ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

PROTOTIPO DE SISTEMA DE FIDELIZACIÓN DE COMERCIO ELECTRÓNICO USANDO TECNOLOGÍA DOCKER Y SERVICIOS DE AMAZON

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

RICARDO DAVID ORTIZ HEREDIA

DIRECTOR: DR. LUIS FELIPE URQUIZA AGUILAR

Quito, marzo 2022

AVAL

Certifico que el presente trabajo fue desarrollado por Ricardo David Ortiz Heredia, bajo mi supervisión.

_____

DR. LUIS FELIPE URQUIZA AGUILAR

DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo, Ricardo David Ortiz Heredia, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejo constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

_____

RICARDO DAVID ORTIZ HEREDIA

DEDICATORIA

Dedico este trabajo a la madre de mi corazón Janneth Ortiz, quien me amado como a un hijo y me ha brindado su apoyo incondicional siempre.

AGRADECIMIENTO

Agradezo mi familia Janneth, Diego y Cristina por su apoyo incondicional. A Nicole por su cariño y compañía durante el desarrollo de este trabajo. Al Dr. Luis Urquiza, director de este proyecto, por su tiempo dedicado al éxito de este trabajo.

ÍNDICE DE CONTENIDO

AVAL II
DECLARACIÓN DE AUTORÍA III
DEDICATORIA IV
AGRADECIMIENTO V
ÍNDICE DE CONTENIDO VI
RESUMEN IX
ABTRACT X
1. INTRODUCCIÓN 1
1.1. OBJETIVOS 1
1.2. ALCANCE 1
1.3. MARCO TEÓRICO 2
1.3.1. NEGOCIOS ELECTRÓNICOS 3
1.3.2. COMERCIO ELECTRÓNICO 4
1.3.3. COMERCIO ELECTRÓNICO EN EL CONTEXTO DE LA PANDEMIA POR COVID 19 6
1.3.4. TIENDA ELECTRÓNICA 8
1.3.5. FIDELIZACIÓN 9
1.3.6. DOCKER 10
1.3.7. DOCKER COMPOSE 13
1.3.8. JAVASCRIPT 14
1.3.9. NODE 16
1.3.10. TYPESCRIPT 17
1.3.11. NESTJS 18
1.3.12. REACT 19
1.3.13. IONIC REACT 20
1.3.14. PRUEBAS QA 24
1.3.15. HTTP 25
1.3.16. REST 27
1.3.17. CEZERIN 28
1.3.18. SERVICIOS EN LA NUBE PROVISTOS POR AWS 29
2. METODOLOGÍA 31
2.1. ANÁLISIS DE REQUERIMIENTOS 31
2.1.1. ACTORES DEL SISTEMA 32
2.1.2. HISTORIAS DE USUARIO 32

2.1.3. REQUERIMIENTOS DE DOCKERIZACIÓN....34
2.1.4. REQUERIMIENTOS DEL SISTEMA DE FIDELIZACIÓN....35
2.2. DISEÑO....36
2.2.1. ARQUITECTURA LÓGICA....36
2.2.2. ARQUITECTURA FÍSICA....38
2.2.3. DISEÑO BASE DE DATOS RELACIONAL POSTGRESQL PARA SISTEMA DE FIDELIZACIÓN....39
2.2.4. DISEÑO SERVIDOR WEB DE SISTEMA DE FIDELIZACIÓN....40
2.2.5. DISEÑO APLICATIVO WEB CLIENTE Y ADMINISTRATIVO DEL SISTEMA DE FIDELIZACIÓN....42
2.2.6. DISEÑO MIDDLEWARE PARA SISTEMA DE FIDELIZACIÓN Y CEZERIN 44
2.2.7. DISEÑO DE E-COMMERCE DE CÓDIGO ABIERTO....46
2.2.8. DISEÑO BASE DE DATOS NO RELACIONAL MONGODB....48
2.2.9. PLAN DE PRUEBAS....49
2.3. DESARROLLO EN LOCAL....49
2.3.1. LEVANTAMIENTO Y CONFIGURACIÓN DE SERVIDOR SMTP EN LOCAL UTILIZANDO IMAGEN DE DOCKER DE MAILHOG.....49
2.3.2. LEVANTAMIENTO Y CONFIGURACIÓN DE LA BASE DE DATOS DOCUMENTAL MONGODB EN AMBIENTE LOCAL....50
2.3.3. LEVANTAMIENTO Y CONFIGURACIÓN DE LA BASE DE DATOS RELACIONAL POSTGRESQL EN AMBIENTE LOCAL.....52
2.3.4. DESARROLLO SERVIDOR WEB PARA SISTEMA DE FIDELIZACIÓN 53
2.3.5. DESARROLLO APLICATIVO WEB CLIENTE PARA SISTEMA DE FIDELIZACIÓN.....54
2.3.6. DESARROLLO APLICATIVO WEB ADMINISTRATIVO PARA SISTEMA DE FIDELIZACIÓN....55
2.3.7. DESARROLLO SERVIDOR WEB PARA E-COMMERCE BACKEND USANDO CEZERIN2.....58
2.3.8. DESARROLLO APLICATIVO WEB PARA E-COMMERCE USANDO CEZERIN2....60
2.3.9. DESARROLLO APLICATIVO WEB PARA DASHBOARD ADMINISTRATIVO USANDO CEZERIN2.....63
2.3.10. DESARROLLO SERVIDOR WEB PARA INTEGRACIÓN DE SISTEMA DE TARJETAS CON EL SISTEMA DE E-COMMERCE. 64
2.3.11. DESARROLLO SCRIPT DE DOCKER COMPOSE....67
3. RESULTADOS Y DISCUSIÓN 70
3.1. DESPLIEGUE EN LA NUBE....70
3.1.1. APROVISIONAMIENTO INFRAESTRUCTURA EN LA NUBE 70
3.1.2. PREPARACIÓN DEL AMBIENTE EN LA NUBE....72

3.1.3. CONFIGURACIÓN Y LEVANTAMIENTO DEL SISTEMA CON UN COMANDO 75
3.2. EJECUCIÓN PLAN DE PRUEBAS 77
3.2.1. AUTENTICACIÓN EN APLICATIVO WEB. 77
3.2.2. AUTENTICACIÓN EN APLICATIVO WEB ADMINISTRATIVO 79
3.2.3. REVISIÓN DE CATÁLOGOS DE PRODUCTOS 79
3.2.4. PROCESO DE COMPRA 80
3.2.5. CREACIÓN, MODIFICACIÓN Y ELIMINACIÓN DE REGISTRO DE UN PRODUCTO 82
3.2.6. CREACIÓN, MODIFICACIÓN Y ELIMINACIÓN DEL REGISTRO DE UNA CATEGORÍA 85
3.2.7. ASOCIACIÓN DE UN PRODUCTO A UNA CATEGORÍA 87
3.2.8. CONSULTA DE ÓRDENES 89
3.2.9. CONSULTA DE CLIENTES 89
3.2.10. DEFINICIÓN DE TARJETA DE FIDELIZACIÓN Y ASIGNACIÓN A UN USUARIO DEL E-COMMERCE 90
3.2.11. REVISIÓN DE TARJETAS ASIGNADAS POR EL USUARIO CLIENTE DEL ECOMMERCE 93
3.2.12. COMPRA RESULTA EN AGREGAR PERFORACIÓN A TARJETA 94
3.2.13. COLOCACIÓN DE MARCA DE PREMIOS REDIMIDOS A TARJETA DE FIDELIZACIÓN 95
3.3. PRUEBAS ADICIONALES AL SISTEMA 95
3.3.1. REDES CON CONTENEDORES DOCKER 96
3.3.2. CONFIGURACIONES DE GRUPOS DE SEGURIDAD DE AMAZON 101
4. CONCLUSIONES Y RECOMENDACIONES 103
4.1. CONCLUSIONES 103
4.2. RECOMENDACIONES 105
5. REFERENCIAS BIBLIOGRÁFICAS 106
ANEXOS I

RESUMEN

El rápido desarrollo del Internet ha contribuido al agigantado aumento de transacciones electrónicas. Estas son ubicuas en la vida cotidiano de millones de personas. Los sistemas de fidelización constituyen una mejora a un sistema de e-commerce. Aunque existen una gran variedad de software para sistemas de e-commerce, la integración con sistemas de fidelización suele ser muy limitada o inexistente. Este proyecto es una contribución al desarrollo de los sistemas de e-commerce, ya que se ha desarrollado un sistema de fidelización que se integra a cualquier sistema de e-commerce que utiliza el protocolo HTTP y REST. Para ilustrar esta capacidad se ha realizado, también como parte de este proyecto, una integración al sistema de e-commerce de código abierto Cezerin. Todo el sistema se ha desplegado en la nube utilizando la tecnología de virtualización Docker y servicios de Amazon.

En el primer capítulo se revisa la teoría del comercio electrónico, los sistemas de fidelización, la tecnología de virtualización Docker, las tecnologías de computación en la nube de AWS y algunas de las tecnologías de programación para desarrollo web utilizadas en este proyecto: JavaScript, Node, React, Ionic, y NestJS. Para mantener sistemas desacoplados se ha utilizado el protocolo REST y HTTP para establecer comunicación entre sistemas independientes. Se revisa los conceptos principales de estos protocolos.

En el segundo capítulo se describe el proceso de diseño y de desarrollo en ambiente local. Primeramente, se definen los actores del sistema, las tecnologías a utilizar en sus versiones específicas, los roles de los subsistemas y las funcionalidades del sistema de fidelización a desarrollar. Se incluyen los esquemas, diagramas y figuras que se elaboraron para guiar el proceso de desarrollo. También, se definió el plan de pruebas que guía el proceso de validación del funcionamiento del sistema. Finalmente, se incluye en esta sección el proceso en desarrollo local, incluyendo los principales comandos, las respuestas de la instancia local a los mismos, las secciones de código más relevantes y las configuraciones realizadas.

En el tercer capítulo se describe el despliegue del sistema utilizando servicios de AWS para computación en la nube. Más adelante, se documentan los resultados de la ejecución del plan de pruebas definido previamente. Finalmente, se documenta pruebas adicionales realizadas al sistema que ilustran funcionalidades de Docker como tecnología de virtualización y de AWS como tecnología de computación en la nube.

PALABRAS CLAVE: *E-commerce*, *Fidelización, Docker, Computación en la nube, REST.*

ABTRACT

Fast development of the Internet has contributed to a significant growth of electronic transactions. This are ubiquitous in the day-to-day life of millions of people. Loyalty systems are an improvement for e-commerce systems. However, the integration of an e-commerce with a loyalty system is usually very limited or not available in many popular products. This project is a contribution to the development of e-commerce by providing a loyalty system that can be incorporated with any e-commerce that uses the HTTP protocol and adjusts to REST principles. To illustrate this functionality, this project includes an integration with the open-source system Cezerin. This fully functional system was deployed using cloud computing services provided by Amazon.

The first chapter includes a review on e-commerce theory, loyalty systems, the virtualization, technology Docker, the cloud computing technologies offered by Amazon and some of the main software technologies used in this project, namely: JavaScript, Node, React, Ionic and NestJS. In order to keep the systems decoupled this project uses the HTTP protocol as well as REST principles in order to establish communication between systems that are independent from each other. Thus, some of the core principles of this protocols are reviewed as well on the first chapter.

Chapter two includes a description go the design process and the local development. Firstly, the system actors are defined, then the technologies chosen for this project, the roles of each subsystem as well as the core functionalities of the system. This section includes diagrams, schemes and figures produced during the design process. As part of the design process the testing plan was established as a mean to guide the validation process. Finally, this section includes the development in the local environment, it includes many of the main instructions, responses to instructions, code sections and configurations that were done during the development process.

The third chapter describes the system deployment using AWS cloud computing services. Then, it includes the documentation of results obtained after executing the test plan. Finally, additional testing results are included as an illustration of the capabilities of the Docker technology and the AWS cloud computing services.

KEYWORDS: *E-commerce*, *Loyalty, Docker, Cloud computing, REST.*

1. INTRODUCCIÓN

Un e-commerce, o comercio electrónico es un esquema de negocio que hace uso de las tecnologías de telecomunicaciones y computación. Los productos relacionados, como tiendas electrónicas han tenido un crecimiento sostenido. Los sistemas de fidelización complementan a los sistemas de e-commerce. Un sistema de fidelización es esencialmente un programa de recompensas que mejora la relación del cliente con el negocio. Existe una gran variedad de sistemas de e-commerce de código abierto y propietario, sin embargo, son escasos los que presentan un sistema de fidelización. En los casos que el sistema de fidelización está presente, su implementación suele ser exclusiva para el sistema con el que opera y suelen ser característicamente limitados. Este proyecto permite agregar a cualquier sistema de e-commerce un sistema de fidelización por medio de una integración usando el protocolo de comunicación HTTP con la arquitectura REST.

OBJETIVOS

El objetivo general de este trabajo de titulación es desarrollar un prototipo de un sistema de fidelización de comercio electrónico usando la tecnología Docker y servicios de Amazon.

Asimismo, los objetivos específicos son:

- Analizar los e-commerce, los sistemas de fidelización, la tecnología de virtualización con contenedores Docker, la computación en la nube con servicios de Amazon y las tecnologías de programación para desarrollo de aplicaciones de red.
- Implementar un servicio web de tarjetas de fidelización que expone un API REST para la integración con sistemas de e-commerce que utilicen el protocolo REST.
- Integrar un e-commerce de código abierto con el sistema de fidelización desarrollado.
- Analizar y documentar los resultados obtenidos mediante la ejecución de un plan de pruebas previamente establecido.

ALCANCE

Este prototipo proveerá un servicio web para un sistema de fidelización de tarjetas perforadas con las siguientes funcionalidades: crear, listar y borrar premios por completar una tarjeta; asignar premios a esquemas de tarjetas que se usan luego para crear tarjetas asignadas a usuarios; crear, listar y borrar esquemas de tarjetas; crear listar y borrar tarjetas asignadas a

usuarios; agregar, listar y borrar perforaciones de una tarjeta asignada a un usuario; y agregar, listar y remover marcas de premios redimidos a una tarjeta asignada a un usuario.

El prototipo contará con los siguientes componentes: un sistema de e-commerce de código abierto, un sistema web de fidelización y un servidor web conectando ambos sistemas. El sistema de e-commerce incluye: un servidor web que expone un API REST, un aplicativo web para el cliente un aplicativo web administrativo, una base de datos no relacional y un servidor para correos (ilustrativo). El sistema de fidelización incluye un servidor web que expone un API REST, un aplicativo web para el cliente, un aplicativo web administrativo y una base de datos relacional.

Para el desarrollo de este proyecto se utilizará la metodología de desarrollo ágil SCRUM. La base de datos relacional usará el gestor Postgres y la base de datos no relacional usará el gestor MongoDB. Por otra parte, la base de datos relacional es parte del sistema de fidelización mientras que la base de datos no relacional es parte del sistema de e-commerce.

El servicio web que expone el API REST, backend, será desarrollado en Javascript con Typescript y NestJS, para la conexión con la base de datos se usará el ORM: TypeORM y para el frontend se usará Javascript con React y Ionic. Se elaborará un plan de pruebas de aseguramiento de calidad para identificar: errores y defectos, para lo cual se utilizarán dos ambientes: desarrollo y producción para facilitar las pruebas.

Finalmente, se levantará el sistema de e-commerce Cezerin y el prototipo de sistema de fidelización utilizando el servicio de Amazon: Elastic Computing Cloud (EC2). Se ejecutará el plan de pruebas y se documentarán los resultados. De esta manera, el proyecto incluye un producto final demostrable: el prototipo de fidelización desarrollado integrado a un sistema de e-commerce de código abierto desplegado con infraestructura en la nube.

MARCO TEÓRICO

El presente proyecto constituye una contribución al desarrollo de los e-commerce. Es así como en esta primera sección se revisa la teoría de los e-commerce como un elemento de los negocios electrónicos, su importancia, crecimiento y evolución en su contexto general y en particular en el contexto de la pandemia del Covid-19. Adicional, se repasa la evolución y estado de estos en el Ecuador. Para este trabajo se toma como elemento primordial los sistemas de fidelización, por lo que se explora su función y los beneficios que aporta en particular a los e-commerce.

Para el desarrollo del proyecto se han utilizado lenguajes de programación en conjunto con tecnologías adicionales como librerías, frameworks y motores para compilación. Se utilizan

gestores de bases de datos para la persistencia de datos, la tecnología de virtualización Docker y servicios de computación en la nube. En esta sección se describen estas tecnologías de forma general.

El prototipo realizado consiste principalmente en un sistema web que utiliza protocolos de capa de aplicación y tecnologías de red. Es así como se revisan conceptos de los protocolos principales que están en uso por el sistema del prototipo propuesto.

[redacted]NEGOCIOS ELECTRÓNICOS

Muchas compañías han movido a la web algunos de sus procesos como promoción, venta, administración y relación con el cliente, entre otros. Esto ha creado un conjunto de procesos y dinámicas nuevas, que se lo refiere con el término de negocio electrónico [1]. Los participantes en las transacciones electrónicas se clasifican en tres grupos: A, B y C. En el grupo "A" se ubican los gobiernos y sus instituciones, en el grupo "B" las compañías y organizaciones, finalmente en el grupo "C" los clientes o usuarios finales. Esta categorización se utiliza para nominar a las relaciones comerciales en los negocios electrónicos. La figura 1.1 muestra los tipos de relaciones en los negocios electrónicos [1].

		Consumidor de servicios		
		Usuarios	**Compañías**	**Gobiernos**
Proveedor de servicio	**Usuarios**	**Usuario a usuario (C2C)** Ejm: propaganda en páginas personales	**Usuario a compañías (C2B)** Ejm: página web con personal calificado	**Usuario /ciudadanos a Gobiernos (C2A)** Ejm: ciudadanos evaluan un proyecto público
	Compañías	**Compañías a usuarios (B2C)** Ejm: productos y servicios en tiendas on line	**Compañias a compañías (B2B)** Ejm: órdenes a proovedores (cadena de logística)	**Compañías a Gobiernos (B2A)** Ejm: servicios electrónicos para la administración pública
	Gobiernos	**Gobiernos a Consumidores/ ciudadanos (A2C)** Ejm: posibilidad de elecciones electrónicas	**Gobiernos a compañías (A2B)** Ejm: entretenimiento público en proyectos	**Gobiernos a Gobiernos (A2A)** Ejm: formas de cooperación en comunidades virtuales

Figura 1.1 Tipos de relaciones en los negocios electrónicos [1]

Una tienda electrónica es de tipo B2C, un servicio de elecciones usando una aplicación móvil es de tipo A2C, un servicio donde clientes escriben su opinión sobre un servicio o producto en forma de evaluación es de tipo C2B. Los procesos de venta de los e-commerce suelen ser de tipo "B2C" o "B2B" que significa que son relaciones comerciales entre compañías o entre una compañía y clientes finales. Por ejemplo, AWS ofrece servicios en la nube que son contratados por empresas como Netflix, McDonalds, Toyota y Coursera [2, 3], este es un ejemplo de relación B2B. Las empresas clientes de AWS, por otro lado, presentan sistemas web que ponen a disposición de los usuarios para información promoción o ventas, lo que constituye una relación B2C. Ambos son ejemplos de e-commerce [1].

COMERCIO ELECTRÓNICO

El comercio electrónico, e-commerce, es un subconjunto de los negocios electrónicos de tipo B2B o B2C que genera valor a través del Internet con el intercambio de servicios y productos [1]. En los países desarrollados, las compras en tiendas electrónicas son diarias para muchas personas, en contraste con los países en vías de desarrollo. En ambos casos, el crecimiento de este tipo de transacciones es considerable [4].

La Cámara de Comercio Electrónico del Ecuador registra en el Ecuador un crecimiento en transacciones digitales, transacciones e-commerce, ventas digitales y ventas e-commerce. Estas definidas de la siguiente manera [5]:

- Transacciones digitales: Operaciones que se realizan por medios electrónicos, independientemente de si se realizan de forma presencial o no presencial.
- Transacciones e-commerce: Operaciones que se realizan a través de medios electrónicos no presenciales.
- Ventas digitales: Monto en dólares en ventas que se efectúan con medio electrónicos independientemente de si se realizan de forma presencial o no presencial.
- Ventas e-commerce: Monto en dólares de ventas que se realizan de forma no presencial exclusivamente.

La Figura 1.2 muestra el crecimiento de las transacciones digitales y de los e-commerce en el Ecuador en los últimos años, mientras que la Figura 1.3 muestra el crecimiento de ventas digitales y ventas e-commerce en el país. Ambas reflejan un crecimiento sostenido en los últimos años.

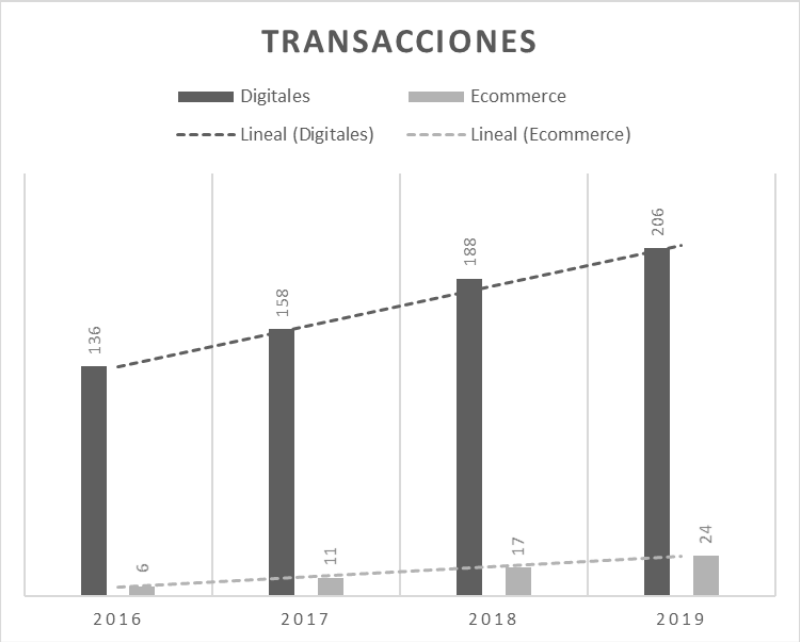


Figura 1.2 Evolución de las transacciones digitales y de e-commerce en el Ecuador [6]

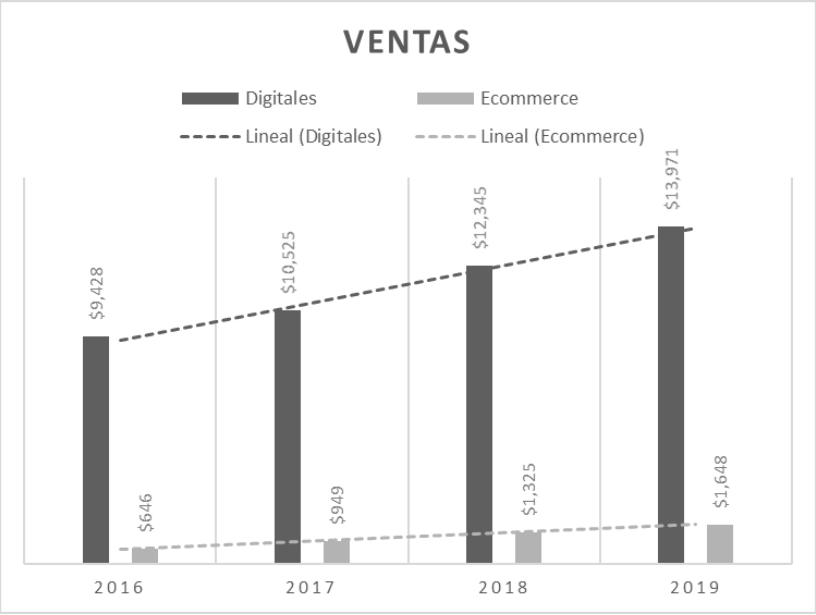


Figura 1.3 Evolución de las ventas digitales y de e-commerce en el Ecuador [6]

La Figura 1.4 muestra las ventas del sector del e-commerce en Ecuador en comparación a su Producto Interno Bruto (PIB). La contribución de las ventas por comercio electrónico correspondió a un 1.53% del PIB en 2019, es decir un crecimiento del 25% con respecto al año anterior [5]. En comparación con otros sectores económicos es inferior en volumen de ventas, siendo por ejemplo menos de un décimo del comercio tradicional [7].

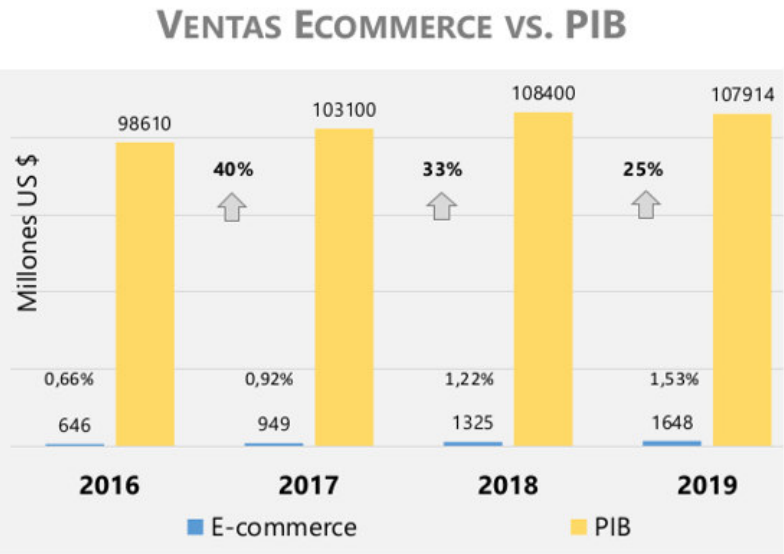


Figura 1.4 Ventas Ecommerce vs. PIB [5]

COMERCIO ELECTRÓNICO EN EL CONTEXTO DE LA PANDEMIA POR COVID 19

La empresa inglesa Kantar [8] ha realizado un estudio internacional desde el inicio de la pandemia sobre la influencia del COVID-19 en el comportamiento de los consumidores, sus actitudes y expectativas en diferentes áreas del mercado. Una de las conclusiones del mencionado estudio referentes al comercio electrónico proyecta que su crecimiento será mayor al del comercio tradicional. La consultora "Dcode Economic and Financial Consulting" coloca al comercio electrónico, e-commerce, como uno de los potenciales beneficiados en este ciclo. La Figura 1.5 muestra los sectores económicos potencialmente beneficiados y los afectados por la pandemia de COVID-19. Se observa en esta Figura al e-commerce entre los potenciales beneficiados. En el Ecuador la frecuencia transacciones por medios digitales ha aumentado con la pandemia. Es así que la frecuencia de compras antes y después del inicio de la pandemia, creció, pues las personas que realizan una vez por semana compras en línea aumentó en un 13%, las personas que compran cada 15 días en un 7%, mientras que las que hacen una compra al mes son un 21% mayor [7].

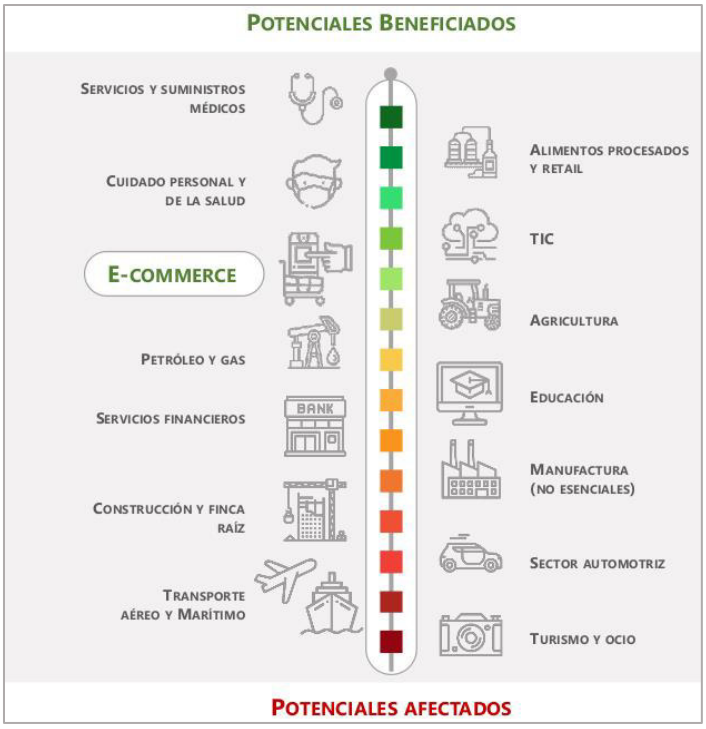


Figura 1.5 Sectores económicos potencialmente beneficiados y afectados por el COVID-19 [9]

En Ecuador podemos ver esta expansión del comercio electrónico, por ejemplo, al observar las descargas de aplicaciones para dispositivos móviles. Antes de la pandemia las diez aplicaciones más descargadas en el Ecuador para el sistema operativo iOS eran de entretenimiento y redes sociales exclusivamente. Después del inicio de la pandemia, se establecen tres aplicaciones de comercio en este grupo selecto: KFC, Rappy y Globo [5]. La Figura 1.6 muestra en detalle el top 10 de descargas de aplicativos móviles antes y después de la pandemia en el sistema operativo antes mencionado.

Figura 1.6 Top 10 descargas de APPs en dispositivos iOS en Ecuador [5]

S ha mantenido una tendencia al crecimiento del uso de e-commerce en los últimos años. Sin embargo, el incremento desde que inicio la pandemia es considerablemente mayor [5]. Las razones para incrementar la frecuencia de uso del comercio electrónico son principalmente un resultado de la pandemia. La Figura 1.7 muestra los motivos para el incremento en la frecuencia de compras no presenciales en el Ecuador.

Figura 1.7 Motivos de aumento de compras no presenciales en el Ecuador [7]

[redacted]TIENDA ELECTRÓNICA

Un sistema de comercio electrónico, e-commerce, puede hacer uso de una o más tiendas electrónicas. Una tienda electrónica, también denominada e-Shop, es un sistema web que permite a los clientes de un negocio electrónico realizar procesos de negocio como registrar productos, revisar catálogo, ordenar, pagar, dar direcciones de entrega, monitorear proceso de entrega, entre otros. Existe una gran variedad de opciones en el mercado de software: de código abierto y gratuitas, como también opciones propietarias de cientos de miles de dólares [1]. La Figura 1.8 esquematiza los procesos y participantes de una tienda electrónica.

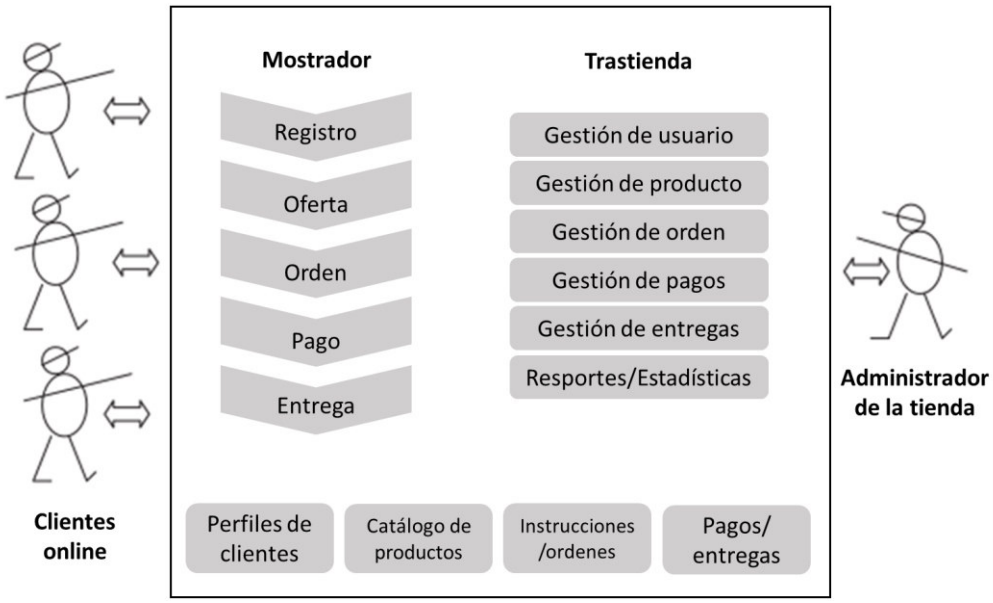


Figura 1.8 Procesos y participantes de una tienda electrónica [1]

FIDELIZACIÓN

El propósito de los sistemas de fidelización es motivar al cliente a favorecer una marca y resistir ofertas de competidores. De esta manera si un competidor, por ejemplo, para expandir mercado al abrir una nueva tienda reduce los precios con descuentos, la compañía que ha fidelizado a sus clientes no los perderá a esta nueva oferta. Los programas de fidelización pueden extenderse a los empleados para volver a la empresa más atractiva en el mercado laboral. Un programa típicamente se reduce a premios que recibe el cliente, sin embargo, estos son ampliamente extendidos y las compañías invierten millones en su estructuración, ejecución, mantenimiento y actualización. Un ciudadano americano en promedio estará involucrado en 18 programas de fidelización [10].

Se puede clasificar los programas de fidelización en tres grupos o fases. El primer grupo se conoce como Lealtad 1.0 o Loyalty 1.0, que se caracteriza por establecer una relación transaccional, ejemplos de este son los programas de cash-back y promociones con tarjetas perforadas. Asimismo, Lealtad 2.0 son programas de fidelización que aprovechan información de segmentación del mercado, de manera que la empresa empieza a utilizar información que tiene sobre los clientes con el fin de personalizar el programa [10]. Estos pueden traducirse, por ejemplo, en campañas de correos electrónicos. No obstante, esta fase fue perdiendo efectividad por resultar en una abrumadora cantidad de correos para el cliente por parte de una gran cantidad de compañías que realizaban la misma práctica.

Por otra parte, algunos elementos como la motivación, Big Data y la ludificación fueron necesarios para llegar al programa de fidelización de tipo Loyalty 3.0. Investigaciones en ciencias sociales han hecho avances que permiten comprender que compele y motiva el comportamiento humano [10]. Del lado de tecnologías de la información ha habido avances que permiten recoger, organizar y analizar grandes cantidades de información. El área de estudio de la informática que se refiere a este tema se conoce como Big Data. Adicionalmente investigaciones sobre los juegos para ocio, educación y estimulación han arrojado importante información y técnicas para lograr participación y compromiso de los usuarios [10].

La experiencia del usuario al realizar transacciones electrónicas en el Ecuador ha sido calificada como buena, placentera, satisfactoria, beneficiosa, útil y favorable. De estas últimas, “placentera” es la característica que, aunque alta entre los usuarios, es la que puntúa más bajo. Un sistema de fidelización mejora la experiencia del usuario en la compra lo que elevaría la puntuación de ese parámetro [7].

DOCKER

Docker software

Docker es un software que sirve principalmente para crear y administrar contenedores. Incluye funcionalidades adicionales como la administración de imágenes y su creación, reportes e interacción con repositorios de imágenes en la nube. Está principalmente desarrollado en Golang, como parte del proyecto en código abierto Moby. Moby es el proyecto creado por la empresa Docker Inc que tiene el fin de desarrollar Docker en comunidad de forma abierta, antes de su lanzamiento Docker era software propietario. Es mantenido principalmente por la compañía estadounidense: Docker Inc. Por su naturaleza de código abierto existen múltiples contribuidores de todo el mundo, individuales y corporativos donde destacan RedHat, Microsoft, IBM, Cisco, y HPE (Hewlett Packard Enterprise) [11].

Docker está disponible para Linux, Windows y MAC. El proyecto busca ajustarse a los estándares establecidos por la OCI (Open container initiative). Actualmente destacan dos estándares publicados por esta entidad en el 2017: image-spec y runtime-spec [11].

Docker Engine

El "Docker Engine" es el software central de Docker que permite correr y administrar contenedores, es análogo a ESXi en la tecnología de máquinas virtuales y es modular en su arquitectura, una tendencia reciente que aún está en proceso, por ejemplo, con el desacoplo de

"runc", un componente que crea contenedores [11]. El "Docker engine" se encuentra compuesto por:

- El cliente: una interfaz en línea de comandos que permite correr comandos de Docker al usuario;
- El "Docker deamon": el proceso que provee el API y otras funcionalidades para el interfaz;
- El componente containerd: un proceso supervisor que administra el ciclo de vida de los contenedores (start, stop, pause). Adicionalmente, tiene funcionalidades de administración de imágenes que fueron agregadas por conveniencia para su uso en Kubernetes. Fue desarrollado por Docker Inc. y donado a la Cloud Native Computing Foundation;
- Componente shim: permite la independencia de funcionamiento con el "Docker daemon". Esto da paso, por ejemplo, a actualizar Docker sin finalizar los procesos de los contenedores. Reporta el estado de los contenedores al "Docker daemon" para la administración centralizada. Existe un componente shim por cada contenedor;
- Runc: es una implementación del estándar runtime-spec de la OCI (Open Container Initiative). Es un software ligero que envuelve el software libcontainer. Su función es la creación de contenedores. Libcontainer es un software desarrollado por Docker Inc. de plataforma agnóstica que permite administrar los recursos de la instancia host para la virtualización. El proceso runc corre en la creación de cada contenedor y se extingue una vez creado, no permanece a diferencia del proceso shim.

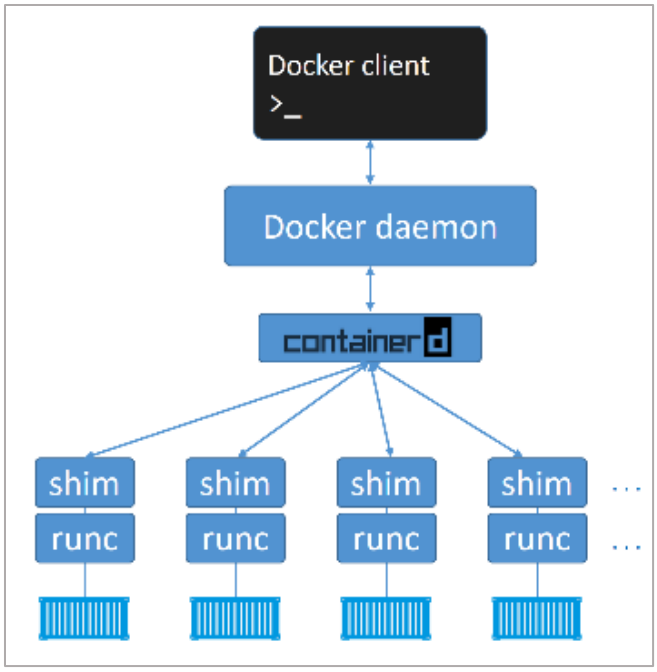


Figura 1.9 Arquitectura de Docker [11]

[redacted]Contenedores

Los contenedores son una tecnología de virtualización de instancias que facilita correr múltiples aplicaciones en un mismo servidor físico. Un contenedor permite crear instancias con su propio conjunto de puertos, establecer redes entre ellos o con el servidor host. Están estandarizados por la OCI (Open container initiative) en el estándar runtime-spec [11].

Un predecesor a la tecnología de contenedores son las máquinas virtuales. La principal diferencia entre ambas tecnologías es que las máquinas virtuales requieren cada una su propio sistema operativo mientras que los contenedores requieren uno solo que pueden compartir. Cada sistema operativo requiere recursos como CPU, memoria RAM y almacenamiento. Los contenedores al compartir un sistema operativo usan menos recursos que las máquinas virtuales por lo tienen más recursos disponibles para aplicaciones. Adicionalmente, cada sistema operativo requiere mantenimiento que debe realizarse por separado y puede requerir una licencia de pago lo que elevaría costos al uso de máquinas virtuales.

[redacted]Imágenes en Docker

Las imágenes en la tecnología de contenedores son equivalentes a las plantillas en la tecnología de máquinas virtuales [11]. Las imágenes permiten crear contenedores. Las imágenes, en Docker, están estructuradas en capas y cada capa es un cambio sobre la capa anterior.

Adicionalmente las imágenes tienen meta data, es así que al momento de crear una imagen cada línea de comando genera una nueva capa y a su vez agrega meta data a la imagen.

Redes para contenedores

Los contenedores pueden comunicarse entre sí con el host o con un host de una red externa. Pueden descubrirse por nombre gracias al servidor DNS embebido que tiene el módulo de Docker para red, así como el sistema de resolución de DNS que tiene cada contenedor.

Volúmenes

Los volúmenes son una funcionalidad de Docker que permiten persistir información de forma conveniente. La información de los contenedores se persiste en un espacio de memoria disponible para el contenedor. Al borrar un contenedor se libera el espacio de memoria del contenedor y se pierde la información guardada en su interior. Los volúmenes son la herramienta para persistir la información de forma independiente del estado del contenedor además de brindar una forma más conveniente de migración y compartición de información. La interfaz para la administración de volúmenes es por medio de APIs y comandos de consola que están incluidos en la distribución de Docker para cualquier sistema operativo [11].

DOCKER COMPOSE

Docker Compose es un software propiedad de Docker Inc., que permite usar un archivo declarativo que describe el sistema y adicionalmente permite levantarlo usando un simple comando. Docker Compose está disponible en Linux, MAC y Windows. Docker Compose utiliza archivos YAML, el archivo por defecto es docker-compose.yml. Se definen cuatro parámetros principales: "version", "services", "networks" y "volumes" [11]. Adicionalmente existen los parámetros de "secrets" y "configs" que se utilizan en el modo Swarm. En "services" se pueden definir el número de réplicas, la política de reinicio, entre otros. En "networks" se pueden crear redes y definir las instancias dentro de las mismas. El parámetro "volumes" permite definir los espacios de memoria para cada servicio a ser usados para persistencia. La versión se refiere a la versión del archivo, no a la del software Docker Compose ni Docker. Se deberá revisar la compatibilidad de la versión de docker-compose, Docker y el archivo YAML.

Una vez definido el archivo se puede levantar el sistema usando el comando "docker-compose up". Para bajar los servicios del sistema se utiliza el comando: docker-compose down". Existen comandos adicionales que permiten revisar logs, reiniciar el sistema y otros comandos del ciclo

de vida de los servicios y de sus procesos que se pueden revisar en la documentación oficial en https://docs.docker.com/compose/.

