTTP Request	N/A	Solicitud HTTP con proxy a Backend Server	N/A
	/socket			Solicitud HTTP con proxy a Backend Server	Sin autenticación, todos son permitidos	HTTP Request	N/A	Solicitud HTTP con proxy a Backend Server	N/A
	/user		POST	Función para crear un User, consumiendo el servicio del Backend	Sin autenticación, todos son permitidos	En el body del HTTP Request: User (a crear)	JSON	En el body del HTTP Response: User (si se creó el user) o Error si no se pudo crear el user	JSON
		/user/username	GET	Función para validar si ya existe el username, consumiendo el servicio del Backend	Sin autenticación, todos son permitidos	En el header del HTTP Request: Access Token En la url del HTTP Request: Username	JSON	En el body del HTTP Response: Username o Error de consulta	JSON

INTERFACES

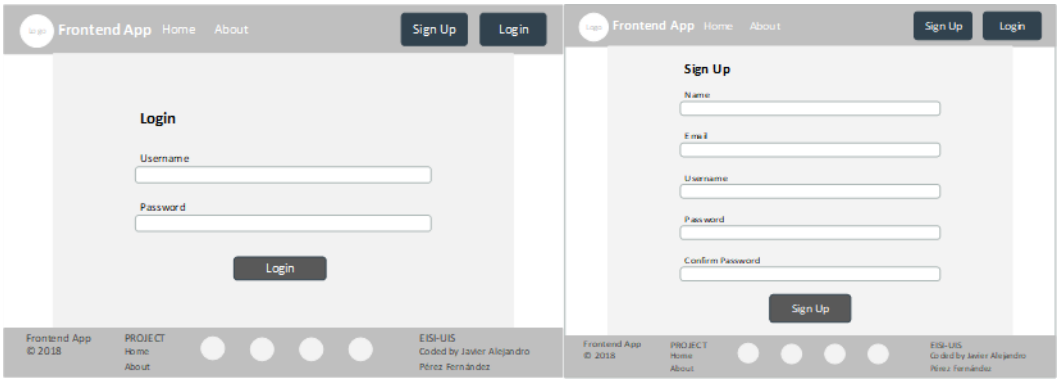
Los siguientes bosquejos de la Figura 40 a la Figura 45 muestran interfaces de la aplicación base:

Figura 40. Maqueta de Home and About



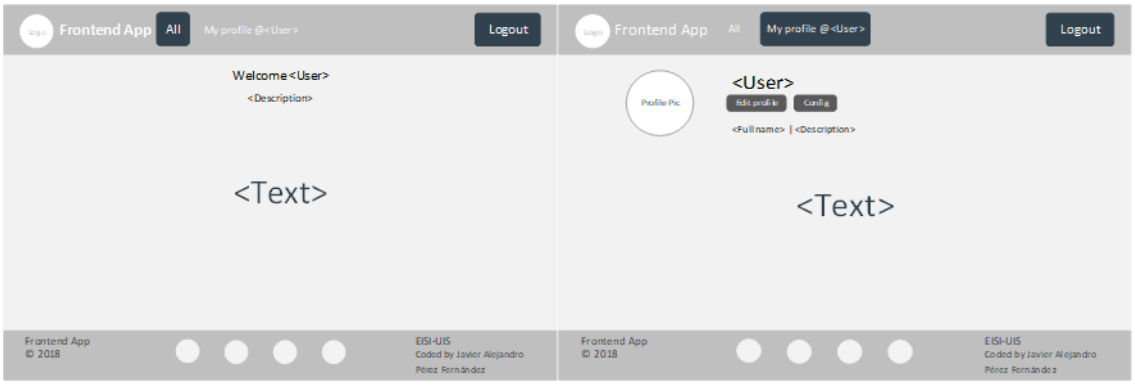
Fuente: Autor

Figura 41. Maqueta de Login and Sign Up



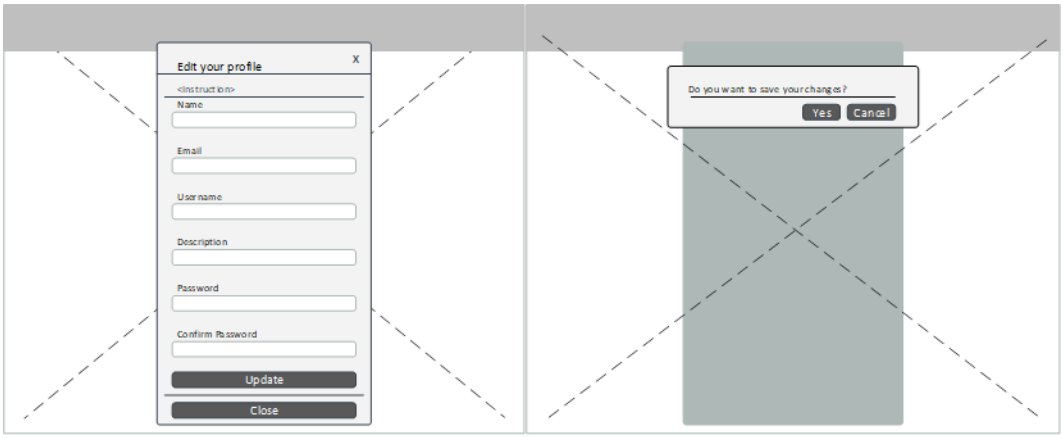
Fuente: Autor

Figura 42. Maqueta de Home (Authenticated) and User Profile



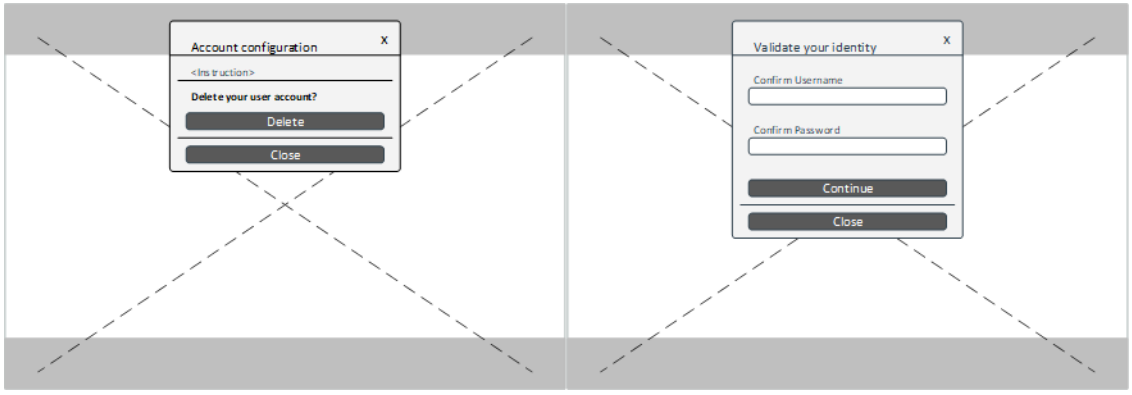
Fuente: Autor

Figura 43. Maqueta de Edit your profile Modal and Confirm edit Modal



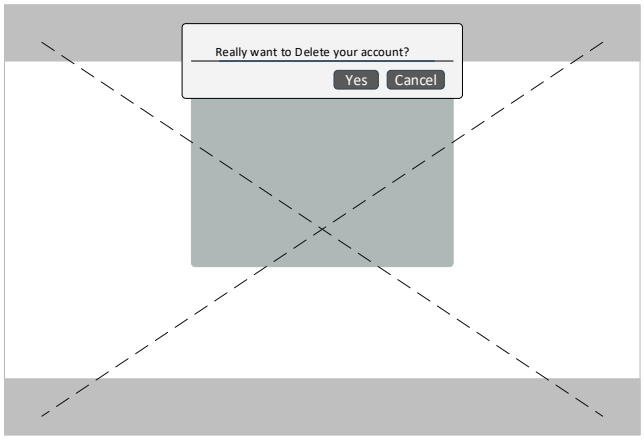
Fuente: Autor

Figura 44. Maqueta de Configuration Modal and Delete user account Modal



Fuente: Autor

Figura 45. Maqueta de Confirm delete user account Modal



Fuente: Autor

Anexo D MVP Prototipo UPost.

DESCRIPCIÓN Y OBJETIVO DEL PROTOTIPO UPost

El prototipo UPost es una aplicación web con propósito educativo, para compartir publicaciones entre los miembros de una comunidad académica. En esta aplicación cada uno de los miembros tiene su propio perfil, donde puede compartir fotos y un Home donde puede ver las fotos que comparten los demás. El objetivo a nivel técnico es utilizar el FWD construido aprovechando las ventajas que ofrece y ampliando algunas funcionalidades para adaptarlo a las necesidades propias de la construcción de este prototipo, como es el uso de un servidor de estáticos en la nube. Este prototipo está limitado a la capa funcional de cara a los usuarios, no tiene un BackOffice para el control y gestión administrativa. El objetivo a nivel funcional es definir la estrategia para el uso del FWD, con el fin de que sea replicado en otros proyectos que lo requieran.

STAKEHOLDERS DE UPost

Los siguientes son los roles implicados en la aplicación web.

Tabla 10. Stakeholders de UPost

Stakeholder	Descripción
UPost	Aplicación web donde se comparten fotos de momentos de interés, de cada miembro de la comunidad académica UIS.
Usuario	Miembro de la comunidad académica UIS que utiliza la aplicación web UPost.
Desarrollador	Persona que realiza el desarrollo de la aplicación web Upost

LISTA DE FUNCIONALIDADES

A continuación, se describe una lista de las funcionalidades que va a integrar la aplicación UPost. Al final en una tabla se encuentra un resumen con todas las funcionalidades y con su respectiva priorización.

Tabla 11. Detalle funcionalidad 1 UPost

Funcionalidad:	*Usuario se registra en la página.*	**id:**	**P-01**
Stakeholders:	Usuario, UPost		
Descripción:	El usuario ingresa a la aplicación y se le solicita un Nombre de usuario, Correo Institucional y una contraseña a través de un formulario.		
Restricciones:	- El usuario debe tener un correo institucional. - No puede haber dos usuarios con el mismo correo o nombre de usuario.		
Entrada:	Ingresa al módulo de registro de la aplicación, y envía su correo institucional y una contraseña.		
Salida:	El usuario es registrado en la base de datos, se le notifica que se registró correctamente, se inicia su sesión y es redirigido al Inicio de la aplicación.		
Salida negativa:	- El usuario no se puede registrar porque ya se encuentra registrado, y es notificado con el error. - El usuario ingresó un correo o contraseña de forma errónea, y es notificado con el error. - La aplicación presenta fallas, o no se encuentra en línea, en el dispositivo del cliente.		

Tabla 12. Detalle funcionalidad 2 UPost

Funcionalidad:	*Usuario inicia sesión en la página.*	**id:**	**P-02**
Stakeholders:	Usuario, UPost		
Descripción:	El usuario ingresa a la aplicación y se le solicitan los datos de Correo Institucional y la contraseña.		
Restricciones:	- El usuario debe estar registrado en la aplicación UPost - El usuario debe digitar correctamente los datos solicitados.		
Entrada:	Ingresa al módulo de inicio de sesión, y envía su correo institucional y una contraseña.		
Salida:	El usuario se valida en la base de datos, se le notifica que se autenticó el usuario, se inicia su sesión y es redirigido al Inicio de la aplicación.		
Salida negativa:	- El usuario no puede iniciar sesión porque digitó mal sus datos de registro. - La aplicación presenta fallas, o no se encuentra en línea en el dispositivo del cliente.		

Tabla 13. Detalle funcionalidad 3 UPost

Funcionalidad:	*Usuario comparte una foto y crea una publicación.*	**id:**	**P-03**
Stakeholders:	Usuario, UPost		
Descripción:	El usuario inicia sesión en la aplicación, da clic en el botón para compartir fotos, sube una foto de su dispositivo, escribe una descripción y da clic en enviar para subir la foto y se publica.		
Restricciones:	- El usuario debe haber iniciado sesión en la aplicación UPost. - El usuario debe subir únicamente imágenes y en formato (.jpg o .png).		
Entrada:	Ingresa al módulo para compartir fotos, selecciona una imagen que desee compartir y una descripción de la publicación.		
Salida:	El usuario se valida en la base de datos, se le notifica que se publicó la imagen y es redirigido al Inicio de la aplicación.		
Salida negativa:	- El usuario no puede subir la imagen porque su dispositivo no se encuentra en línea. - El usuario no ha iniciado sesión. - La aplicación presenta fallas y no está en línea.		

Tabla 14. Detalle funcionalidad 4 UPost

Funcionalidad:	*Usuario consulta su perfil.*	**id:**	**P-04**
Stakeholders:	Usuario, UPost		
Descripción:	El usuario inicia sesión en la aplicación, ingresa su perfil de usuario donde puede visualizar sus datos personales y las fotos que ha compartido.		
Restricciones:	- El usuario debe haber iniciado sesión en la aplicación UPost.		
Entrada:	Ingresar a su perfil personal.		
Salida:	Se muestran los datos personales almacenados en la base de datos, y se listan las imágenes que ha compartido.		
Salida negativa:	- El usuario no puede visualizar su perfil si no ha iniciado sesión. - La aplicación presenta fallas y no está en línea.		

Tabla 15. Detalle funcionalidad 5 UPost

Funcionalidad:	*Usuario modifica sus datos personales.*	**id:**	**P-05**
Stakeholders:	Usuario, UPost		
Descripción:	El usuario inicia sesión en la aplicación, ingresa al módulo para la edición de datos personales.		
Restricciones:	- El usuario debe haber iniciado sesión en la aplicación UPost.		
Entrada:	Ingresar al módulo de edición de datos personales. Modificar sus datos y guardar los cambios.		
Salida:	Se actualizan los datos en la base de datos y se redirige al perfil de usuario para validar que estén correctos.		
Salida negativa:	- El usuario modificó los datos y hay un error, por duplicidad de información o no la envío con los formatos especificados. - El usuario no puede cambiar los datos sin haber iniciado sesión. - La aplicación presenta fallas y no está en línea.		