[redacted]JAVASCRIPT

JavaScript es un lenguaje de programación de alto nivel, interpretado y orientado a objetos y a programación funcional, donde las variables son débilmente tipadas. Inició como un lenguaje de script, sin embargo, ha ido evolucionado a un lenguaje robusto multipropósito utilizado en una gran diversidad de proyectos de ingeniería. JavaScript es el lenguaje más popular en el mundo, según stackoverflow.com por octavo año consecutivo ha mantenido este puesto [12]. Es usado por la gran mayoría de páginas web y soportado por todos los exploradores web modernos de PC, tabletas y teléfonos celulares. Node ha contribuido a la popularización al expandir las posibilidades del lenguaje ya que ha propagado su funcionamiento al sistema operativo pues permite el control de directorios, archivos, etc. La Figura 1.10 muestra algunos de los lenguajes más comunes según la encuesta realizada por stackoverlfow.com.

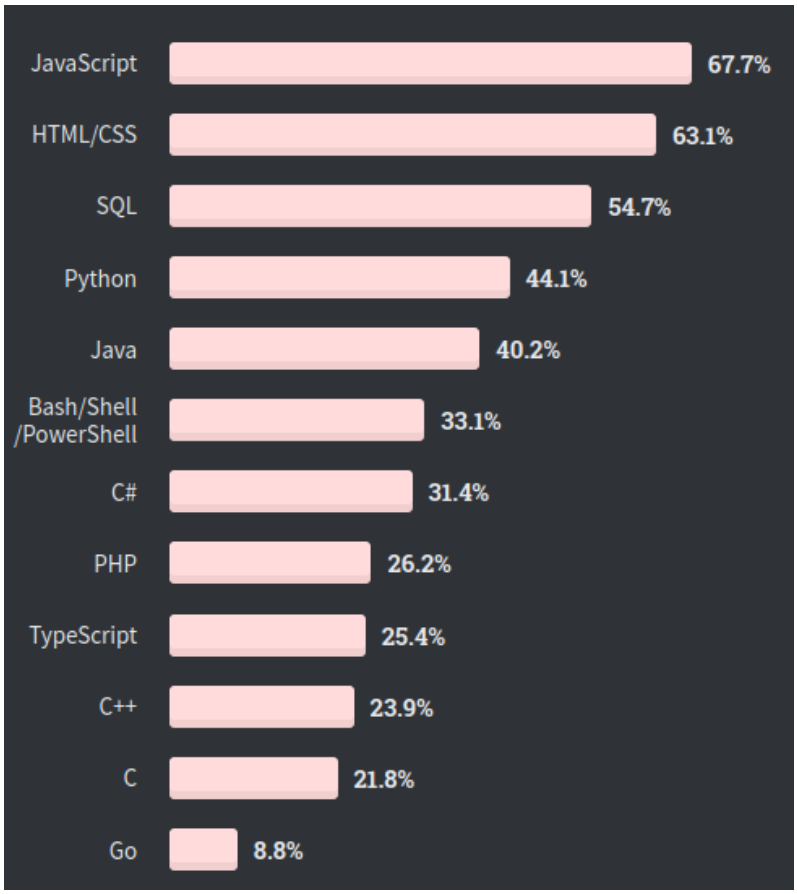


Figura 1.10 Lenguajes de programación más comunes según stackoverlfow.com [12]

Las librerías de JavaScript nucleares presentan un mínimo de funcionalidad para operar con números, texto, arreglos, sets, mapas, entre otros. Tradicionalmente el host ha sido un explorador web, con plataformas como Node o Deno y se agrega la posibilidad de que el host sea un sistema operativo. Entre sus funcionalidades, JavaScript recolecta basura automática que consiste en liberar memoria de variables cuando estas ya no se utilizarán [13]. El interpretador de JavaScript identifica cuando una variable ya no es referenciable y por ende el valor al que apunta ya no se utilizará en el futuro y puede ser liberado de memoria. Por esto, el desarrollador no tiene que liberar explícitamente memoria ni encargarse de la destrucción de variables.

Sintaxis

JavaScript distingue entre mayúsculas y minúsculas en su sintaxis, por ejemplo, en la nominación de las variables; ignora los espacios y saltos de línea; acepta los caracteres de tabulación ASCII adicionalmente al de espacio y acepta caracteres Unicode de espacios [13].

En JavaScript se denomina identificadores a los nombres que se dan a variables, funciones, clases, entre otros. Los identificadores deben comenzar con una letra, un guion bajo o un signo de dólar, las letras subsiguientes pueden ser cualquiera de los caracteres anteriores o ser un digito. Algunas palabras son reservadas por el lenguaje y no pueden ser utilizadas para nombrar variables, funciones o clases ya que tienen un significado especial en el lenguaje, pues son utilizadas para declarar clases, lazos, estructuras de datos, etc. La Figura 1.11 lista algunas de las palabras reservadas. Algunas de estas palabras no tienen una implementación, por ejemplo: enum, implements y public.

```
as      const      export     get         null     target
void
async   continue   extends    if          of       this
while
await   debugger   false      import      return   throw
with
break   default    finally    in          set      true
yield
case    delete     for        instanceof  static   try
catch   do         from       let         super    typeof
class   else       function   new         switch   var
```

Figura 1.11 Palabras reservadas por JavaScript [13]

JavaScript utiliza el "punto y coma" para separar instrucciones. Sin embargo, el uso de este es opcional y la gran mayoría de las veces un salto de línea es suficiente, JavaScript asumirá que la instrucción de la nueva línea es una diferente a la anterior. Por esto existen dos estilos de programación en cuanto al uso de "punto y coma", el primero es utilizarlo siempre aun cuando la

inferencia de JavaScript es suficiente. Otro estilo es usarlo únicamente cuando la inferencia de JavaScript no es suficiente y en cualquier otro caso omitirlo.

Tipos, valores y variables

JavaScript tiene dos categorías de tipos: tipos primitivos y objetos. Los tipos primitivos son: "number" para números, "string" para texto, "boolean" para valores de verdadero o falso, "undefined" y "null" para identificar la ausencia de valor generalmente. Los objetos son una colección de propiedades con pares de nombre y valor. En JavaScript las clases no son reglas sintácticas que el lenguaje impone para verificación de consistencia, sino que son objetos que incluyen parámetros y funciones, envolviéndolas en un ente lógico que puede operar sobre otros o ser su argumento.

JavaScript no realiza una verificación de consistencia de los tipos, sino que sigue reglas para convertir tipos y permitir que una instrucción se realice sin levantar errores. Esta característica es deseable cuando se desea evitar caídas en un sistema web, aunque al costo de errores lógicos son detectables únicamente en la operación. Este comportamiento es altamente indeseable en aplicativos más robustos y que exigen precisión lo que ha sido una de las principales motivaciones para el desarrollo de Typescript que si realiza una y exige consistencia en tiempo de compilación

Las constantes y variables en JavaScript son una forma de utilizar nombres para denominar valores. La declaración se hace sin utilizar tipos. Los tipos son asignados por JavaScript una vez que se inicializan y se infieren del valor asignado. El Código 1.1 muestra la declaración e inicialización de la constante "a" y la declaración de la variable "b".

```
const a = 5;
let b;
```

Código 1.1 Ejemplo de declaración en JavaScript

JavaScript soporta el tipo booleano que puede tomar únicamente los valores de verdadero o falso. Se puede utilizar los operadores AND "&&", OR "||" y NOT "!" para realizar operaciones booleanas en valores booleanos. Todo valor booleano, true-false, truthy-falsy, puede ser usado en estructuras de control como lazos, iteraciones, instrucciones condicionales, etc.

NODE

Node, también conocida como NodeJs, es una plataforma creada por Ryan Dahl que permite utilizar JavaScript en el lado del servidor y construir servidores web de alto rendimiento. Está

basado en JavaScript y está estructurado con el principio de programación de eventos y naturaleza asíncrona. Utiliza el popular y altamente eficiente motor V8 mismo que está desarrollado en C++ y que es utilizado por el explorador Chrome. Dicho motor fue desarrollado en el proyecto Chromium para el explorador del mismo nombre, es de código abierto y puede correr en Windows 7, macOs y Linux. Uno de los principales aportantes al desarrollo del proyecto Chromium es la empresa Google [14].

Node incluye el software complementario NPM (Node Package Manager) que administra dependencias. Es esencial en el uso de node y pone a disposición del usuario una interfaz en línea de comandos que permite obtener las dependencias de un proyecto, así como también funcionalidades adicionales que ponen a disposición como iniciar un proyecto, correr scripts del proyecto, publicar, verificar la versión, buscar proyectos en la nube, entre otros. NPM define las dependencias en un directorio denominado "node_modules", describe las dependencias necesarias en el archivo "package.json" y adicionalmente utiliza el archivo package-lock.json para indicar las dependencias instaladas. El archivo "package.json" también contiene información del proyecto como: nombre, licencia, versión, scripts disponibles, entre otros [14].

TYPESCRIPT

Typescript es uno de los lenguajes más populares en la actualidad para la creación de aplicativos web [14]. Es desarrollado y mantenido por Microsoft en un proyecto de código abierto usando la licencia Apache 2.0, y se lo puede considerar una extensión de JavaScript. Su código es válido, también, en Typescript. Typescript agrega funcionalidades, siendo la principal: tipos de datos; a esto debe su nombre. Además, es débilmente tipado por lo que errores por tipo de datos son comunes, y se desarrollan en tiempo de ejecución, indetectables en etapas anteriores como tiempo de compilación. Typescript resuelve este problema introduciendo tipos y permitiendo la detección de inconsistencias en el proceso de transpilación [14].

La transpilación es un proceso que consiste en la conversión del código a JavaScript. JavaScript luego será transformado a lenguaje de más bajo nivel por algún motor designado para esta tarea como por ejemplo el motor V8 de Node. La transpilación se lleva a cabo por la herramienta "tsc", desarrollado por Microsoft, o con herramientas como Babel.

Adicionalmente a la introducción de tipos de datos, Typescript permite utilizar versiones modernas de JavaScript y transformarlas a versiones anteriores para que puedan ser soportadas por exploradores menos modernos o versiones anteriores de Node. Typescript incluye estructuras comunes a otros lenguajes modernos que no están disponibles en JavaScript como

interfaces, anotaciones y genéricos. También permite integraciones con ambientes integrados para desarrollo, IDEs, lo que permite detectar errores mientras se escribe el código [15].

Typescript puede ser usado para el frontend o el backend en una aplicación web. Se pueden usar en conjunto con Typescript frameworks y librerías populares como React, Angular o Nest.

NESTJS

NestJS es un framework para construir servidores web con Javascript e incluye soporte para Typescript. Combina elementos de los paradigmas de programación orientada a objetos, funcional y funcional reactiva [16]. Estas herramientas constituyen librerías, anotaciones y envolturas de otras herramientas populares como TypeORM, ExpressJS, Dotenv, Axios y Passport. Algunas de las herramientas de NestJS son:

- módulos (modules): un módulo es una de las estructuras principales en NestJS. Una aplicación de NestJS tiene un módulo raíz por proyecto que permite organizar los elementos de la aplicación en bloques, ocultar la implementación de sus funcionalidades y tener control sobre las funciones que expone.
- controladores (controllers): Permiten definir endpoints utilizando anotaciones como: @Get, @Post, @Put, @Delete y @Patch que corresponded a los verbos HTTP: GET, POST, PUT, DELETE, PATCH respectivamente.
- proveedores (providers): Son clases que se pueden utilizar para construir servicios con la lógica de un recurso REST. En este caso será el punto central para procesamiento, persistencia y transformación de datos. Se definen con una clase y la anotación @Injectable. Puede ser inyectada en otros proveedores o en un controlador.
- @nestjs/typeorm: envuelve la librería TypeORM. La librería TypeORM es una herramienta para conexión con una gran variedad de bases de datos como: MySQL, Postgres, SQLite, MSSQL, MongoDB, y muchos otros.
- HttpService de @nestjs/common: envuelve la librería Axios. La librería Axios permite realizar consultas HTTP con NodeJS. Utiliza la funcionalidad de promesas de Javascript para permitir la transmisión de información de forma asíncrona.

NestJS pone a disposición una interfaz en líneas de comandos (CLI) que permite desarrollar aplicaciones de NestJS [15]. Está disponible como un módulo de npm identificado como @nestjs/cli. Algunos de los comandos de esta interfaz son:

- new: crea una nueva aplicación de NestJS
- generate: permite agregar a una aplicación NestJS controladores, decoradores, filtros, interfaces, interceptadores, módulos y más estructuras de NestJS
- build: compila la aplicación a un directorio externo al código fuente
- start: corre la aplicación
- update: actualiza las dependencias del proyecto definidas en el archivo package.json a su versión más actual

REACT

React es una librería de JavaScript que se usa para desarrollar interfaces de usuario de aplicativos web, no es un framework integral, como Angular [14]. Es desarrollado y mantenido por Facebook y es de código abierto desde 2013. Es de las herramientas más populares para frontend junto a Angular y Vue. Es utilizado por aplicativos como: Flipboard y Airbnb. React consiste en cuatro elementos principales: componentes web, propiedades, estado y estilo. Los componentes web son un parte fundamental en React, son abstracciones lógicas de partes de la página que pueden ser rehusados. Los componentes pueden tener diferentes comportamientos según las propiedades y su estado. Las propiedades y el estado son similares en tanto que significan características del componente, la diferencia entre ellos es que las propiedades son inmutables.

React extiende JavaScript introduciendo JSX. JSX permite utilizar HTML (HyperText Markup Language) junto a JavaScript, con esto se logra una programación declarativa en lugar de imperativa por pasos. La programación imperativa se caracteriza por definir instrucciones que corren una por una en orden. La programación imperativa permite describir un componente HTML en lugar de construirlo con instrucciones. JSX soporta todas las etiquetas de HTML y permite crear nuevas customizadas. JSX no es soportado por los exploradores web ni por sistemas como V8 de Node por lo que requiere un paso adicional de preprocesamiento. En este preprocesamiento se utilizan herramientas como Babel y Webpack. Babel realiza la transformación de JSX a JavaScript y HTML. Webpack realiza un proceso de minificación lo que significa condensar archivos de estilo CSS (Cascading Style Sheets), los archivos de programación y los archivos HTML en el mínimo posible, si se puede, uno sólo por cada tipo.

React también se lo refiere con el nombre de React Web cuando se desea especificar su uso en aplicativos de red. Esto se debe a que su concepto se extiende al uso de componentes en general

incluyendo otros escenarios interactivos por ejemplo en un aplicativo móvil, a este se conoce como React Native. Los componentes de React tienen un ciclo de vida que permiten definir comportamientos según eventos. Las etapas del ciclo se identifican con un método. La Figura 1.12 muestra los ciclos de los componentes de React en su renderización y en la re-renderización.

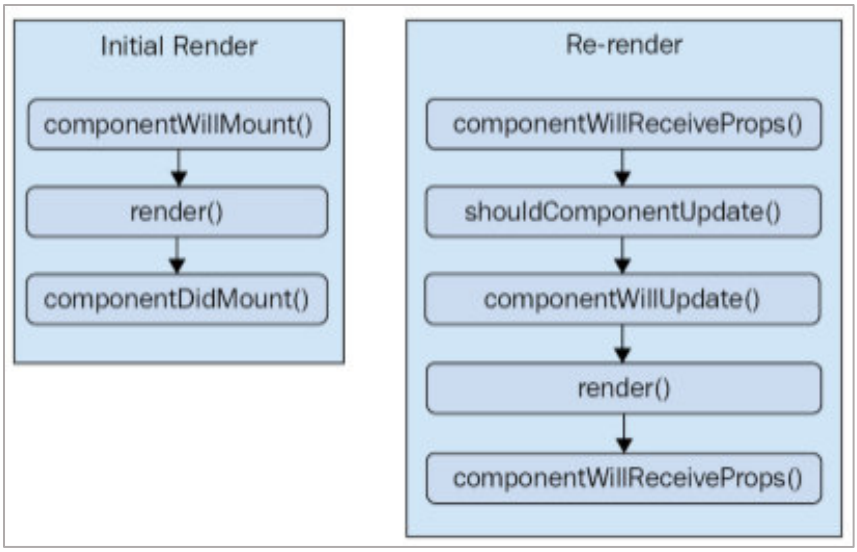


Figura 1.12 Ciclos de los componentes de React [17]

[redacted]IONIC REACT

Ionic es un framework para desarrollo de aplicaciones web progresivas y móviles multiplataforma. Esta desarrollado en un proyecto en código abierto, su código fuente se encuentra publicado en https://github.com/ionic-team/ionic-framework. Utiliza las tecnologías HTML, CSS y Javascript y se integra a las populares herramientas Angular, React y Vue. Ionic se utiliza para construir aplicaciones utilizando elementos tradicionales de HTML, como botones, input boxes, radio buttons, y labels; permite hacer uso de animaciones y gestos que son elementos típicos en aplicativos móviles y posibilita mantener un solo código fuente que puede ser utilizado versátilmente en distintos escenarios como aplicativos web, iOS y aplicativos de escritorio [18]. Para reforzar esta funcionalidad Ionic presenta la funcionalidad Adaptive Styling que adapta la renderización del código según el objetivo de compilación.

Ionic React es la versión de Ionic que utiliza React como elemento central. Incluye más de un centenar de componentes, animaciones y gestos para construir aplicaciones. Utiliza librerías populares y las envuelve en componentes con la misma sintaxis consabida, pero incluyendo funcionalidades adicionales. Con Ionic React se puede mantener un código fuente y utilizarlo

para desplegar aplicaciones móviles de iOS, Android utilizando Capacitor o Cordova de la fundación Apache, aplicaciones web y aplicaciones de escritorio con Electron. Asimismo, soporta cualquier componente compatible con React, de esta manera permitiendo el uso del gran número de librerías creadas para React [18].

Para el mapeo de rutas Ionic React hace uso de la librería React Router a través del elemento de Ionic que lo envuelve: IonReactRouter. Este componente puede ser utilizado para definir rutas hacia un componente predefinido, hacia otra ruta o asociarlo a un bloque de JSX definido dentro del mismo componente de la aplicación raíz. Otro componente importante para la navegación es el elemento IonRouterOutlet, pues permite la renderización de páginas. Ionic organiza la aplicación web o móvil en "páginas" definidas con el componente IonPage [18].

El Código 1.2 muestra un ejemplo del uso de IonReactRouter para definir una ruta hacia http:servidor-ejemplo/dashboard con el componente DashboardPage.

```
const App: React.FC = () => (
  <IonApp>
    <IonReactRouter>
      <IonRouterOutlet>
        <Route path="/dashboard" component={DashboardPage} />
      </IonRouterOutlet>
    </IonReactRouter>
  </IonApp>
);
```

Código 1.2 Ejemplo del uso de IonReactRouter

Ionic provee una serie de componentes para construir una aplicación. Algunos elementos notables son:

- IonApp: es un elemento contendor para una aplicación de Ionic React. Debe existir un solo componente IonApp por proyecto;
- IonPage: es un componente que envuelve cada vista en una aplicación de Ionic React App. A su vez, cada vista a la que se navega usando el método de ruteo de IonicRouter debe incluir un componente IonPage. Además, permite usar IonHeader y IonContent para organizar la información en la vista;
- IonAlert: Es un elemento de diálogo que se presenta al usuario como alerta o como solicitud de confirmación. Puede recoger información utilizando elementos de entrada. La Figura 1.13 muestra un ejemplo de una alerta que solicita una confirmación;

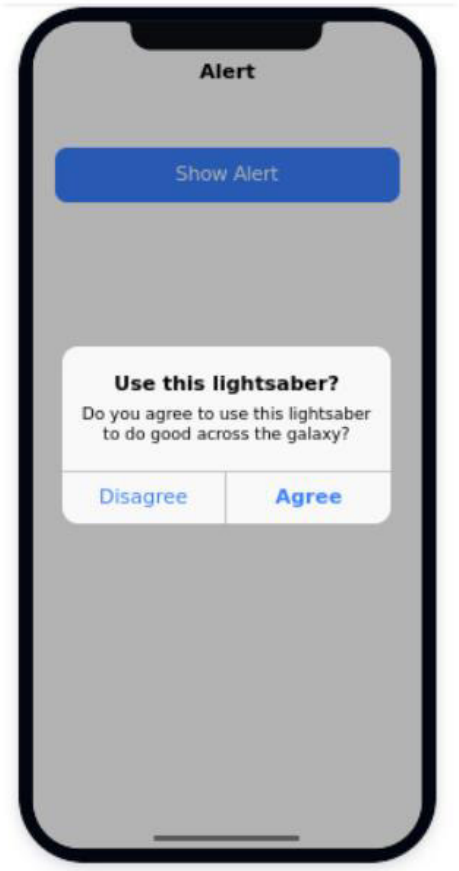


Figura 1.13 Ejemplo de una alerta

- IonButton, IonCheckbox, IonSelect: son envolturas a los elementos nativos de HTML button, checkbox y select. Agregan estilos para mejorar la estética y, en el caso de select permite abrir las opciones en un cuadro de diálogo para mejor la visualización. Las Figura 1.14 muestra el uso de los elementos IonButton, IonCheckbox y IonSelect;

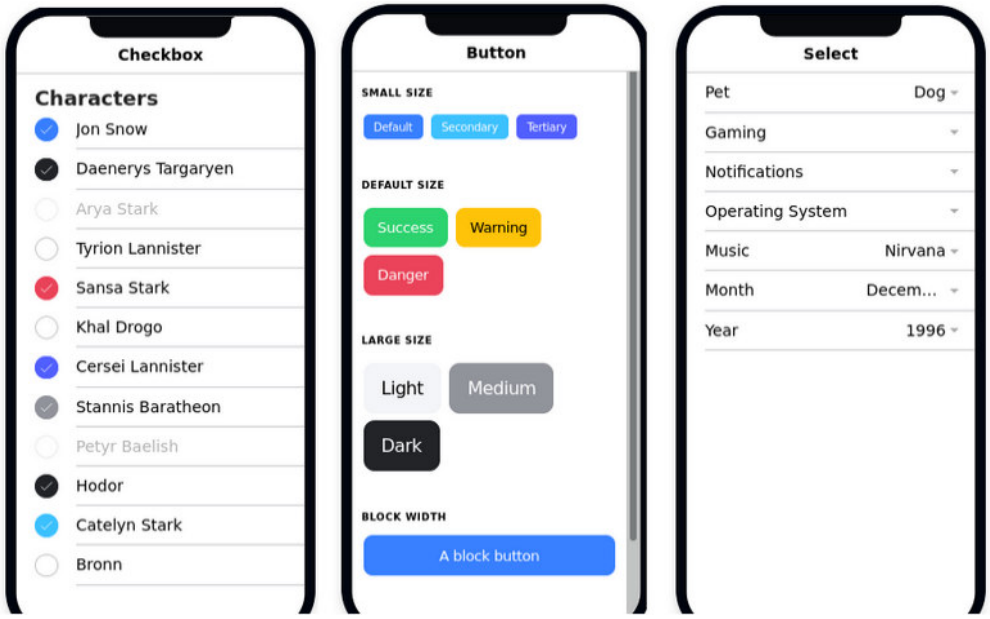


Figura 1.14 Uso de IonButton, IonCheckbox y IonSelect

- IonCard: es un componente contenedor, constituye un elemento que establece un contorno típico para tarjetas. Se puede separar en cabecera (IonCardHeader) y contenido (IonCardContent). La Figura 1.15 ilustra el uso de dos componentes IonCard.

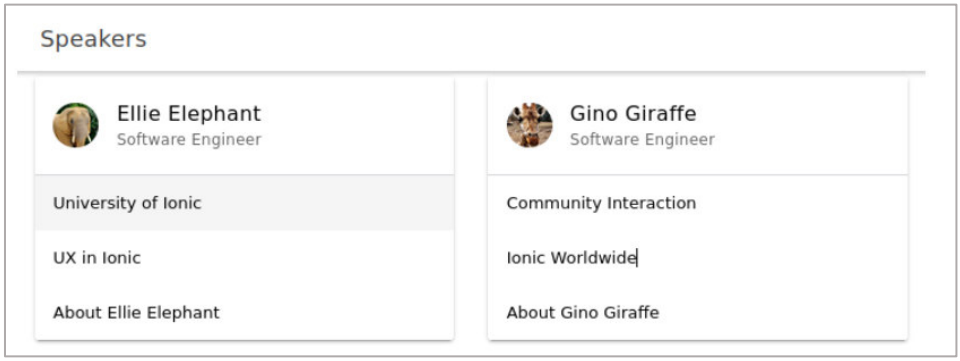


Figura 1.15 Uso de componentes IonCard

Ionic provee una interfaz en línea de comando para el desarrollo de aplicaciones con Ionic. Esta desarrollado con TypeScript y Node.js. Tiene soporte para Node 10.3 en adelante. Es un proyecto de código abierto y su código fuente se encuentra en https://github.com/ionic-team/ionic-cli. Algunos de los comandos principales son:

- deploy: permite el despliegue de la aplicación utilizando la infraestructura de Ionic;
- start: permite crear un nuevo proyecto de Ionic;

- build: crea un directorio optimizado para despliegue en producción;
- capacitor: permite utilizar operaciones de Capacitor para la compilación del código fuente de un proyecto de Ionic para crear un aplicativo móvil;
- serve: inicia un servidor web para desarrollo de un proyecto de Ionic.

La Figura 1.16 muestra la respuesta de la ejecución del comando ionic info dentro del proyecto de ejemplo de ionic que se encuentra en https://github.com/ionic-team/ionic-conference-app.

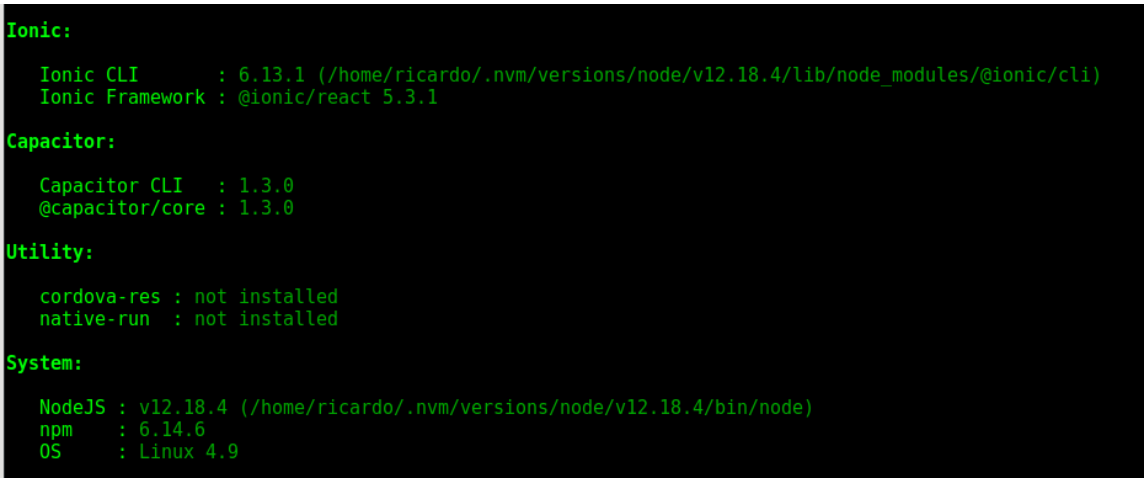


Figura 1.16 Respuesta de la ejecución del comando ionic info

[redacted] PRUEBAS QA

En el desarrollo de software es importante considerar en todo el proceso la calidad. Los procesos de aseguramiento de la calidad de software se pueden dividir en dos categorías:

- Procesos estáticos: son las examinaciones de documentos y código fuente, la inspección de valores persistidos y algoritmos usados, en general las revisiones continuas del software del sistema sin ejecutarlo;
- Procesos dinámicos: es la inspección del funcionamiento del software a través de su ejecución controlando sus datos de entrada, el ambiente y la inspección de los resultados.

En el proceso de aseguramiento de la calidad se definen dos conceptos importantes para la evaluación de software: verificación y validación. La verificación es una actividad que evalúa la exactitud y precisión del desarrollo en una fase del proceso, de esta manera permite asegurarse que se está realizando el proceso de desarrollo de forma correcta, es decir el desarrollo del producto correcto. Por otro lado, la validación constituye una actividad para confirmar que las funcionalidades del software desarrollado estén acorde a las expectativas de las

especificaciones. En este sentido, para las actividades de validación la ejecución del software debe realizarse en sus condiciones reales. Utilizar ambientes que simulan las condiciones reales es una práctica común y facilita la validación, pero tiene sus limitaciones según qué tan similar es el ambiente simulado al real.

Una actividad importante en el proceso de aseguramiento de la calidad es la realización de pruebas. Se definen tres términos comunes en el léxico de resultados de pruebas: falla, error y defecto [20]. Una falla ocurre cuando el comportamiento del sistema no se ajusta a sus especificaciones y expectativas. Un error es un estado del sistema y a su vez un sistema con errores tendrá fallas en su ejecución. Los defectos son las falencias responsables de los errores, por ejemplo, las arquitecturas inconvenientes y los algoritmos ineficientes o equivocados. Previo a la realización de pruebas, es convenientes planificarlas y documentarlas en un documento conocido como el "Plan de pruebas". Un estándar para documentación de pruebas es el estándar IEEE 829 que tiene los siguientes propósitos [19]:

- establecer un contexto común para las actividades de pruebas en cualquier fase del ciclo de desarrollo;
- definir para la documentación de pruebas los conceptos de diseño de pruebas, casos de pruebas, reporte de anomalías, registro de pruebas, nivel de pruebas;
- definir un "Plan de pruebas Master" y un "Plan de pruebas de nivel".

HTTP

HTTP, siglas para HyperText Transfer Protocol, es un protocolo de capa de aplicación definido en el RFC 1945 y RFC 2616. HTTP utiliza una arquitectura cliente-servidor. Los clientes son los exploradores web. Un servidor web puede ser, por ejemplo, un programa desarrollado en JavaScript utilizando Node y Express o en Java utilizando Spring [21].

Los exploradores web solicitan y despliegan páginas web a un servidor que implementa HTTP del lado del servidor. Una página Web es una colección de objetos, donde los objetos son archivos HTML, imágenes JPEG, applets o videos a los que se pueden identificar con una URL. HTTP no define como mostrar el contenido de un archivo HTML sino, el cómo realizar la comunicación a través del intercambio de mensajes. HTTP utiliza el protocolo de capa de transporte TCP [21]. Existen dos tipos de mensajes en HTTP: mensajes de consultas y mensajes de respuestas. Los mensajes de consulta están escritos en ASCII.

La Figura 1.17, esquematiza las partes de un mensaje de consulta de HTTP. Se puede dividir el mensaje en tres partes. La primera línea es la línea de consulta incluye el método HTTP, la URL y la versión. Ejemplos de métodos HTTP son: GET, POST, PUT, PATCH. A continuación de la línea de consulta se encuentran las líneas de cabecera, estas son pares de nombre y valor.

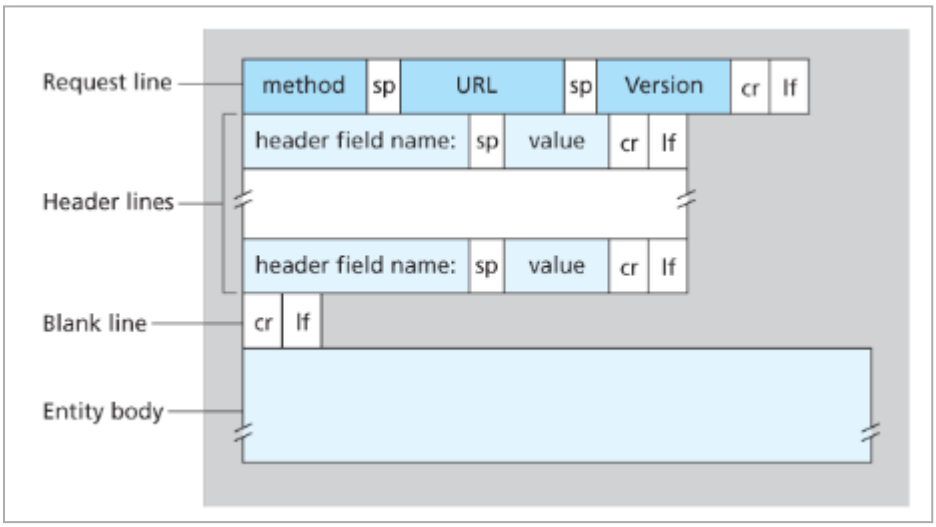


Figura 1.17 Esquema de las partes de un mensaje de consulta de HTTP [21]

La Figura 1.18 esquematiza las partes de un mensaje de respuesta de HTTP. La primera línea se conoce como línea de estado, esta incluye la versión, un código de estado y una frase asociada. Los códigos de estado están estandarizados en el RFC 7231. Algunos códigos de estado comunes, su frase asociada y su significado a continuación:

- 200 OK: la operación solicitada se ha realizado con éxito;
- 201 CREATED: un nuevo registro ha sido persistido en memoria, por ejemplo, en una base de datos;
- 400 BAD REQUEST: el formato de la data enviada es inválido, lo que ha ocasionado un error;
- 401 UNAUTHORIZED: autenticación requerida;
- 404 NOT FOUND: recurso no encontrado;
- 405 Method NOT ALLOWED: la URL no soporta el método enviado;
- 500 INTERNAL SERVER ERROR: error del lado del servidor.

Luego de la línea de estado se tiene las cabeceras. Algunas cabeceras típicas en una respuesta HTTP son:

- Date: para indicar la fecha y hora de la creación del mensaje de respuesta;

- Server: identifica al servidor que realiza la respuesta, ejemplo: Apache/2.2.3 CentOs;
- Last-modified: indica la fecha y hora de la última modificación del objeto de la respuesta;
- Content-Length: indica la longitud en bytes del objeto retornado;
- Content-Type: indica el tipo de objeto en la respuesta, tipos comunes son HTML, JSON, XML;

Luego de las cabeceras separado por una línea vacía se encuentra el objeto retornado.

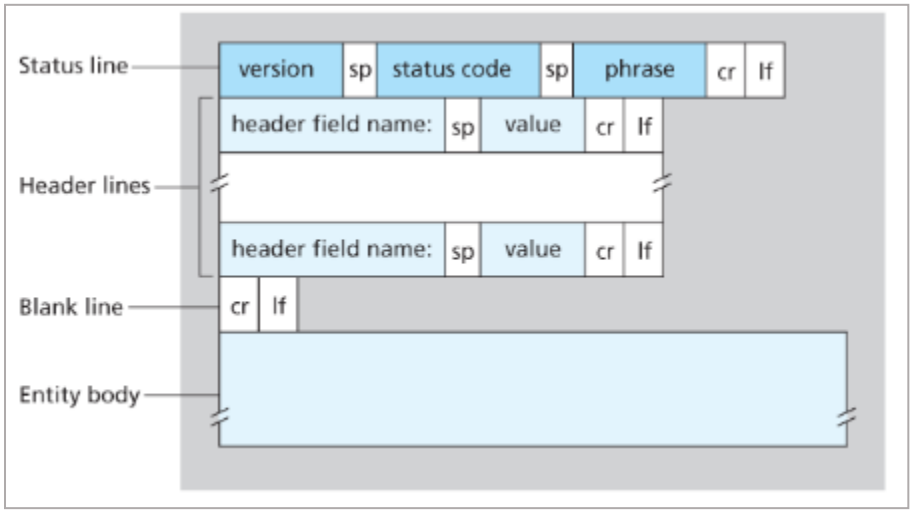


Figura 1.18 Esquema de las partes de un mensaje de respuesta de HTTP [21]

[redacted]REST

Representational State Transfer (REST) es un conjunto de principios que describen como HTTP puede ser utilizado para definir una interfaz para un sistema remoto. Aunque en teoría REST no está limitado a ser usado con HTTP en la práctica no se utilizan otros protocolos. REST es independiente de un lenguaje de programación o sistema operativo [14]. Es una lista de buenas prácticas y patrones de diseño para crear un API de una forma estándar. La red informática mundial, World Wide Web, es una implementación de REST.

Los componentes de un API según REST son los recursos. Los recursos se usan para modelar un sistema, pueden ser un registro de un usuario, un registro de un producto, el catálogo de productos de una tienda, etc., describen al elemento que representan utilizando JSON o XML. XML, Extensible Markup Language es un lenguaje demarcado para documentos que contienen información estructurada. La raíz de XML es el documento, que puede representar un gráfico, una transacción de comercio electrónico, ecuaciones matemáticas, registros de bases de datos

y muchos otros. XML es similar a HTML, pero es más flexible ya que el significado de sus etiquetas no es fijo ni limitado a un conjunto definido.

JSON, JavaScript Object Notation es un formato ligero para intercambio de datos que es fácil de leer para humanos, así como fácil de procesar por computadoras. JSON no está restringido a un lenguaje de programación, se utiliza ampliamente en C, C++, Java, JavaScript, Perl, Python, entre otros. Además, define sus registros en pares de campo y valor, también permite agrupar estos en arreglos. JSON suele ser preferido sobre XML porque es más ligero y más rápido de procesar, aunque a costo de rigidez lo que dificulta su validación.

REST define tres componentes: URLs, métodos HTTP y el formato de datos de los recursos. La URL permite identificar un objeto que puede ser un recurso o una acción [14]. Se establece el uso de sustantivos en la URL para identificar recursos, por ejemplo, la URL para definir un producto podría ser http://my-e-commerce/product. Las acciones sobre el recurso se definen con una cabecera especial de HTTP, la cabecera de "method". Típicamente se usa GET para listar información del recurso, POST y PUT para realizar cambios en recursos y DELETE para borrar recursos en el sistema. Los mensajes de respuesta REST incluyen un campo que identifica el estado de la respuesta pudiendo ser este de éxito o error. La cabecera que se utiliza es la de "statusCode". Esta tendrá un valor numérico que identifica el estado de la respuesta y se utilizan los códigos de estado de HTTP.

[redacted]CEZERIN

Cezerin es un software de código abierto y de libre uso bajo la licencia MIT. Información sobre el proyecto se puede encontrar en la página del proyecto [22]. Asimismo, la documentación del API, que es pertinente para el prototipo se puede encontrar en su página oficial [23]. Está compuesto por tres elementos que son:

- store: el backend que corre en el servidor. Gestiona los productos, categorías de productos, clientes, órdenes, autenticación, etc.;
- web: el aplicativo web para el explorador para los usuarios finales;
- admin: el aplicativo web para operadores del e-commerce.

Cezerin está desarrollado en JavaScript utiliza Node, de versión superior a la 8, y React. Entre las librerías destacadas que usa están: express, babel, webpack, moment, lodash, eslint. Una base de datos no relacional MondoDB, es necesaria por el componente backend para la persistencia de información de productos, clientes, etc. La configuración se realiza con variables de ambiente, y el repositorio no cuenta con un archivo de Docker Dockerfile.

Ofrece dos temas para el aplicativo web de clientes: un tema deafult y el tema "Plusha" que está en venta en la página oficial [23]. Los temas son variaciones estéticas del aplicativo web, y se pueden realizar cambios adicionales en la presentación del aplicativo web por medio de cambios en el código. El componente del servidor ofrece un API Rest para la administración de la tienda electrónica que incluye también funcionalidades de campos personalizables para envíos, webhooks para notificación de eventos, access tokens para autenticación y renderización desde el servidor que soporta SEO (Search Engine Optimization).

SERVICIOS EN LA NUBE PROVISTOS POR AWS

La computación en la nube se refiere al uso remoto de instancias para suplir requerimientos de infraestructura. Esto permite a un usuario crear aplicaciones reduciendo la necesidad de invertir recursos en infraestructura y personal para mantenerla. Algunos proveedores comunes de infraestructura para computación en la nube son Amazon Web Services (AWS), Microsoft Azure, Huawei, IBM y Alibaba Cloud, y son utilizados como servidores web, para procesamiento de cálculos, bases de datos relacionales y no relacionales, servicios de autenticación para seguridad, networking, etc.

Una tecnología central en cualquier sistema de computación en la nube es la virtualización, que permite dividir los recursos de hardware de un sistema físico. La computación en la nube presenta las ventajas de ser escalable, elástico y más barato que la alternativa tradicional de un servidor físico. Este último tiene un alto costo en la compra, más los recursos que utiliza de electricidad ventilación, mantenimiento y puesta en marcha.

AWS incluye instancias de tipo EC2 (Elastic Cloud Computing) que son multipropósito, instancias de tipo RDS (Relational Database System) que son específicamente para bases de datos relacionales, private keys que son llaves de seguridad para el acceso con SSH, entre otros componentes [24]. AWS mantiene una red interna y permite la comunicación entre instancias utilizando la red privada de Amazon sin salir a la red pública del Internet.

Elastic Cloud Computing, EC2, es un servicio de AWS que provee una versión virtual de un servidor. Una instancia EC2 puede ser configurada con características como CPU, memoria, almacenamiento, e interfaces de red personalizadas a la medida de los requerimientos de la aplicación para la que serán utilizadas [24]. Su despliegue es rápido, se lo realiza en una interfaz web que también permite comandos básicos para detener, reiniciar y encender una instancia. Una instancia puede correr un gran número de gestores populares de bases de datos

relacionales como: Postgres, Mysql, MariaDb, entre otros. Las características de memoria, CPU, almacenamiento en disco son personalizables.

AWS permite construir aplicaciones que interactúen con sus servicios en la nube por medio del uso de sus SDK (Software Development Kit). AWS actualmente provee SDKs para nueve lenguajes de programación en los que se incluyen: JavaScript, Java, .Net y Python. También provee SDKs para desarrollo de aplicativos móviles en Android y iOS. También existen plugins para ambientes integrados de desarrollo como Eclipse y Visual Studio Code [24].

2. METODOLOGÍA

Para la planificación del sistema se realizó un análisis de requerimientos y se llevó a cabo un proceso de diseño. El análisis de requerimientos es una descripción del contexto y de las expectativas en general del sistema. Este análisis ayuda a direccionar el desarrollo y establece algunos flujos necesarios en el sistema y ciertas condiciones básicas de funcionamiento. De esta manera, el primer paso fue identificar los actores del sistema y los procesos en los que participan. Estos procesos, conocidos como casos de uso, se los describen en forma de historias de usuario utilizando la metodología ágil SCRUM [25, 26]. Con este análisis de requerimientos como base, se diseña el prototipo.

En el proceso de diseño se definieron las tecnologías a usar en específico, entrando a detalle en la versión, librerías, y métodos a utilizar. Además, se definieron: la arquitectura lógica del sistema, la arquitectura física de la infraestructura, los flujos de procesos, las estructuras de datos para la persistencia de información y los microservicios que componen el sistema. Las arquitecturas se documentaron con esquemas que detallan los componentes del sistema organizado de forma lógica o física según corresponda. Los flujos de procesos, por ejemplo, el registro de un usuario, se definen con diagramas de flujo, mientras que todo microservicio del sistema se define con las tecnologías que se usaron en su desarrollo y su rol dentro de la arquitectura lógica, física y en los procesos en los que participa. Finalmente, como parte del diseño se elaboró el plan de pruebas compuesto por casos de prueba. Este plan permitió verificar que se han seguido los lineamientos del análisis de requerimientos, que los procesos definidos en las historias de usuario se pueden realizar sin errores y se pueden explorar funcionalidades adicionales del sistema.

Una vez realizado el diseño se realizó el desarrollo en local del sistema. Para esto se levantaron los sistemas de correos y bases de datos y se realizó su configuración. Se levantó en local el sistema de e-commerce. Luego, se desarrolló el código de para el servidor web, el aplicativo web, el aplicativo web administrativo y el servidor web de integración del sistema de fidelización. Se realizaron las configuraciones necesarias para integrar todos los sistemas. Finalmente, se desarrolló el script de Docker y de Docker Compose.

ANÁLISIS DE REQUERIMIENTOS

Inicialmente se identificaron y definieron los actores del sistema y los procesos en los que participan al hacer uso del prototipo. Luego, se definieron los requerimientos generales para el

desarrollo del sistema. Este análisis de requerimientos fue una guía para el desarrollo y buscó describir cualitativamente las expectativas del sistema.

[redacted]ACTORES DEL SISTEMA

Los usuarios del sistema se clasificaron según el rol que tienen en el sistema. Se identificaron dos roles: operador y usuario del e-commerce y se definieron de la siguiente manera:

Usuario e-commerce: usuario que realiza compras en una tienda virtual. Puede revisar catálogos de productos, detalles de productos, como fotografías y precios, información registrada de su usuario, revisar tanto información de una orden como su estado. También puede revisar información de fidelización que le corresponden, como sus tarjetas de fidelización.

Operador: es el usuario administrador el e-commerce. Puede cargar nuevos productos al sistema, crear nuevas categorías de productos, asignar productos a una categoría, revisar estadísticas, revisar estado de órdenes, revisar información de clientes en el sistema, realizar configuraciones en el sistema de e-commerce. Puede, también, revisar la información de fidelización de todos los usuarios, asignar tarjetas a usuarios y marcar como redimidas cuando ya se han otorgado los premios a los que se ha hecho acreedor un usuario.

[redacted]HISTORIAS DE USUARIO

Para la pila de productos o (product backlog) se utilizaron en este proyecto las historias de usuario [27]. Estas historias de usuario no constituyen un conjunto estricto de requerimientos a cumplir, sino una guía de desarrollo, es así que se pueden realizar ajustes cuando es conveniente para el diseño y según la prioridad de las funcionalidades. Por esto, algunas de estas historias se cambiaron o eliminaron. A continuación, la Tabla 2.1 muestra las historias de usuario definidas:

Tabla 2.1 Historias de usuario definidas

No. Historia de usuario	**Titulo**	**Actores del sistema**	**Resumen**
1	Autenticación en aplicativo web	Usuario e-commerce.	Usuario accede a pantalla de login donde utiliza sus credenciales: nombre de usuario y contraseña. Cuando son correctas, el sistema permite el acceso.

2	Autenticación en el aplicativo web de administración	Operador	Un operador del e-commerce accede al aplicativo web administrativo que le solicita en un formulario su nombre de usuario y su contraseña. El operador llena el formulario. Si las credenciales son correctas. El sistema le permite continuar y utilizar el aplicativo web.
3	Revisión de catálogos de productos	Usuario e-commerce	El usuario puede observar una lista de productos disponibles, esta lista puede estar acotada por conveniencia, pero permite revisar todos los productos. El usuario puede hacer uso de herramientas de búsqueda para localizar un producto.
4	Proceso de compra	Usuario e-commerce	Un usuario e-commerce puede realizar una orden de uno o varios productos, indicando la cantidad que desee y la dirección de entrega.
5	Creación, modificación y eliminación de registro de un producto	Operador	Un operador en el aplicativo web administrativo puede crear nuevos registros de productos, definiendo nombre, precio, e imágenes asociadas, puede editar y borrar registros existentes.
6	Creación, modificación y eliminación del registro de una categoría	Operador	Un operador usando el aplicativo administrativo crea una nueva categoría de productos definiendo su nombre, también puede editar o borrar registros existentes.
7	Asociación de un producto a una categoría	Operador	Un operador usando el aplicativo web puede asociar un producto del e-commerce a una categoría. Puede también remover un producto de una categoría.
8	Consulta de órdenes	Operador	Un operador puede revisar las ordenes creadas por todos los usuarios en el sistema: los productos ordenados, el monto y la dirección de entrega definida por el usuario al realizar la compra.
9	Consulta de clientes.	Operador	Un operador puede revisar una lista de los usuarios registrados, información de los mismos y las ordenes que han realizado.

10	Definición de tarjeta de fidelización y asignación a un usuario del e-commerce.	Operador	Un operador puede definir una tarjeta de fidelización con un título, un número de perforaciones y un conjunto de premios que recibirá el cliente al completar la tarjeta.
11	Revisión de tarjetas asignadas.	Usuario e-commerce	Un usuario puede revisar las tarjetas que le han sido otorgadas, revisar los premios que recibirá al completarlas y las perforaciones que ha obtenido, así como si la tarjeta ya ha sido redimida.
12	Compra resulta en agregar perforación a tarjeta.	Usuario e-commerce.	Cuando un usuario realiza una compra en el sistema de e-commerce y tiene tarjetas asignadas que no han sido completadas, obtendrá una perforación en una de estas tarjetas como resultado de su compra, acercándole a la obtención de un premio.
13	Colocación de marca de premios redimidos a tarjeta de fidelización.	Operador	Un operador puede agregar una marca de premio redimido a una tarjeta que ha sido completada y cuyos premios asociados han sido ya entregados al usuario al que le corresponde la tarjeta.

REQUERIMIENTOS DE DOCKERIZACIÓN

El sistema no requirió que el host tenga instalado software para soportar lenguajes, entornos de desarrollo integrados (IDES), administradores de paquetes propios de un lenguaje como npm, librerías globales o locales de node. El sistema necesitó únicamente: Docker, Docker Compose y el script de Docker Compose correspondiente al sistema exclusivamente.

El script de Docker Compose fue guardado en un repositorio en la nube que utilizó el software de control de versiones Git. El script usó la versión 3.8 y variables de ambiente para distinguir entre los dos ambientes del proyecto: desarrollo y producción. Asimismo, cada subsistema tuvo su propio archivo Dockerfile para la dockerización. La ubicación del archivo Dockerfile se conoce como el "contexto de construcción" y su ubicación preferente es la raíz del proyecto. La presencia

de este fue documentada en el archivo README.md ubicado comúnmente en la raíz del proyecto. Se utilizó el archivo ".dockerignore" para excluir los archivos del directorio del contexto de construcción y de esta manera reducir el tamaño de la imagen.

Se utilizó la imagen más ligera posible. En este sentido se eligieron las imágenes basadas en una versión Slim de Debian o una imagen basada en Linux Alpine por las ventajas de tamaño y seguridad. Las imágenes de Linux Alpine tienen un tamaño de menos de 6MB. Se documentaron con comentarios sólo cuando se consideró necesario.

Para incluir información de licencia, autor o cualquier otra información no funcional se utilizó la instrucción "LABEL". Se debe utilizar la instrucción EXPOSE para indicar los puertos que utiliza la imagen descrita por el Dockerfile, esto es únicamente con fines de documentación y no tiene un objetivo funcional, sin embargo, es lo recomendado como buena práctica. Docker provee dos instrucciones con idéntica funcionalidad de copia de archivos: ADD y COPY, la primera tiene la capacidad adicional de descompresión y de búsqueda de archivos remotos. No obstante, se debe preferir COPY para copiar archivos, excepto en los casos que requieran descompresión y lectura de archivos remotos. La lectura de archivos remotos debe ser evitada en lo posible. Se debe utilizar una ruta absoluta como parámetro de la instrucción WORKDIR. Se deben evitar rutas relativas y no se deben utilizar "RUN cd .." como sustituto de WORKDIR porque dificulta la lectura y mantenimiento [16].

Las imágenes de Docker, definidas en el archivo Dockerfile, deben ser "efímeras". Docker Inc. define como imágenes efímeras a las imágenes que requieren el mínimo de configuración: deben ser fáciles de reconstruir y reemplazar. El sistema debe lanzarse con un solo comando y en pocos minutos debe estar listo para ser utilizado. Al reiniciar la instancia el sistema debe levantarse automáticamente. Finalmente, todo el código necesario para correr el prototipo debe estar disponible en un repositorio público, preferentemente el repositorio en la nube GitHub [16].

[redacted]REQUERIMIENTOS DEL SISTEMA DE FIDELIZACIÓN

El sistema de fidelización construyó en un sistema de tarjeta de fidelización. Las tarjetas tienen definido un número de perforaciones que se obtienen al realizar una compra. Al completar el número de perforaciones definidas por la tarjeta, el usuario del e-commerce se hace acreedor a uno o varios premios. El sistema de fidelización incluyó un servidor web que expone un API REST, un aplicativo web para los usuarios en general y un aplicativo web administrativo.

Para el desarrollo del servidor web se utilizó lenguaje de programación JavaScript con Typescript, el motor Node y el framework NestJS. Para la persistencia de datos se utilizó el gestor de bases

de datos PostgreSQL [28]. En el servidor web se utilizó la popular librería Express y el módulo HTTP de NestJS para comunicación HTTP y la librería TypeORM para la conexión del servidor web con la base de datos. Se utilizó el módulo de nestjs/config para la configuración del sistema utilizando variables de ambiente.

Para el frontend se utilizó el lenguaje JavaScript, Typescript con la librería React y el framework Ionic. Se prefirió siempre los componentes de Ionic sobre componentes de otras librerías o componentes de HTML. Se manejará la librería axios para realizar consultar HTTP, mientras que para la administración del estado de la aplicación se utilizó la librería "Redux". Todos los subsistemas: servidor y aplicativos web fueron provistos de un script de Docker. El desarrollo se realizó usando los módulos "eslint" y "prettier" para mantener una buena estructura de código.

DISEÑO

En esta sección se describe el diseño del sistema en términos de los elementos que lo componen y su función, la arquitectura física, las tecnologías utilizadas especificando versiones, librería y configuraciones. Además, se presentan el diseño del servidor en el formato OpenAPI donde se especifican los endpoints que se exponen, los parámetros y ejemplos de respuestas; los diseños de pantallas para los aplicativos web; y los esquemas de las bases de datos, así como los productos de Amazon su configuración y su rol en el prototipo.

Finalmente se muestra el plan de pruebas empezando con los casos de prueba como base para luego entrar a detalle en cómo se conducirán las pruebas: herramientas de software que se utilizarán, cuantes veces se probará, en qué momento, que ambiente y los criterios de aceptación.

ARQUITECTURA LÓGICA

La arquitectura lógica especifica los componentes del sistema de acuerdo a funcionalidad independientemente de la infraestructura utilizada para su despliegue. Se especifican las funciones de cada servidor y software en forma general. La Figura 2.1 muestra la arquitectura lógica del sistema.

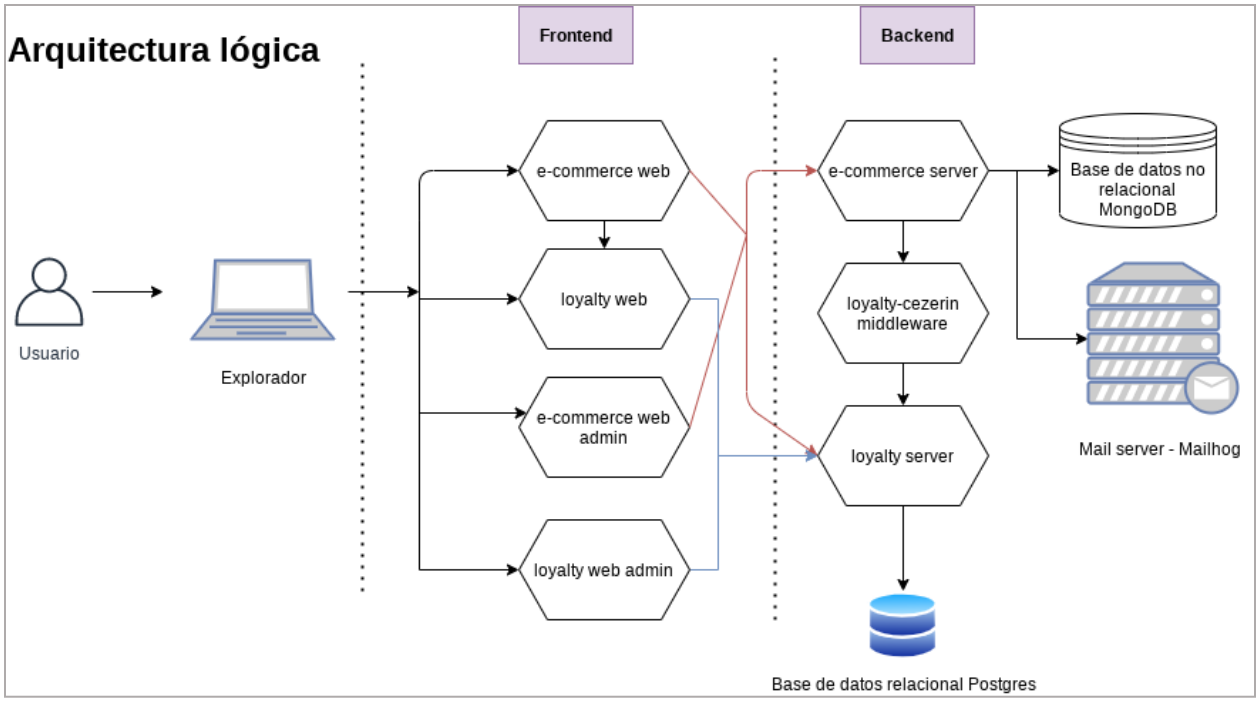


Figura 2.1 Arquitectura lógica

El sistema está compuesto por:

- un aplicativo web para el e-commerce: es la tienda electrónica que un cliente puede acceder para consultar el catálogo de productos y realizar compras. Contiene una sección para consultar información sobre fidelización;
- un aplicativo web administrativo para el e-commerce: es el interfaz desde el cual un operador puede realizar configuraciones del sistema, administrar productos, revisar listas de clientes, ordenes realizadas, administración del e-commerce en general;
- un aplicativo web del sistema de fidelización: muestra información de tarjetas de fidelización y permite filtrar por clientes;
- un aplicativo web administrativo del sistema de fidelización: aplicativo desde el cual un operador puede administrar el sistema de fidelización: crear nuevas tarjetas, asignarles premios, revisar las tarjetas existentes y marcarles como redimidas;
- un servidor web de e-commerce de código abierto;
- un servidor web de fidelización;
- una base de datos no relacional mongo-db: base de datos utilizada por el servidor de e-commerce;

- una base de datos relacional, postgresql: base de datos utilizada por el servidor de fidelización;
- un servidor de correos: utilizado por el servidor de e-commerce.

ARQUITECTURA FÍSICA

Los elementos del sistema definidos de forma lógica según su rol en el prototipo deben ser desplegados en un instancia o conjuntos de instancias conectados por redes. Es deseable tener una instancia para cada componente del sistema, de manera que un error en una instancia no cause que todo el sistema falle. Sin embargo, una mayor cantidad de instancia causa incurrir en costos más altos. La virtualización es una solución común para tener modularizar el sistema dentro de una sola instancia física. En este prototipo se utilizó Docker para construir una instancia virtual para cada componente dentro de una instancia en la nube.

Para la instancia en la nube se utilizó el producto de Amazon Web Services Elastic Computing Cloud o EC2. La instancia requiere un sistema operativo, Docker y Docker-compose instalado. El sistema operativo preferido será Ubuntu Server 18.04, las instancias EC2 permiten configuraciones de red, por ejemplo, bloquear ciertos puertos, direcciones IP, entre otras. Se realizaron de manera ilustrativa el bloqueo de ciertos puertos para explorar estas opciones.

Las instancias tradicionalmente requieren la instalación de paquetes y software que permita correr los servidores web, bases de datos, entre otros. Sin embargo, por el uso de Docker, esta solo requiere Docker y Docker-compose. Esta tecnología simplificó considerablemente la puesta en marcha, limitándola a un ejecutar un comando.

Docker permite configuraciones de redes en los contenedores, se realizaron configuraciones simples de manera ilustrativa. Por ejemplo, permitiendo el tráfico hacia la base de datos de Docker solo desde el backend del e-commerce. La Figura 2.2 muestra un diagrama de la arquitectura física que gracias a la virtualización se simplifica a una sola instancia. El sistema está compuesto por:

- Instancia EC2: es una instancia en la nube provista por Amazon, utiliza el sistema operativo Ubuntu y tiene instalado Docker y Docker-compose.

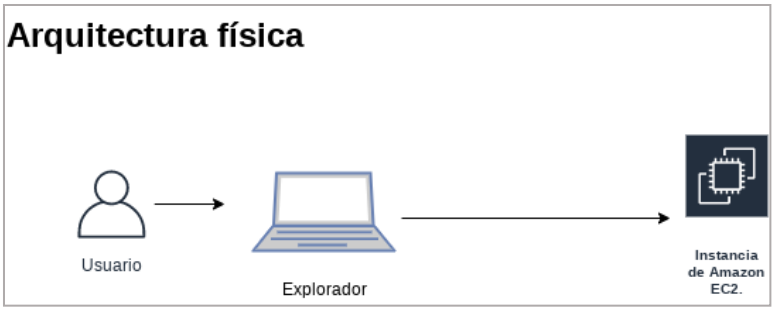


Figura 2.2 Arquitectura física

DISEÑO BASE DE DATOS RELACIONAL POSTGRESQL PARA SISTEMA DE FIDELIZACIÓN

La base de datos relacional usó el gestor Postgresql en la versión 13. Se utilizó la imagen de Docker postgres-13-alpine. La configuración se limitó a la creación de un usuario y la asignación de una contraseña usando las variables de ambiente: POSTGRES_USER y POSTGRES_PASSWORD. Adicionalmente, se utilizó un volumen de Docker para persistir la información de la base en un directorio y las tablas serán creadas por el microservicio y definidas en los archivos de entidad dentro del código fuente. A continuación, se presenta la Figura 2.3 con el esquema con el diseño para las estructuras de datos.

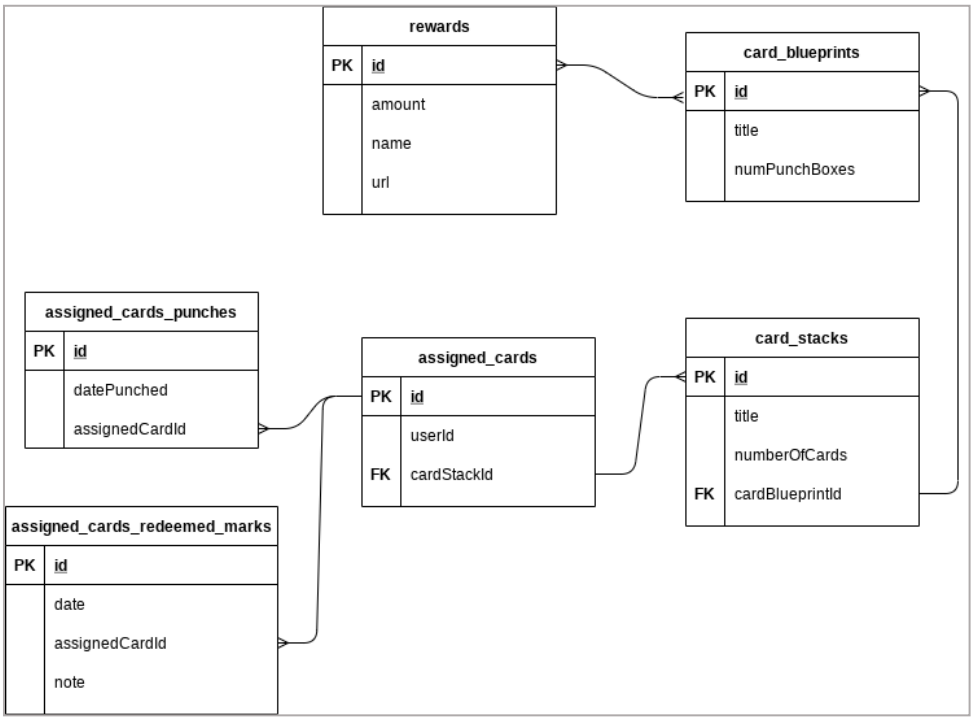


Figura 2.3 Esquema de diseño para las estructuras de datos

PK (Primary Key) indica que es una llave primaria, es decir un identificador único del registro. FK (Foreign Key) indica una referencia a otro registro. Para la relación de muchos a muchos se utilizó una tabla intermedia.

DISEÑO SERVIDOR WEB DE SISTEMA DE FIDELIZACIÓN

El servidor web de fidelización es un servicio web con un aplicativo REST. El rol de este microservicio es procesar la información de fidelización del usuario, esto es: administrar tarjetas de fidelización, esquemas de tarjetas, pilas de tarjetas, premios, perforaciones de tarjetas y marcas de premios reclamados, adicionalmente asociar premios a esquemas de tarjetas. Tiene un endpoint para identificar si el sistema está operando.

Para la persistencia se eligió una base de datos de Postgres. El lenguaje de programación para este microservicio elegido fue JavaScript con Typescript y el framework NestJS. Algunas de las librerías principales que se seleccionaron para este sistema y su función a continuación:

- Express: construir un servidor HTTP;
- TypeORM: conexión con la base de datos;
- @nestjs/config: permitir uso de variables de ambiente para configuración.

Adicional se definió el desarrollo de un script de Docker complementario. La configuración se definió sea por medio de variables de ambiente. Las variables de ambientes definidas para configuración son:

- TYPEORM_CONNECTION;
- TYPEORM_HOST;
- TYPEORM_USERNAME;
- TYPEORM_PASSWORD;
- TYPEORM_DATABASE;
- TYPEORM_SCHEMA;
- TYPEORM_PORT;
- TYPEORM_SYNCHRONIZE;
- TYPEORM_LOGGING;
- TYPEORM_ENTITIES.

Todas estas variables corresponded a la conexión a la base de datos. En anexo el diseño completo en formato Open API que incluye todos los endpoints con las rutas, métodos y ejemplos de respuestas se encuentra definido en Open Api. Las rutas de este servidor y su función definidas son:

- /: identificar el estado del servicio;
- /rewards: administrar premios de tarjetas;
- /card-blueprints: administrar los esquemas de tarjetas;
- /card-blueprint-to-rewards: asignar premios a esquemas de tarjetas;
- /card-stacks: administrar pilas de tarjetas;
- /assigned-cards: administrar tarjetas de fidelización;
- /assigned-cards/{assignedCardId}/punches: administrar perforaciones de tarjetas;
- /assigned-cards/{assignedCardId}/redeemed-marks: administrar marcas de premios reclamados.

Para los endpoints se definieron estructuras asociadas a algunos endoints, en este caso para todos excepto para el de estado del servicio. Cada estructura incluyó una lista de parámetros, con la definición de su tipo. Se usa integer para números enteros y de punto flotante y string para cadenas de caracteres. La Figura 2.4 ilustra, la estructura para tarjetas de fidelización.

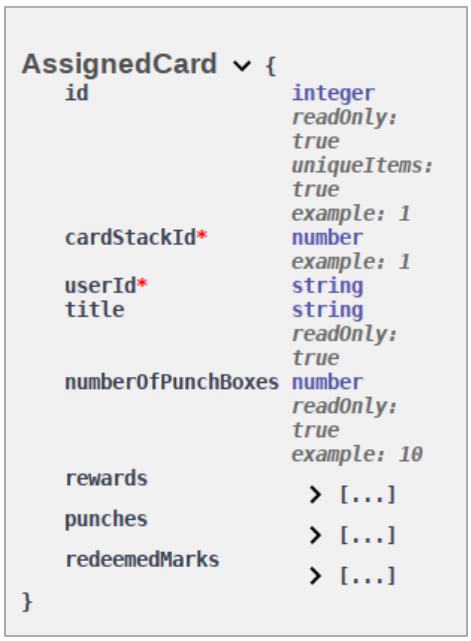


Figura 2.4 Ejemplo de estructura de tarjetas de fidelización

Esta estructura ilustrada en la Figura 2.4, es una representación visual de la definición en archivo yaml del diseño del API que se obtuvo utilizando un intérprete. En este caso se utilizó un plugin de VSCode. El asterisco sobre un campo indica que deberá tener asignado un valor obligatoriamente cuando se lo utiliza como parámetro de entrada. Los parámetros no definidos como string o number sino como un arreglo hacen referencia a otro esquema. En este caso rewards, punches y redeemedMarks están haciendo referencia a otros esquemas. Para mantener una buena estructura de código se usaron las librerías eslint y prettier. Estas librearías se configuran usando el archivo.eslintrc.js que será definido como lo muestra el Código 2.1.

```
module.exports = {
  parser: '@typescript-eslint/parser',
  parserOptions: {
    project: 'tsconfig.json',
    sourceType: 'module',
  },
  plugins: ['@typescript-eslint/eslint-plugin'],
  extends: [
    'plugin:@typescript-eslint/recommended',
    'prettier/@typescript-eslint',
    'plugin:prettier/recommended',
  ],
  root: true,
  env: {
    node: true,
    jest: true,
  },
  rules: {
    '@typescript-eslint/interface-name-prefix': 'off',
    '@typescript-eslint/explicit-function-return-type': 'off',
    '@typescript-eslint/explicit-module-boundary-types': 'off',
    '@typescript-eslint/no-explicit-any': 'off',
  },
};
```

Código 2.1 Archivo eslintrc.js

[redacted] DISEÑO APLICATIVO WEB CLIENTE Y ADMINISTRATIVO DEL SISTEMA DE FIDELIZACIÓN

El sistema de fidelización tuvo dos aplicativos web, uno para el uso de clientes y el otro con fines administrativos. El aplicativo web para clientes permitió a los clientes consultar las tarjetas de fidelización que han obtenidos y la información de las mismas: perforaciones, premios asociados, y si han sido reclamados los premios asociados. Por otra parte, el aplicativo web administrativo permitió a un operador administrar el sistema de fidelización.

Los aplicativos web fueron desarrollado en el lenguaje Javascript con Typescript, usando la librearía React y el framework Ionic. Algunas de las librerías importantes utilizadas fueron:

- redux: permite mantener un estado global para toda la aplicación e inyectarlo a componentes de React.
- redux-thunk: permite agregar un middleware en redux. En este proyecto se utiliza para imprimir información del estado global en consola.
- axios: permite realizar consultas HTTP.

Ambos aplicativos web estuvieron provisto de un script de Docker. Para la configuración se tuvieron, en ambos casos, a disposición las siguientes variables de ambiente:

- REACT_APP_LOYALTY_CARDS_API_HOST: define la URL del servidor de tarjetas de fidelización.
- ESLINT_NO_DEV_ERRORS: usada para definir si las recomendaciones de buenas prácticas de la librería deberían impedir la compilación del código.

Para el diseño de los aplicativo web se ha utilizado la herramienta Pencil y se han dibujado las pantallas. Diseño completo en anexo para web de clientes y administrativo. Las pantallas no definen el "look and feel" final, pero dan una guía de los botones disponibles y la organización de los componentes web. La Figura 2.5 muestra la pantalla del aplicativo web de clientes.

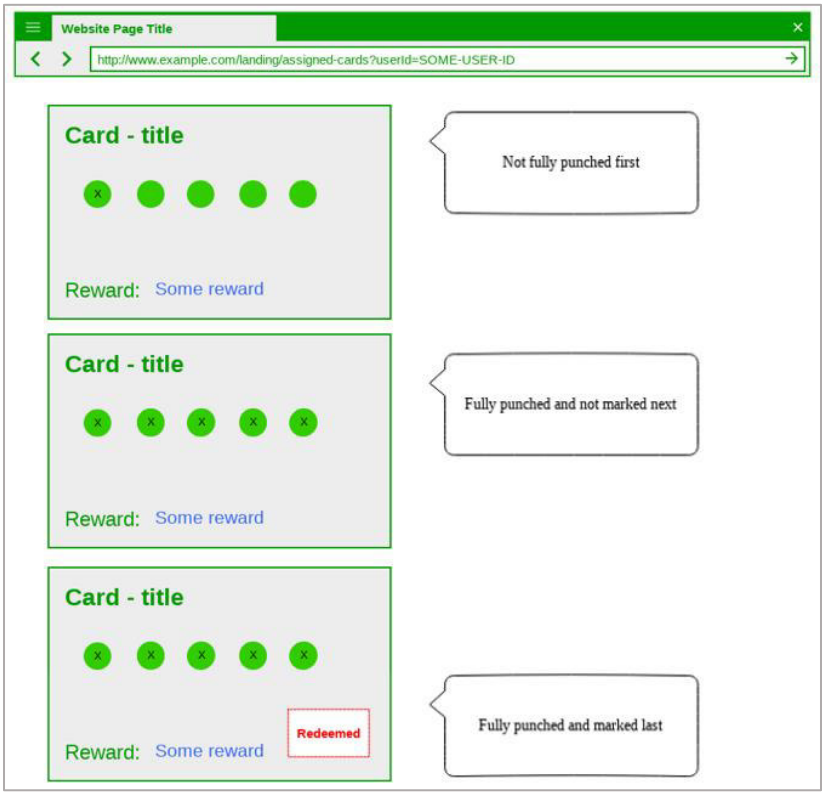


Figura 2.5 Pantalla del aplicativo web de clientes

La Figura 2.6 muestra un ejemplo del aplicativo web administrativo: la pantalla de premios para el aplicativo web administrativo

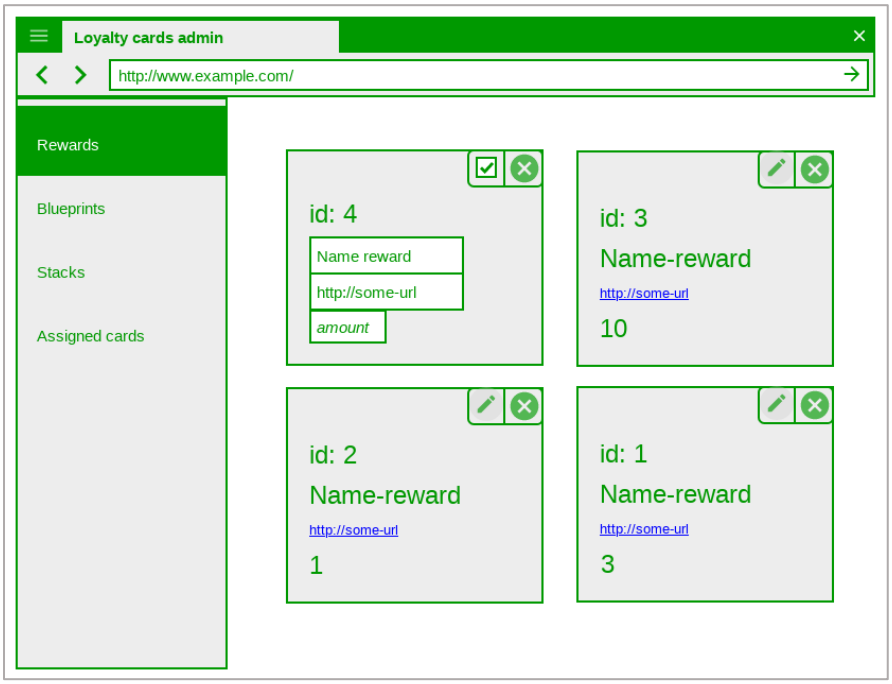


Figura 2.6 Ejemplo del aplicativo web administrativo

[redacted]DISEÑO MIDDLEWARE PARA SISTEMA DE FIDELIZACIÓN Y CEZERIN

Este microservicio provee un endpoint a usar por el e-commerce cuando nuevas órdenes se hayan realizado. Este microservicio escucha un evento de nueva orden y realiza la perforación de una tarjeta de fidelización del cliente que realizó la orden siempre que el cliente tenga una tarjeta asignada a la que aun falten perforaciones para completar. Se selecciono Javascript con Typescript y el framework NestJS para su desarrollo.

Paran la persistencia se definió el gestor de bases de datos Postgres. Se determinó utilizar la misma base de datos que el servidor web de fidelización. Se determinó el desarrollo de un script de Docker complementario. Las variables de ambiente disponibles para conexión con la base de datos son:

- TYPEORM_CONNECTION
- TYPEORM_HOST
- TYPEORM_USERNAME
- TYPEORM_PASSWORD

- TYPEORM_DATABASE
- TYPEORM_SCHEMA
- TYPEORM_PORT
- TYPEORM_SYNCHRONIZE
- TYPEORM_LOGGING
- TYPEORM_ENTITIES

Adicionalmente, se determinó la variable LOYALTY_CARDS_HOST para indicar la URL del servidor web de fidelización. En ANEXO A, se presenta el diseño completo que incluye todos los endpoints con las rutas, métodos y ejemplos de respuestas se encuentra definido en Open Api. El servidor tiene una sola ruta: "/punchers/purchase-punch-requests" utilizada por el e-commerce para solicitar que se perfore una tarjeta cuando una compra ha sido realizado. La estructura de una solicitud de perforación por compra (PurchasePunchRequest) se observa en la Figura 2.7:

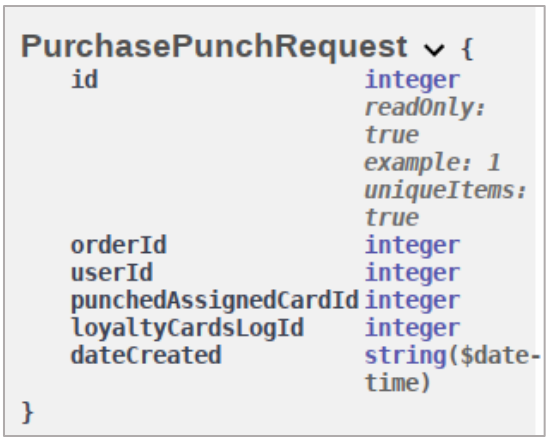


Figura 2.7 Estructura de una solicitud de perforación por compra

Para mantener una buena estructura de código se usarán las librerías eslint y prettier. Estas librearías se configuran usando el archivo .eslintrc.js que será definido según el Código 2.2:

```
module.exports = {
  parser: '@typescript-eslint/parser',
  parserOptions: {
    project: 'tsconfig.json',
    sourceType: 'module',
  },
  plugins: ['@typescript-eslint/eslint-plugin'],
  extends: [
    'plugin:@typescript-eslint/recommended',
    'prettier/@typescript-eslint',
    'plugin:prettier/recommended',
  ],
  root: true,
  env: {
    node: true,
    jest: true,
  },
  rules: {
    '@typescript-eslint/interface-name-prefix': 'off',
    '@typescript-eslint/explicit-function-return-type': 'off',
    '@typescript-eslint/explicit-module-boundary-types': 'off',
    '@typescript-eslint/no-explicit-any': 'off',
  },
};
```

Código 2.2 Archivo .eslintrc.js

[redacted]DISEÑO DE E-COMMERCE DE CÓDIGO ABIERTO

Se utilizó el software de código abierto Cezerin, así como también los tres componentes del sistema de forma separada [29]. Es decir, cada componente tendrá su propio repositorio y cada uno estará por sí sólo en un contenedor de Docker exclusivo. Cada uno, individualmente, tendrá su propio script de Docker y tendrá sus propias configuraciones. La Figura 2.8 muestra los tres componentes del e-commerce que ofrecen una interfaz a los usuarios desde el componente web y el componente web administrativo; muestra también la conexión que existe del e-commerce con un componente SMTP y una base de datos no relacional MongoDB, esto desde el componente store (servidor).

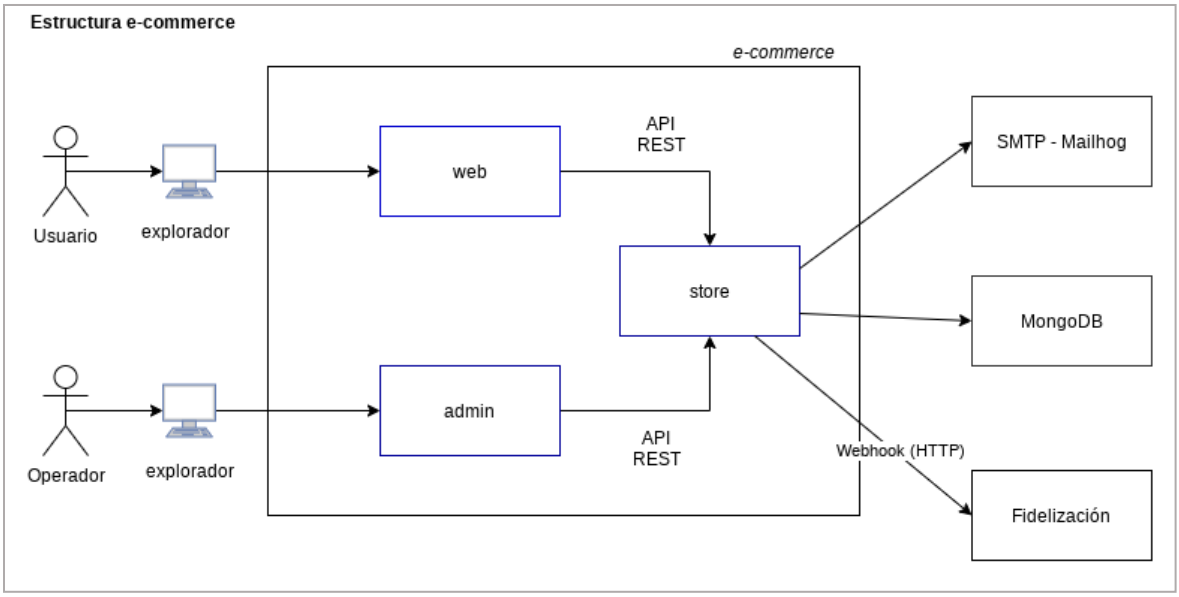


Figura 2.8 Componentes del e-commerce

Se utilizarán las funcionalidades de administrar productos, categorías de productos, crear usuarios, realizar órdenes, administrar estados de órdenes. Otras funcionalidades de Cezerin no se exploran en este prototipo. Adicionalmente se usará la funcionalidad de webhooks que permite notificar a otros microservicios de eventos, por ejemplo, la creación de un usuario, la realización de una nueva orden pagada. El microservicio corre un servidor web con endpoints REST y no tiene información ni funcionalidades de fidelización.

El servidor web expone un API REST, usa la versión de Node 10 o también conocida como dubnium. Las variables que utiliza para la configuración son:

- SMTP_HOST: host del servidor SMTP;
- SMTP_PORT: puerto del servidor SMTP para comunicación de mensajes;
- SMTP_SECURE: bandera que indica si se requiere autenticación con el servidor SMTP;
- DB_HOST: host de la base de datos Mongo;
- DB_PORT: puerto de la base de datos Mongo;
- DB_NAME: nombre del esquema dentro de la base de datos Mongo;
- DB_USER: usuario para la base de datos Mongo para autenticación;
- DB_PASS: contraseña de la base de datos Mongo para autenticación;
- FRONT_URL: URL del aplicativo web para control CORS.

Por otra parte, para el aplicativo web se utiliza la versión de Node 12, también conocida como erbium. Se utiliza el tema predefinido y las variables de configuración que utiliza son:

- REACT_APP_SERVER_URL: URL para conexión con el servidor backend desde el servidor del aplicativo web;
- REACT_APP_CLIENT_URL: URL para conexión con el servidor backend desde el cliente en el explorador web.

A pesar de que estas dos variables pueden tener el mismo valor, la diferencia es importante cuando se desea crear una ruta adicional desde el servidor web por razones de seguridad de red. Por otra parte, para el aplicativo de administración, "admin", se utiliza una instancia virtualizada aparte del backend y el aplicativo web. Se utiliza Node versión 12, erbium y para su configuración se utilizan dos variables:

- REACT_APP_CLIENT_URL: URL para conexión con el servidor backend desde el cliente del sistema administrativo en el explorador web;
- REACT_APP_WEB_SOCKET: URL para el websocket entre el admin y el servidor.

Para el cliente administrativo es necesario realizar la configuración de dominio para el correcto despliegue de las imágenes.

DISEÑO BASE DE DATOS NO RELACIONAL MONGODB

La base de datos no relacional para el e-commerce usará el gestor MongoDB en la versión 4. Se utilizará la imagen de Docker oficial de tag mongo 4.4 publicada en el repositorio público https://hub.docker.com/_/mongo. Se usará la funcionalidad de volúmenes de Docker para persistencia y la configuración se limita a crear un usuario y una base dentro del gestor. Las colecciones serán creadas por la instancia del e-commerce backend con el comando provisto por los desarrolladores de Cezerin: "npm run setup".

Las colecciones que se crearon de esta manera son: customers, emailTemplates, orders, pages, paymentMethods, productCategories, products, settings y shippingMethods. Algunas de estas colecciones ya tendrán información, por ejemplo, la colección pages contendrá 9 registros correspondientes a las páginas disponibles en el aplicativo web: home, checkout, checkout-success, about, login, register, customer-account, forgot-password y reset-password.