Tabla 16. Detalle funcionalidad 6 UPost

Funcionalidad:	*Usuario elimina su perfil y datos personales.*	**id:**	**P-06**
Stakeholders:	Usuario, UPost		
Descripción:	El usuario inicia sesión en la aplicación, ingresa al módulo para eliminar el perfil de usuario, debe digitar la contraseña y aceptar que va a eliminar su usuario. Posteriormente el usuario es eliminado de la base datos con todos los datos que ha compartido.		
Restricciones:	- El usuario debe haber iniciado sesión en la aplicación UPost.		
Entrada:	Ingresar a su perfil personal.		
Salida:	Se muestran los datos personales almacenados en la base de datos, y se listan las imágenes que ha compartido.		
Salida negativa:	- El usuario no puede visualizar su perfil si no ha iniciado sesión. - La aplicación presenta fallas y no está en línea.		

Tabla 17. Detalle funcionalidad 7 UPost

Funcionalidad:	*Usuario elimina una publicación.*	**id:**	**P-07**
Stakeholders:	Usuario, UPost		
Descripción:	El usuario inicia sesión en la aplicación, ingresa al módulo para eliminar publicaciones, selecciona la que desea eliminar y confirma que desea eliminarla.		
Restricciones:	- El usuario debe haber iniciado sesión en la aplicación UPost. - El usuario debe seleccionar una foto para eliminar.		
Entrada:	Ingresa al módulo para eliminar publicaciones, selecciona una imagen que desee compartir y una descripción de la publicación.		
Salida:	El usuario se valida en la base de datos, se le notifica que se publicó la imagen y es redirigido al Inicio de la aplicación.		
Salida negativa:	- El usuario no puede subir la imagen porque su dispositivo no se encuentra en línea. - El usuario no ha iniciado sesión. - La aplicación presenta fallas y no está en línea.		

Tabla 18. Detalle funcionalidad 8 UPost

Funcionalidad:	*Usuario consulta el mapa con las publicaciones.*	**id:**	**P-08**
Stakeholders:	Usuario, UPost		
Descripción:	El usuario inicia sesión en la aplicación, ingresa al módulo del mapa para visualizar las publicaciones, y se le despliegan sobre un mapa con la ubicación donde fue capturada.		
Restricciones:	- El usuario debe haber iniciado sesión en la aplicación UPost.		
Entrada:	Ingresa al módulo del mapa para visualizar las publicaciones.		
Salida:	Se busca en la base de datos la ubicación de cada publicación, se ubica en un mapa y se muestra el mapa al usuario.		
Salida negativa:	- El usuario no puede visualizar el mapa si no ha iniciado sesión. - La aplicación presenta fallas y no está en línea.		

Tabla 19. Detalle funcionalidad 9 UPost

Funcionalidad:	*Usuario edita la descripción de una publicación.*	**id:**	**P-09**
Stakeholders:	Usuario, UPost		
Descripción:	El usuario inicia sesión en la aplicación, ingresa al módulo editar publicaciones, edita el texto y guarda los cambios.		
Restricciones:	- El usuario debe haber iniciado sesión en la aplicación UPost.		
Entrada:	Ingresa al módulo editar publicaciones, edita y se envía el texto.		
Salida:	Se actualiza la base de datos y la publicación.		
Salida negativa:	- El usuario no puede editar la publicación si no ha iniciado sesión. - La aplicación presenta fallas y no está en línea.		

Tabla 20. Detalle funcionalidad 10 UPost

Funcionalidad:	*Usuario busca a otros usuarios.*	**id:**	**P-10**
Stakeholders:	Usuario, UPost		
Descripción:	El usuario inicia sesión en la aplicación, ingresa al buscador y envía el texto con el nombre de usuario o datos del usuario objetivo.		
Restricciones:	- El usuario debe haber iniciado sesión en la aplicación UPost.		
Entrada:	Ingresa al buscador, digita y envía texto.		
Salida:	Se consulta en la base de datos y se listan los usuarios encontrados.		
Salida negativa:	- El usuario no puede buscar si no ha iniciado sesión. - La aplicación presenta fallas y no está en línea.		

Tabla 21. Detalle funcionalidad 11 UPost

Funcionalidad:	*Usuario da 'me gusta' a las publicaciones.*	**id:**	**P-11**
Stakeholders:	Usuario, UPost		
Descripción:	El usuario inicia sesión en la aplicación, encuentra una aplicación que le gusta y da clic en el icono de 'me gusta'. Se le suma uno a la cantidad de 'me gusta' y se actualiza la base de datos.		
Restricciones:	- El usuario debe haber iniciado sesión en la aplicación UPOST.		
Entrada:	Da clic sobre el icono de 'me gusta' en una publicación que le gusta.		
Salida:	Se modifica la cantidad de 'me gusta' en la base de datos, y se actualiza la vista del usuario.		
Salida negativa:	- El usuario no puede buscar si no ha iniciado sesión. - La aplicación presenta fallas y no está en línea.		

Tabla 22. Detalle funcionalidad 12 UPost

Funcionalidad:	*Usuario puede listar las publicaciones que le gustan.*	**id:**	**P-12**
Stakeholders:	Usuario, UPost		
Descripción:	El usuario inicia sesión en la aplicación, ingresa al módulo para ver las publicaciones que le gustan, y se listan todas las que le han gustado.		
Restricciones:	- El usuario debe haber iniciado sesión en la aplicación UPOST.		
Entrada:	Ingresa al módulo para ver las publicaciones que le gustan.		
Salida:	Se consulta en la base de datos y se listan las publicaciones que le gustan.		
Salida negativa:	- El usuario no puede listar las publicaciones que le gustan si no ha iniciado sesión. - La aplicación presenta fallas y no está en línea.		

Consolidado final con su respectiva priorización. Las categorías son: *Must*, *Should* y *Nice to have*.