Como principio de diseño para esta base se evitará realizar cambios en la estructura de datos, de la misma manera no se modificarán índices. Se agregarán nuevos registros, se cambiarán valores, se removerán registros desde un aplicativo web del sistema y no directamente.

PLAN DE PRUEBAS

De manera general, el plan de pruebas es una guía que contiene las actividades de prueba con el fin garantizar el funcionamiento óptimo del sistema. Es así que para la presente investigación se utilizó el estándar IEEE 829 [19] y la plantilla de la Universidad de California Northdrige [20] mismas que fueron aplicadas al sistema completo, es decir al e-commerce con el sistema de fidelización de tarjetas perforadas. El plan completo en anexo. Algunos puntos para destacar son:

- Funcionalidades para probar: todas las listadas en las historias de usuario del proyecto;
- Funcionalidades que no ser probarán:
    - No se probarán funcionalidades no utilizadas en ese sistema integrado del sistema de e-commerce ya que su desarrollo no fue parte de este proyecto;
    - No se probará la capacidad de soportar grandes cantidades de consultas (conocidas como pruebas de carga);
    - No se probarán sofisticadas funcionalidades de seguridad como control de phishing, hacking, deny of service, etc.
- Requerimientos de ambientes: el ambiente está constituido por una instancia EC2 de Amazon que tiene instalado Docker y Docker Compose, y se utilizará un ambiente destinado a pruebas o el ambiente de producción sin preferencia de uno sobre otro.

DESARROLLO EN LOCAL

LEVANTAMIENTO Y CONFIGURACIÓN DE SERVIDOR SMTP EN LOCAL UTILIZANDO IMAGEN DE DOCKER DE MAILHOG.

Para levantar el servidor de correos de Mailhog se puede utilizar el script de Docker que se encuentra en el repositorio público publicado en GitHub [30]. Para esto, primeramente, se clona el proyecto utilizando el comando de "git clone". El primer argumento de la función clone es el repositorio de Mailhog en GitHub y el segundo indica copiarlo en el directorio actual. Luego, se construyó una imagen de Docker usando el comando “docker build -t mailhog”. El argumento de la instrucción build es "." que indica crear la imagen utilizando el script presente en el directorio actual. Se utiliza la opción "-t" para colocar la etiqueta "mailhog".

Una vez creada la imagen se procede a correr un contenedor a partir de esta imagen. El Código 2.3 muestra la instrucción para correr el contenedor de mailhog. Esta instrucción tiene como argumento "mailhog:latest" que es la etiqueta de la imagen construida previamente. Adicionalmente se utiliza la opción "-d" para liberar la consola, la opción "--name" para denominar al contenedor: "smtp". La opción "-p" se usa dos veces para mapear los puertos 1025 y 8025 del contenedor a los puertos del mismo número en el ambiente local.

```
ricardo@riunturi:~/riu-fs-2019/home-tori$ docker run -
p 8025:8025 -p 1025:1025 -d --name=smtp mailhog:latest
```

Código 2.3 Comando del contenedor de mailhog

Mailhog provee un dashboard para revisar los correos que han pasado por el sistema. El dashboard es un aplicativo web al que se puede acceder utilizando un explorador web. La Figura 2.9 muestra una captura de pantalla con el dahsboard es su página inicial.

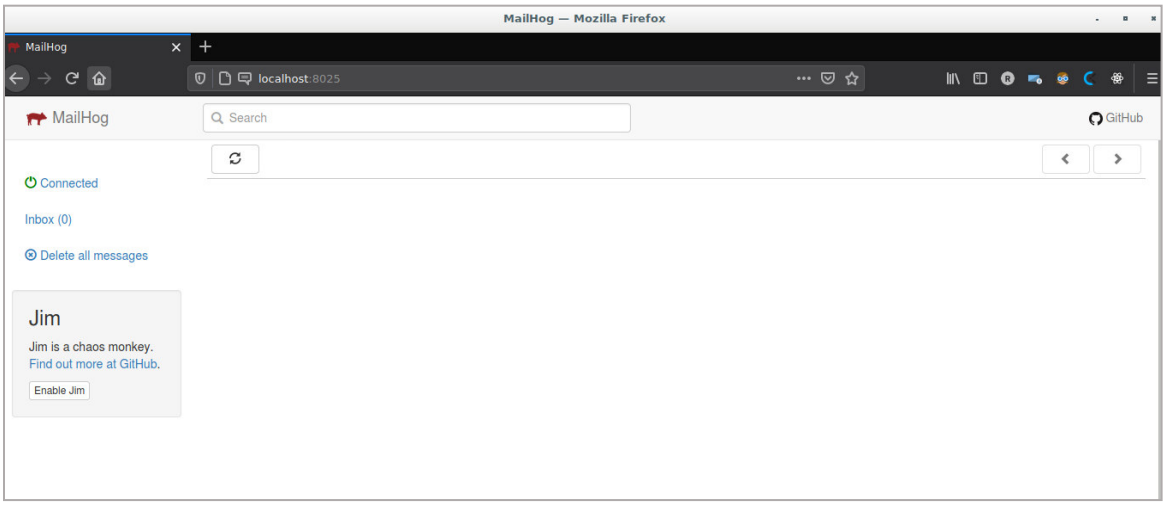


Figura 2.9 Dashboard de Mailhog

[redacted]LEVANTAMIENTO Y CONFIGURACIÓN DE LA BASE DE DATOS DOCUMENTAL MONGODB EN AMBIENTE LOCAL.

Para el levantamiento de la base de datos de MongoDB en local se puede utilizar la imagen de Docker: "mongo:4.4". Para esto se ejecuta el comando en el Código 2.4:

```
docker run --name mongo -p 27017:27017 -d -v /home/ricardo/docker/volumes/mongo -e
MONGO_INITDB_ROOT_USERNAME=root -e MONGO_INITDB_ROOT_PASSWORD=some-
password mongo:4.4
```

Código 2.4 Comando para instalar la imagen de Docker

Esta instrucción de Docker levanta una instancia de MongoDB en la versión 4.4, le etiqueta con el nombre "mongo", y le asocia al puerto 27017 del host. Para la persistencia utiliza un volumen asociado a un directorio del mismo nombre. Se utilizan dos variables para la configuración:

- MONGO_INITDB_ROOT_USERNAME: identificador del usuario;
- MONGO_INITDB_ROOT_PASSWORD: contraseña.

La base de datos no contendrá información hasta ejecutar el comando "npm run setup" de Cezerin. Este comando crea las colecciones y carga información de ejemplo en la base de datos. La Figura 2.10 muestra el contenido de la colección "shop" obtenidos con la herramienta MongoDBCompass. En la Figura 2.10 se observan siete colecciones, el tamaño de cada colección, número de documentos, número de índices y tamaño promedio de cada documento.

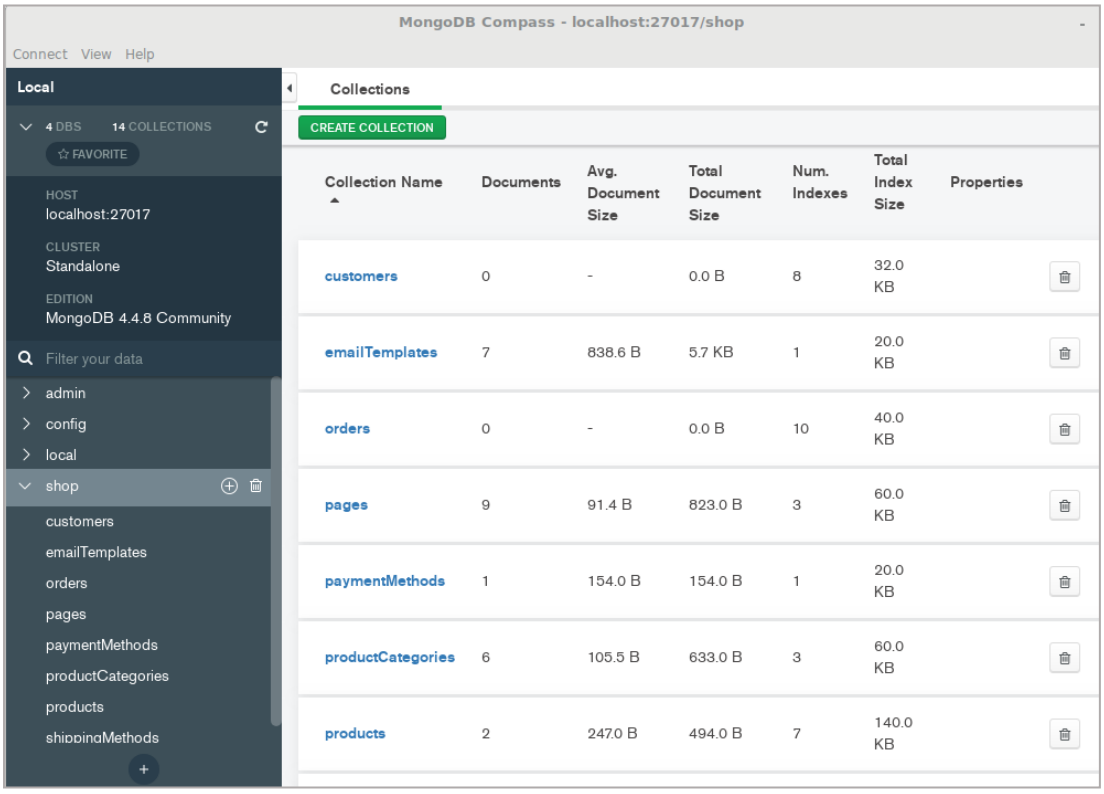


Figura 2.10 Contenidos de la colección "shop"

[redacted]LEVANTAMIENTO Y CONFIGURACIÓN DE LA BASE DE DATOS RELACIONAL POSTGRESQL EN AMBIENTE LOCAL.

Para el levantamiento en ambiente local de la base de datos PostgreSQL se puede utilizar una imagen Docker [28]. Para esto, se corre el comando del Código 2.5. Se utilizó un contenedor de la imagen "postgres:12-alpine". Más adelante se utilizó la opción "-e" dos veces para configurar las variables POSTGRES_USER y POSTGRES_PASSWORD para definir el usuario principal y su contraseña respectivamente. La opción "--name" se usa para asignar el nombre "db-pg" a la instancia virtual creada. La opción "-p" asocia el puerto 5432 de la instancia virtual al puerto del mismo número en el ambiente local. La opción "-d" permite liberar la consola, y finalmente, para la persistencia se utiliza un volumen en un directorio de nombre "postgres" en el ambiente local.

```
docker run --name db-pg -e POSTGRES_USER=root -e POSTGRES_PASSWORD=thesis1234 -d -p
5432:5432 -v /home/ricardo/docker/volumes/postgres:/var/lib/postgresql/data postgre
s:12-alpine
```

Código 2.5 Comando para correr un contenedor de bases de datos

Usando TypeORM se crearon las tablas para la administración de datos. La Figura 2.11 muestra la lista de tablas en la base de datos luego de ser creadas por los servidores web. Esta Figura fue obtenida con la herramienta DBeaver que provee una interfaz visual para inspeccionar y administrar bases de datos.

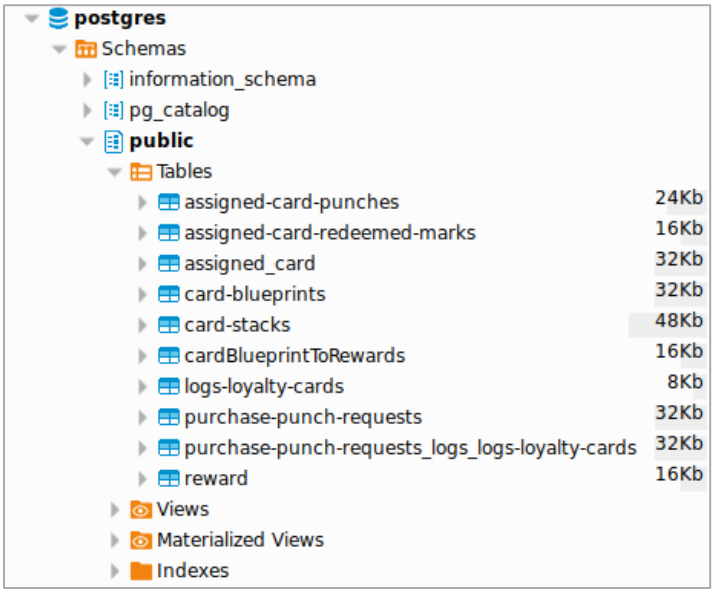


Figura 2.11 Lista de tablas en la base de datos

DESARROLLO SERVIDOR WEB PARA SISTEMA DE FIDELIZACIÓN

Se creó un nuevo proyecto indicando el nombre "loyalty-cards". Esto crea un nuevo directorio que incluye el código base de un proyecto de NestJS. Luego, se instalan las librerías a utilizar como pg, typeorm, eslint-config-prettier y eslint-plugin-prettier. La librería pg se usa para la integración con bases de datos Postgres.

Continuando con el diseño, se crean los módulos: rewards, card-blueprints, card-stacks, assigned-cards, assigned-card-punches y assigned-card-redeemed-marks. Algunos esquemas utilizados en este proyecto son: resource, controller, interface, module y service. El Código 2.6 muestra un método del controlador para los endpoints del esquema assigned-cards. En este método se utiliza la anotación @Get que corresponde al método GET de HTTP que construye este método. La anotación @Query se utiliza para asociar los parámetros de la cadena de consulta de una URL a variables. Esté método usa el método findAll de servicio assignedCards y devuelve su respuesta.

```
@Get()
async findAll(
@Query('onlyFullyPunched') onlyFullyPunched = false,
@Query('onlyNotFullyPunched') onlyNotFullyPunched = false,
@Query('userId') userId: string,
) {
return await this.assignedCardsService.findAll(
  onlyFullyPunched,
  onlyNotFullyPunched,
  userId,
);
}
```

Código 2.6 Método del controlador para los endpoints

Para la configuración del microservicio se utilizan variables de ambiente y el módulo @nestjs/config. Se desarrolló un script de Docker con la imagen "node:erbium-alpine". Esta imagen corresponde a la versión 12 de Node, también llamada erbium y el sistema operativo Linux Alpine. El Código 2.15 muestra el script de Docker desarrollado y que se encuentra en la raíz del proyecto.

```
FROM node:erbium-alpine
LABEL maintener='Ricardo David Ortiz'
WORKDIR /loyalty-cards
COPY package*.json ./
RUN npm install
COPY ./ ./
EXPOSE 8000
CMD npm run start
```

Código 2.7 Script de Docker con la imagen "node:erbium-alpine"

DESARROLLO APLICATIVO WEB CLIENTE PARA SISTEMA DE FIDELIZACIÓN.

Se crea un nuevo proyecto utilizando el comando "ionic start". Ionic consultará: el framework, el nombre del proyecto y la plantilla. Se elige React, "loyalty-cards-web" y "blank", que corresponde a la plantilla en blanco respectivamente, lo que crea un directorio y código base. Para instalar las librerías se utilizó el comando "npm install", y a la vez se instalaron las librerías: axios, ionicons, lodash, redux, redux-logger y redux-thunk.

Para el aplicativo web de clientes, siguiendo el diseño, se construye una página para las tarjetas de fidelización. El Código 2.8 muestra el componente de la página para tarjetas. Este código muestra el uso de la función "map" de JavaScript para asociar un componente a cada tarjeta de fidelización obtenida del servidor web con una consulta HTTP.

```
export const LandingAssignedCards:
React.FC<LandingAssignedCardsProps> = ({cards}) => {
  return (
  <IonPage>
  <IonContent fullscreen>
    {cards.map((card) => (
      <AssignedCard assignedCard={card}></AssignedCard>
    ))}
  </IonContent>
  </IonPage>
)};
```

Código 2.8 Componente de la página para tarjetas

Esto permite mostrar las tarjetas de fidelización correspondientes a un usuario identificado por su id único. Para esto se usó el módulo de node queryString y el objeto location de React. Se usó el React Hook useEffect que permite el uso del ciclo de vida de React para ejecutar métodos antes de la carga de un componente.

La configuración de este aplicativo se la realizó con un archivo de variables de ambiente en la raíz del proyecto con el nombre ".env". En este archivo se define la variable REACT_APP_LOYALTY_CARDS_API con el valor "http://localhost:8000". Se desarrolló el script de Docker del aplicativo utilizando la imagen "node:erbium-alpine". En el script se documenta el puerto 3000 que utiliza el aplicativo por defecto. En el script se utiliza el comando "npm run start" que crea un directorio optimizado y lo corre. El Código 2.9 muestra el script de Docker.

```
FROM node:erbium-alpine
LABEL maintener='Ricardo David Ortiz'
WORKDIR /loyalty-cards
COPY package*.json ./
RUN npm install
COPY ./ ./
EXPOSE 3000
CMD npm run start
```

Código 2.9 Ejecución del comando "npm run start" de Docker

Para correr en local se utilizó el comando "ionic serve". Una vez desarrollado el proyecto y al levantarlo se pude revisar las páginas del aplicativo utilizando un explorador web. La Figura 2.12 muestra la página de tarjetas de fidelización del aplicativo vista desde el explorador web Firefox para la URL http://localhost:8100/landing/assigned-cards?userId=6104533c490bc900adf7b9eb. Esto luego que al usuario con id "6104533c490bc900adf7b9eb" se le ha asignado una tarjeta.

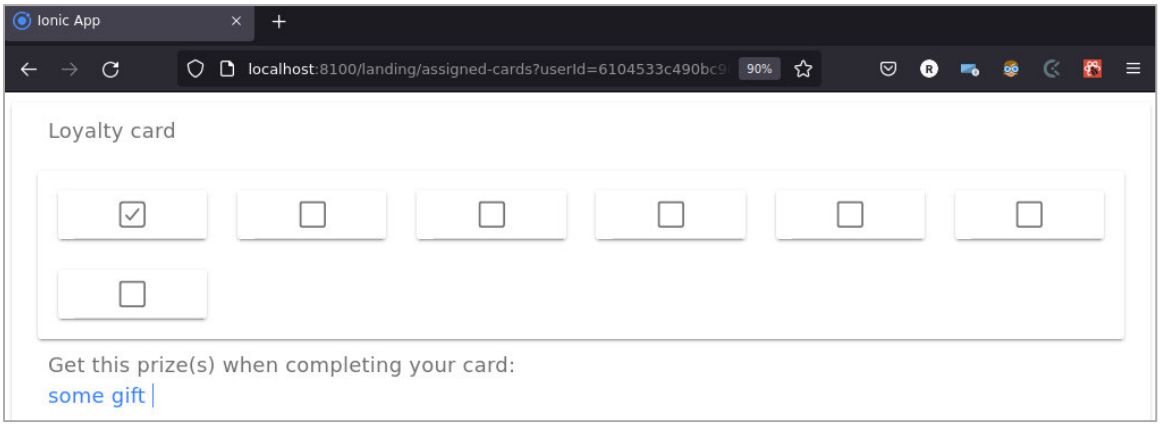


Figura 2.12 Página de tarjetas de fidelización en explorador web

DESARROLLO APLICATIVO WEB ADMINISTRATIVO PARA SISTEMA DE FIDELIZACIÓN

El desarrollo fue similar al del aplicativo de clientes, se utilizó Ionic como base, las mismas librerías y Redux para mantener un estado global. Sin embargo, este aplicativo es más complejo:

incluye más páginas y acepta ingreso de datos del usuario para administrar el sistema de tarjetas. El sistema para clientes es una herramienta visual que no permite ingreso de datos ni operaciones de administración.

Se construyeron las siguientes páginas: assigned-cards, blueprints, rewards y stacks. Cabe recalcar que la página de tarjetas de este aplicativo es distinta al del aplicativo de clientes. En este caso permite ver todas las páginas del sistema, eliminarlas y editar las marcas de premios reclamados, en contraste con la página del aplicativo de clientes, pues solo permite ver las tarjetas y usar filtros. Todas las páginas de este aplicativo utilizan los componentes IonPage y IonContent como base y el componente AppToolBar. Recurrentemente se usa el componente IonGrid para construir una tabla con los elementos correspondientes a la página: premios, esquemas de tarjetas, pilas de tarjetas o tarjetas asignadas a usuario. El Código 2.10 muestra el componente de la página de tarjetas.

```
export const AssignedCards: React.FC<AssignedCardsProps> = ({
  pageTitle,
  assignedCards = [],
  isLoading = false,
  removeAssignedCardAndReloadRequested,
  addRedeemedMarkToAssignedCardAndReloadRequested: addRedeemedMarkToAssignedCard,
  removeRedeemedMarkFromAssignedCardAndReloadRequested: removeRedeemedMarkFromAssignedCard,
}) => (
  <IonPage>
    <IonHeader>
      <AppToolbar pageTitle={pageTitle} />
    </IonHeader>
    <IonContent>
      <IonLoading isOpen={isLoading} />
      <IonGrid>
        <IonRow>
          {assignedCards.map((assignedCard) => (
            <IonCol size="6" key={`assigned-card-${assignedCard.id}`}>
              <AssignedCard
                removeAssignedCard={removeAssignedCardAndReloadRequested}
                addRedeemedMarkToAssignedCard={addRedeemedMarkToAssignedCard}
                removeRedeemedMarkFromAssignedCard={removeRedeemedMarkFromAssignedCard}
                {...assignedCard}
              />
            </IonCol>
          ))}
        </IonRow>
      </IonGrid>
    </IonContent>
  </IonPage>
);
```

Código 2.10 Componente de la página de tarjetas

El aplicativo contiene un script de Docker que utilizó la imagen "node: erbium-alpine". Se documenta el puerto 3000 que utiliza el aplicativo por defecto. En el script se utilizó el comando "npm run start" que crea un directorio optimizado y lo corre. El Código 2.11 muestra el script de Docker.

```
FROM node:erbium-alpine
LABEL maintener='Ricardo David Ortiz'
WORKDIR /loyalty-cards
COPY package*.json ./
RUN npm install
COPY ./ ./
EXPOSE 3000
CMD npm run start
```

Código 2.11 Comando "npm run start" de Docker

Se levantó el proyecto usando el comando ionic serve. La Figura 2.13 muestra la página de tarjetas de fidelización utilizando el explorador Firefox y utilizando la URL "http://localhost:8101/assigned-cards". En esta página se observa el menú a la izquierda, las tarjetas a la derecha con un botón para eliminarlas en la esquina superior derecha, una sección para marcas de premios reclamados "Redeemed Marks", y las marcas indicando la hora que se agregaron y provistas de un botón para eliminarlas. Además, en esta sección hay un botón en el centro de la sección de premios reclamados que sirve para agregar una nueva marca con la hora y fecha actual.

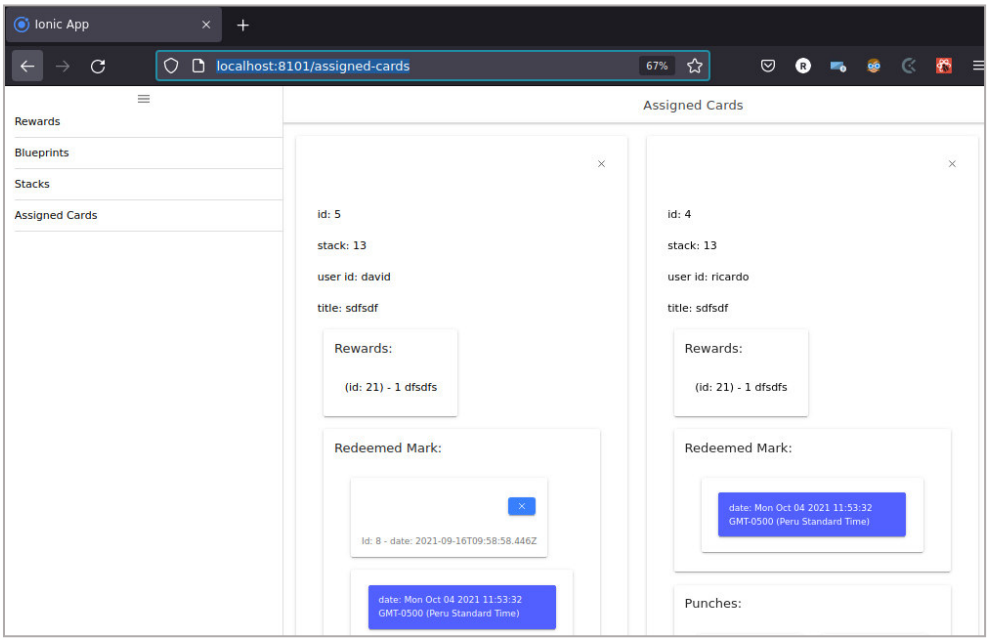


Figura 2.13 Página de tarjetas de fidelización utilizando un explorador web

DESARROLLO SERVIDOR WEB PARA E-COMMERCE BACKEND USANDO CEZERIN2.

Para el desarrollo del servidor web de e-commerce se utilizó Cezerin. Para esto se clonó el proyecto de la URL https://github.com/Cezerin2/cezerin2 utilizando el comando "git clone" y se creó un nuevo directorio y proyecto de GitHub exclusivo este subsistema. Este proyecto se le asignó la URL: https://github.com/riuGlobal/cezerin2.git.

La configuración de Cezerin se la puede hacer con cambios en el código, pero esto no es conveniente para el despliegue con Docker que se pretende. Para lograr el despliegue con Docker se realizaron cambios en el archivo config/server.js, el archivo destinado a la configuración, para permitir el uso de variables de ambiente. Se introducen las variables de ambiente utilizando la variable global de Node: "process.env". El Código 2.12 muestra la asignación de variables de ambiente a las variables de configuración de Cezerin.

```
const dbHost = process.env.DB_HOST || '127.0.0.1';
const dbPort = process.env.DB_PORT || 27017;
const dbName = process.env.DB_NAME || 'shop';
const dbUser = process.env.DB_USER || 'root';
const dbPass = process.env.DB_PASS || 'pass';
const dbCred =
        dbUser.length > 0 || dbPass.length > 0 ? `${dbUser}:${dbPass}@` : '';

const dbUrl =
        process.env.DB_URL ||
        `mongodb://${dbCred}${dbHost}:${dbPort}/${dbName}?authSource=admin&readPreference=primary`;
```

Código 2.12 Variables de ambiente a las variables de configuración de Cezerin

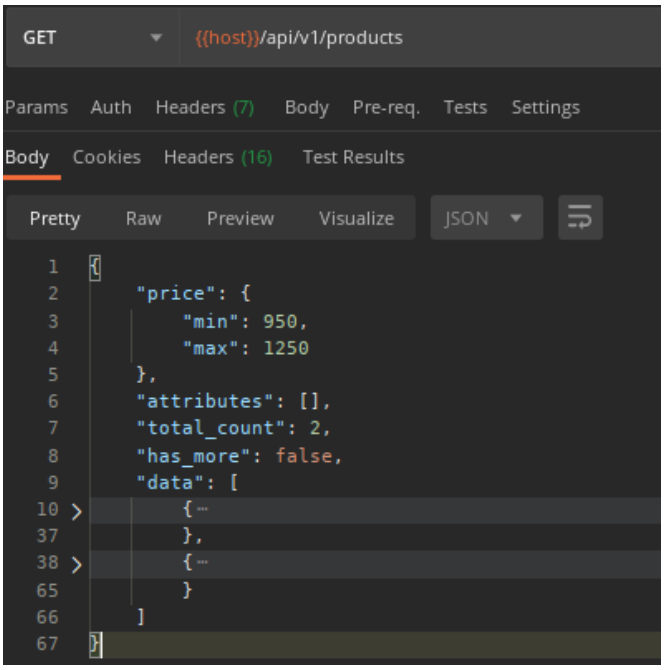
El servidor web también incluye un servidor de contenido estático para imágenes que operará en el puerto 3000 en lugar del puerto 3001 que se usa para el API REST. El código original de Cezerin no incluye un script de Docker. Se utilizó la imagen oficial de Node: "node:dubnium-buster", misma que corresponde a la versión 10 de Node utilizando la base del sistema operativo Debian versión 10 conocida como Debian Buster. El servidor de Cezerin no soporta versiones más actuales de Node y el uso de Linux Alpine presenta problemas de librerías. Para levantar este subsistema se utilizaron los comandos: "npm run setup", "npm run build" y "npm run start". El comando "npm run setup" configura la base de datos, sino ha sido configurada, que crea las colecciones e índices necesarios. También inyecta en la base de datos información de prueba. El comando "npm run build" crea un directorio optimizado para producción. Finalmente, el comando "npm run start" corre el subsistema. En el script se documentaron los puertos: 3001

para el servidor web y 3000 para el servidor de imágenes estático. El Código 2.13 muestra el script de Docker para este subsistema:

```
FROM node:dubnium-buster
LABEL mainteiner='Ricardo David Ortiz'
WORKDIR /var/www/html/cezerin2
COPY package*.json ./
RUN npm install
COPY ./ ./
EXPOSE 3001 3000
CMD npm run setup \
&& npm run build \
&& npm run start
```

Código 2.13 Script para levantar el subsistema

Para levantar en local se utilizaron los mismos comandos definidos en el script de Docker. Una vez levantado el sistema los endpoints estarán listos para consultar desde un aplicativo web o alguna herramienta como Postman. El Código 2.14 muestra la consulta al endpoint "api/v1/products" utilizando Postman. Dentro de la respuesta existe un objeto "prices" que contiene los rangos de precios para esta consulta. Se puede observar que existen dos objetos los cuales corresponden cada uno a un producto.



Código 2.14 Consulta al endpoint "api/v1/products"

DESARROLLO APLICATIVO WEB PARA E-COMMERCE USANDO CEZERIN2

Para el aplicativo web del e-commerce se utilizó el aplicativo web de Cezerin. Para esto se clonó el proyecto de la URL https://github.com/Cezerin2/cezerin2 utilizando el comando "git clone", y se creó un nuevo directorio y proyecto de GitHub exclusivo para este subsistema. Este proyecto se le asigna la URL: https://github.com/riuGlobal/cezerin-web-example.

Al igual que en el caso del servidor web, Cezerin provee una configuración con variables de un archivo para el aplicativo web. Esto no es conveniente para el despliegue con Docker. Se realizaron cambios en el archivo config/server.js para el uso de variables de ambiente. El Código 2.15 muestra el uso del objeto global "process.env" para asignar variables de ambiente del archivo .env en la raíz del proyecto a las variables clientUrl y serverUrl del archivo de configuración.

```
const clientUrl = process.env.REACT_APP_CLIENT_URL || 'http://localhost:3001';
const serverUrl = process.env.REACT_APP_SERVER_URL || 'http://localhost:3001';
```

Código 2.15 Uso del objeto global "process.env"

Para el uso de variables de ambiente en el aplicativo web fue necesario utilizar adicionalmente la herramienta Webpack, por lo que se hicieron cambios en el archivo para la configuración del webpack "webpack.config.store.js" ubicado en la raíz del proyecto. Se usó el plugin DefinePlugin.

El aplicativo web de Cezerin incluye una página para la información de un usuario. Cuando un usuario registrado ingresa al aplicativo con sus credenciales, esta página está disponible en "/customer-account". Esta página tiene tres secciones: Profile, Orders y Logout. La Figura 2.14 muestra esta página de Cezerin en ambiente local.

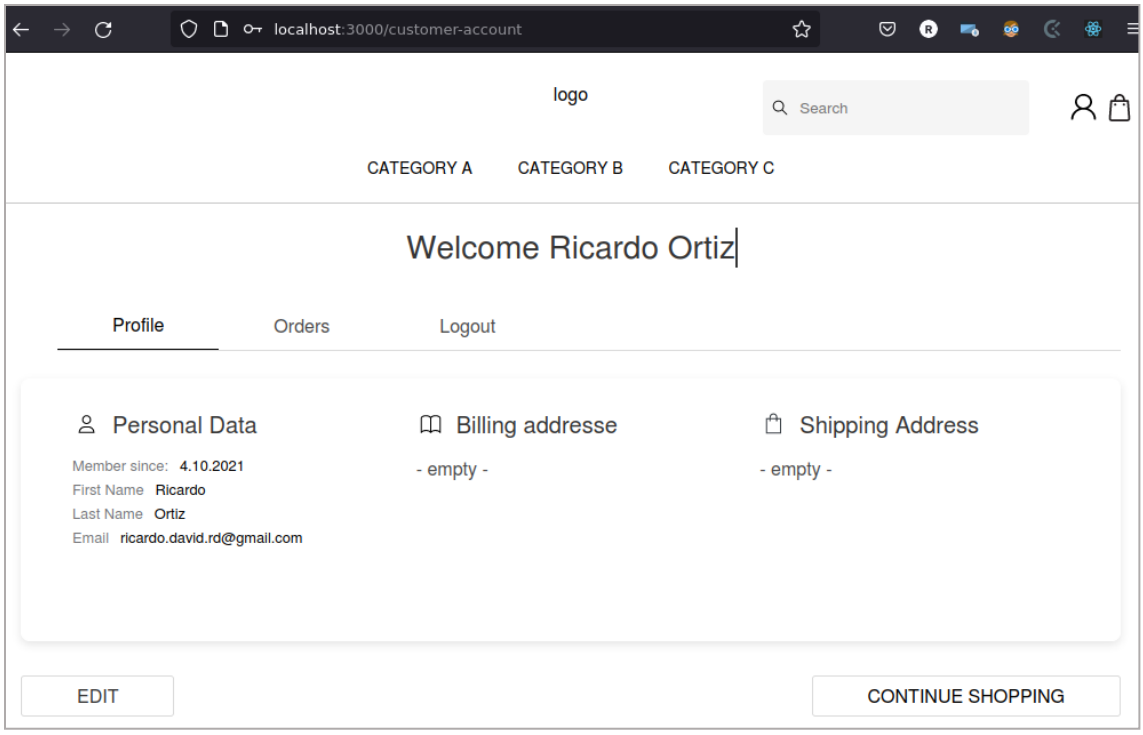


Figura 2.14 Página de Cezerin en ambiente local

Es en esta página se construyó una sección para la información de fidelización. Para esto se utilizó el componente "iframe" de html que permite crear un espacio dentro de una página con contenido de otra. Para este caso se utilizó el contenido del aplicativo web de clientes de fidelización, específicamente en la página de tarjetas de fidelización.

Un cliente podrá observar las tarjetas de fidelización que le han sido asignadas, sus premios asociados, sus perforaciones y si los premios han sido reclamados o no. La Figura 2.15 muestra la página con esta nueva sección para un cliente que tiene una tarjeta de fidelización asignada.

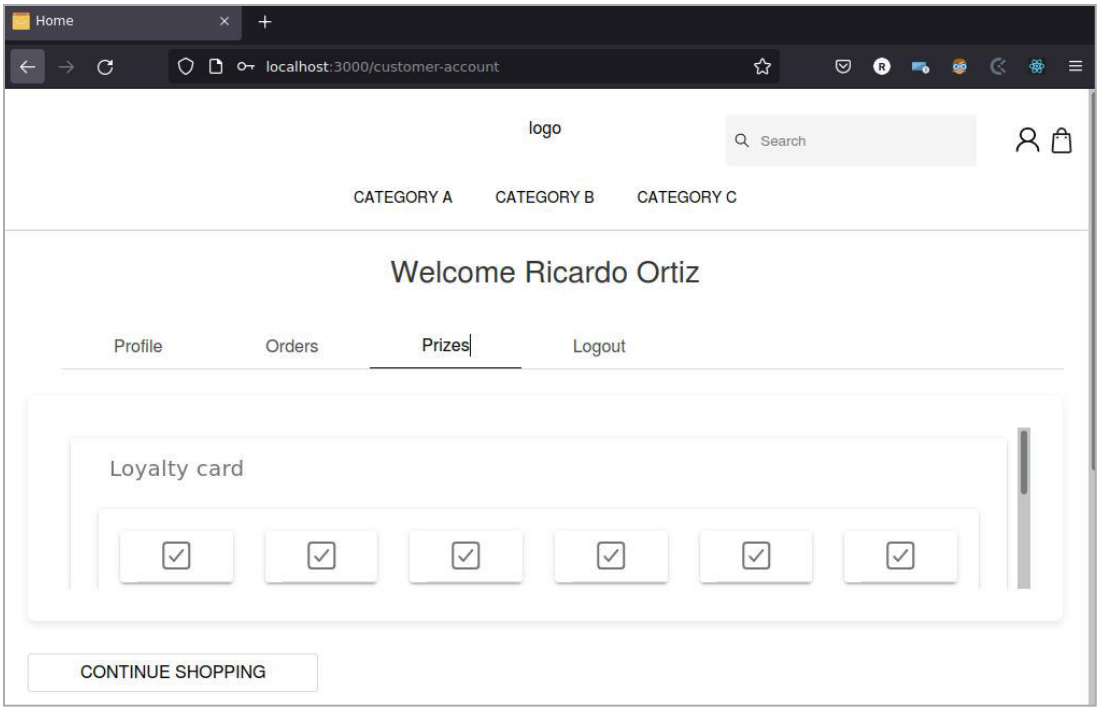


Figura 2.15 Página con información de tarjeta de fidelización asignada

Para el desarrollo del script de Docker, que no está presente en el componente Cezerin original, se utilizó la imagen "node:erbium-stretch-slim". Para el aplicativo web se puede utilizar una versión más actual que para el servidor, en este caso la versión 12. Los problemas con las librearía, sin embargo, persisten por lo que no se utiliza Linux Alpine y se prefiere Debian en su versión 9 conocida también como Debian Stretch. En el script se documentó el puerto 3000 que utiliza el aplicativo por defecto. Para levantar el sistema en el script se utilizó los comandos: "npm run build" para crear un directorio optimizado para producción y "npm run start-store" para iniciar el aplicativo web de clientes. El Código 2.16 muestra el script de Docker.

```
FROM node:erbium-stretch-slim
LABEL mainteiner='Ricardo David Ortiz'
WORKDIR /var/www/html/cezerin-web-example
COPY package*.json ./
RUN npm install
COPY ./ ./
EXPOSE 3000
CMD npm run build && npm run start-store
```

Código 2.16 Script de Docker

Para levantar en local se utilizaron los comandos del script de Docker. Una vez hecho esto se puede navegar por el e-commerce utilizando un explorador web. La Figura 2.16 muestra la página de inicio del e-commerce para clientes.

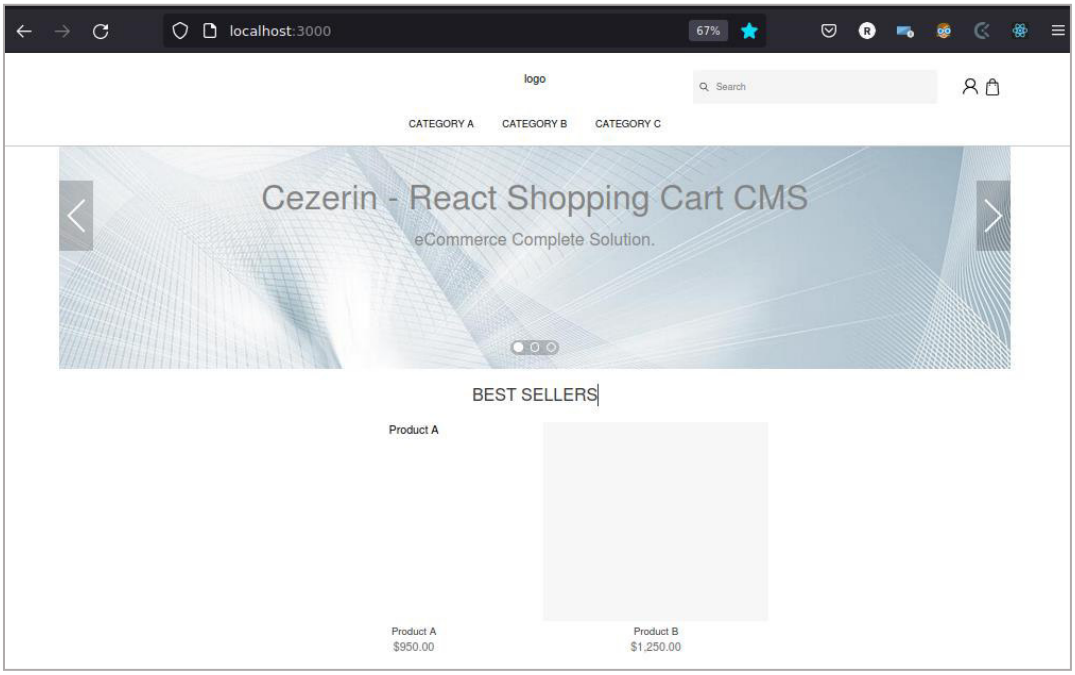


Figura 2.16 Página de inicio del e-commerce para clientes en navegador web

DESARROLLO APLICATIVO WEB PARA DASHBOARD ADMINISTRATIVO USANDO CEZERIN2.

Para el desarrollo del aplicativo web administrativo del e-commerce se utilizó el aplicativo web administrativo de Cezerin. Para esto se clonó el proyecto de la URL https://github.com/Cezerin2/cezerin2 utilizando el comando "git clone", y se creó un nuevo directorio y proyecto de Github exclusivo para este subsistema. A este nuevo proyecto se le asignó la URL: https://github.com/riuGlobal/cezerin-admin.git.

Primero se realizaron los cambios en el archivo config/server.js para introducir configuración por variables de ambiente de manera similar al aplicativo para clientes. De la misma manera que en el aplicativo de clientes se realizaron los cambios en el archivo de configuración de Webpack que en este caso es "webpack.config.admin.js".

Para el despliegue con Docker se utilizó el mismo script que para el aplicativo web. Esto ya que los requisitos son iguales y porque Cezerin utiliza los mismos comandos para levantar el

aplicativo. Se utiliza la imagen "node:erbium-stretch-slim". El Código 2.17 muestra el script de Docker para este aplicativo.

```
FROM node:erbium-stretch-slim
LABEL mainteiner='Ricardo David Ortiz'
WORKDIR /var/www/html/cezerin-web-example
COPY package*.json ./
RUN npm install
COPY ./ ./
EXPOSE 3000
CMD npm run build && npm run start-store
```

Código 2.17 Script de Docker

Una vez levantado en local se puede navegar por las páginas utilizando un explorador web. La Figura 2.17 muestra la página de inicio del aplicativo web administrativo levantado en local. En esta página se observan las ordenes realizadas en el último mes.

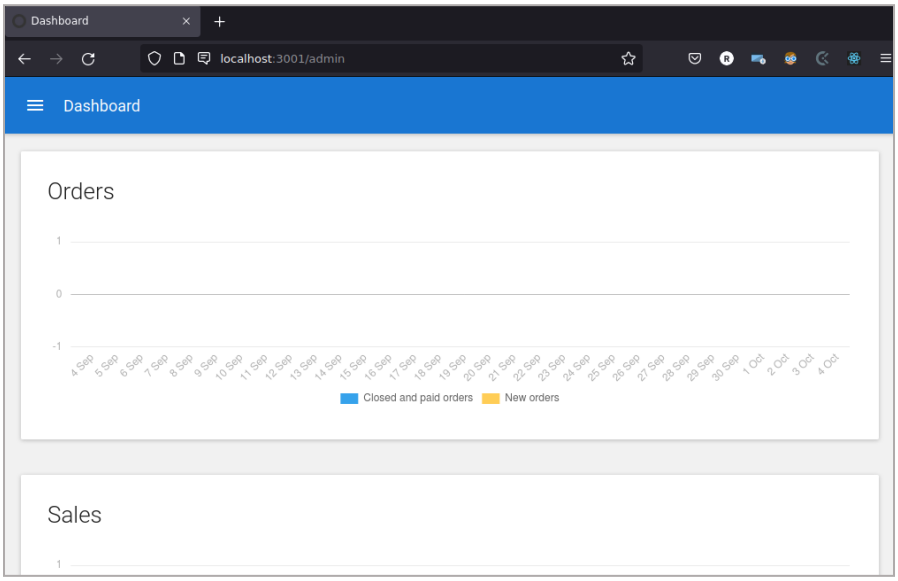


Figura 2.17 Página de inicio del aplicativo web administrative

DESARROLLO SERVIDOR WEB PARA INTEGRACIÓN DE SISTEMA DE TARJETAS CON EL SISTEMA DE E-COMMERCE.

Para empezar, se utilizó el interfaz de líneas de comando de NestJS para crear un nuevo proyecto. El nombre de este componente del sistema es: "loyalty-cards-cezerin-middleware". Luego, se instalaron las librerías: pg,typeorm axios y rxjs. Para la instalación se utiliza el comando "npm install".

Se desarrollo un solo esquema: PurchasePunchRequest, mismo que corresponde a la solicitud de perforación que ocurre cada que se realiza una orden. Si el cliente realiza una orden exitosamente y tiene una tarjeta de fidelización asignada que no tiene todas sus perforaciones, recibe una perforación adicional como premio por su compra. El Código 2.18 muestra la clase PunchersPunchRequest del archivo "punchers-purchase-punch-request.entity.ts" con la definición de la tabla purchase-punch-requests.

```
@Entity('purchase-punch-requests')
export class PunchersPurchasePunchRequest {
  @PrimaryColumn()
  orderId: string;

  @Column({ nullable: true })
  userId: string;

  @Column({ nullable: true })
  punchedAssignedCardId: number;

  @ManyToMany(() => LoyaltyCardsLog, (logs) => logs.purchasePunchRequests)
  @JoinTable()
  logs: LoyaltyCardsLog;
}
```

Código 2.18 Clase PunchersPunchRequest

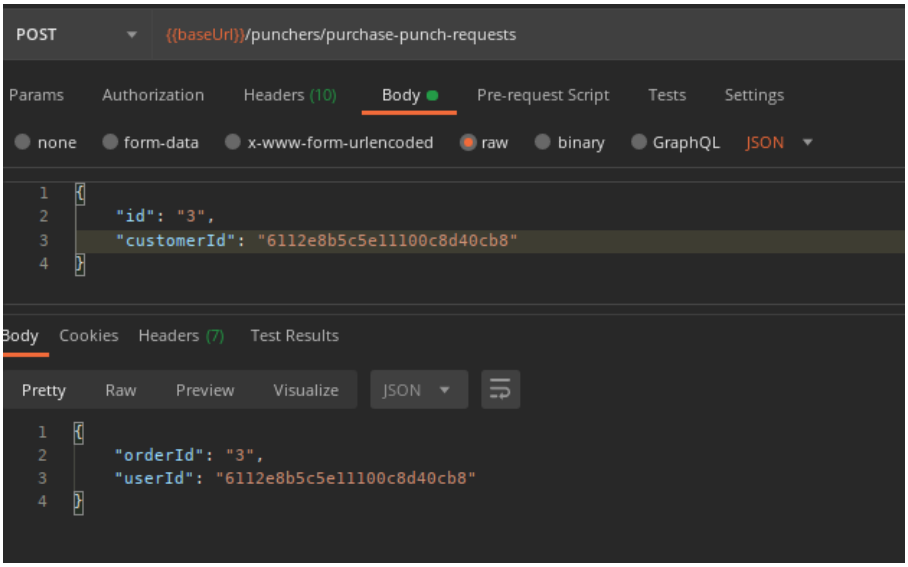
De manera similar al servidor web de tarjetas de fidelización, se utilizó el módulo @nestjs/config para utilizar variables de ambiente. Este microservicio utiliza las variables de TypeORM y la variable LOYALTY_CARDS_HOST para definir el host del servidor web de fidelización.

A diferencia del servidor web de fidelización este microservicio realiza consultas HTTP a otro servidor. Para esto se usó el módulo httpService para realizar consultas y la librería RxJS. El Código 2.19 muestra el método create del servicio AssignedCardPunchesService. En este método el servicio httpService utiliza el método post para realizar una consulta HTTP con el verbo POST. El servicio httpService toma el parámetro genérico "unknown" que indica el tipo de datos devuelto por la llamada. Utilizar "unknown" es una alternativa a definir una interfaz o clase con el tipo retornado cuando el control de la respuesta se la realiza de otra manera. En este caso el control se lo realizó con el valor del código de respuesta, siendo 200 el valor que indica una respuesta exitosa. Por último, el método "create" devuelve el "observable" para ser leído por otros servicios.

```
create(assignedCardId: number): Observable<AxiosResponse<unknown>> {
    const URL = `${this.LOYALTY_CARDS_HOST}${this.getSlug(assignedCardId)}`;
    const httpRequest = this.httpService.post<unknown>(URL).pipe(
      tap({
        next(axiosResponse) {
          console.log(` --- Log
          statuscode: ${axiosResponse.status}
          resposeBody: ${JSON.stringify(axiosResponse.data)}
          requestBody: null
          url: ${URL}
          `);
        },
      }),
      catchError((error) => {
        console.error('========---!!!!=== ERROR', error);
        throw new Error('---Error when puncing cards');
      }),
    );

    return httpRequest;
  }
```

Código 2.19 Método create del servicio AssignedCardPunchesService

Este sistema, incluye un script de Docker idéntico al del sistema de tarjetas de fidelización. Esto se debe a que las tecnologías utilizadas son en base las mismas. Para levantar en local se utilizó el comando "npm run start: dev". Una vez levantado, los endpoints están disponibles para usarse por medio de consultas HTTP. El Código 2.20 muestra la respuesta del endpoint al realizar la consulta por medio de la herramienta Postman de Google.



Código 2.20 Respuesta del endpoint

La respuesta del endpoint es informativa y no es utilizada por otro microservicio o aplicativo web. En la base de datos se tiene registros más detallados del proceso. La Figura 2.18 muestra el contenido de la tabla purchase-punch-requests con un registro que indica que la orden con id 3 que corresponde al usuario con id 6112e8b5c5e11100c8d40cb8 ha resultado en la perforación de la tarjeta con id 6.

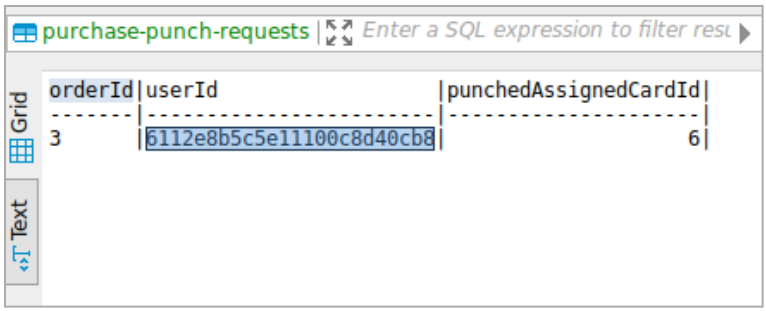


Figura 2.18 Contenido de la tabla purchase-punch-requests

[redacted]DESARROLLO SCRIPT DE DOCKER COMPOSE

Para el desarrollo del script de Docker Compose, que sirve para levantar todo el sistema, se inició con la creación de un directorio y un proyecto de Github. Se le asignó la URL https://github.com/riuGlobal/thesis-launcher.git. Se escribió el archivo principal, "docker-compose.yml" y se definieron los servicios dentro del script que corresponden, cada uno, a un componente del sistema. Los servicios definidos dentro del archivo principal son:

- loyalty-cards-admin: aplicativo web administrativo de tarjetas de fidelización;
- loyalty-cards-web: aplicativo web de clientes para del sistema de tarjetas de fidelización;
- loyalty-cards: servidor web del sistema de tarjetas de fidelización;
- loyalty-cards-cezerin-middleware: servidor web para integración del sistema de tarjetas de fidelización con el sistema de e-commerce;
- smtp: servidor de correos de Mailhog utilizador por el sistema de e-commerce;
- ecommerce: servidor web para el sistema de e-commerce;
- web: aplicativo web para el sistema de e-commerce;
- admin: aplicativo web administrativo para el sistema de e-commerce.

Cada uno contiene: un atributo build con la URL de su repositorio correspondiente, un atributo environment con las variables de ambiente que requiere para configuración, un atributo ports

asociando los puertos de la instancia virtual a crear con la del host, y opcionalmente un atributo depends_on para listar los servicios de los que depende. El Código 2.21 muestra el servicio "loyalty-cards-admin". Para este caso se utilizó la URL git@github.com:riuGlobal/loyalty-cards-admin.git junto a una variable que determina una sección del repositorio identificado por un hash de un commit o el nombre de una rama de git. Se asocia el puerto 3003 del host al puerto 3000 de la instancia y se definen los valores de las variables de ambiente: ESLINT_NO_DEV_ERRORS toma el valor de una variable de su mismo nombre del archivo ".env" mientras que REACT_APP_LOYALTY_CARDS_API_HOST toma el valor de la variable LOYALTY_CARDS_API_HOST del mismo archivo.

```
loyalty-cards-admin:
    build: git@github.com:riuGlobal/loyalty-cards-admin.git#${LOYALTY_CARDS_ADMIN_VERSION:-master}
    ports:
      - 3003:3000
    environment:
      - ESLINT_NO_DEV_ERRORS
      - REACT_APP_LOYALTY_CARDS_API_HOST=${LOYALTY_CARDS_API_HOST}
```

Código 2.21 Servicio "loyalty-cards-admin"

Las bases de datos se declararon de forma separada de manera que se puedan levantar opcionalmente. Se desarrolló un archivo "docker-compose.restart.yml" que agrega el atributo restart a todos los servicios del archivo principal. Este atributo permite definir una política de reinicio, por lo que se escogió la política unless-stopped que reinicia la instancia cuando sufre fallos o en caso de reinicio del host pero no cuando se ha indicado explícitamente que se detenga la instancia por medio de un comando. Estos atributos están declarados en un archivo distinto al principal para permitir su uso de forma opcional. En ambiente local y en casos de prueba puede no ser deseable el reinicio automático.

Finalmente, se levantó todo el proyecto: sistema de fidelización y de e-commerce, con el comando "docker-compose up -d". La opción "-d" permite liberar la consola. El Código 2.22 muestra el listado de los contenedores que corren en el sistema local luego que estos han sido inicializados de esta manera. Para listarlos se ha usado el comando "docker ps". En este ejemplo se observa que todos los servicios están listos para usar.

```
CONTAINER ID        IMAGE                                              COMMAND                  CREATED
    STATUS              PORTS                                              NAMES
b7c09221cc78        thesis-launcher_loyalty-cards-cezerin-middleware   "docker-entrypoint.s…"   18 hours ago
    Up 18 hours         0.0.0.0:8031->8000/tcp                             thesis-launcher_loyalty-cards-cezerin-m
iddleware_1
096dd39b8966        thesis-launcher_loyalty-cards-admin                "docker-entrypoint.s…"   18 hours ago
    Up 18 hours         0.0.0.0:3003->3000/tcp                             thesis-launcher_loyalty-cards-admin_1
30170521d34f        thesis-launcher_loyalty-cards-web                  "docker-entrypoint.s…"   18 hours ago
    Up 18 hours         8000/tcp, 0.0.0.0:3002->3000/tcp                   thesis-launcher_loyalty-cards-web_1
1d8e71ca964b        thesis-launcher_web                                "docker-entrypoint.s…"   24 hours ago
    Up 18 hours         0.0.0.0:3000->3000/tcp                             thesis-launcher_web_1
505cc52c2e68        thesis-launcher_admin                              "docker-entrypoint.s…"   24 hours ago
    Up 18 hours         0.0.0.0:3001->3000/tcp                             thesis-launcher_admin_1
8b2e6277aaef        thesis-launcher_ecommerce                          "docker-entrypoint.s…"   24 hours ago
    Up 18 hours         0.0.0.0:8017->3000/tcp, 0.0.0.0:8016->3001/tcp     thesis-launcher_ecommerce_1
3c3d726ca273        thesis-launcher_loyalty-cards                      "docker-entrypoint.s…"   24 hours ago
    Up 18 hours         0.0.0.0:8030->8000/tcp                             thesis-launcher_loyalty-cards_1
50c077259874        mongo:4.4                                          "docker-entrypoint.s…"   24 hours ago
    Up 18 hours         0.0.0.0:27017->27017/tcp                           thesis-launcher_database_1
b00f815f9986        postgres:13-alpine                                 "docker-entrypoint.s…"   24 hours ago
    Up 18 hours         0.0.0.0:15432->5432/tcp                            thesis-launcher_db-pg_1
556076e2eaf2        thesis-launcher_smtp                               "MailHog"                24 hours ago
    Up 18 hours         1025/tcp, 0.0.0.0:8025->8025/tcp                   thesis-launcher_smtp_1
```

Código 2.22 Servicios en local

3. RESULTADOS Y DISCUSIÓN

En esta sección se documenta el proceso del despliega del sistema en la nube del sistema completo previamente desarrollado en local y los resultados de la ejecución del plan de pruebas. Además, se realizaron pruebas adicionales que evidencian funcionalidades de las tecnologías utilizadas.

DESPLIEGUE EN LA NUBE

APROVISIONAMIENTO INFRAESTRUCTURA EN LA NUBE

Para el despliegue en la nube, primeramente, se requiere de una instancia en la misma. Para su provisionamiento se el producto EC2 o Elastic Cloud Computing. Para crear una instancia se utilizó el dashboard de AWS. La Figura 3.1 muestra el dashboard de creación de instancias de AWS.

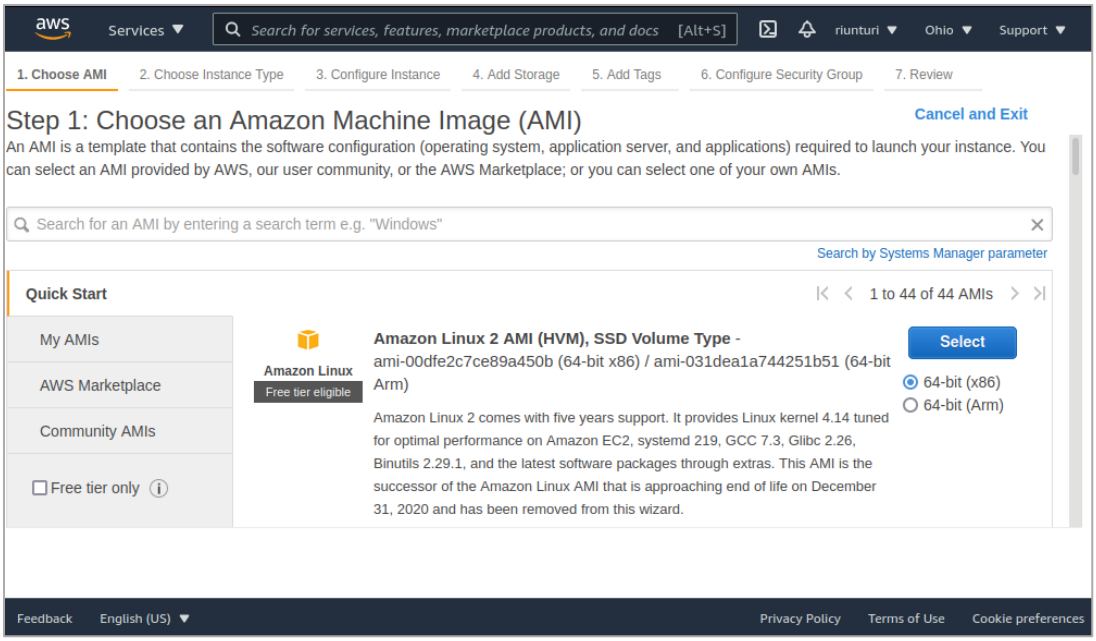


Figura 3.1 Dashboard de creación de instancias de AWS

A través de una serie de pasos, AWS consulta información de las instancias a crear. Dos pasos son obligatorios:

Elija AMI: AMI o Amazon Image es la imagen de un sistema operativo a utilizar. Para este caso se eligió el sistema operativo Ubuntu Server 18.04 en su versión de 64 bits con procesador x86 de Intel.

Elegir tipo de instancia: El tipo de instancia definirá el número de CPU y la memoria RAM disponible para la instancia. Al correr el proyecto en local con todos los sistemas y consultar el uso de recursos se obtiene un uso de 5.37GB de memoria RAM al iniciar los servicios. Esta información se obtuvo usando el comando htop. La Figura 3.2 muestra la respuesta de este comando. En Amazon, existe una opción de 6GB, pero esta no deja mucho margen por lo que se prefiere el siguiente nivel que corresponde a 8GB, por esto se utilizó el tipo t2.large que incluye 2 CPUs y 8GB de memoria RAM.

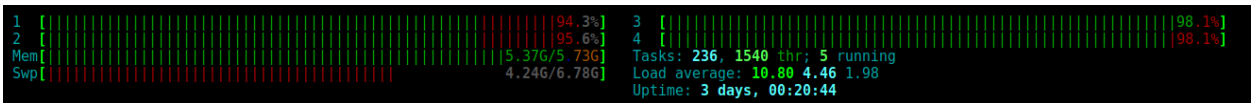


Figura 3.2 Respuesta del comando htop

Luego de seguir estos pasos obligatorios se puede elegir la opción "Revisar y lanzar". Esto elegirá las opciones por defecto para las demás opciones. Antes de lanzar la instancia, AWS pedirá el par de claves a usar para el acceso a la nueva instancia. Se pueden elegir un par o crear un nuevo par. Luego de esto la instancia será creada o "lanzada" en términos de AWS.

Para habilitar el tráfico hacia o desde un determinado puerto, se crean reglas indicando el puerto al que corresponde la regla, el protocolo TCP o UDP, la red o IP que tiene permiso para utilizar ese puerto y una breve descripción informativa. La Figura 3.3 muestra algunas reglas definidas para el grupo de seguridad a usar en las instancias de este proyecto.

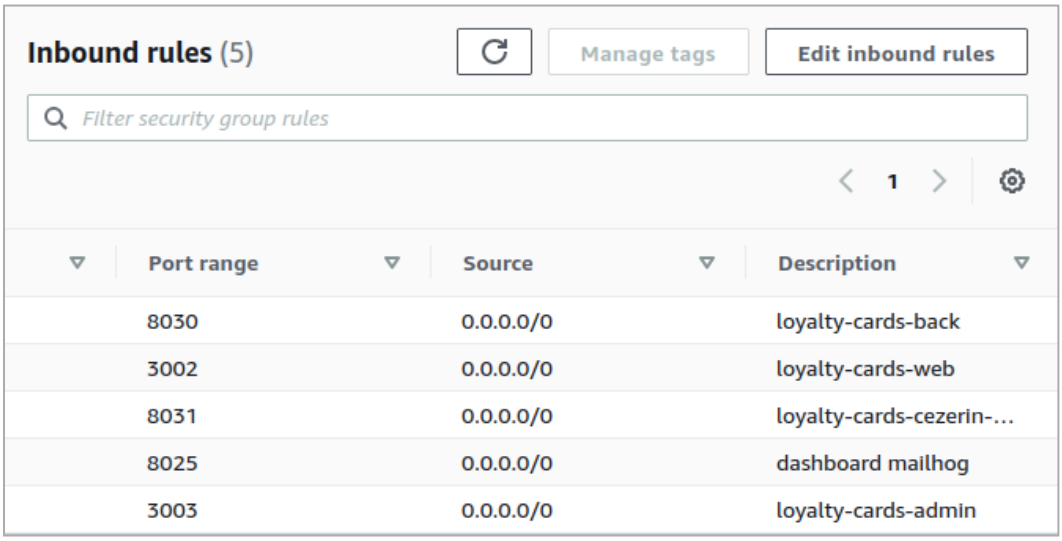


Figura 3.3 Reglas definidas para el grupo de seguridad

Una vez realizado esto se puede ingresar a la instancia utilizando un cliente SSH. La Figura 3.4 muestra la consola que tiene acceso a la nueva instancia. En esta se observa la IP pública establecida por AWS para la instancia: 3.144.159.144.

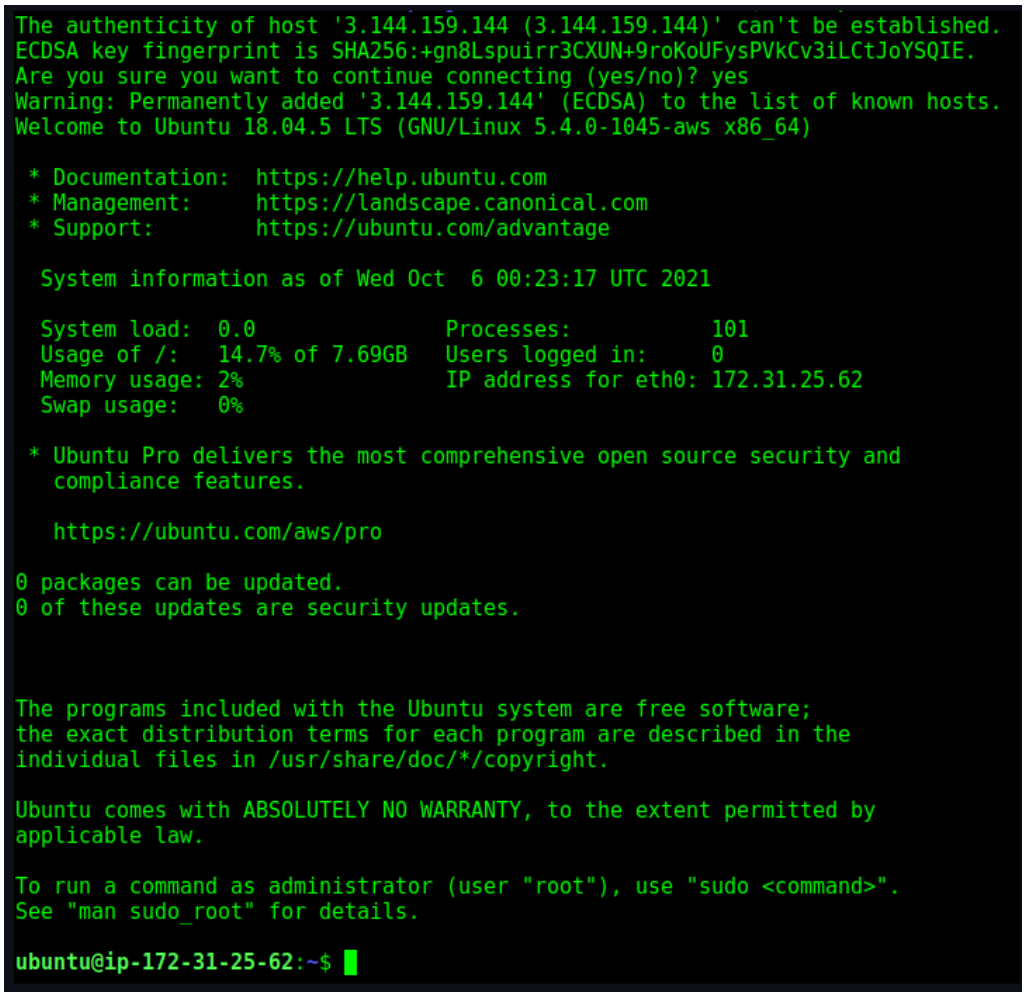


Figura 3.4 Consola que tiene acceso a la nueva instancia

[redacted]PREPARACIÓN DEL AMBIENTE EN LA NUBE

La instancia necesita únicamente Docker, Docker-compose y el script de Docker Compose del proyecto. El Código 3.1 muestra la instrucción para acceder a la instancia en la nube del ambiente de producción utilizando ssh.

```
- ssh -i ~/riu-fs-2019/secriu/pems/riunturi-otim.pem ubuntu@3.144.159.144
```

Código 3.1 Comando para acceder al ambiente de producción

La instrucción requiere como argumento el usuario y la dirección IP pública de la instancia. Adicionalmente se usa la opción "-i" que permite pasar la ruta del archivo pem que contiene la llave privada que se usa para permitir el acceso a la instancia. La Figura 3.5 muestra la terminal con el acceso a la consola en la nube, las instrucciones que se corran en esta terminal se ejecutarán en la instancia en la nube. La terminal muestra como bienvenida información de: el sistema operativo, el último ingreso, el uso de recursos, entre otros.

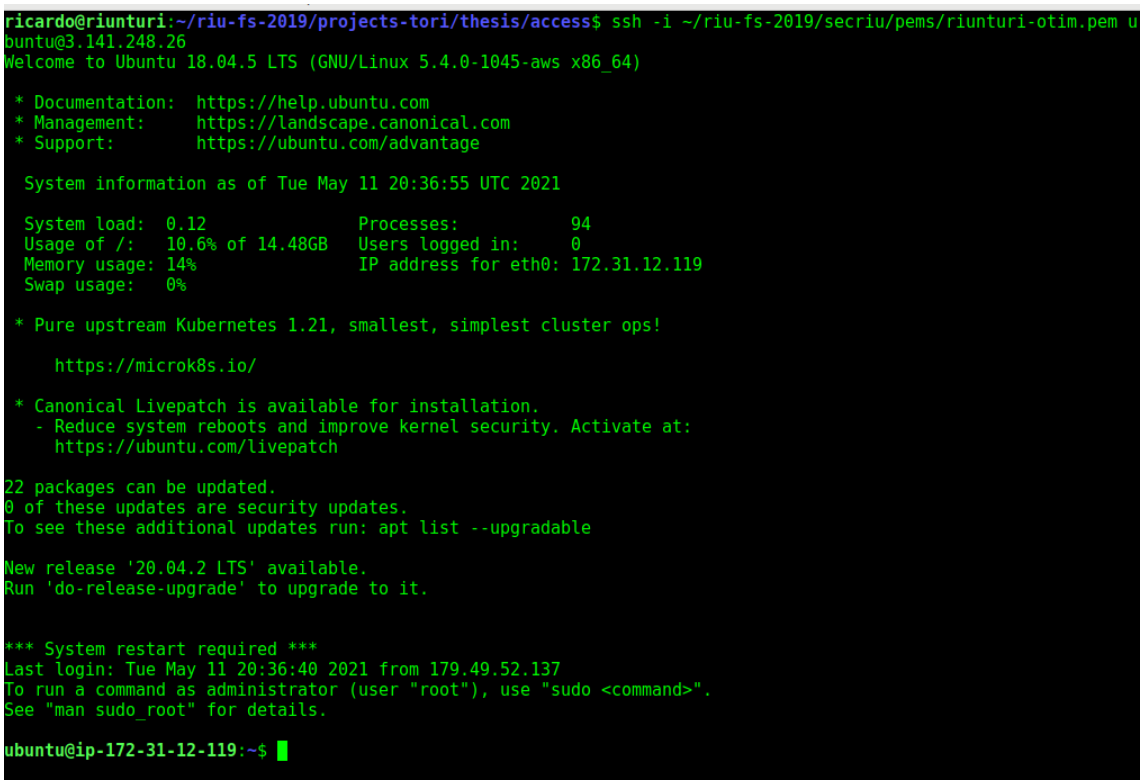


Figura 3.5 Terminal con el acceso a la consola en la nube

Para la instalación de Docker se utilizó la guía de Digital Ocean publicada en https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04. Para instalar Docker usando este script, primero se crea un directorio.

Luego, se clonó el proyecto que contiene los scripts. Una vez clonado, se corrió el script correspondiente al sistema operativo en este caso Ubuntu, mediante el comando “bash install-docker-ubuntu.sh”. La Figura 3.6 muestra parte de la respuesta de la consola donde se indica la instalación de algunos componentes de Docker como "containerd" elemental para su funcionamiento.

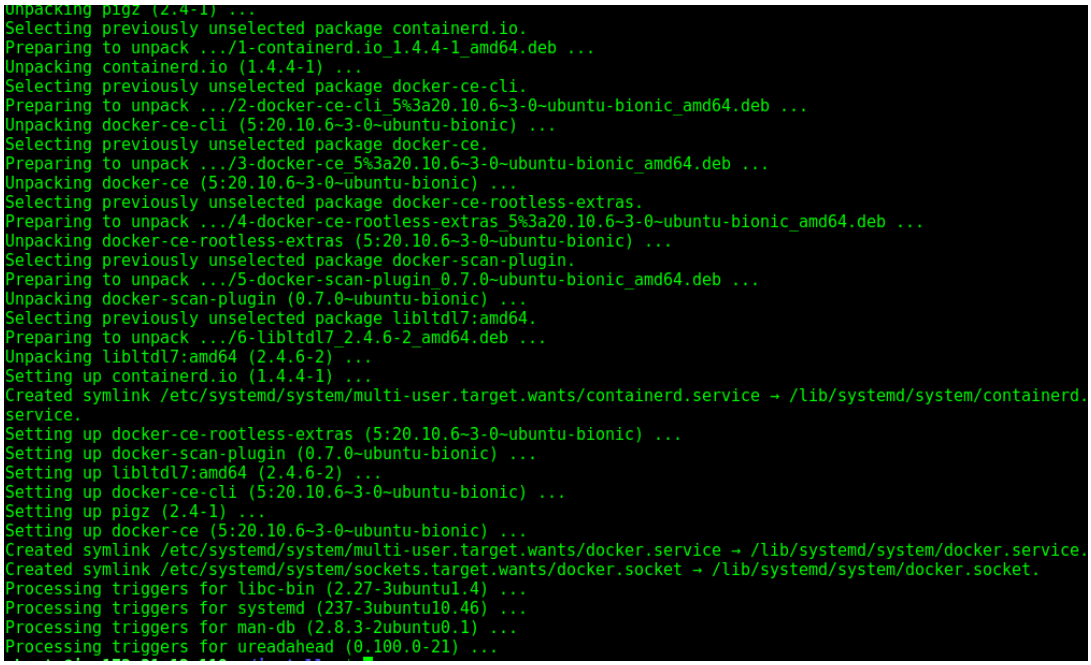


Figura 3.6 Respuesta de consola a instalación

Se comprueba la instalación de Docker usando la instrucción del comando “docker -v”. La Figura 3.7 muestra respuesta de la consola que indica que se encuentra instalada la version 20.10.6.

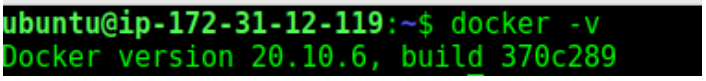


Figura 3.7 Respuesta de la consola con el mensaje de éxito de la instalación

Para instalar Docker Compose en la instancia en la nube se usó el script del repositorio https://github.com/riuGlobal/docker-installers-tori. Este ya fue clonado para la instalación de Docker. El Código "bash install-docker-compose.sh” muestra la instrucción para correr el script de instalación. Esto debe ser ejecutado en el directorio donde se encuentra el script o a su vez utilizar la ruta hacia el script desde el directorio actual. Para verificar la instalación se ejecutó la instrucción “docker-compose -v”. La Figura 3.8 muestra la respuesta de la consola indicando que se encuentra instalada la versión 1.25.3.

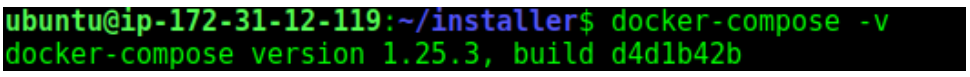


Figura 3.8 Respuesta de la consola con la instalación de la versión 1.25.3

Para obtener el script de Docker Compose desarrollado para lanzar el sistema se utilizó el comando “git clone https://github.com/riuGlobal/thesis-launcher.git”. Finalmente es recomendable reiniciar la instancia usando el comando “sudo reboot”.

CONFIGURACIÓN Y LEVANTAMIENTO DEL SISTEMA CON UN COMANDO

Antes de lanzar la instancia fue necesaria configurarla. Para la configuración de la instancia se utilizó un archivo ".env" en el directorio donde se encuentra el script de Docker Compose. Para crear un nuevo archivo se utilizó la instrucción “vim .env”. En este archivo se definieron los archivos YAML a usar por Docker Compose, las URL de los servidores que los aplicativos pasaron a los exploradores web para que puedan hacer consultas HTTP, las ramas a usar de cada repositorio y los usuarios con sus contraseñas para las bases de datos. El Código 3.2 muestra una parte del archivo de configuración utilizado para el ambiente de producción. Algunas variables que destacan son:

- COMPOSER_FILE: esta es una variable de Docker Compose para indicar los archivos YAML a usar al momento de levantar el sistema;
- LOYALTY_CARDS_API_HOST: Es la URL del servidor web de tarjetas de fidelización que utilizará los exploradores web para realizar consultas HTTP;
- LOYALTY_CARDS_WEB_VERSION: Es la rama del repositorio a utilizar del aplicativo web para clientes de fidelización;
- PG_DB_HOST: Es el host de la base de datos a usar por el servidor web de fidelización y el servidor web para integración. Se utiliza el nombre de host db-pg gracias al servidor DNS de Docker que permite traducirlo a la IP correspondiente.

```
COMPOSE_FILE=docker-compose.yml:docker-compose.db.yml:docker-compose.db-pg.yml

LOYALTY_CARDS_API_HOST=http://3.132.111.116:8030
#Loyalty-cards-admin
LOYALTY_CARDS_ADMIN_VERSION=release/v1
ESLINT_NO_DEV_ERRORS=true

#Loyalty-cards-web
LOYALTY_CARDS_WEB_VERSION=release/v1

# Loyalty-cards
LOYALTY_CARDS_VERSION=release/v1

PG_DB_HOST=db-pg
PG_DB_PORT=5432
PG_DB_USER=root
PG_DB_PASSWORD=thesis1234
LOYALTY_CARDS_PG_DB_DATABASE=postgres
# LOYALTY_CARDS_PG_DB_SCHEMA=
LOYALTY_CARDS_PG_DB_SYNC=true

# Loyalty-cards-cezerin-middleware
LOYALTY_CARDS_CEZERIN_MW_VERSION=release/v1

LOYALTY_CARDS_CEZERIN_MW_PG_DB_DATABASE=postgres
LOYALTY_CARDS_CEZERIN_MW_PG_DB_SYNC=true
#Database

MONGO_INITDB_ROOT_USERNAME=root
MONGO_INITDB_ROOT_PASSWORD=thesis1234
```

Código 3.2 Parte del archivo de configuración del ambiente de producción

Las líneas que inician con "#" son comentarios para facilitar la legibilidad de este archivo. Finalmente, se levantó el sistema con el comando “docker-compose up -d”. El comando "up" levanta todos los servicios mientras que la opción "-d" libera la consola. Una vez levantado se pueden usar los aplicativos y los servidores. La Figura 3.9 muestra la página de inicio del e-commerce del ambiente de producción desplegado en la nube.

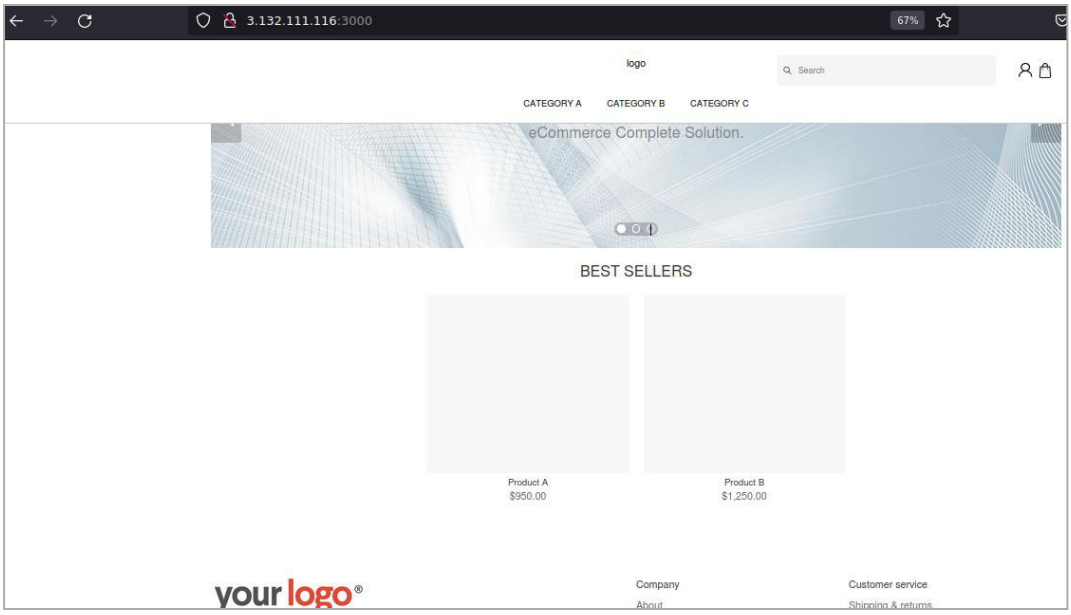


Figura 3.9 Página de inicio del e-commerce del ambiente de producción

EJECUCIÓN PLAN DE PRUEBAS

Las historias de usuario definidas previamente describen los objetivos de funcionalidad del sistema. El plan de pruebas los define como las funcionalidades a probar. Se realizaron pruebas para comprobar y documentar que el sistema desarrollado incluye todas las funcionalidades definidas en las historias de usuario. Para esto se utilizó el ambiente de desarrollo que corresponde a una instancia desplegada en la nube.

AUTENTICACIÓN EN APLICATIVO WEB.

Para verificar que un usuario registrado pueda ingresar con sus credenciales al sistema, es necesario, primeramente, realizar el registro de un usuario. Para esto se utiliza la URL: http://3.141.248.26/register. Para esta prueba se completó el formulario de registro con los siguientes datos:

- Nombre: Ricardo
- Apellido: Ortiz
- email: ricardo.ortiz+1@epn.edu.ec
- password: 123456789

Una vez completado el formulario se utilizó el botón "Register". Luego de esto, el sistema indicó que se ha realizado un preregistro y que se ha enviado un correo electrónico. El sistema

implementa un servidor de correos de pruebas que captura los correos electrónicos enviados desde el sistema de e-commerce. Para revisar estos correos está disponible en un aplicativo web que para este ambiente está asociado a la URL: http://3.141.248.26:8025. La Figura 3.10 muestra el correo correspondiente al registro realizado en esta prueba visualizado con la herramienta Mailhog.

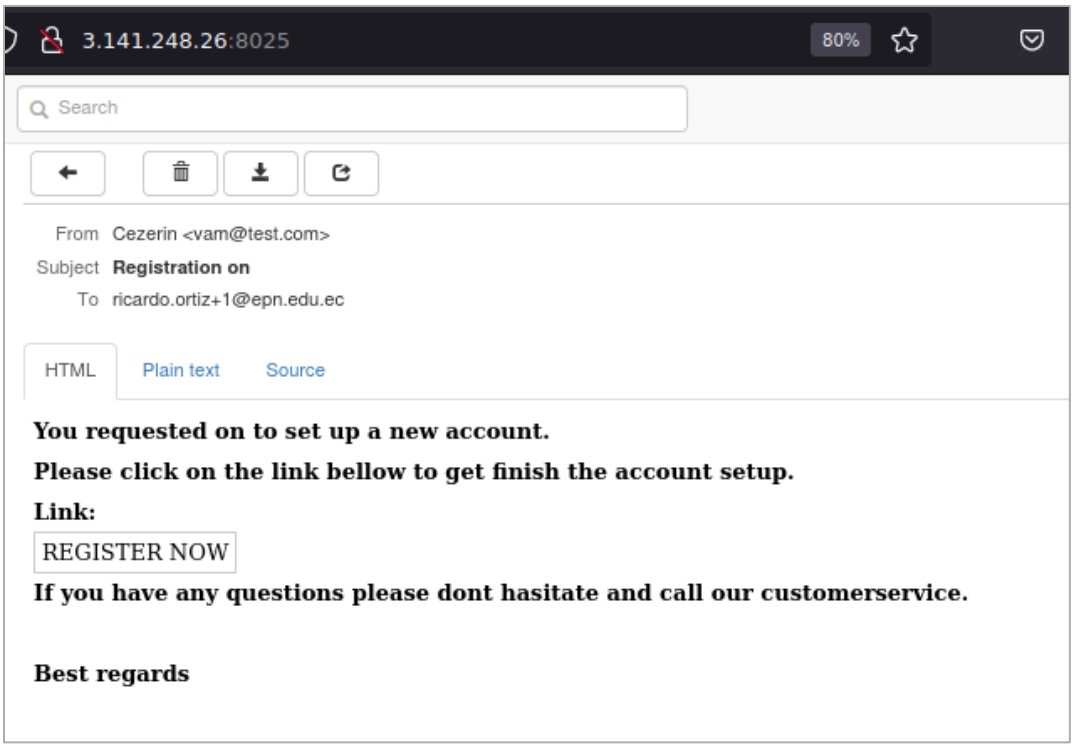


Figura 3.10 Correo electrónico correspondiente al registro

Con este correo se completa el registro usando el botón destinado a este propósito. Al completar el registro el botón llevará al usuario a la página del e-commerce donde se tendrá un mensaje de éxito del registro. Una vez realizado el registro se puede realizar el ingreso al sistema con el nuevo usuario registrado. Esto se lo realizó en la URL "http://3.141.248.26/login" a la que se puede llegar con el botón provisto en la página con el mensaje de éxito o desde el ícono de usuario presente en la cabecera de la página. Una vez ingresadas las credenciales, para esta prueba: ricardo.ortiz+1@epn.edu.ec y 123456789, el usuario ha completado el proceso de autenticación.