Tabla 23. Priorización de funcionalidades UPost

Código	Funcionalidad	Priorización
P-01	*Usuario se registra en la página.*	**Must**
P-02	*Usuario inicia sesión en la página.*	**Must**
P-03	*Usuario comparte una foto y crea una publicación.*	**Must**
P-04	*Usuario consulta su perfil.*	**Must**
P-05	*Usuario modifica sus datos personales.*	**Should**
P-06	*Usuario elimina su perfil y datos personales.*	**Should**
P-07	*Usuario elimina una publicación.*	**Should**
P-08	*Usuario consulta el mapa con las publicaciones.*	**Nice to have**
P-09	*Usuario edita la descripción de una publicación.*	**Should**
P-10	*Usuario busca a otros usuarios.*	**Nice to have**
P-11	*Usuario da 'me gusta' a las publicaciones.*	**Must**
P-12	*Usuario puede listar las publicaciones que le gustan.*	**Nice to have**

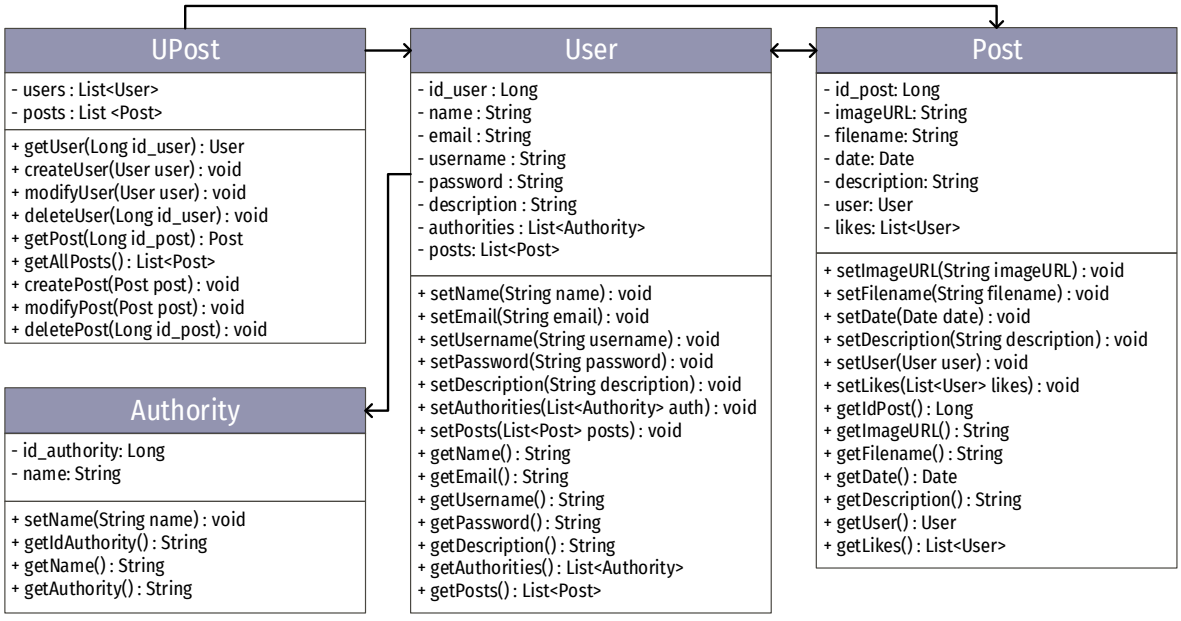
Solo se van a desarrollar en el MVP las funcionalidades que fueron seleccionadas con la categoría '*Must*' y '*Should*' durante el ciclo de desarrollo. Todos los pasos siguientes se harán con base en estas funcionalidades seleccionadas, es posible que el modelo conceptual, la base de datos, contratos de servicio e interfaces de usuario se diseñen para que las demás funcionalidades se implementen en ciclos posteriores de desarrollo.

DIAGRAMACION

Diagrama conceptual

El siguiente diagrama de la Figura 46, satisface las necesidades y funcionalidades identificadas que tiene la aplicación UPost.

Figura 46. Diagrama conceptual UPost



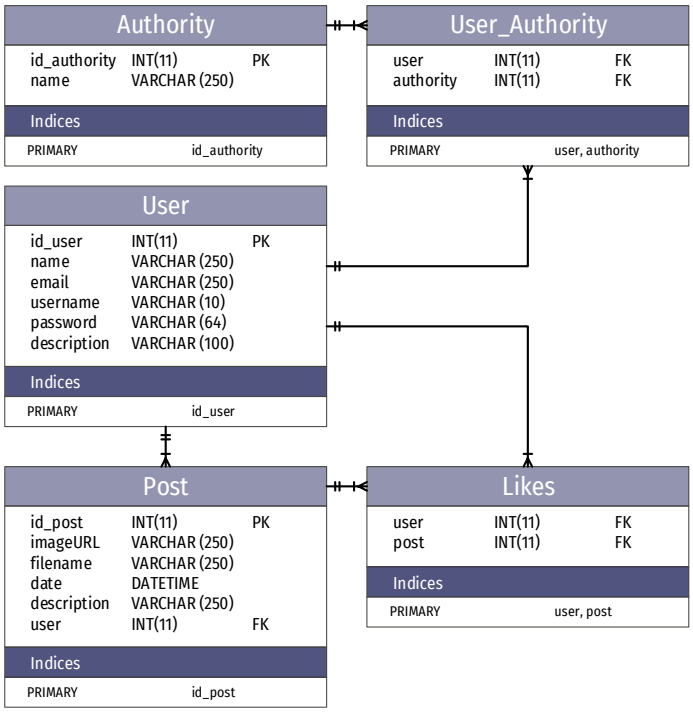
Fuente: Autor

Incluye a los stakeholders involucrados y además cumple con todos los principios SOLID. Al desarrollar la aplicación se reemplazan las funciones y métodos de la clase UPost en las 3 capas presentadas en la arquitectura de N-capas.

Diagrama de datos

Se propone el siguiente diagrama del modelo de datos creado a partir del modelo conceptual y reutilizando el modelo planteado en el FWD (ver Figura 47).

Figura 47. Diagrama de datos UPost



Fuente: Autor

CONTRATOS DE SERVICIO

Se reutilizan los servicios del FWD, y se añaden otros específicos para UPost. Se reutilizan los mismos roles de autorización del FWD.