El sistema web muestra al usuario la página "http://3.141.248.26/customer-account", donde se puede revisar información del usuario, sus órdenes y la nueva funcionalidad desarrollada de premios de lealtad. Adicionalmente, existe una viñeta para cerrar la sesión. La Figura 3.11 muestra la página de información del usuario correspondiente al usuario utilizado en esta prueba.

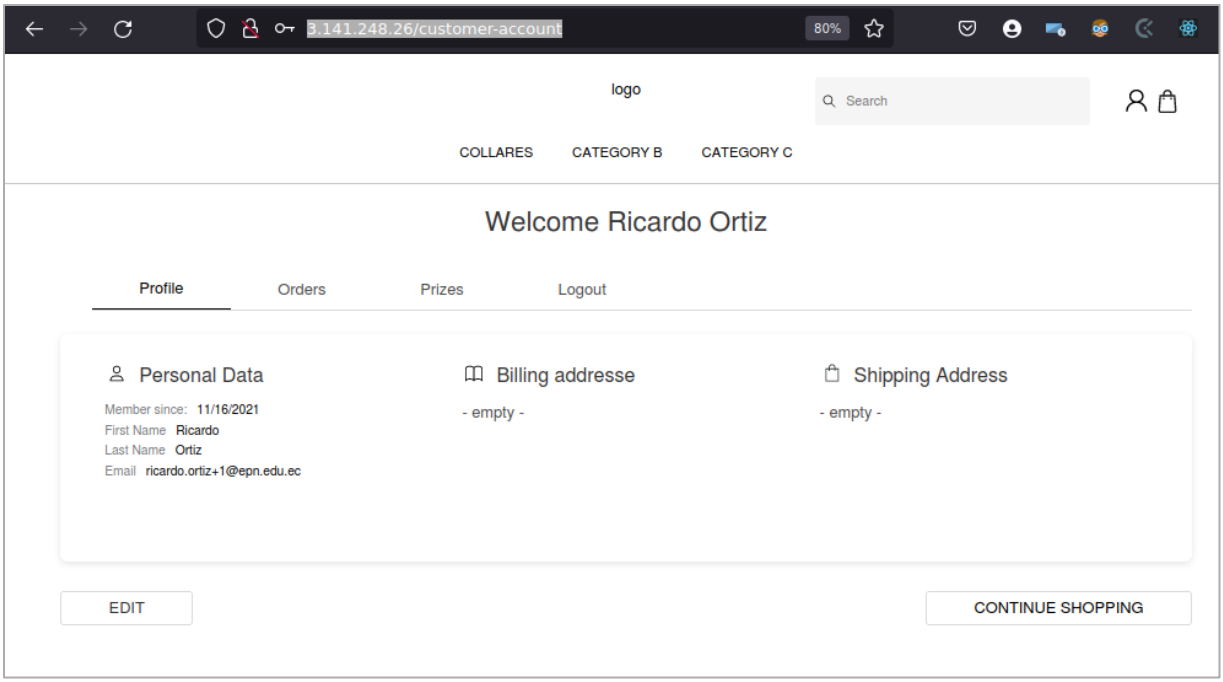


Figura 3.11 Página de información de Usuario

[redacted]AUTENTICACIÓN EN APLICATIVO WEB ADMINISTRATIVO

No existe un mecanismo de autenticación para el sistema administrativo. El control de acceso se lo realiza con un sistema de listas blancas de las direcciones IP de origen permitidas a nivel de infraestructura. De esta manera, se limita el acceso a usuarios con una IP configurada en el grupo de seguridad correspondiente. El grupo de seguridad incluye reglas asociadas a un puerto de la instancia. La regla asociada al puerto 3001 es la que corresponde al sistema web administrativo del e-commerce para el ambiente de desarrollo. Para las pruebas no se realizan bloqueos de manera que sea más sencillo acceder al aplicativo.

[redacted]REVISIÓN DE CATÁLOGOS DE PRODUCTOS

El catálogo de productos se encuentra en la página inicial del aplicativo web. Estos se encuentran listados debajo del banner informativo. Para esta prueba se verificó la existencia de tres productos, los cuales no tienen imágenes asociadas y uno, el producto "Producto C", no está disponible por falta de existencia en inventario. Estos productos fueron configurados de manera automática al levantar el sistema. La Figura 3.12 muestra los productos listados al momento de realizar esta prueba.

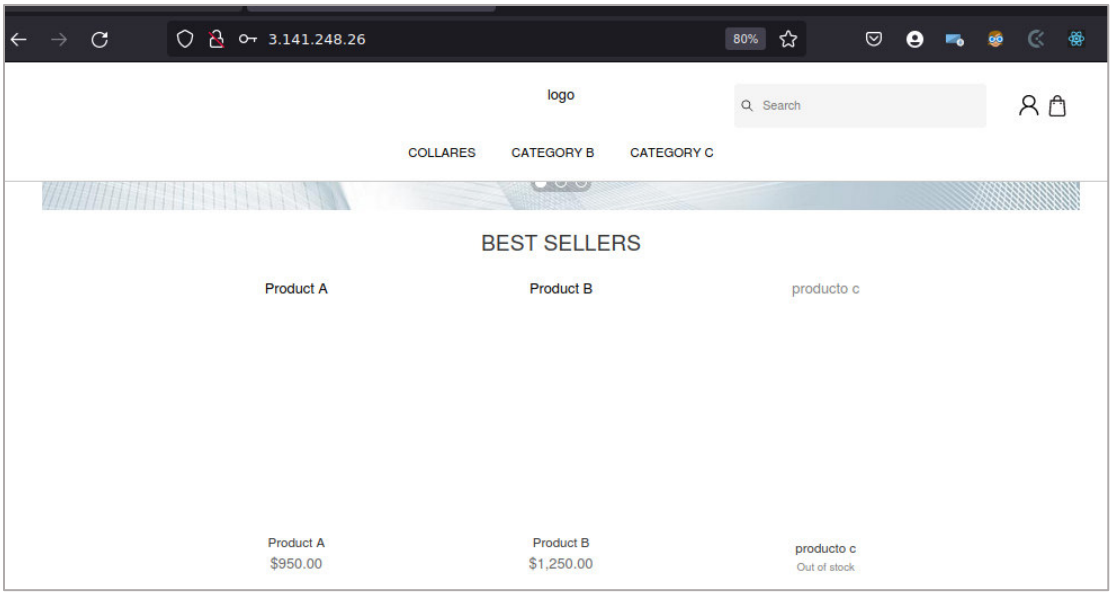


Figura 3.12 Productos listados

[redacted]PROCESO DE COMPRA

Para realizar una compra se debe agregar al menos un producto al carrito de compras. Para esta prueba se agregaron dos: el producto "Product A" y el producto "Product B", ambos se han agregado desde la página inicial. Desde la barra de herramientas superior se puede controlar los productos en el carrito. Además, desde la barra se procede al siguiente paso "Checkout" con el botón destinado para ese propósito. La Figura 3.13 muestra la página inicial con el carrito de compras desplegado y mostrando que contiene dos productos.

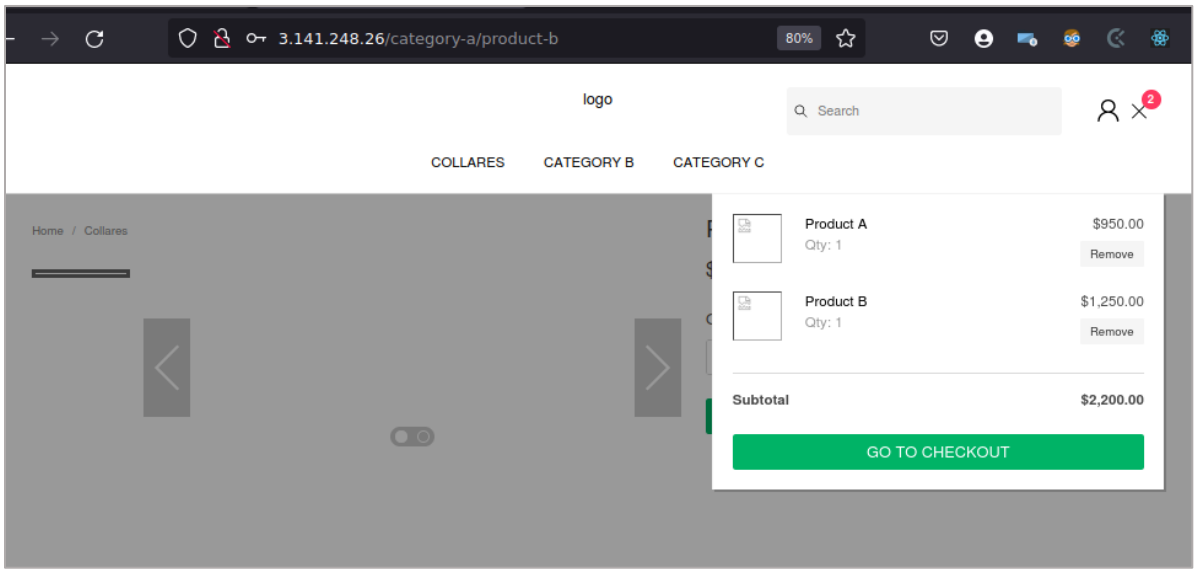


Figura 3.13 Página inicial con el carrito de compras desplegado

Luego, se procedió al siguiente paso con el botón "Go to checkout" de manera que el sistema muestra la página "http://3.141.248.26/checkout". Se completa la información requerida para el pedido como teléfono del usuario, dirección de entrega y opción de pago. El sistema presenta una opción de pago: efectivo contra entrega. La Figura 3.14 muestra el formulario lleno para la prueba realizada.

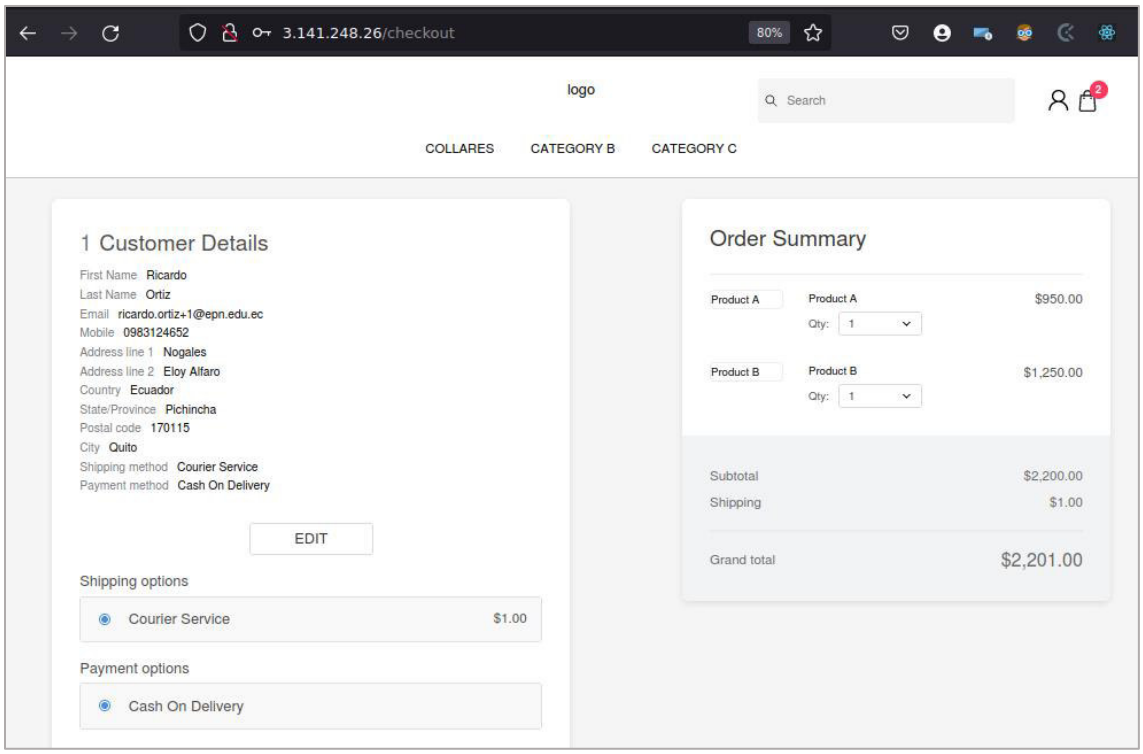


Figura 3.14 Formulario para culminar la prueba

Una vez llenado el formulario se procede a completar la orden con el botón destinado a ese propósito, lo que llevará al usuario a la página: "http://3.141.248.26/checkout-success". En esta página se muestra el detalle de la orden realizada y un mensaje de que el proceso ha sido exitoso. La Figura 3.15 muestra el mensaje de éxito para la orden 1011 realizada en este evento de prueba.

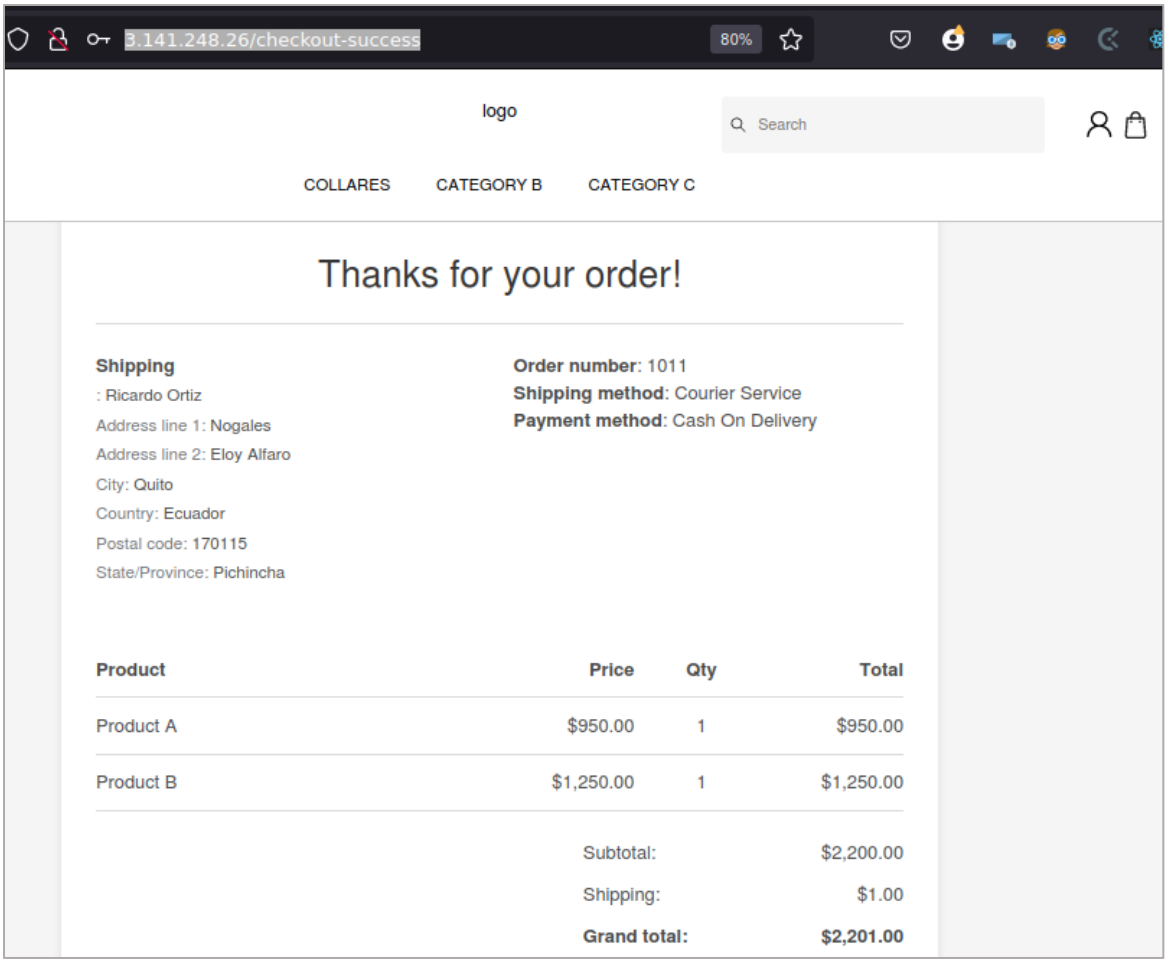


Figura 3.15 Mensaje de éxito para la orden de la prueba

El detalle de la compra realizada se puede revisar, también, desde el aplicativo administrativo. Para verificar esto, se ingresó al aplicativo y se utilizó el menú en el costado izquierdo. Se ingresó a la sección "Orders" y se eligió la orden correspondiente a este evento de prueba

[redacted] CREACIÓN, MODIFICACIÓN Y ELIMINACIÓN DE REGISTRO DE UN PRODUCTO

La creación de un producto es posible realizarla utilizando el aplicativo administrativo web, específicamente la ruta "/admin/products". Para esta prueba se inicia la creación de un producto con el botón dedicado a este propósito ubicado en la esquina superior derecha identificado con un ícono de un signo de cruz. Al utilizarlo se desplegó un formulario, donde se utilizó la siguiente información para completarlo:

- Nombre de producto: Producto de prueba
- Descripción: un producto de prueba para la ejecución del plan de pruebas

- Precio: $10
- Categoría B
- Cantidad en stock: 5 unidades.

Adicionalmente a esta información se asocia al producto una imagen. Esto se hace en la última sección del formulario. La imagen utilizada tiene el nombre "Improv circles pillow" del autor "stitchindye" y está publicada en "https://search.creativecommons.org/photos/49a22b8d-4bf3-4791-96d3-84fe2a7649bd". La licencia de esta imagen es la licencia BY-NC-SA 2.0 que permite el uso libre para fines no comerciales. Una vez completado el formulario se completa la creación de un producto utilizando el botón "Save". La Figura 3.16 muestra el formulario para creación de producto lleno en el aplicativo administrativo.

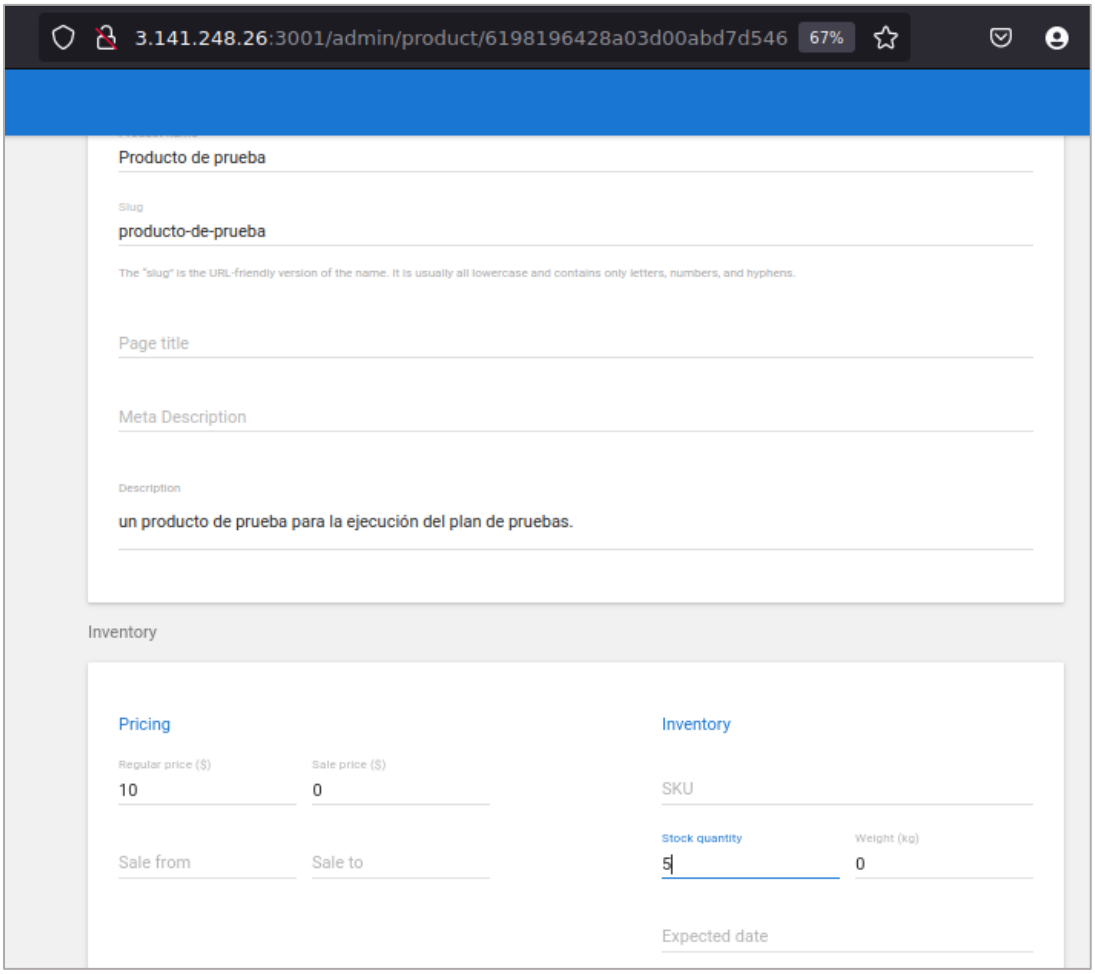


Figura 3.16 Formulario para crear un producto

Finalizado el proceso de creación del producto se pudo observar que este se encuentra listado en la página del e-commerce para clientes. Fue necesario, previamente, refrescar la página. La Figura 3.17 muestra el producto creado en esta prueba.

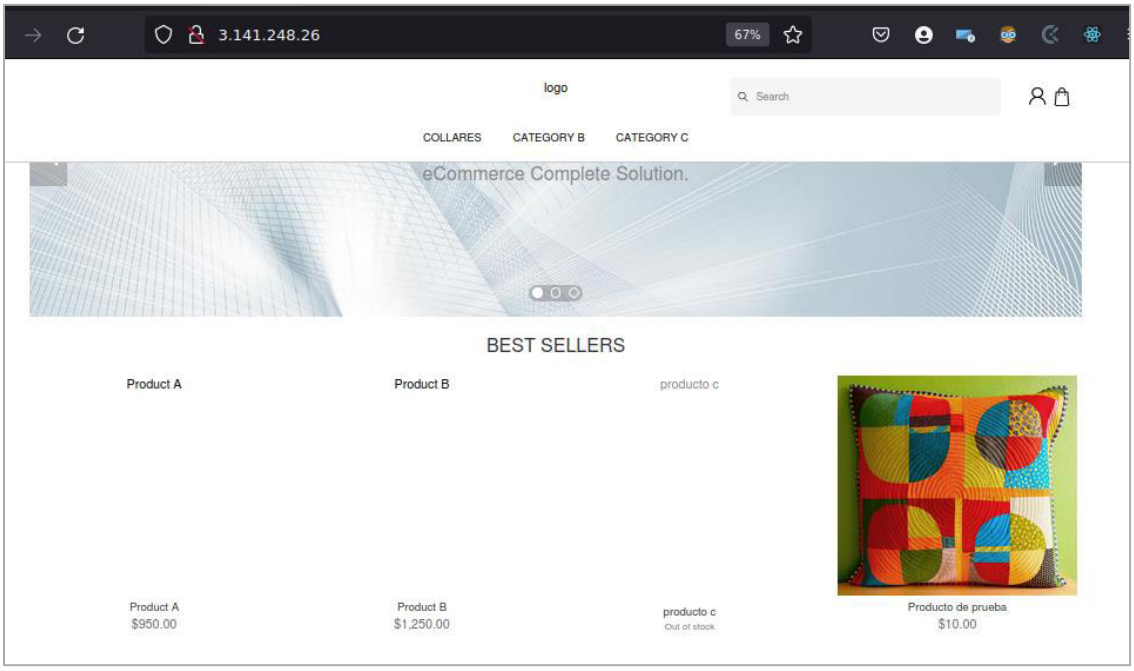


Figura 3.17 Producto creado

Para realizar prueba de modificación se ingresó al detalle del producto que se deseaba modificar, en este caso era el nuevo producto denominado "Producto de prueba". Gran parte de los parámetros se pueden modificar como el precio, la cantidad de existencia en inventario, el nombre, la descripción, la porción de URL asociado, la categoría, entre otros parámetros. Para esta prueba se modificó el precio, pasó de $10 a $20. Una vez realizado el cambio se puede verificar que el producto muestra la lista el nuevo precio en el aplicativo web de clientes. La Figura 3.18 muestra el producto modificado con el nuevo precio que había sido modificado.

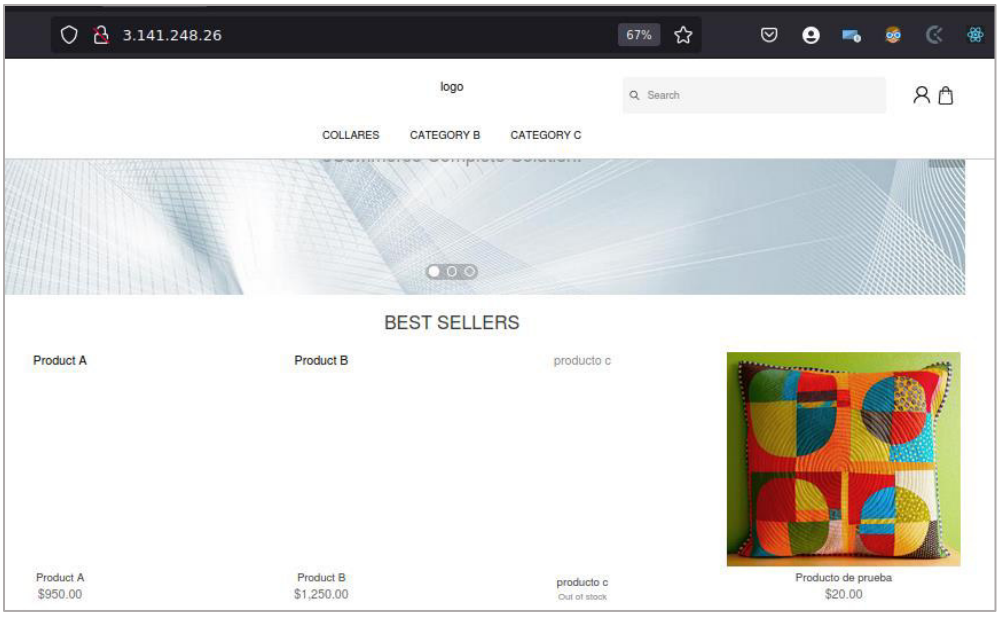


Figura 3.18 Producto con cambios

Para la eliminación del producto está disponible un botón en el detalle del producto. Para esta prueba se eliminó el producto creado previamente.

[redacted] CREACIÓN, MODIFICACIÓN Y ELIMINACIÓN DEL REGISTRO DE UNA CATEGORÍA

Para la creación de una categoría se utiliza el aplicativo administrativo. Para esta prueba se utilizó la ruta "http://3.141.248.26:3001/admin/products/categories". Para la creación se usó el botón de la esquina superior derecha con un ícono de cruz. Asimismo, para esta prueba se utilizó la siguiente información en el formulario de creación de categoría:

- Nombre de categoría: categoría de prueba;
- Descripción: categoría creada con fines de prueba.

La Figura 3.19 muestra el formulario para la creación de categoría correspondiente de esta prueba.

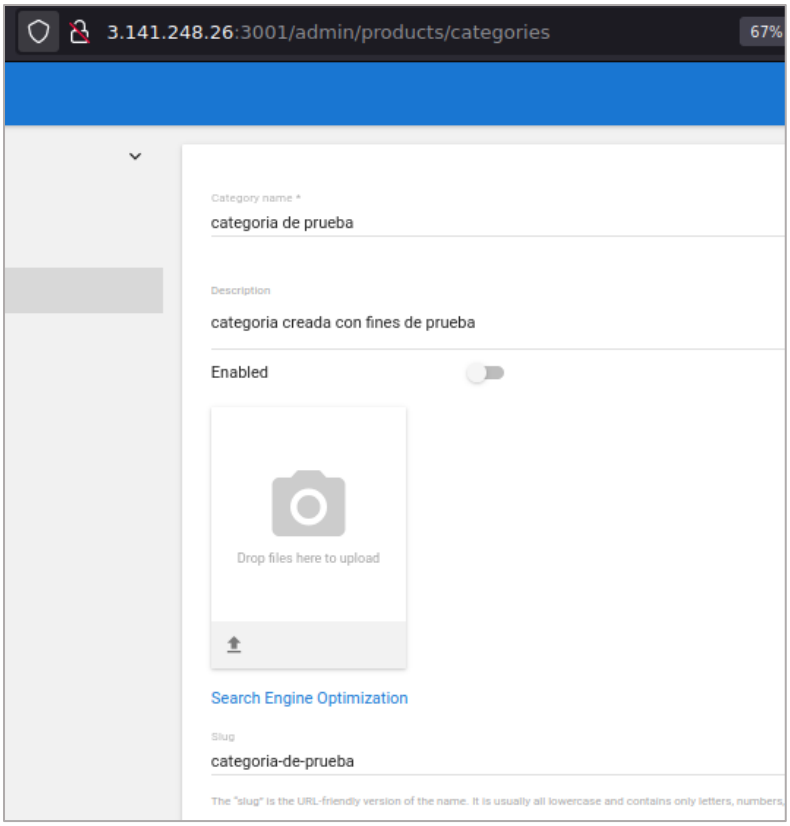


Figura 3.19 Formulario para la creación de categorías

Para observar la categoría se debe, previamente, habilitarla en el mismo formulario. La Figura 3.20 muestra la categoría creada para la ejecución de esta prueba en el aplicativo web.

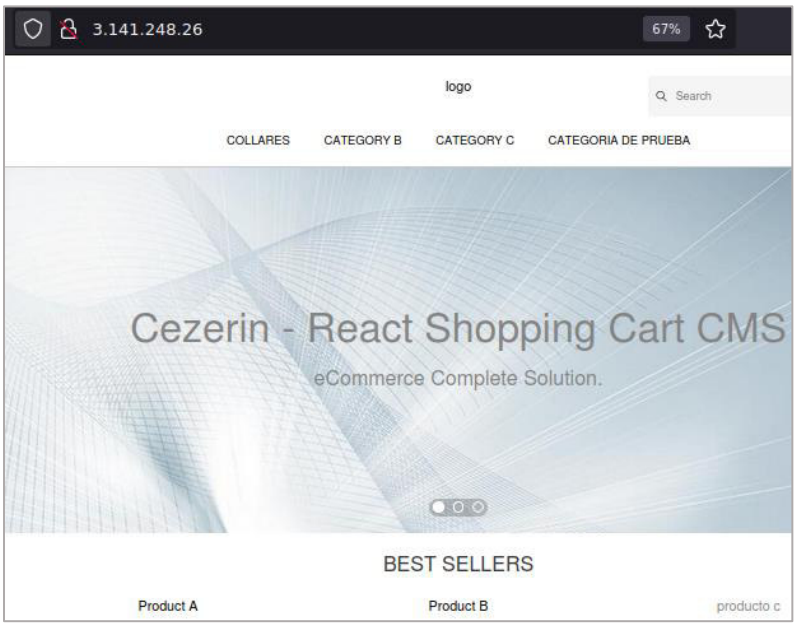


Figura 3.20 Ejemplo de categoría creada

Para la edición de la categoría se utiliza el dashnoard administrativo. La ruta para esto es la del detalle de la categoría que se desea editar. Para esta prueba se modificó el nombre de la categoría, el nuevo nombre colocado es: "Categoría de prueba modificado". Una vez realizado la modificación, se puede verificar la modificación en el aplicativo web de clientes. La Figura 3.21 muestra el aplicativo web con la categoría modificada que ahora muestra el nuevo nombre.

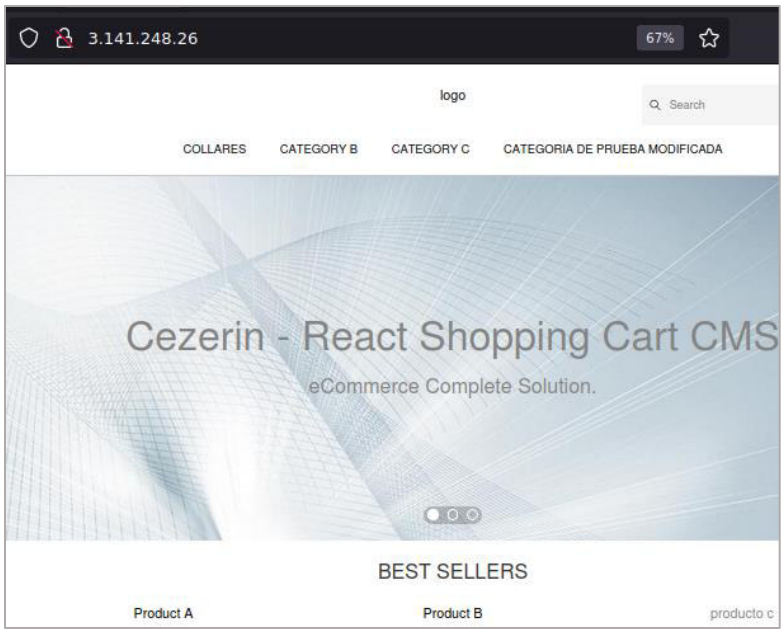


Figura 3.21 Ejemplo de categoría modificada

Para la eliminación de la categoría se utiliza el aplicativo web administrativo, específicamente el botón para eliminación ubicado en la barra de herramientas en la parte superior derecha. Para esta prueba se eliminó la categoría creada previamente.

[redacted]ASOCIACIÓN DE UN PRODUCTO A UNA CATEGORÍA

La asociación de un producto se la realiza en la página del detalle del producto al que se desea asociar una nueva categoría. Para esta prueba se utilizó el producto "Product b". Este producto se creó automáticamente con el lanzamiento del sistema inicial. Adicional, se asoció el producto a la categoría "Category C". La Figura 3.22 muestra la página de edición del producto y la ventana para la elección de la categoría a asociar.

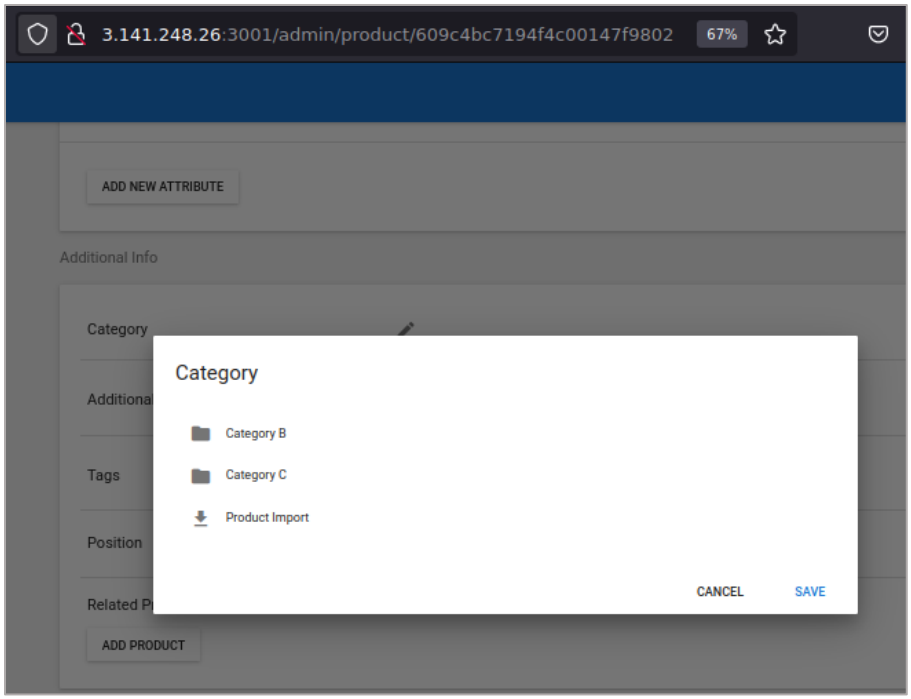


Figura 3.22 Ventana para la elección de la categoría a asociar con el cambio

Una vez realizada esta asociación se pudo verificar en el aplicativo web de clientes que el producto "Product B" se encuentra listado en la categoría "Category C". La Figura 3.23 muestra la página de la categoría "Category C" en el aplicativo web de clientes con el producto "Product B" listado correctamente.

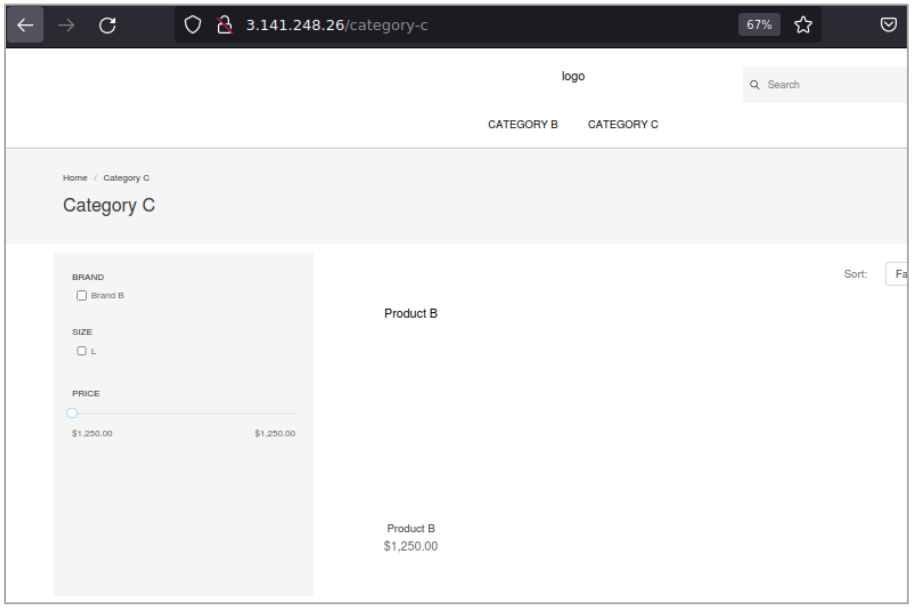


Figura 3.23 Ejemplo de categorías y productos listados

CONSULTA DE ÓRDENES

Se pueden consultar las ordenes realizadas por un usuario, haciendo uso del aplicativo web administrativo. Para eso se usó la ruta "/admin/orders". El aplicativo pone a disposición del usuario administrador una lista de órdenes en una tabla que incluye la siguiente información:

- identificador de orden junto con la fecha de la realización de la compra;
- usuario que realizó la compra;
- total del valor de la compra.

La Figura 3.24 muestra el listado de ordenes en el aplicativo administrativo al momento de la realización de esta prueba.

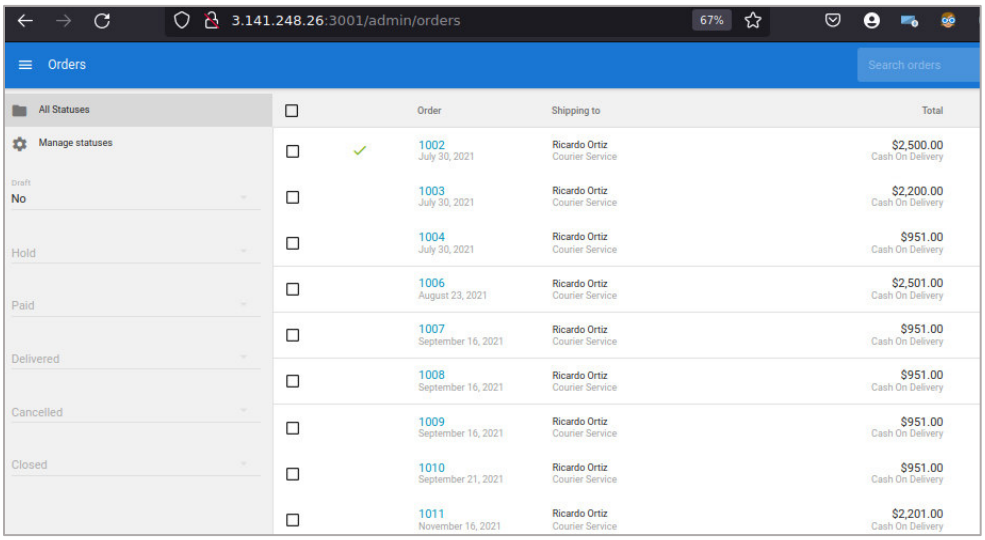


Figura 3.24 Listado de ordenes en el aplicativo administrativo

CONSULTA DE CLIENTES

Para consultar el listado de clientes que se han registrado en el sistema web, el aplicativo web pone a disposición la ruta "/admin/customers". La página del aplicativo web administrativo lista los clientes registrados junto con la siguiente información:

- nombre del usuario;
- número de ordenes;
- total gastado por el usuario en compras.

El total gastado por un cliente en compras se refiere a compras que están en el estado "Cerrado". El aplicativo administrativo para el listado de usuarios permite realizar un filtrado de clientes por

nombre. La Figura 3.25 muestra la lista de clientes al momento de realizar esta prueba. Se observan dos clientes que se han creado al momento, ambos con el mismo nombre. El primero ha realizado una compra y tiene un total gastado de cero dólares. El segundo ha realizado ocho compras y ha gastado un total de 2500 dólares.

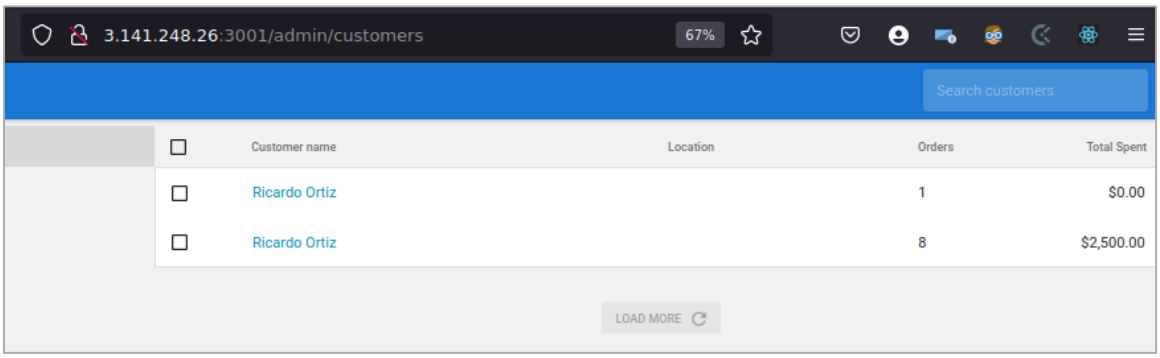


Figura 3.25 Ejemplo de lista de clientes

DEFINICIÓN DE TARJETA DE FIDELIZACIÓN Y ASIGNACIÓN A UN USUARIO DEL E-COMMERCE

La administración de tarjetas de fidelización se la realiza en el aplicativo administrativo de tarjetas de fidelización y se utilizó la URL: "http://3.141.248.26:3003". Para la configuración de una tarjeta de fidelización, primeramente, se define un premio que estará asociado a una tarjeta, de esta manera el cliente recibirá este premio si completa la tarjeta de fidelización. Para la administración de premios el aplicativo administrativo de tarjetas pone a disposición del administrador la ruta "/rewards". La primera tarjeta en esta página es un formulario para crear un premio. Para este evento de prueba se utiliza la siguiente información:

- Nombre: producto-b
- URL: URL del producto "Product B" del ecommerce. Esto indicará que el premio a recibir es el producto "Product B.
- Cantidad: 1

Con el fin de completar la creación de un registro de premio se utiliza el botón de la esquina superior derecha que tiene un símbolo de cruz. La Figura 3.26 muestra la página de premios del aplicativo administrativo de tarjetas que lista el nuevo premio con número de identificador 3.

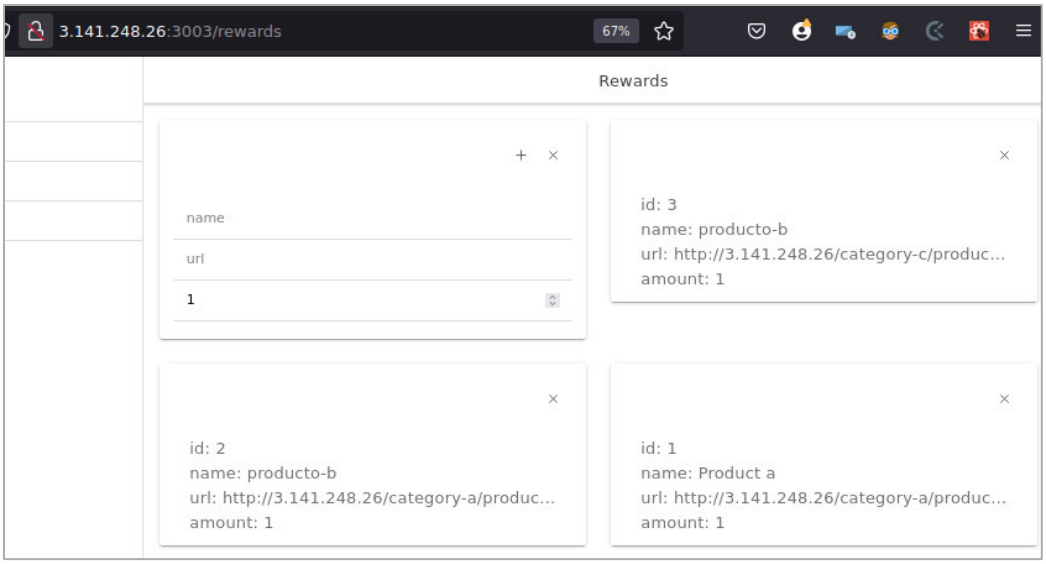


Figura 3.26 Página de premios del aplicativo administrativo

Luego, se debe definir un esquema de tarjeta. Para esta prueba se creó un esquema con la siguiente información:

- Título: esquema-prueba;
- Número de perforaciones: 2.

Una vez creado el esquema se utilizó el cuadro en el centro de la tarjeta asociada al esquema para asignar un premio al recientemente creado esquema. La Figura 3.27 muestra el esquema creado en el aplicativo administrativo de tarjetas para este evento de prueba. Este nuevo esquema tiene el identificador número 4.

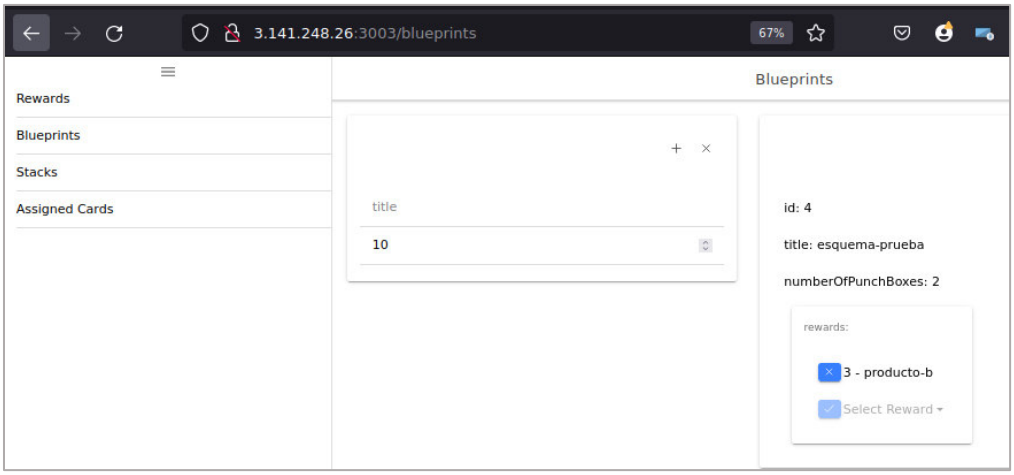


Figura 3.27 Ejemplo de esquema creado en el aplicativo administrativo

Luego de crear un esquema de tarjeta se crea una pila de tarjetas con este esquema. Para esto, el aplicativo administrativo de tarjetas pone a disposición la página en la ruta "/stacks". Para esta prueba se utilizó la siguiente información para crear una nueva pila de tarjetas:

- Título: pila-tarjetas-prueba;
- Número de tarjetas en la pila: 100;
- Esquema: Esquema con número de identificador: 4. Este corresponde al creado previamente en este proceso de prueba.

Una vez completado el formulario y creada la pila esta se puede visualizar junto a la tarjeta de formulario. La Figura 3.28 muestra la pila creada, con el número de identificador 4.

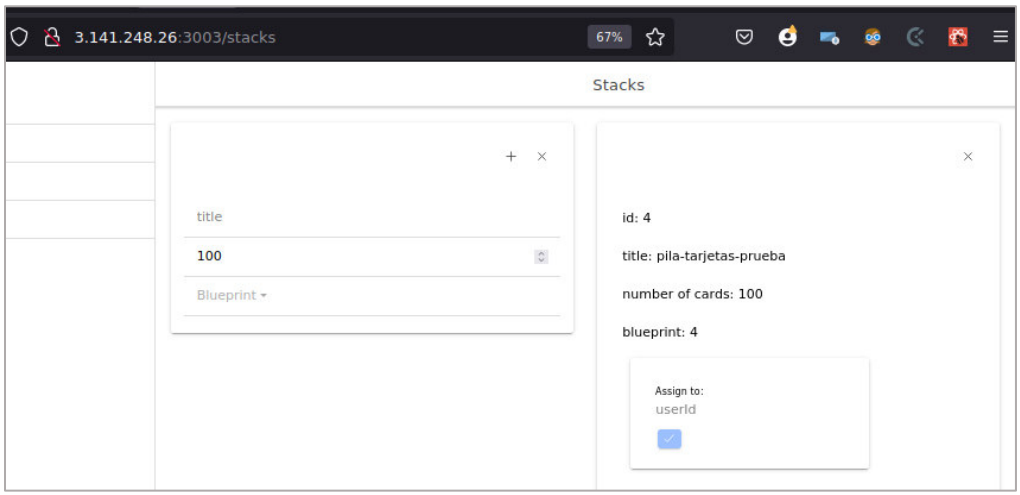


Figura 3.28 Ejemplo de pila creada

La asignación de tarjetas se hace desde la página de pilas de tarjeta. Para esto se utiliza el cuadro que se encuentra en la parte inferior del recuadro de una pila. En ese se debe incluir el identificador único de usuario. Para este escenario se utiliza el identificador del usuario en el e-commerce. Este identificador se lo encuentra en la página de detalle de usuario del aplicativo administrativo del e-commerce. La Figura 3.29 muestra la página para el usuario con identificador: 61941f1aec3fc900b0e37d22. Este identificador no se encuentra en la página, sino, que se lo puede observar como parte de la URL.

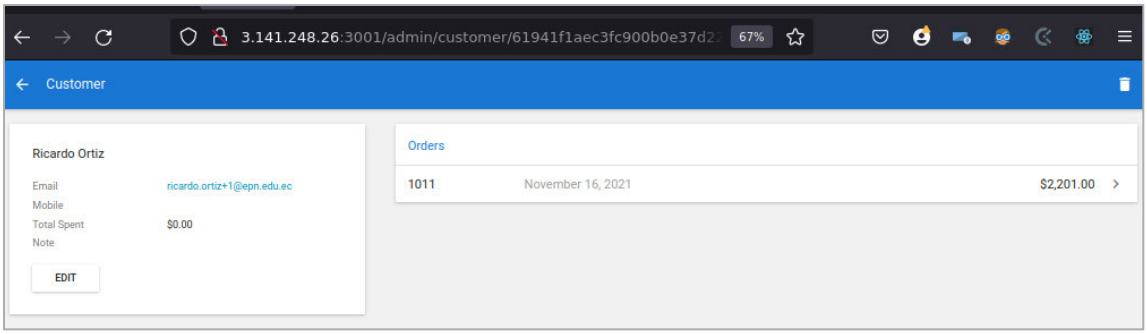


Figura 3.29 Ejemplo de página para Usuario

Con el identificador y utilizando el recuadro para asignación de tarjetas de una pila en el aplicativo administrativo de tarjetas se realizó la asignación de una tarjeta. La verificación de la asignación de la tarjeta se la puede realizar desde el aplicativo administrativo de tarjetas en la ruta "/assigned-cards". Esta página lista las tarjetas asignadas detallando información de la pila de origen, el usuario asignado, información de perforaciones y marcas de premios reclamados. La Figura 3.30 muestra la tarjeta asignada con número de identificador 7.

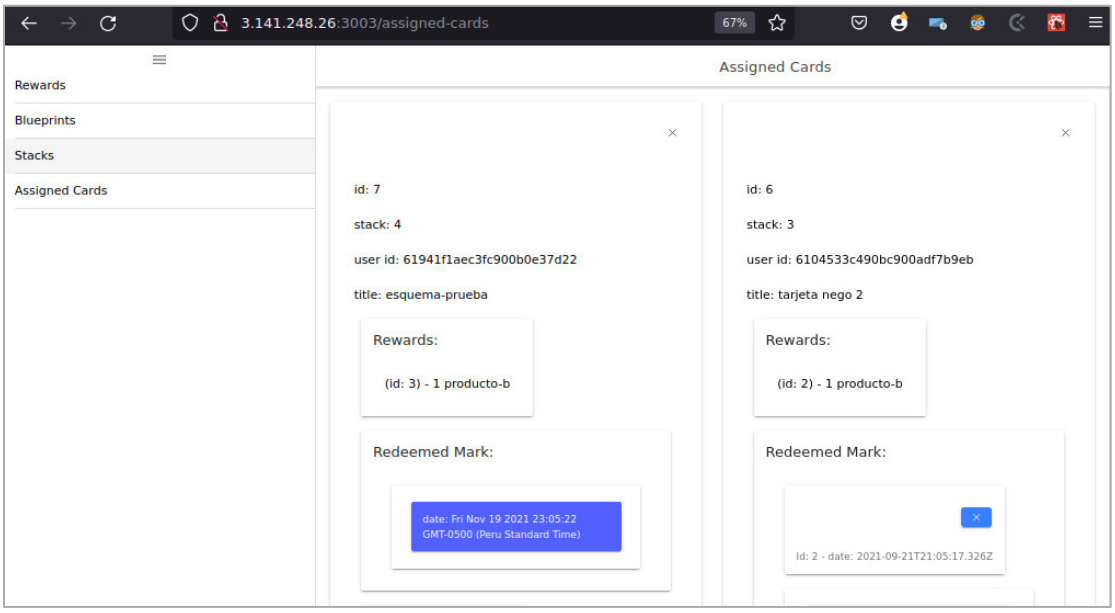


Figura 3.30 Ejemplo de tarjeta asignada

[redacted]REVISIÓN DE TARJETAS ASIGNADAS POR EL USUARIO CLIENTE DEL ECOMMERCE

Un usuario puede revisar desde el aplicativo de clientes las tarjetas que le han sido asignadas, a través del proceso de autenticación. Para esta prueba se utilizó el usuario creado previamente para el caso de prueba de autenticación. Luego de realizar el proceso de autenticación el

aplicativo mostrará la página con ruta "/customer-account", en esta se escoge la opción "Prizes" del menú respectivo. La Figura 3.31 muestra las tarjetas asignadas al usuario vistas desde el aplicativo web.

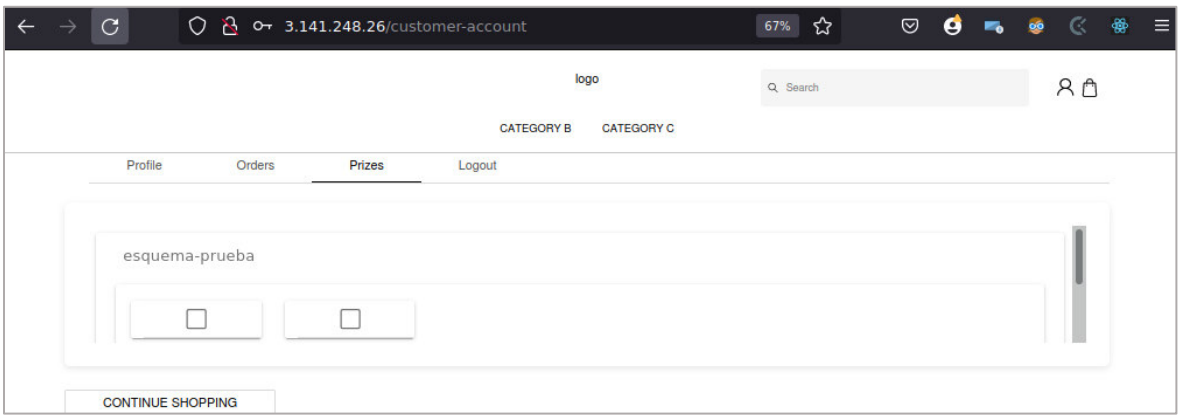


Figura 3.31 Tarjetas asignadas al Usuario

[redacted]COMPRA RESULTA EN AGREGAR PERFORACIÓN A TARJETA

Al realizar una compra se debe verificar que, si el usuario tiene una tarjeta de fidelización, esta se marque con una perforación. Para esta prueba se utilizó el usuario con correo electrónico: ricardo.ortiz+1@epn.edu.ec utilizado previamente en este evento de prueba. Se completo la orden con información de método de entrega y pago. Luego se verificó que la tarjeta ha recibido una perforación desde el aplicativo web de clientes. La Figura 3.32 muestra la ruta "/customer-account", con el panel de premios seleccionado. En este, se visualiza que una marca se ha agregado a la tarjeta.

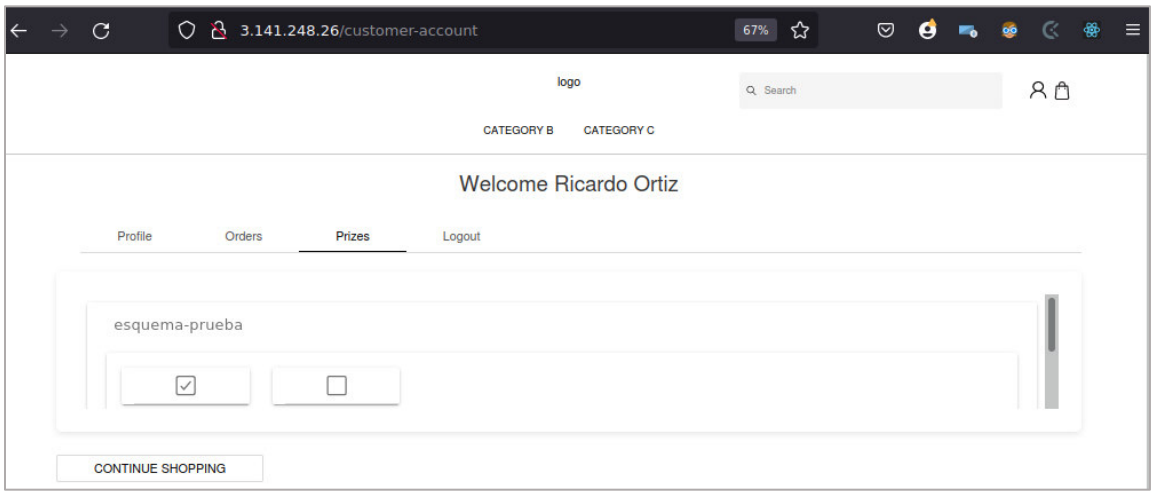


Figura 3.32 Ejemplo de tarjeta perforada

COLOCACIÓN DE MARCA DE PREMIOS REDIMIDOS A TARJETA DE FIDELIZACIÓN

El sistema permite marcar una tarjeta de fidelización con una marca que documente que el premio asociado a la tarjeta ha sido ya entregado. Esto se lo realiza desde el aplicativo administrativo del sistema de tarjetas de fidelización, específicamente en la ruta "assigned-cards". Aquí, inicialmente se ubica la tarjeta completa que se desea marcar. Luego, para marcar se utiliza el botón del recuadro "Redeemed Mark". Para esta prueba se utilizó la tarjeta asignada previamente. Esta tiene el número de identificador único: 7. La Figura 3.33 muestra la tarjeta con la marca. Esta indica que ha sido agregada el 20 de Noviembre de 2021.

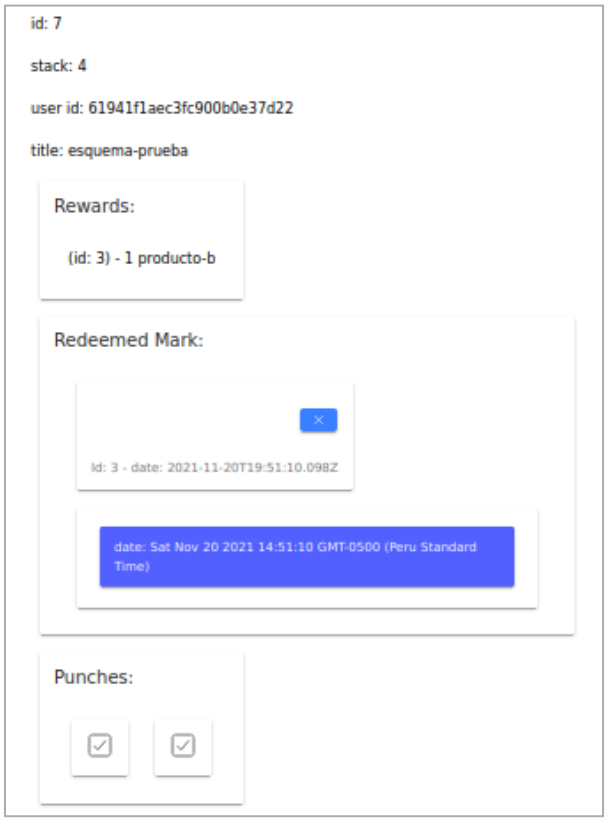


Figura 3.33 Ejemplo de tarjeta marcada

PRUEBAS ADICIONALES AL SISTEMA

Adicional a las pruebas definidas en el plan de pruebas, se realizan pruebas experimentales que ilustran funcionalidades de las tecnologías elegidas y permiten expandir la descripción del sistema. Estas pruebas no constituyen un proceso de validación y verificación. Se revisarán las funcionalidades de red y virtualización de Docker, así como también las opciones de red de Amazon AWS.

REDES CON CONTENEDORES DOCKER

La tecnología de virtualización Docker permite establecer redes virtuales entre los contendores administrados con este software. Para ilustrar algunas de las opciones de red de Docker en esta prueba se realiza lo siguiente:

- revisar la red virtualizada del sistema de fidelización integrado al e-commerce;
- verificar la comunicación entre contendores que no se encuentran en una misma red;
- crear una nueva red y un contenedor de prueba que se agrega a esta red;
- verificar que no existe comunicaciones entre contenedores que no están en la misma red.

En el ambiente de desarrollo se verificó las redes existentes con el comando "docker network ls". La Figura 3.34 muestra la respuesta de la consola a la ejecución de este comando.

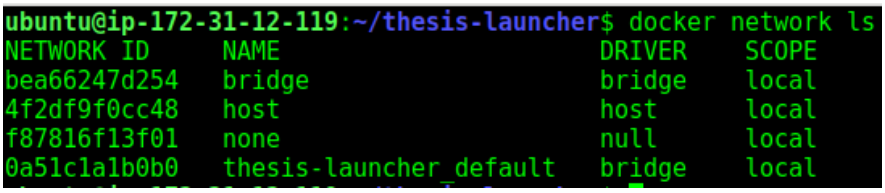


Figura 3.34 Respuesta de la consola al comando "docker network ls"

Al momento de esta prueba se listaron cuatro redes. Las redes: bridge, host son redes que se generan por defecto por Docker, mientras que "node" se refiere a los contenedores que no tienen una red asignada. La red "thesis_launcher_default" es la red que se genera para todos los contenedores definidos en el archivo de Docker Compose" de este proyecto. El nombre se construye con el directorio donde se encuentra el archivo de script que tiene el nombre: "thesis-launcher". Para revisar detalles sobre esta red se utiliza el comando: "docker inspect thesis_launcher_default". La Figura 3.35 y 3.36 muestran la respuesta de la consola en el ambiente de desarrollo al utilizar este comando.

```
[
    {
        "Name": "thesis-launcher_default",
        "Id": "0a51c1a1b0b03d6c00dfbfdad65956a19c4e9bc3aeb31610d45af23d9b3aa4ce",
        "Created": "2021-07-30T17:46:16.241842966Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.19.0.0/16",
                    "Gateway": "172.19.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": true,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "0374cae93f622ac581e9adf18897d214365491fc8fb0fc47dc08f0ba3ad3a1e2": {
                "Name": "thesis-launcher_db-pg_1",
                "EndpointID": "ea938305f4eea9899d73010adaba1ba0270da2a7598209e74e1df994b82e69b2",
                "MacAddress": "02:42:ac:13:00:02",
                "IPv4Address": "172.19.0.2/16",
                "IPv6Address": ""
            },
            "040eea689dab037ca2c06afab4f128d511cadeb4a50dd61d947b4c0cf951c26b": {
                "Name": "thesis-launcher_web_1",
                "EndpointID": "d5becf8a174baea27d0caec96fb907fe85f474ac96345091b505c1349f3b98ea",
                "MacAddress": "02:42:ac:13:00:09",
                "IPv4Address": "172.19.0.9/16",
                "IPv6Address": ""
            },
            "25bf4fe1f6ae9948887cd8d4e997939ecd3b30401acf98f18e7cd6093a79b47a": {
                "Name": "thesis-launcher_loyalty-cards-cezerin-middleware_1",
                "EndpointID": "c24a37b68b3d62dd62a7a4476248011ed0d5385da06a2cb7a19aaf29c3b66935",
                "MacAddress": "02:42:ac:13:00:0b",
                "IPv4Address": "172.19.0.11/16",
                "IPv6Address": ""
            },
            "46d3beb91f45751774e10efac84eb8aef03eeed7145848086fe248be01af6302": {
                "Name": "thesis-launcher_loyalty-cards-web_1",
                "EndpointID": "26c91e1f12dbd5acb29f6ff9333a25712a12a6113328c047c176a0bc0b6cf022",
                "MacAddress": "02:42:ac:13:00:06",
                "IPv4Address": "172.19.0.6/16",
                "IPv6Address": ""
```


Figura 3.35 Respuesta al comando para listar redes en Docker

```
            "73b4433a5ff2092f6bb2db164cf02d46e5e0dc1c9b0cbf228b4ecc7d0b8a2ea7": {
                "Name": "thesis-launcher_admin_1",
                "EndpointID": "9cd1cf008d3a134e19cb652a790d5243cce14b973adc1e840c639dae892b1131",
                "MacAddress": "02:42:ac:13:00:0a",
                "IPv4Address": "172.19.0.10/16",
                "IPv6Address": ""
            },
            "9c9b548bb4564c5b39eba56e975ceb46d5c168c3363b45b45020bd9c54f2afde": {
                "Name": "thesis-launcher_smtp_1",
                "EndpointID": "780a40e6c4e36f25800276f52383dbd50ffdeb90b92c4e8cdd95a089cc9ee67a",
                "MacAddress": "02:42:ac:13:00:07",
                "IPv4Address": "172.19.0.7/16",
                "IPv6Address": ""
            },
            "adce550d3bb5e73fef61d5e4a9ad9456c46159de5873460e4aeb046217aab3f5": {
                "Name": "thesis-launcher_database_1",
                "EndpointID": "e57ae7a8ef250fffcf8d2f6846b8e3587c5cc80ed54f322b18c25090640de82d",
                "MacAddress": "02:42:ac:13:00:03",
                "IPv4Address": "172.19.0.3/16",
                "IPv6Address": ""
            },
            "ba9c5fd62456c32540fe36f08667b0b67f95a0859839ba62c7a7669f66bdc51f": {
                "Name": "thesis-launcher_loyalty-cards-admin_1",
                "EndpointID": "454ab7d2ffdfd7a7f49388709b621bbeb662f78047597aedc49157d841de7c7b",
                "MacAddress": "02:42:ac:13:00:04",
                "IPv4Address": "172.19.0.4/16",
                "IPv6Address": ""
            },
            "cad59423a191f28322fcb71f231c1747fab57c40d166a0694082eba20abbdcc9": {
                "Name": "thesis-launcher_loyalty-cards_1",
                "EndpointID": "9dd4225c6615313a1da1c870efca3e05d5a1d41f50519999a497a83aad8f4428",
                "MacAddress": "02:42:ac:13:00:05",
                "IPv4Address": "172.19.0.5/16",
                "IPv6Address": ""
            },
            "e21faebcb4802bc1c84935f3e3614aded79fd162aa63e9949a344b7e9b39c885": {
                "Name": "thesis-launcher_ecommerce_1",
                "EndpointID": "40cdbdb44ad4c4c41e21a41fd679197779909e15586dc4aeddc4c77d89e11a13",
                "MacAddress": "02:42:ac:13:00:08",
                "IPv4Address": "172.19.0.8/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {
            "com.docker.compose.network": "default",
            "com.docker.compose.project": "thesis-launcher",
            "com.docker.compose.version": "1.29.1"
        }
    }
]
```


Figura 3.36 Respuesta al comando para listar redes en Docker

Notablemente se observa la subred creada para este sistema en el parámetro: "IPAM.Config.Subnet". IPAM es un acrónimo para Administración de direcciones IP (IP Address Management). Para esta prueba la subred está identificada con 172.19.0.0/16. El valor de la puerta de entrada está identificado con la dirección 172.19.0.1. Este comando también lista las direcciones IP y Mac de cada instancia virtual. Por ejemplo, la instancia del e-commerce en esta prueba se observó que tiene asignada la IPv4 y Mac: "172.19.0.8/16" y "02:42:ac:13:00:08" respectivamente, la instancia del aplicativo web de clientes por otro lado tiene asignada la IPv4 "172.19.0.6/16" y la Mac "02:42:ac:13:00:06".

Dado que los contenedores están dentro de la misma subred pueden descubrirse. Para ilustrar esto se utilizará la herramienta ping para verificar comunicación entre los contenedores correspondientes al aplicativo web de clientes del e-commerce y el contenedor del servidor web del e-commerce. Para esto, primeramente, se ejecutó el comando: "docker exec -it thesis-launcher_web_1 /bin/bash" para tomar control de la consola del contenedor del aplicativo web del e-commerce. Luego se instala la herramienta ping utilizando el comando "apt-get update && apt-get install iputils-ping". Se completó la instalación confirmando con la letra "y" la pregunta de la consola de instalar la herramienta. Luego, se utilizó la herramienta ping con el siguiente comando: "ping 172.19.0.8". La Figura 3.37 muestra la respuesta de la consola al correr este comando en el ambiente de desarrollo en la instancia en la nube.

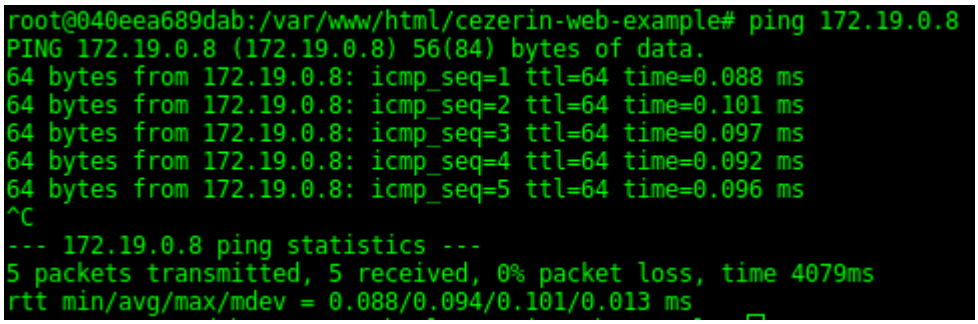


Figura 3.37 Respuesta de la consola al comando "ping 172.19.0.8"

Para mostrar el caso de contenedores en distintas redes, se creó un nuevo contendor y se lo configuró de manera que esté en una red distinta a los contenedores del sistema de fidelización. Para esta prueba se creó un contendor utilizando el comando "docker run --network=bridge --name contenedor-prueba -dit ubuntu". Este comando de Docker utiliza la función "run" que crea un contenedor. Las opciones utilizadas fueron "network=bridge" para asignar la red bridge al nuevo contenedor, "--name contenedor-prueba" para asignarle el nombre de "contenedor-prueba y "-dit" que corresponde a las opciones "d", "i" y "t" en conjunto que permiten liberar la consola,

asignar una psuedo-consola de control y mantener un proceso interactivo permanente, respectivamente. Finalmente, el argumento "ubuntu" del comando indica que el contenedor deberá utilizar la imagen de ubuntu más actual. Para listar la configuración de este nuevo contenedor se puede utilizar el comando "docker inspect contenedor-prueba". El Código 3.38 muestra parte de la respuesta de la consola de la instancia en la nube al correr dicho comando.


```
...
"NetworkSettings": {
            "Bridge": "",
            "SandboxID": "55bb143586e42691ee4801502e454fd175344ed191e27741c702cf2de6173a58",
            "HairpinMode": false,
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "Ports": {},
            "SandboxKey": "/var/run/docker/netns/55bb143586e4",
            "SecondaryIPAddresses": null,
            "SecondaryIPv6Addresses": null,
            "EndpointID": "ac3050d2548128706a41c2c246d75a008a444d946fe6ac55ba72707b902890ce",
            "Gateway": "172.17.0.1",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "IPAddress": "172.17.0.2",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "MacAddress": "02:42:ac:11:00:02",
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID": "bea66247d254f7a2ef92c67f53118db5e2da0e9c35cb7ba3e10fb2ace81196fb",
                    "EndpointID": "ac3050d2548128706a41c2c246d75a008a444d946fe6ac55ba72707b902890ce",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.2",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:02",
                    "DriverOpts": null
                }
            }
        }
```


Figura 3.38 Parte de la respuesta al comando "docker run --network=bridge --name contenedor-prueba -dit ubuntu"

La respuesta es un objeto JSON que en su parámetro "NetworkSettings" incluye información de red del contenedor. Se observa que la IP asignada para esta instancia es 172.17.0.2/16. Observando la IP y la máscara de red se determina que Docker ha colocado esta instancia en una subred distinta a la de las instancias del sistema de fidelización, siendo 172.17.0.0/16 para el contenedor de prueba y 172.19.0.0/16 para los contenedores del sistema de fidelización. Por

esta configuración se espera que no exista comunicación entre el contenedor "contenedor-prueba" y un contenedor del sistema de fidelización. Con el fin de verificar esto se utilizó la herramienta "ping" desde el contenedor del aplicativo web del e-commerce, para lo cual, primeramente, se ejecuta el comando "docker exec -it thesis-launcher_web_1 /bin/bash" de manera que tengamos a disposición una consola que nos permita ejecutar comando desde el contenedor. La herramienta "ping" ya fue instalada previamente por lo que este paso se omite. Luego, se corre el comando "ping 172.17.0.2". La Figura 3.39 muestra la respuesta obtenida al correr este comando. Se puede observar que todos los paquetes se han perdido por lo que se verifica que no existe comunicación entre estas dos instancias virtuales.

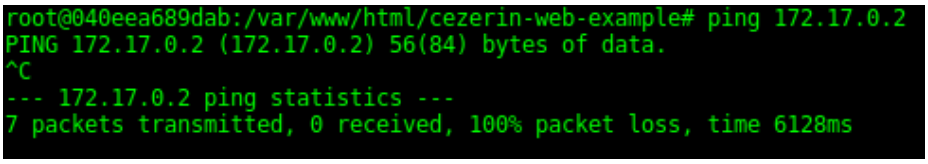


Figura 3.39 Respuesta al comando "ping 172.17.0.2"

CONFIGURACIONES DE GRUPOS DE SEGURIDAD DE AMAZON

Amazon permite realizar configuraciones de red, una de estas es el bloqueo de comunicación con la instancia usando ciertos puertos discriminando por dirección IP de origen. Esto se lo puede realizar por medio de los grupos de seguridad. Para ilustrar este funcionamiento se bloqueó el acceso a la instancia del ambiente de prueba para el puerto 3001 a cualquier IP de origen distinta a "200.7.247.74". El puerto 3001 corresponde al aplicativo web administrativo del e-commerce. La IP pública de la instancia física utilizada para esta prueba es "190.110.43.76". Con la configuración propuesta se espera que la instancia física no pueda establecer comunicación con la instancia en la nube en el puerto 3001.

Para realizar la configuración en la instancia nube se utilizó la consola provista por Amazon y se utilizó el menú de servicios para seleccionar EC2. Luego, en el menú lateral izquierdo se elige la opción "Security Groups" del sub-menú "Network and Security". Para esta prueba, primeramente, se navegó al panel de grupos de seguridad según descrito anteriormente y se modificó el grupo de seguridad con id: sg-0910d7b6a89861973. La Figura 3.40 muestra las reglas asociadas al grupo de seguridad con id: sg-0910d7b6a89861973 luego de la modificación a la configuración de acceso para el puerto 3001.

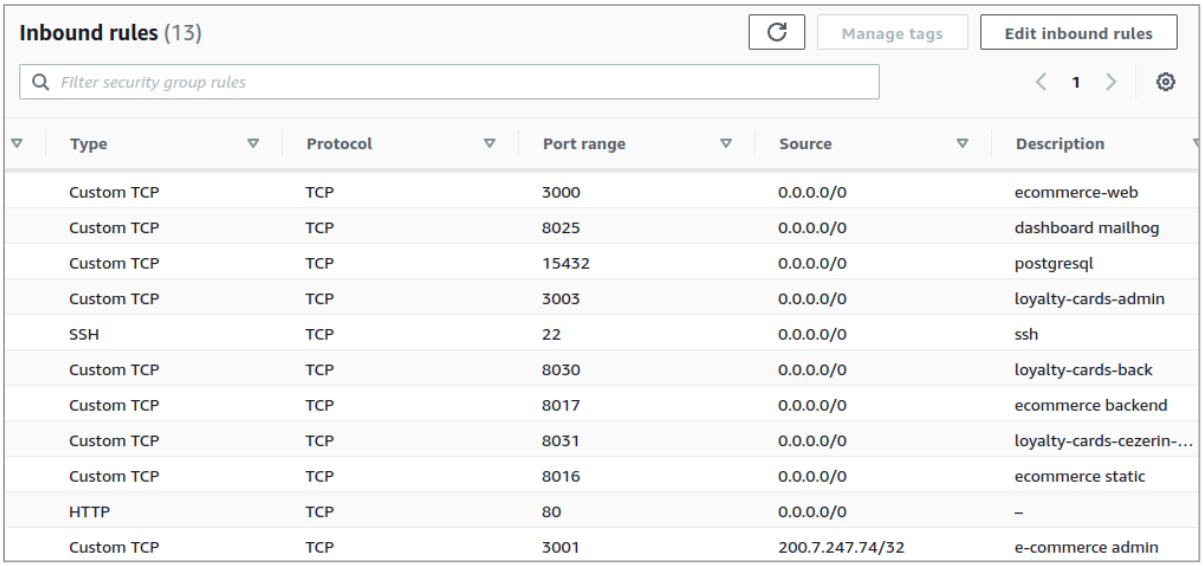


Figura 3.40 Reglas asociadas al grupo de seguridad

Una vez realizado el cambio se verifica la comunicación con la instancia en la nube en el puerto 3001 utilizando la herramienta "telnet". Para esto se corre desde la instancia física el comando "telnet 3.141.248.26 3001". La Figura 3.41 muestra la respuesta de la consola al ejecutar este comando en la consola de la instancia física.

```
ricardo@riunturi:~/riu-fs-2019/home-tori$ telnet 3.141.248.26 3001
Trying 3.141.248.26...
telnet: Unable to connect to remote host: Connection timed out
```

Figura 3.41 Respuesta de la consola al comando "telnet 3.141.248.26 3001"

Para finalizar la prueba se regresa a la configuración anterior donde no existen restricciones de IP de origen y se repite el comando con la herramienta telnet: "telnet 3.141.248.26 3001". La Figura 3.42 muestra la respuesta de la consola de este comando con la nueva configuración del grupo de seguridad.

```
ricardo@riunturi:~/riu-fs-2019/home-tori$ telnet 3.141.248.26 3001
Trying 3.141.248.26...
Connected to 3.141.248.26.
Escape character is '^]'.

^]
telnet>
```

Figura 3.42 Respuesta al comando telnet luego del cambio a la regla de seguridad

4. CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

- Utilizando en conjunto la tecnología de virtualización Docker, los servicios para computación en la nube AWS, y las tecnologías de programación JavaScript, Node, React, Ionic y NestJS se pudo construir un prototipo de un sistema de fidelización para e-commerce. De esta manera este proyecto constituye una referencia para el uso de estas tecnologías en el desarrollo de aplicativos de capa de aplicación.
- Se realizó satisfactoriamente la integración del sistema de fidelización construido con un sistema de e-commerce de código abierto utilizando el estándar REST y el protocolo de comunicación HTTP. REST y HTTP permitieron realizar sistemas independientes desacoplados que utilizan tecnologías distintas y que, sin embargo, pueden comunicarse utilizando un protocolo de comunicación común. Esto evidencia la capacidad del sistema de fidelización construido de integrarse con sistemas de e-commerce que implementen estos estándares comunes de red.
- El análisis de los e-commerce y de los sistemas de fidelización permitió describirlos en el contexto actual y reveló el estado actual de los comercios electrónicos en el Ecuador. El ambiente es favorable para su desarrollo y crecimiento. Plataformas de e-commerce se han impuesto en popularidad a plataformas de entretenimiento y se proyecta que supere al comercio tradicional.
- El contexto de la pandemia de Covid-19 ha acelerado aún más el uso de plataformas virtuales. En específico el e-commerce cuyo crecimiento sostenido ha sido inexorable ha sido acelerado en los últimos tiempos y se proyecta tenga más crecimiento en el futuro.
- Los sistemas de fidelización se han visto favorecidos por la mayor recolección de datos y capacidad de analizarlos. Esto permite tener sistemas más sofisticados y direccionados al usuario destino. Este nuevo contexto es un producto directo del desarrollo de tecnologías de información y computación. Los sistemas de fidelización son populares y constituyen una funcionalidad importante en los sistemas de e-commerce.
- El análisis de algunas de las más populares tecnologías de virtualización, de y computación en la nube y de programación permitió describir el rol de las mismas en la construcción de aplicativos y servidores de red. JavaScript es un lenguaje de programación que permite construir aplicativos y servidores utilizando la programación

orientada a objetos, funcional y asíncrona. La tecnología Node permite utilizar el lenguaje JavaScript en el lado del servidor, que originalmente era únicamente utilizado por los exploradores web. React es una librearía para construir interfaces visuales. NestJS y Ionic son frameworks que incluyen herramientas comunes que ponen a disposición de los desarrolladores. Algunas de estas herramientas son módulos, librerías y métodos para construir endpoints, acceder a bases de datos, dar estilos a bases de datos, asociar eventos a interfaces visuales, etc.

- La ejecución del plan de pruebas permitió verificar el correcto funcionamiento de los servicios del sistema de fidelización: crear, listar y borrar premios redimidos a una tarjeta asignada a un usuario.
- Pruebas adicionales al sistema configurando redes de Docker permitieron comprobar la funcionalidad de redes virtuales de Docker que sirve para establecer redes virtuales entre las instancias también virtuales. Se comprobó que instancias dentro de una misma subred virtual pueden establecer comunicación y las redes que no están en una misma subred, en cambio, no pueden hacerlo.
- Pruebas adicionales al sistema con la herramienta Telnet y configuraciones del servicio de grupos de seguridad de Amazon permitieron revelar la capacidad de controlar el tráfico a instancias de la nube y discriminar por puerto de destino (en la instancia) y dirección IP de origen. Se comprobó que sin establecer bloqueos una instancia local puede, usando Telnet, establecer comunicación con un puerto determinado de la instancia en la nube. Con el bloqueo, sin embargo, la comunicación a ese puerto no es posible y al intentar establecerla con Telnet, esta herramienta indicará un error.