Tabla 24. Contratos de servicio para la aplicación UPost

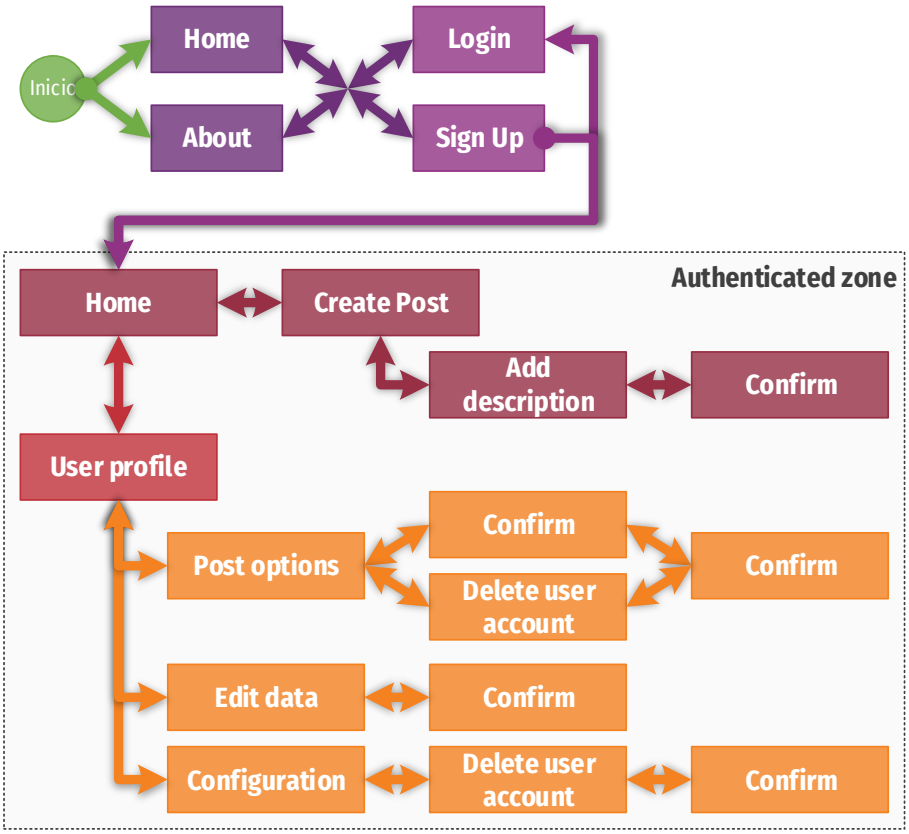
Contratos de servicio									
Server	**Endpoint**	**Subendpoint**	**Método REST**	**Descripción**	**Roles Autorizados**	**Entradas**	**Formato de Entrada**	**Salidas**	**Formato de Salida**
Backend Server (Versión API : /v1)	**/like**		**POST**	Añade el usuario al post sumando un like al post seleccionado	USER	HTTP Request Param - idPost - idUser	application/json	En el body del HTTP Response: String (si se creó el like)	JSON
			DELETE	Elimina el usuario al post quitando un like al post seleccionado	USER	HTTP Request Param - idPost - idUser	application/json	En el body del HTTP Response: String (si se eliminó el like)	JSON
	/post	**/post/{id}**	**GET**	Buscar post por ID retorna el post encontrado	USER	En el body del HTTP Request: id	application/json	En el body del HTTP Response: Post encontrado	JSON
			POST	Crea un nuevo Post	USER	HTTP Form data con las siguientes llaves: file (archivo)	multipart/form-data	En el body del HTTP Response: Post (si se creó el post)	JSON
			PATCH	Modifica los datos de un Post existente	USER	En el body del HTTP Request: Post (modificado)	application/json	En el body del HTTP Response: Post (si se modificó el post)	JSON
			DELETE	Elimina un Post existente	USER	En el body del HTTP Request: Post (a eliminar)	application/json	En el body del HTTP Response: Post (si se eliminó el post)	JSON
		/post/user	**GET**	Lista de los post del usuario en la aplicación	USER	HTTP Request Request: User (user de los posts)	application/json	En el body del HTTP Response: Array JSON de Post	JSON
	/posts		**GET**	Lista de los post que se encuentren en la aplicación	USER	HTTP Request	application/json	En el body del HTTP Response: Array JSON de Post	JSON
	/storage	**/storage/uploadFile**	**POST**	Carga un archivo a Amazon S3	USER	HTTP Form data con las siguientes llaves: - file (archivo)	multipart/form-data	En el body del HTTP Response: String (si se subio el archivo)	JSON
		/storage/deleteFile	**DELETE**	Elimina un archivo de Amazon S3 con el nombre del archivo	USER	HTTP Request Param - filename	application/json	En el body del HTTP Response: String (si se eliminó el archivo)	JSON

Los contratos de servicios van a operar o funcionar sobre la capa de API en el Backend Server y en el Frontend Server.

CUSTOMER JOURNEY

El siguiente es viaje que tendrá el usuario por las interfaces por donde se moverá a través de la aplicación UPost (ver Figura 48).