RECOMENDACIONES

- Para modificaciones de mantenimiento explorar GraphQl, una tecnología que está creciendo en popularidad en el desarrollo web backend. Frameworks modernos como NestJS ofrecen soporte para GraphQl.
- Para actualizar el sistema se debe utilizar el motor de compilación. Deno es una tecnología desarrollada por el creador de NodeJs que ofrece mejoras en seguridad y soporte nativo de Typescript, entre otras mejoras y funcionalidades adicionales.
- Para casos donde se requiere alta escalabilidad de la infraestructura en la nube se recomienda utilizar el servicio de AWS: Elastic Container Registry (ECR). ECR es un repositorio de imágenes de Docker que permite el almacenamiento, administración y despliegue de las mismas. Presenta mayor facilidad de escalabilidad que EC2, aunque a un costo mayor. Este servicio se puede usar en conjunto con Elastic Container Service (ECS) y Fargate.
- Para el desarrollo por equipos de programadores se recomienda utilizar un sistema de CI/CD (Continuos Integration/Continuos Delivery) como CircleCI.
- Para utilizar el prototipo en situaciones donde se requiere aumentar la seguridad por ejemplo si se agrega integraciones con tarjetas de crédito se recomienda modificar la autenticación y agregar encripción con AES. Para la autenticación se recomienda usar servicios de proveedores externos como Firebase de Google. Para la encripción de información sensible como información de tarjetas de crédito o débito utilizadas para el pago se recomienda usar crypto-js.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] A. Meier and H. Stormer, *eBusiness an eCommerce*: *Managing the Digital Value Chain*, Springer, 2009.

[2] AWS, "General computing". AWS. https://aws.amazon.com/es/ec2/customers/. (accedido el 15 de junio de 2021).

[3] AWS, "Microservices". AWS. https://aws.amazon.com/es/containers/customers/. (accedido el 15 de junio de 2021).

[4] S. E. Ullah, *Developing an ecommerce website*. Durgapur, MicroCom, 2016

[5] UESS; CECE, "Comportamiento de la transacciones no presenciales en Ecuador". CECE. https://cece.ec/wp-content/uploads/2021/04/PresentacionMedicionEcommerce2020-UEES-04MAYO2020.pdf. (accedido el 15 de junio de 2021).

[6] Superintendencia de Bancos del Ecuador. "Datos de Servicios Financieros" Superintendencia de Bancos del Ecuador. http://estadisticas.superbancos.gob.ec/portalestadistico/portalestudios/?page_id=1826. (accedido el 15 de junio de 2021).

[7] UESS; CECE. "Transacciones electrónicas en Ecuador durante el Covid-19". CECE. https://cece.ec/. (accedido el 15 de diciembre de 2020).

[8] Kantar. "COVID-19 Barometer: Consumer attitudes, habits and expectations revisited," 14 Mayo 2021. https://www.kantar.com/inspiration/coronavirus/covid-19-barometer-consumer-attitudes-habits-and-expectations-revisited. (accedido el 10 de julio de 2021).

[9] Decode Economic & Financial Consulting, "Decoding the economics of COVID-19," 2020. https://dcodeefc.com/infographics. (accedido el 8 de junio de 2021).

[10] Paharia and Rajat. *Loyalty 3.0: How to Revolutionize Customer and Employee Engagement with Big Data and Gamification*, New York: McGraw-Hill Education, 2013.

[11] N. Poulton, *Docker Deep Dive*, Lean Publishing, 2018.

[12] Stack Overflow. "2020 Developer Survey". Stack Overflow. https://insights.stackoverflow.com/survey/2020. (accedido el 12 de julio de 2021).

[13] D. Flanagan. *JavaScript. The definitive guide. Master the World's Most-Used Programming Language*, Sebastopol: O'Reilly Media, 2020.

[14] F. Zammetti. *Moderm full-stack development. Using TypeScript, React, Node.js, Webpack, and Docker*, Pottstown: Apress Media LLC, 2020.

[15] NestJS. "Introduction". NestJS. https://docs.nestjs.com/. (accedido el 15 de junio de 2021).

[16] Docker. "Best practices for writing Dockerfiles". Docker https://docs.docker.com/develop/develop-images/dockerfile_best-practices/. (accedido el 11 de junio de 2021).

[17] A. Boduch and R. Derks, *React and react native*, Birmingham: Packt, 2020.

[18] Ionic. "Ionic Framework". Ionic. https://ionicframework.com/docs. (accedido el 15 de julio de 2021).

[19] The Institute of Electrical and Electronics Engineers, Inc., *IEEE Standard for Software and System Test Documentation*, New York: The Institute of Electrical and Electronics Engineers, Inc., 2008.

[20] K. Naik and P. Tripathy, *Software testing and quality assurance*, New Yersey: John Wiley & Sons, Inc. , 2008.

[21] J. F. Kurose and K. Ross, *Computer Networking. A Top-Down Approach*, New Jersey: Pearson, 2017.

[22] Cezerin. "FAC". Cezerin. https://cezerin.org/docs/#/faq. (accedido el 15 de julio de 2021).

[23] Cezerin. "Ecommerce Progressive Web Apps". Cezerin. https://cezerin.org/. (accedido el 10 de junio de 2021).

[24] B. Piper and D. Clinton, *AWS Certified Solutions Architect*, Indianapolis: Jhon Wiley & Sons, 2019.

[25] A. Stellman and J. Greene. *Learning Agile. Understanding Scrum, Xp, Lean, and Kanban*, Sebastopol: O'Reilly Media, Inc., 2015.

[26] K. Schwaber and J. Sutherland. *Scrum: Developed and sustained*, Scrum, 2010.

[27] M. Rehkopf. "User stories with examples and a template". Atlassian. https://www.atlassian.com/agile/project-management/user-stories. (accedido el 9 de junio de 2021).

[28] The PostgreSQL Global Development Group. "PostgreSQL 13.5 Documentation". The PostgreSQL Global Development Group. https://www.postgresql.org/docs/13/index.html. (accedido el 15 de junio de 2021).

[29] Vamcart. "Cezerin is React and Node.js based eCommerce platform". React Shopping Cart. https://github.com/Cezerin2/cezerin2#readme. (accedido el 15 de junio de 2021).

[30] G. Edwards, "Documentation. MailHog". MailHog. https://github.com/mailhog/MailHog. (accedido el 15 de junio de 2021).

[31] MongoDB, "Build faster. Build smarter". MongoDB. https://www.mongodb.com/. (accedido el 15 de junio de 2021).

[32] Z. Yung, "MailHog Tutorial". MailHog. https://mailtrap.io/blog/mailhog-explained/.(accedido el 15 de junio de 2021).

[33] Internet World Stats, "World Internet Users and Population Stats". Internet World Stats. https://www.internetworldstats.com/stats2.htm. (accedido el 14 de junio de 2021).

ANEXOS

ANEXO A. Diseño del servidor web de fidelización usando open API

ANEXO B. Diseño aplicativo web administrativo de tarjetas de fidelización

ANEXO C. Diseño aplicativo web de clientes de tarjetas de fidelización

ANEXO D. Diseño del servidor web del middleware para el sistema de fidelización y cezerin

ANEXO E. Plan de pruebas

ANEXO F. Código fuente

ANEXO A

DISEÑO DEL SERVIDOR WEB DE FIDELIZACIÓN USANDO OPEN API

```yaml
openapi: 3.0.0
info:
  version: 1.0.0
  title: loyalty-cards
servers:
  - url: 'http://localhost:3000'
  - url: 'http://192.168.0.33:3000'
paths:
  /:
    description: 'Heartbeat'
    get:
      tags:
        - heartbeat
      summary: 'Get heartbeat to check system health'
      responses:
        '200':
          description: 'Successful heartbeat'
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Heartbeat'

  /rewards:
    get:
      tags:
        - rewards
      summary: 'List reward options'
      responses:
        '200':
          description: 'Success'
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Reward'


  post:
    tags:
      - rewards
    summary: 'Create reward option'
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Reward'
    responses:
      '200':
        description: 'Success'
        content:
```

```
        application/json:
          schema:
            $ref: '#/components/schemas/Reward'
    responses:
      '200':
        description: 'Success'
        content:
          application/json:
            schema:
              $ref:  "#/components/schemas/Success"

/rewards/{id}:
  put:
    tags:
      - rewards
    summary: 'Edit reward option'
    parameters:
      - name: id
        required: true
        in: path
        schema:
          type: integer
    requestBody:
      required: true
      content:


        application/json:
          schema:
            $ref: '#/components/schemas/Reward'
    responses:
      '200':
        description: 'Success'
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Success'
  delete:
    tags:
      - rewards
    summary: 'Delete reward options'
    parameters:
      - name: id
        required: true
        in: path
        schema:
          type: integer
    responses:
      '200':
        description: 'Success'
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Success'

/card-blueprints:
  get:
    tags:
```

```
    - card-blueprints
  summary: List card blueprints
  parameters:
    - name: id
      in: query
      description: "Blueprint's id'"

      required: false
      schema:
        type:
          integer
  responses:
    '200':
      description: success
      content:
        application/json:
          schema:
            type: array
            items:
              $ref: '#/components/schemas/CardBlueprint'
post:
  tags:
    - card-blueprints
  summary: Create new blueprint
  requestBody:
    required: true
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/CardBlueprint'
  responses:
    '200':
      description: 'Success'
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Success'

/card-blueprints/{id}:
  # put:
  #   tags:
  #     - card-blueprints
  #   summary: Edit blueprint


   #   requestBody:
   #     required: true
   #     content:
   #       application/json:
   #         schema:
   #           $ref: '#/components/schemas/CardBluePrint'
   #   parameters:
   #     - name: id
   #       in: "path"
   #       required: true
   #       schema:
   #         type: integer
```

```
#   responses:
#     '200':
#       description: success
#       content:
#         application/json:
#           schema:
#             $ref: '#/components/schemas/Success'

  delete:
    tags:
      - card-blueprints
    summary: Destroy blueprint
    parameters:
      - name: id
        in: "path"
        required: true
        schema:
          type: integer
    responses:
      '200':
        description: success
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Success'


/card-blueprint-to-rewards:
  get:
    tags:
      - card-blueprint-to-rewards
    summary: List blueprint by reward records
    responses:
      '200':
        description: success
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/CardBlueprintToReward'
  post:
    tags:
      - card-blueprint-to-rewards
    summary: Create blueprint by reward records
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/CardBlueprintToReward'
    responses:
      '200':
        description: 'Success'
        content:
          application/json:
            schema:
              $ref:  "#/components/schemas/Success"
```

```
/card-blueprint-to-rewards/{id}:
  # put:
  #   tags:

  #     - card-blueprint-to-rewards
  #   summary: 'Edit blueprint by reward record'
  #   parameters:
  #     - name: id
  #       in: query
  #       description: "Card Blueprint by Reward id'"
  #       required: true
  #       schema:
  #         type:
  #           integer
  #   requestBody:
  #     required: true
  #     content:
  #       application/json:
  #         schema:
  #           $ref: '#/components/schemas/CardBluePrintByReward'
  #   responses:
  #     '200':
  #       description: 'Success'
  #       content:
  #         application/json:
  #           schema:
  #             $ref: "#/components/schemas/Success"
  delete:
    tags:
      - card-blueprint-to-rewards
    summary: 'Delete blueprint to reward relation record'
    parameters:
      - name: id
        in: path
        description: "Card Blueprint to Reward id"
        required: true
        schema:
          type:
            integer

    responses:
      '200':
        description: 'Success'
        content:
          application/json:
            schema:
              $ref:  "#/components/schemas/Success"

  /card-stacks:
    get:
      tags:
        - card-stacks
      summary: List card stacks
      parameters:
        - name: id
          in: query
          required: false
```

```
        schema:
          type: integer
    responses:
      '200':
        description: success
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/CardStack'
  post:
    tags:
      - card-stacks
    summary: Create new cards stack from blueprint
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref:  "#/components/schemas/CardStack"


  delete:
    tags:
      - card-stacks
    summary: Destroy card stack
    responses:
      '200':
        description: Success
        content:
          application-json:
            schema:
              $ref:  "#/components/schemas/Success"

/assigned-cards:
  get:
    tags:
      - assigned-cards
    summary: List assigned-cards
    parameters:
      - name: id
        in: query
        required: false
        schema:
          type:
            integer

      - name: onlyFullyPunched
        in: query
        required: false
        schema:
          type: boolean
```

```
      - name: onlyNotFullyPunched
        in: query
        required: false
        schema:
          type: boolean

      - name: userId
        in: query
        required: false


        schema:
          type:
            string
    responses:
      '200':
        description: success
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/AssignedCard'

  post:
    tags:
      - assigned-cards
    summary: Create assigned card by assigning card from a stack to a user
    requestBody:
      required: true
      content:
        application-json:
          schema:
            $ref:  "#/components/schemas/AssignedCard"
    responses:
      '200':
        description: Success
        content:
          application-json:
            schema:
              $ref:  "#/components/schemas/Success"

/assigned-cards/{id}:
  # put:
  #   tags:
  #     - assigned-cards
  #   summary: Edit assigned card
  #   requestBody:
  #     required: true
```

```
#     content:
#       application-json:
#         schema:
#           $ref:  "#/components/schemas/AssignedCard"
#   responses:
#     '200':
#       description: Success
#       content:
#         application-json:
#           schema:
#             $ref:  "#/components/schemas/Success"

# patch:
#   tags:
#     - assigned-cards
#   summary: Edit assigned card
#   parameters:
#     - name: id
#       in: path
#       required: true
#       schema:
#         type: integer
#   requestBody:
#     required: true
#     content:
#       application-json:
#         schema:
#           $ref:  "#/components/schemas/AssignedCardPatchBody"
  # responses:
  #   '200':
  #     description: Success
  #     content:
  #       application-json:
  #         schema:
  #           $ref:  "#/components/schemas/Success"


delete:
  tags:
    - assigned-cards
  summary: Destroy assigned card
  parameters:
    - name: id
      in: path
      required: true
      schema:
        type: integer
  responses:
    '200':
      description: 'success'
      content:
        application-json:
          schema:
            $ref: '#/components/schemas/Success'
```

```yaml
/assigned-cards/{assignedCardId}/punches:
  get:
    tags:
      - assigned-card-punches
    summary: List assigned card's punches
    parameters:
      - name: assignedCardId
        required: true
        in: path
        schema:
          type: integer
    responses:
      '200':
        description: success
        content:
          application/json:
            schema:
              type: array
              items:
                $ref: '#/components/schemas/CardPunch'

  post:
    tags:
      - assigned-card-punches
    summary: "Punch assigned card of id: assignedCardId"
    parameters:
      - name: assignedCardId
        required: true
        in: path
        schema:
          type: integer
    # requestBody:
    #   content:
    #     application-json:
    #       schema:
    #         $ref: "#/components/schemas/CardPunch"
    #   required: true
    responses:
      '200':
        description: success
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Success'

/assigned-cards/punches/{punchId}:
  put:
    tags:
      - assigned-card-punches
    summary: "Edit punch of id: punchId"
    parameters:
      - name: punchId
        required: true
```

```
        in: path
        schema:
          type: integer
    requestBody:
      content:
        application/json:
          schema:
            $ref:  "#/components/schemas/CardPunch"
    responses:
      '200':
        description: success
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Success'
  delete:
    tags:
      - assigned-card-punches
    summary: 'Remove punch of id: punchId from its corresponding card'
    parameters:
      - name: punchId
        required: true
        in: path
        schema:
          type: integer
    responses:
      '200':
        description: Delete punch from card
        content:
          application-json:
            schema:
              $ref: '#/components/schemas/Success'

/assigned-cards/{assignedCardId}/redeemed-marks:
  get:
    tags:
      - assigned-card-redeemed-marks

    summary: List assigned card redeem marks
    parameters:
      - name: assignedCardId
        required: true
        in: path
        schema:
          type: integer
    responses:
      '200':
        description: Success
        content:
          application-json:
            schema:
              type: array
              items:
                $ref: "#/components/schemas/CardRedeemedMark"
```

```yaml
    post:
      tags:
        - assigned-card-redeemed-marks
      summary: Add redeem marks to assigned card
      parameters:
        - name: assignedCardId
          required: true
          in: path
          schema:
            type: integer
      requestBody:
        required: true
        content:
          application-json:
            schema:
              $ref: "#/components/schemas/CardRedeemedMark"
      responses:
        '200':
          $ref:  "#/components/responses/success"

  /assigned-cards/redeemed-marks/{redeemedMarkId}:
    delete:
      parameters:
        - name: redeemedMarkId
          in: path
          required: true
          schema:
            type: integer
      tags:
        - assigned-card-redeemed-marks
      summary: Remove redeemed mark from assigned card
      responses:
        '200':
          $ref:  "#/components/responses/success"

components:
  responses:
    'success':
      description: Success
      content:
        application-json:
          schema:
            $ref: "#/components/schemas/Success"


  # requestBodies:
  #   PatchAssignedCard:
  #     content:
  #       application-json:
  #         schema:
```

```yaml
schemas:
  Heartbeat:
    type: object
    properties:
      date:
        type: string
        format: date-time
      message:

        type: string
        example: 'beating'
  Reward:
    type: object
    required:
      - name
    properties:
      id:
        type: integer
        readOnly: true
        example: 1
        uniqueItems: true
      amount:
        type: integer
        default: 1
      name:
        type: string
        example: 'Some cool gift'
      url:
        type: string
        example: 'http://my-ecommerce/products/some-cool-gift'
  CardBlueprint:
    type: object
    required:
      - numPunchBoxes
    properties:
      id:
        type: integer
        readOnly: true
        uniqueItems: true
        example: 1
      title:
        type: string

    default: 'Loyalty card'
  numberOfPunchBoxes:
    type: integer
    format: int32
    example: 10
  rewards:
    type: array
    readOnly: true
    minItems: 1
    items:
      $ref: '#/components/schemas/Reward'
```

```yaml
CardBlueprintToReward:
  type: object
  required:
    - cardBlueprintId
    - rewardId
  properties:
    id:
      type: integer
      uniqueItems: true
      example: 1
      readOnly: true
    cardBlueprintId:
      type: integer
      uniqueItems: true
      example: 1
    rewardId:
      type: integer
      uniqueItems: true
      example: 1
CardStack:
  type: object
  required:
    - cardBlueprintId
    - numberOfCards

  properties:
    id:
      type: string
      uniqueItems: true
      readOnly: true
      example: 1
    cardBlueprintId:
      type: integer
      uniqueItems: true
      example: 1
    numberOfCards:
      type: integer
      example: 200
    title:
      type: string
      uniqueItems: true
      description: Title to identify stack
      example: 'New years 2021 loyalty cards'

# AssignedCard:
# AssignedCardPatchBody:

#   allOf:
#     - not:
#         required: [userId]
```

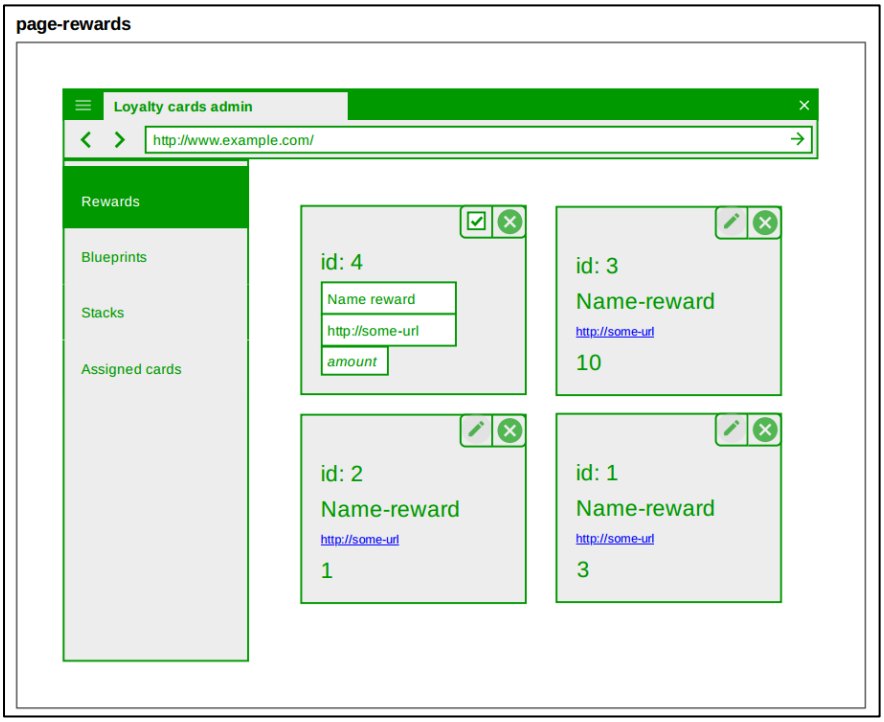
```
#         properties:
#           userId:
#             type: string
    # - type: object
    #   required:
    #     - cardStackId
    #     - userId
    #   properties:
    #     cardStackId:
    #       type: number
    #       example: 1
    #     userId:

    #       type: string
    # - $ref: "#/components/schemas/AssignedCard"
AssignedCard:
# AssignedCardPatchBody:
  type: object
  required:
    - cardStackId
    - userId
  properties:
    id:
      type: integer
      readOnly: true
      uniqueItems: true
      example: 1
    cardStackId:
      type: number
      example: 1
    userId:
      type: string
    # redeemed:
    #   type: boolean
    #   default: false
    title:
      type: string
      readOnly: true
    numberOfPunchBoxes:
      type: number
      readOnly: true
      example: 10
    rewards:
      type: array
      readOnly: true
      items:
        $ref:  "#/components/schemas/Reward"
    punches:
      type: array
      readOnly: true
      items:
```

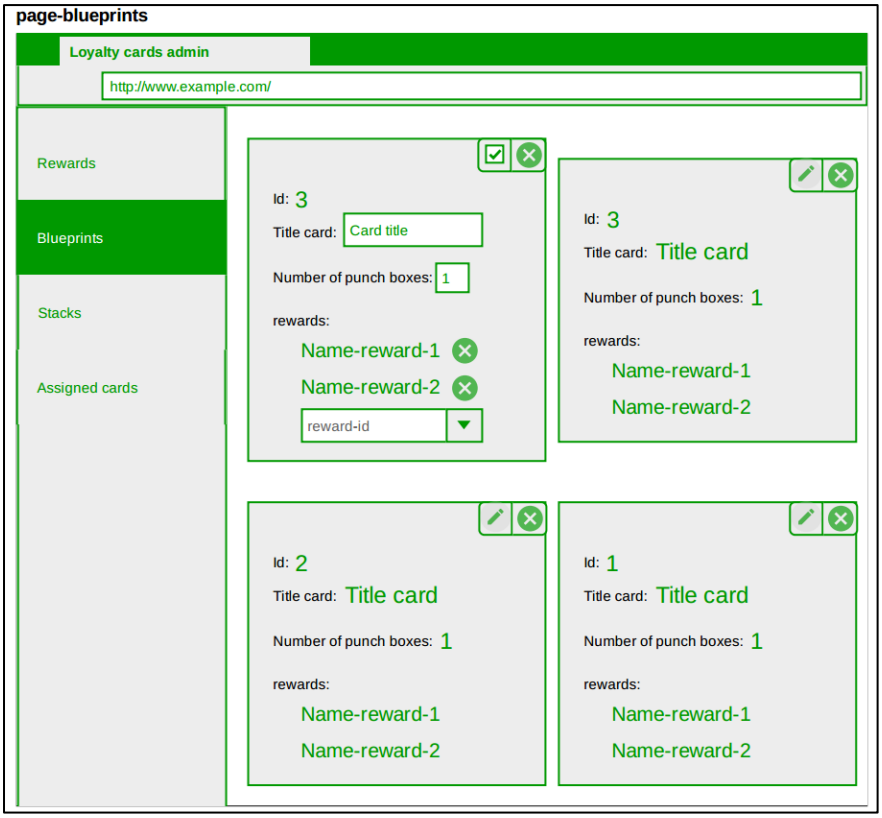
```
          $ref: '#/components/schemas/CardPunch'
      redeemedMarks:
        type: array
        readOnly: true
        items:
          $ref:  "#/components/schemas/CardRedeemedMark"

  CardPunch:
    type: object
    properties:
      id:
        type: integer
        uniqueItems: true
        readOnly: true
        example: 1
      date:
        type: string
        format: date-time

      assignedCardId:
        type: integer
        readOnly: true
        example: 1

  CardRedeemedMark:
    type: object
    required:
      - assignedCardId
      - note
    properties:
      id:
        type: integer
        readOnly: true
        example: 1
        uniqueItems: true


     date:
       type: string
       format: date-time

     assignedCardId:
       type: integer
       uniqueItems: true
       example: 1

     note:
       type: string
       example: "All rewards redeemed."
Error:
  type: object
```

```
      required:
        - message
      properties:
        code:
          type: integer
          format: int32
        message:
          type: string
          example: Error
    Success:
      type: object
      properties:
        message:
          type: string
          example: Success
  securitySchemes:
    BasicAuth:
      type: http
      scheme: basic

tags:
  - name: heartbeat
    description: System health
  - name: rewards
    description: "Reward: A reward option for completing a loyalty card"

  - name: card-blueprints
    description: "Card blueprint: used to create card stacks."
  - name: card-blueprint-to-rewards
    description: "Blueprint-reward associative records"
  - name: card-stacks
    description: "Card stack: created from blueprint. A card from the stack may be assigned to a user."
  - name: assigned-cards
    description: "Assigned Card: A card from a stack assigned to a user."
  - name: assigned-card-punches
    description: "A punch from a punch card (assigned card)"
  - name: assigned-card-redeemed-marks
    description: "Redeemed Mark: Card mark to indicate reward has been redeemed."
```

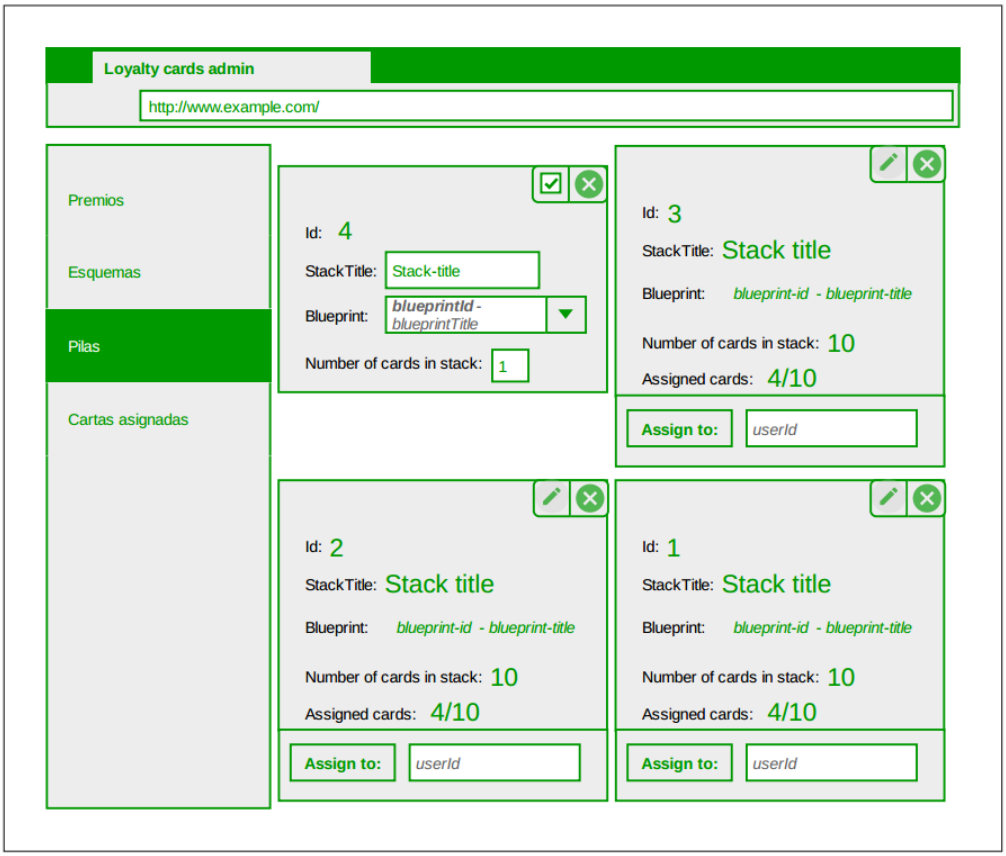
ANEXO B

DISENO DE APLICATIVO WEB ADMINISTRATIVO DE TARJETAS DE FIDELIZACIÓN

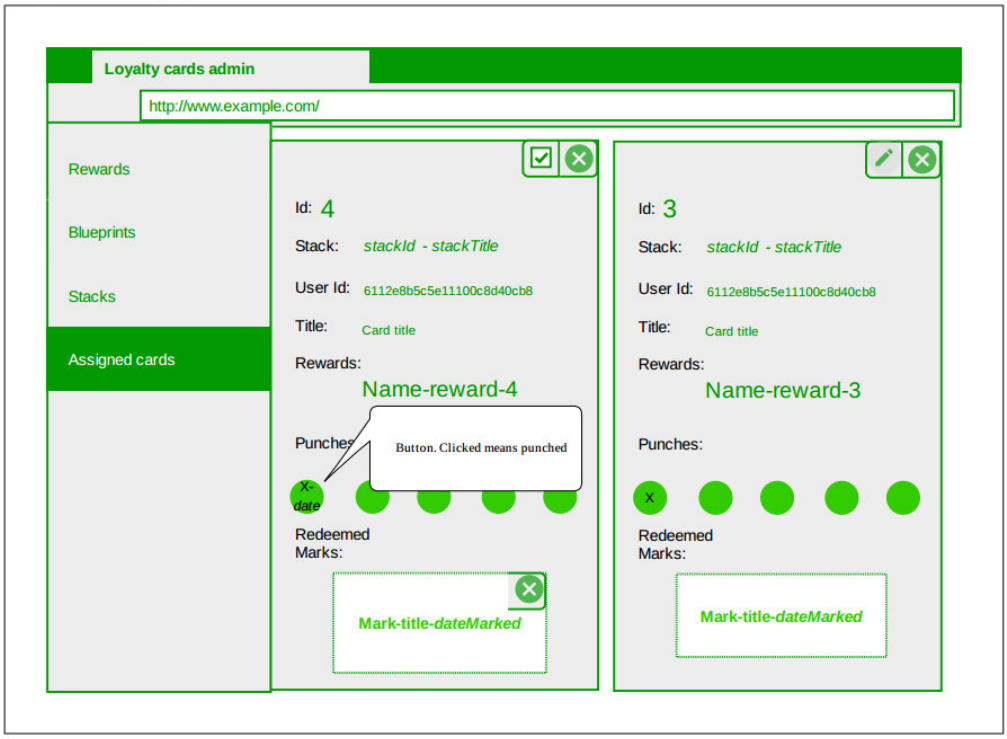




page-stack



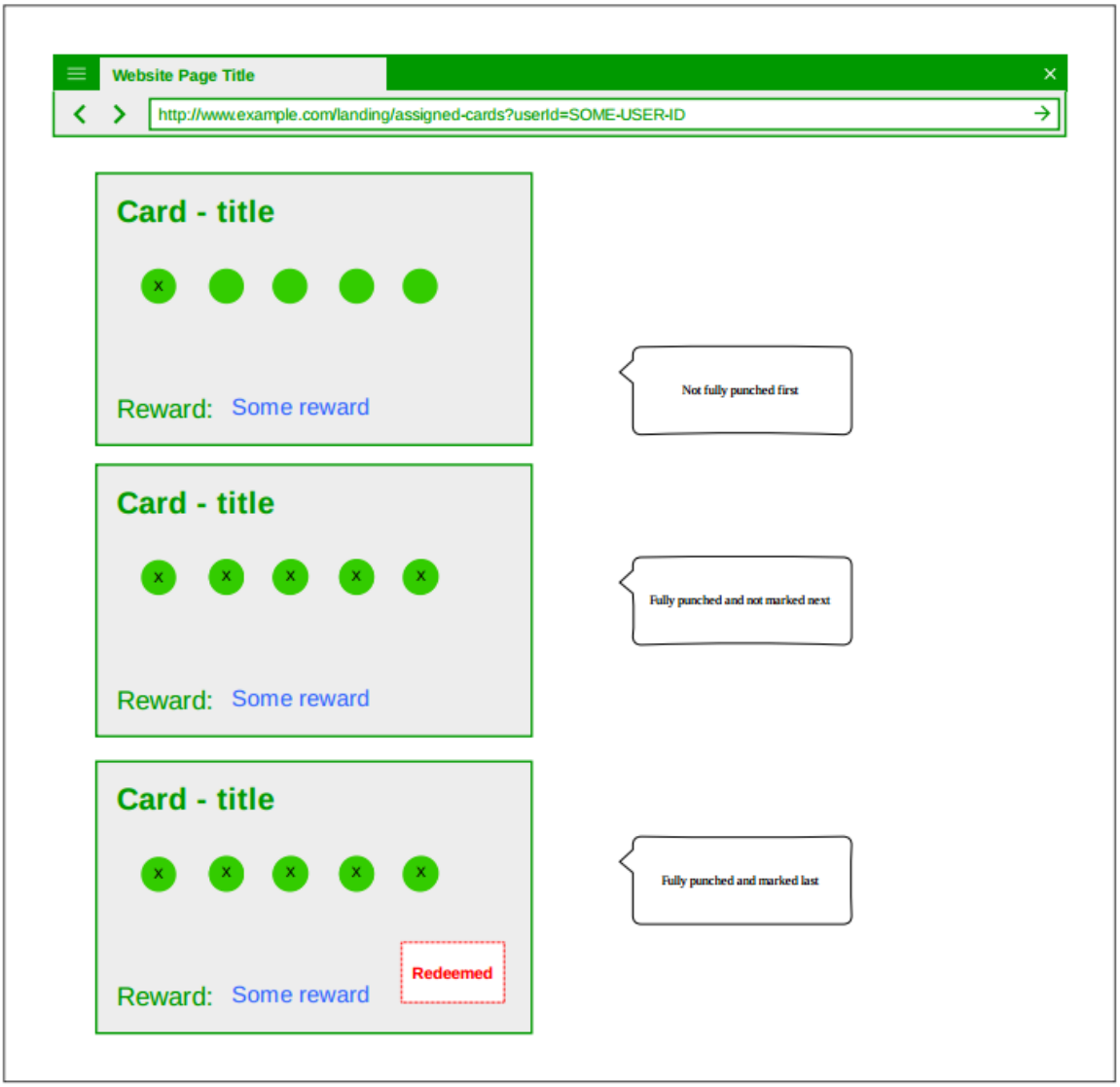
assigned-cards-page



ANEXO C

DISENO APLICATIVO WEB DE CLIENTES DE TARJETAS DE FIDELIZACIÓN

landing-loyalty-cards



ANEXO D

DISEÑO DEL SERVIDOR WEB DEL MIDDLEWARE PARA EL SISTEMA DE FIDELIZACIÓN Y CEZERIN

```yaml
openapi: 3.0.0
info:
  title: loyalty-cards-cezerin-middleware
  version: 1.0.0
components:
  requestBodies:
    CezerinOrder:
      content:
        application-json:
          schema:
            $ref: '#/components/schemas/CezerinOrder'
      description: Cezerin order.
      required: true
  responses:
    Success:
      description: Success
      content:
        application-json:
          schema:
            $ref:  "#/components/schemas/Success"

  schemas:
    CezerinOrder:
      description: Cezerin order
      type: object
      required:
        - id
      properties:
        id:
          type: string
          uniqueItems: true
          example: "6123c02ac1804600ab42f14d"
        customerId:
          type: string
          uniqueItems: true
          example: "6104533c490bc900adf7b9eb"

   PurchasePunchRequest:
     type: object
     properties:
       id:
         type: integer
         readOnly: true
         example: 1
         uniqueItems: true
       orderId:
         type: integer
       userId:
         type: integer
       punchedAssignedCardId:
```

```
  type: integer
loyaltyCardsLogId:
  type: integer
dateCreated:
  type: string
  format: date-time

 LoyaltyCardsLog:
   type: object
   properties:
     id:
       type: integer
     statusCode:
       type: integer
     response:
       type: object
     requestBody:
       type: object
     url:
       type: string


Success:
  type: object
  properties:
    message:
      type: string
      example: 'Success'

 paths:
   /punchers/purchase-punch-requests:
     # get:
     #   responses:
     #     '200':
     #       description: List of purchase punches.
     #       content:
     #         application-json:
     #           schema:
     #             type: array
     #             items:
     #               $ref: '#/components/schemas/PurchasePunch'
     #   tags:
     #     - punchers-purchase-punches
     #   summary: List purchase-punches
```

```
    post:
      requestBody:
        $ref:  "#/components/requestBodies/CezerinOrder"
      responses:
        '200':
          $ref: "#/components/responses/Success"
      tags:
        - punchers-purchase-punch-requests
      summary: Punch card as reward for purchase. Create a PurchasePunchRequest Object and a LoyaltyCardLog

servers:
  - url: 'http://localhost:8000'

tags:
  - name: punchers-purchase-punch-requests
    description: "Purchase punch request: a request to punch card ( if applies) as a reward for a purchase"
```

ANEXO E

PLAN DE PRUEBAS

1. Identificador Plan de Pruebas master-loyalty-cards-v1.0.0
2. Introducción

El propósito de este plan es definir las pruebas para el sistema de tarjetas de fidelización integrado al sistema de ecommerce Cezerin desplegado en infraestructura de Amazon.

3. Elementos de prueba

- Servidor web de e-commerce

- Aplicativo web de e-commerce para clientes

- Aplicativo web de e-commerce administrativo

- Servidor web de tarjetas de fidelización

- Aplicativo web de tarjetas de fidelización para clientes

- Aplicativo web de tarjetas de fidelización administrativo

Script de Docker Compose para lanzamiento del sistema de e-commerce integrado al sistema de tarjetas de fidelización.

4. Funcionalidades a ser probadas

Todas las listadas en las historias de usuario del proyecto

5. Funcionalidades a no ser probadas

* No se probarán funcionalidades no utilizadas en ese sistema integrado del sistema de e-commerce ya que su desarrollo no fue parte de este proyecto.

* No se probará la capacidad de soportar grandes cantidades de consultas (conocidas como pruebas de carga).

* No se probarán sofisticadas funcionalidades de seguridad como control de phishing, hacking, deny of service, etc.

6. Metodología

Se realizarán las pruebas utilizando la herramienta Postman de Google que facilita las consultas HTTP. Se utilizarán usuarios y datos de prueba. Se utilizará un explorador web y se probará el sistema desplegado en el ambiente de producción o en el de pruebas,

documentando cual ha sido usado. Se utilizarán lso aplicativos web de clientes y administrativos tanto del sistema de e-commerce como del sistema de fidelización.

7. Criterios de aceptación

Una vez realizada una prueba, el sistema debe mantenerse en funcionamiento, es decir errores críticos que lo detengan no deben ocurrir. Cada prueba se considerará aceptada si se prueba que el sistema tiene la capacidad suficiente de servir al propósito de la historia de usuario. No se considerará aceptada una prueba si hay fallos en el ingreso de datos válidos, en su persistencia o en su consulta.

8. Criterios para suspensión

Las pruebas se suspenderán si existen cambios extensos en el proyecto: su diseño, tecnologías o propósito.

9. Entregables

Se documentará las pruebas realizadas, indicando los valores de entrada, los resultados obtenidos, fallas, defectos y recomendaciones.

10. Tareas de pruebas

Se realizará una prueba por historia de usuario utilizando valores de entrada válidos.

11. Requerimientos de ambientes

El ambiente está constituido por una instancia EC2 de Amazon que tiene instalado Docker y Docker Compose. Se utilizará un ambiente destinado a pruebas o el ambiente de producción sin preferencia de uno sobre otro.

12. Glosario

Sistema de tarjetas de fidelización. Sistema de tarjetas perforadas elaborado con Node JavaScript y Typescript. Es un proyecto de código abierto publicado en https://github.com/riuGlobal/loyalty-cards. Fue desarrollado como proyecto de titulación con fines académicos.

Sistema de e-commerce Cezerin: Se refiere al sistema de e-commerce mantenido en https://github.com/riuGlobal/cezerin2, https://github.com/riuGlobal/cezerin-web-example y https://github.com/riuGlobal/cezerin-admin que es una versión ligeramente editada de su original. El proyecto original de cezerin es un proyecto de código abierto. Su código fuente se encuentra publicado en https://github.com/Cezerin2/cezerin2

ANEXO F

CÓDIGO FUENTE

El código fuente en anexo se encuentra organizado de la siguiente manera:

1. Sistema de tarjetas de fidelización:
    - Servidor web: En la carpeta fidelizacion-servidor;
    - Aplicativo web de clientes: En la carpeta fidelizacion-web-clientes;
    - Aplicativo web administrativo: En la carpeta fidelizacion-web-administrativo.
2. Sistema de e-commerce Cezerin:
    - Servidor web: en la carpeta ecommerce-servidor;
    - Aplicativo web de clientes: en la carpeta ecommerce-web-clientes;
    - Aplicativo web administrativo: en la carpeta ecommerce-web-administrativo.
3. Script de Docker Compose en la carpeta: script-docker-compose.
4. Servidor web para integración del sistema de fidelización con el sistema de e-commerce: en la carpeta integracion-cezerin-fidelizacion.

El sistema se encuentra disponible en línea en los siguientes enlaces:

- Aplicativo web del e-commerce: http://3.141.248.26/;
- Aplicativo web administrativo del e-commerce: http://3.141.248.26:3001/admin;
- Aplicativo web administrativo del sistema de tarjetas de fidelizacion: http://3.141.248.26:3003.

ORDEN DE EMPASTADO