Figura 48. Customer Journey UPost

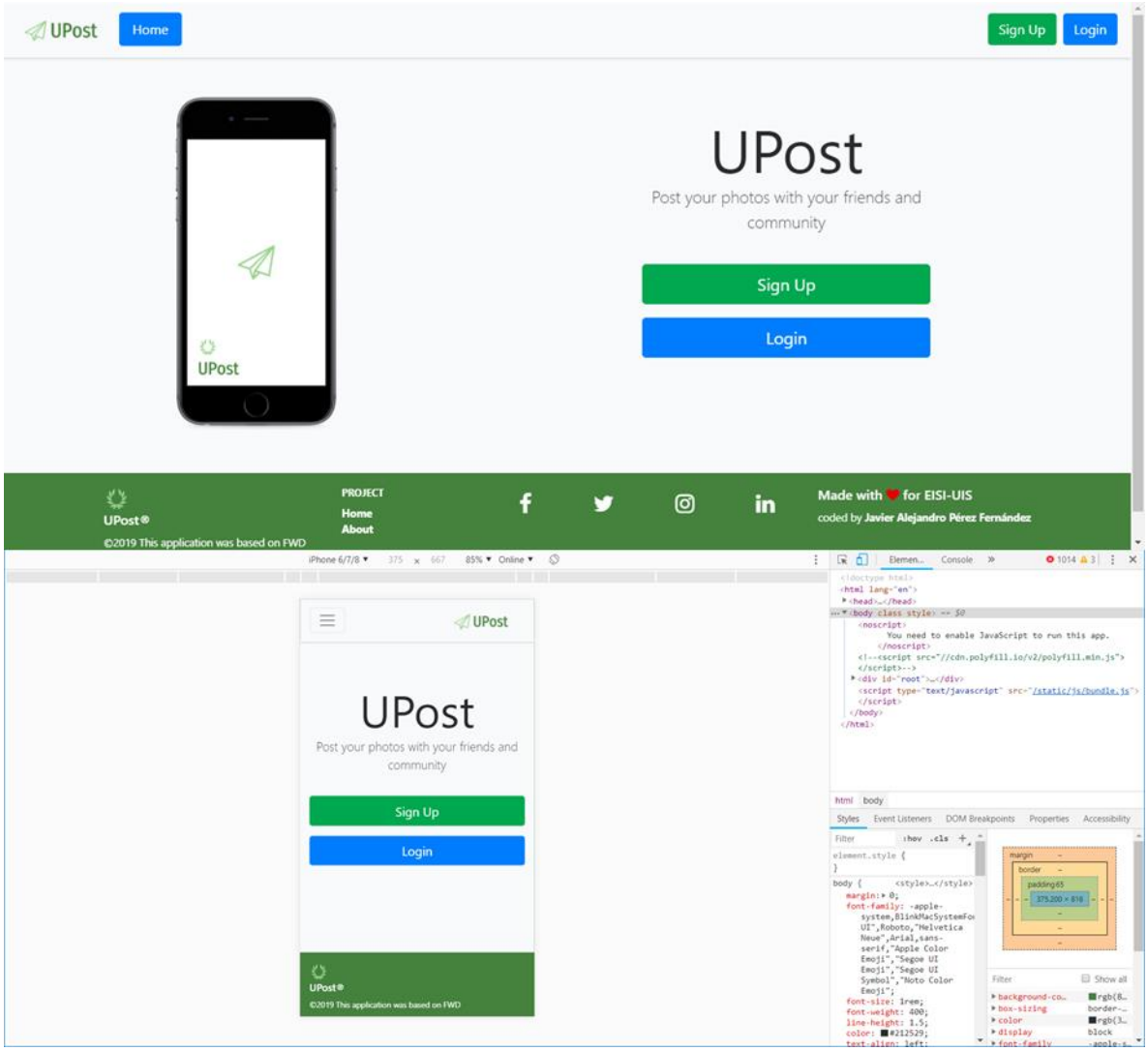


Fuente: Autor

INTERFACES

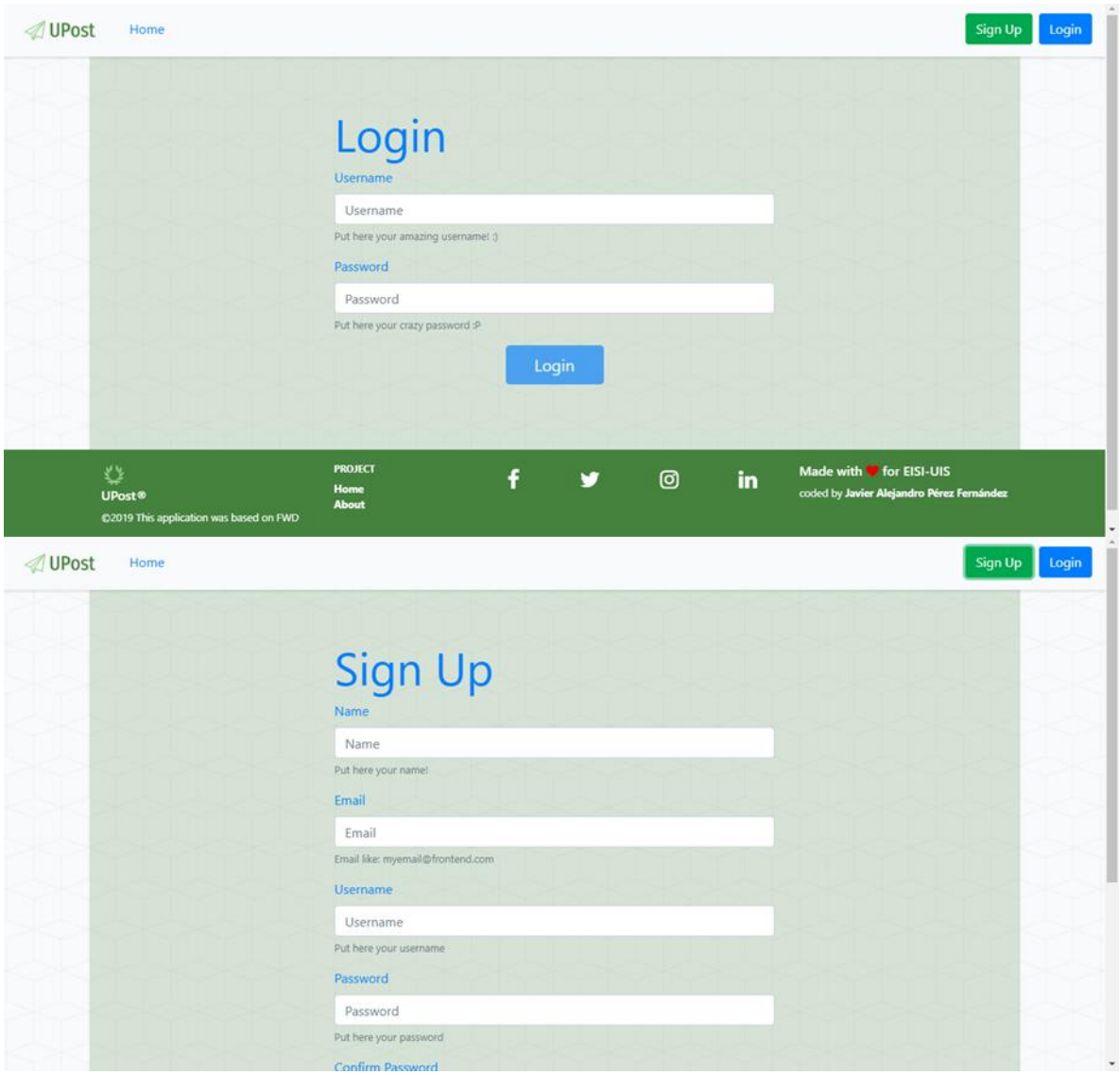
Los siguientes bosquejos de la Figura 51 a la Figura 54, muestran interfaces de la aplicación UPost:

Figura 49. Landing Page y vesión móvil de UPost



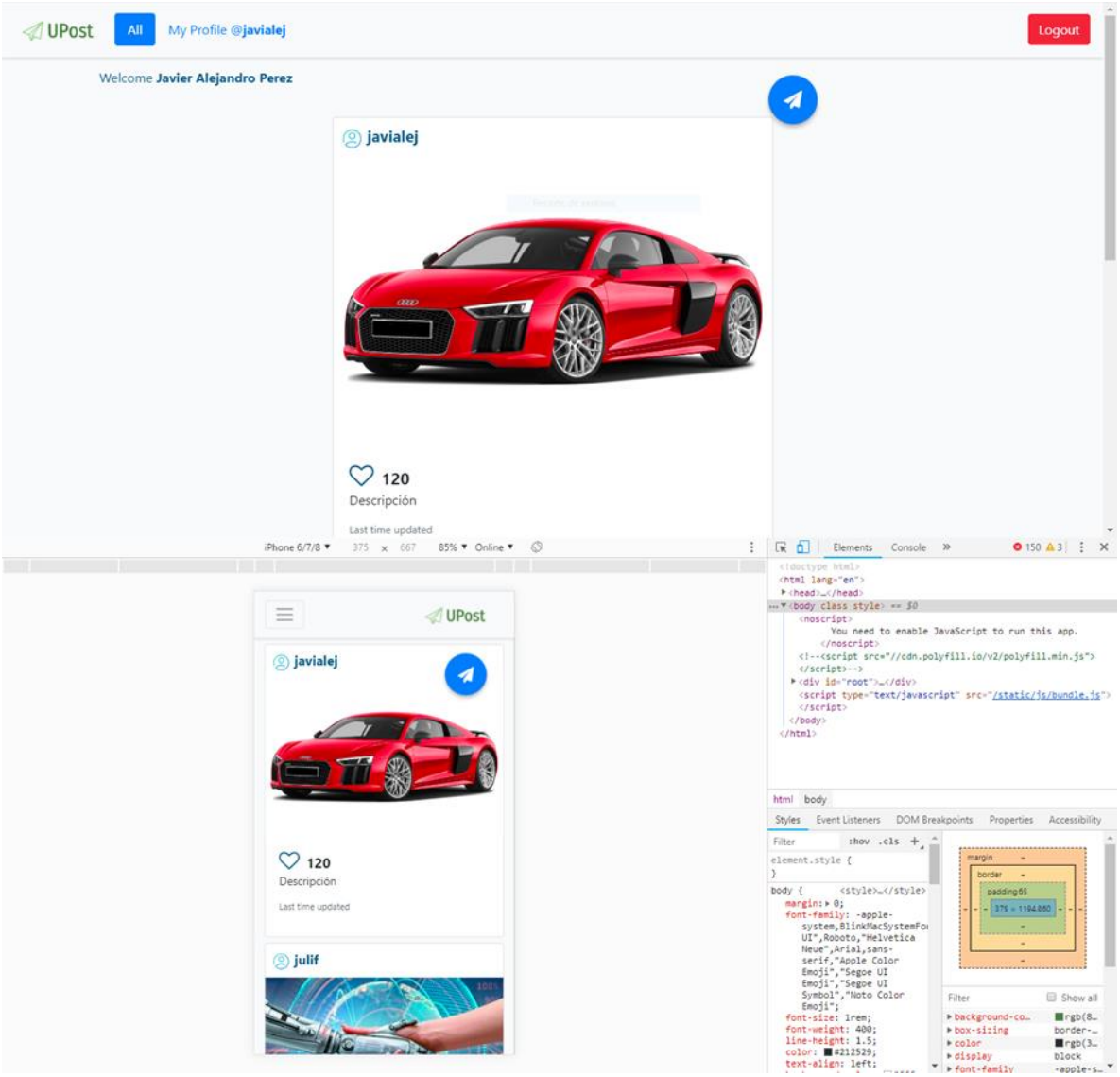
Fuente: Autor

Figura 50. Login y Sign Up de UPost



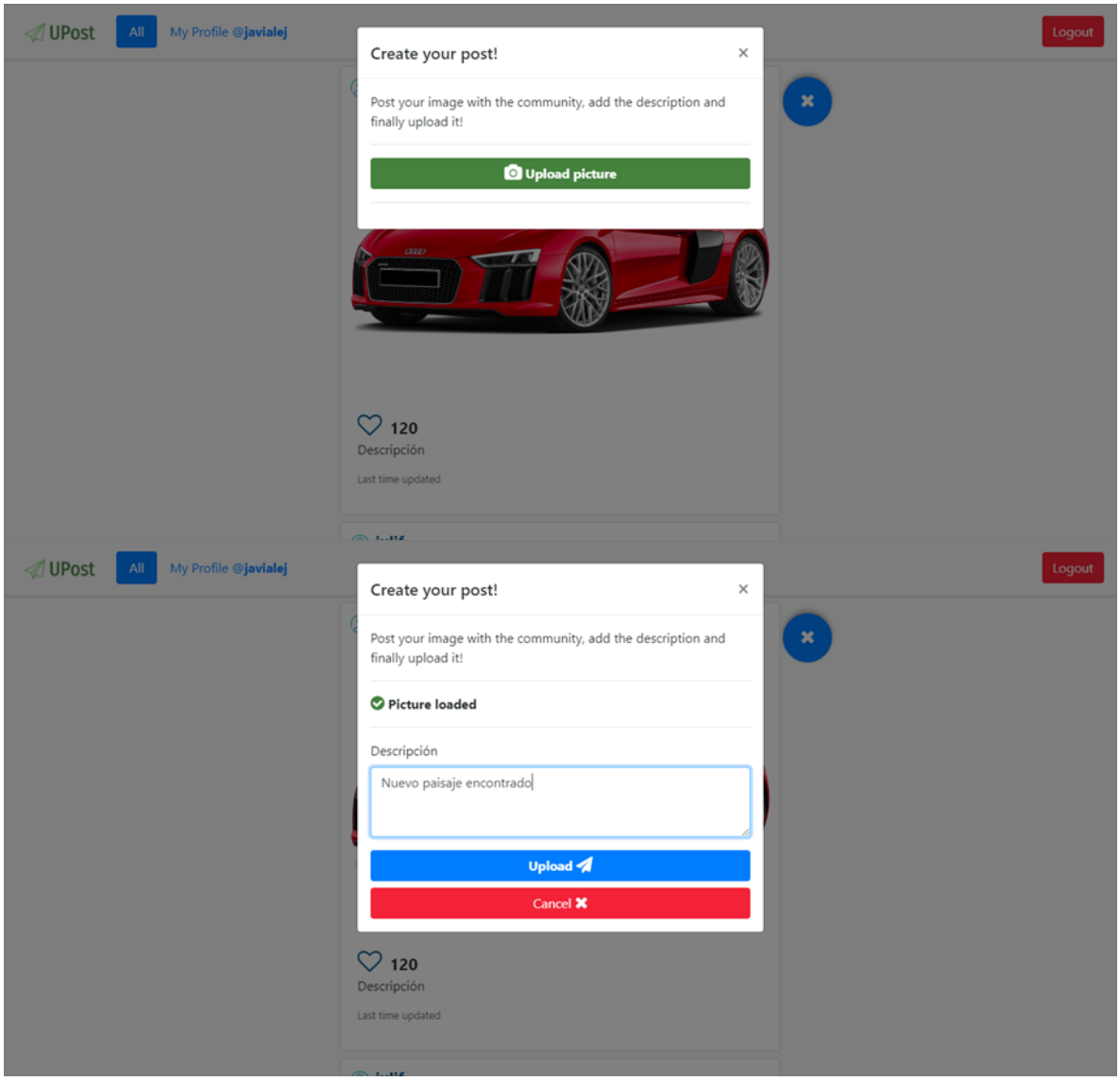
Fuente: Autor

Figura 51. Home page con listado de posts y Home Page versión móvil UPost



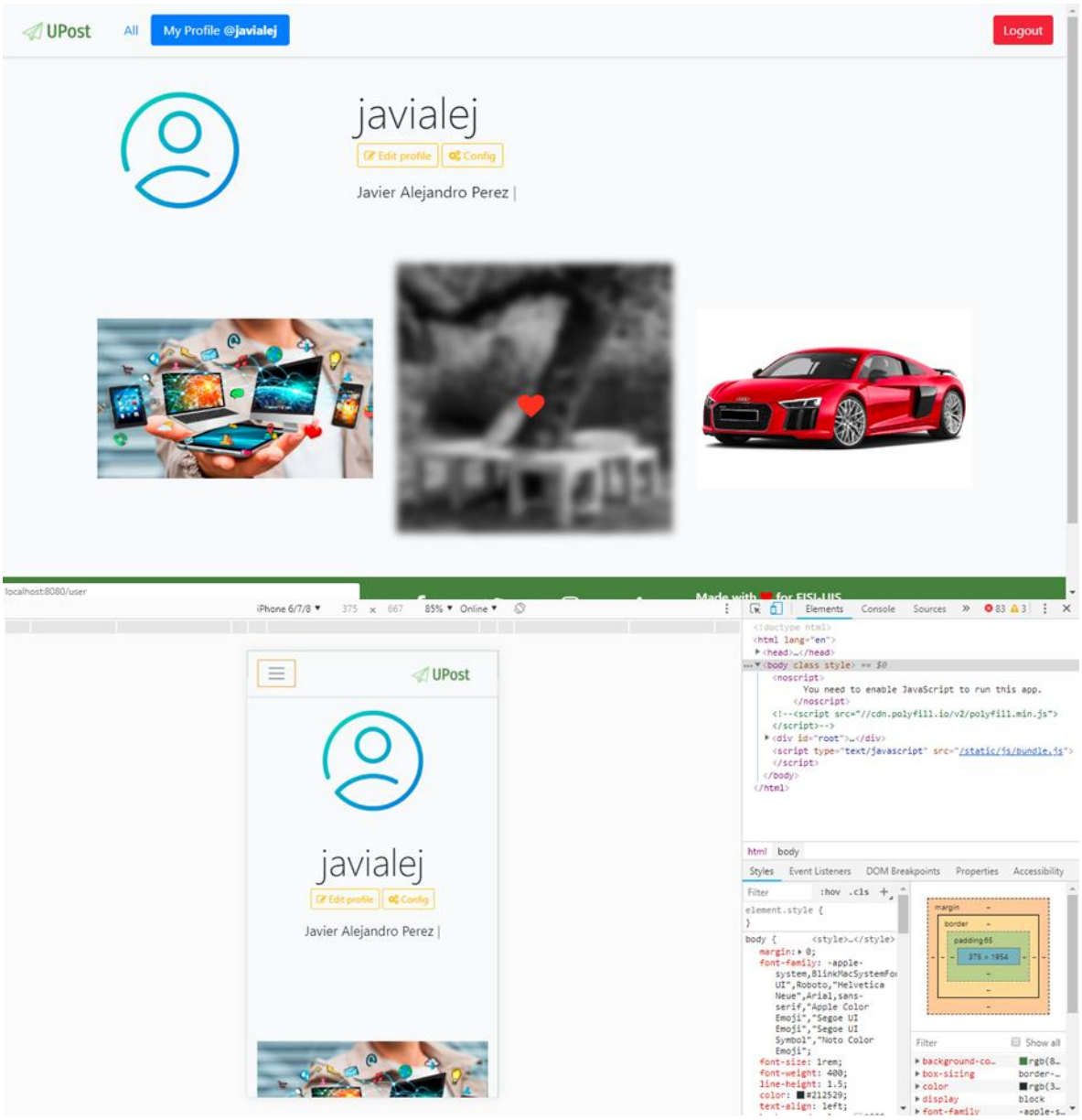
Fuente: Autor

Figura 52. Modal para cargar imagen del post y Modal de confirmación UPost



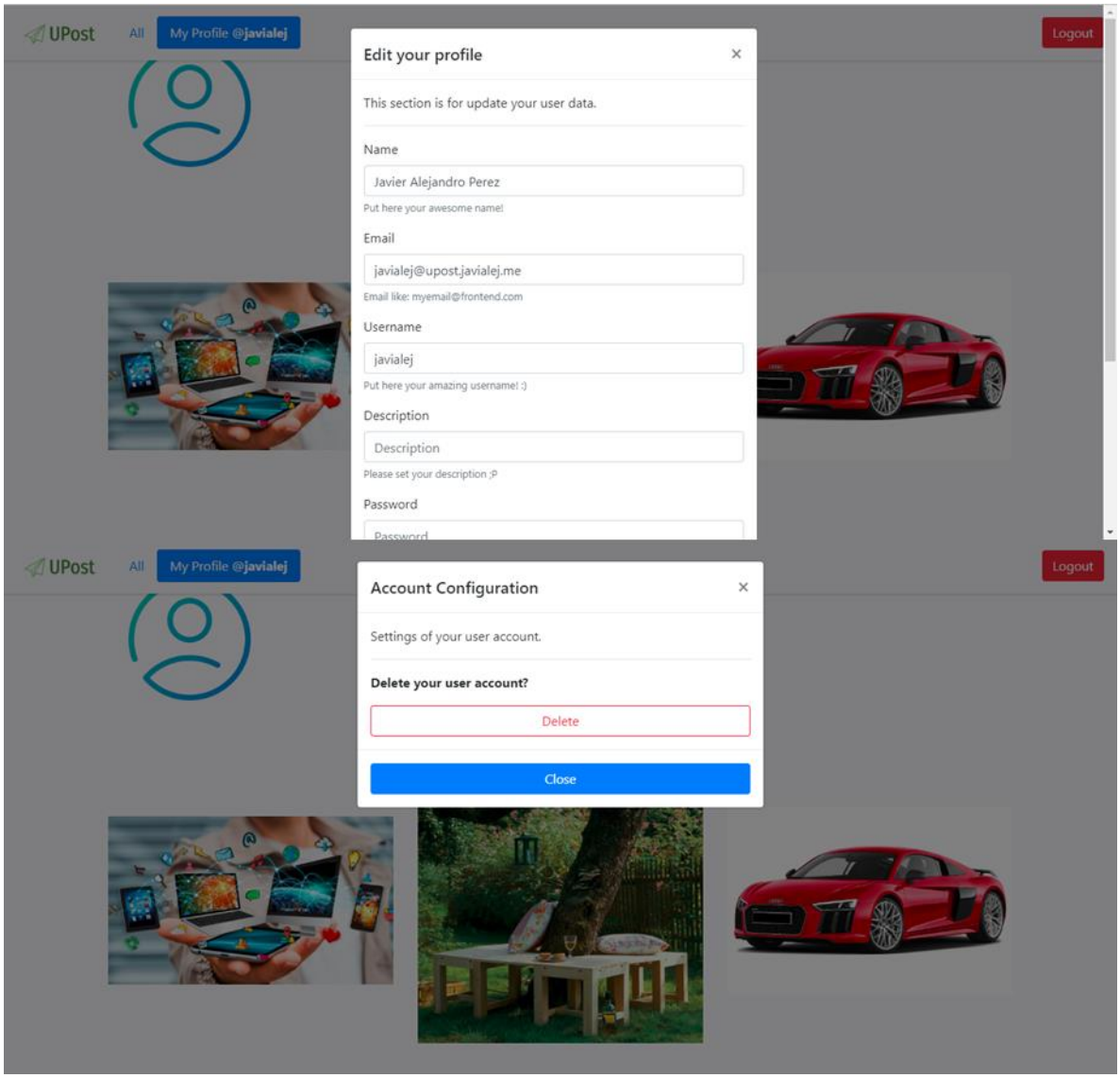
Fuente: Autor

Figura 53. User page con posts del usuario y User page versión móvil UPost



Fuente: Autor

Figura 54. Eliminar y editar usuario UPost



Fuente: Autor