Documento de arquitectura de software del sistema [Nombre del sistema]

[Subtítulo del documento]

## Descripción breve

[BREVE DESCRIPCIÓN DEL SISTEMA O SU IMPORTANCIA]

[FECHA]
[Versión del documento]

## APROBADORES DEL DOCUMENTO

<<Representa las personas que aprobaron para que este documento pudiera ser publicado>>

| Nombre del aprobador | Rol |
|---|---|
| **[nombre del aprobador]** | [Rol del aprobador] |
| | |
| | |

## REVISORES DEL DOCUMENTO

<<Representa las personas que revisaron el documento antes de su publicación>>

| Nombre del revisor | Rol |
|---|---|
| **[nombre del revisor]** | [Rol del aprobador] |
| | |
| | |

## HISTORIAL DE CAMBIOS

<< Indica todos los cambios que ha tenido el documento desde su creación, como la versión del documento, la fecha del cambio, la persona que lo editó y, sobre todo, una descripción de los cambios realizados al documento >>

| Versión | Fecha | Actualizado por | Descripción |
|---|---|---|---|
| **[No Versión]** | [Fecha del cambio] | [Nombre de la persona que modifico el documento] | <Rol del aprobador> |
| | | | |
| | | | |

# Contenido

# INTRODUCCIÓN

## Propósito

*<<Breve descripción sobre el propósito de este documento>>*

<< Ejemplo:

Este documento tiene como finalidad proveer una visión global y detallada de la arquitectura del sistema, mediante el uso de distintas perspectivas arquitectónicas para representar los diferentes aspectos del mismo. Su propósito es capturar y transmitir las decisiones arquitectónicas más relevantes que se han tomado en relación al sistema.

>>

## Alcance

*<<Describe brevemente cual es el alcance de este documento, como los sistemas que cubre y los sistemas o componentes que quedan fuera del documento, básicamente se intenta delimitar lo que encontraremos en este documento.>>*

<< Ejemplo:

Este documento intenta explicar la arquitectura del componente ACME Store API, uno de los componentes más importantes del sistema ACME Store. Este documento está acotado a explicar el funcionamiento de este componente y como este se relaciona con los componentes adyacentes.

>>

## Glosario

*<<Esta sección es para listar cualquier termino que pueda ser confuso o desconocido para un actor externo al sistema>>*

| Término | Descripción |
|---|---|
| **[nombre del término]** | [Descripción del término] |
| ***<<ejemplos>*** | *<<ejemplos>>* |

| | |
|---|---|
| **REST** | Representative State Transfer |
| **DAO** | Data Access Object |
| **DTO** | Data Transfer Object |
| **Spring Framework** | Framework de desarrollo Java de propósito general |
| **Spring boot** | Framework Java para la creación de microservicios. |
| **JMS** | Java Message Service |
| **JWT** | JSON Web Token |
| **JSON** | Javascript Object Notation |

# ARQUITECTURA GENERAL

## Descripción general de la aplicación

*<<Realice una breve descripción de la aplicación que está documentando>>*

<<Ejemplo:

ACME Store es una aplicación de comercio electrónico cuyo objetivo principal es procesar pedidos de clientes a través de internet. Aunque pueda parecer una aplicación sencilla, en realidad consta de varios componentes que realizan diversas tareas, desde la capa de presentación (una versión web y otra móvil) hasta el backend, que incluye un API REST y un proceso de sincronización de pedidos con el ERP. Este documento explica la arquitectura del sistema ACME Store, incluyendo los aspectos más relevantes de la arquitectura de software.

>>

## Objetivos generales y restricciones de la arquitectura

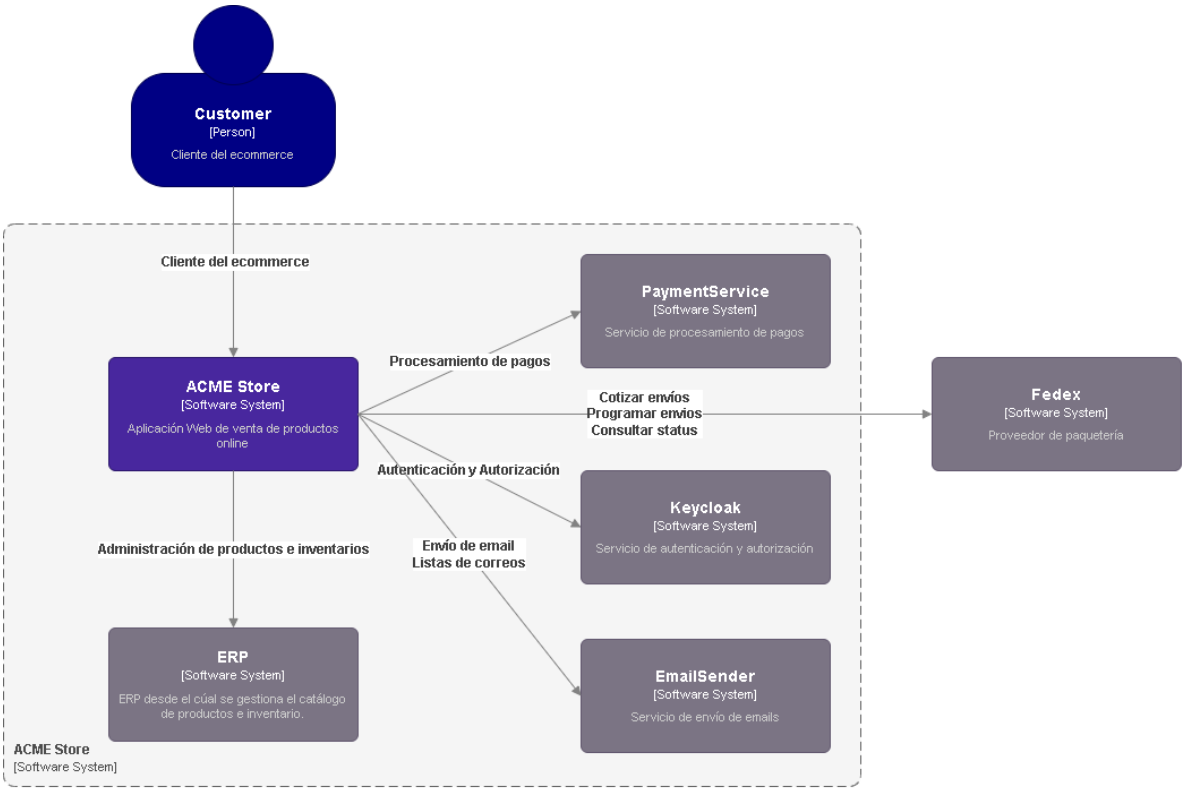
<<Los siguientes requerimientos no funcionales fueron identificados y tomados en cuenta para diseñar la arquitectura del sistema>>

| Requisitos no funcionales | Descripción |
|---|---|
| **[Requisito funcional]** | <<Descripción de requisito funcional y como este afecta a la arquitectura>> |
| ***[Ejemplo]*** | *[Ejemplo]* |
| **Seguridad** | Dado que el sistema procesa pagos en línea, es necesario contar con mecanismos que ayuden a procesar de forma segura los cargos y evitar que los datos bancarios sean mal utilizados, almacenados o explotados por personas no autorizadas. |
| **Escalabilidad** | Dada las ambiciones de la compañía, se espera que ACME Store pueda funcionar en varios países sin verse afectada por el tráfico que este tenga. De la misma forma, es necesario tomar en cuenta los picos de demanda como las fechas festivas para no detener su operación |

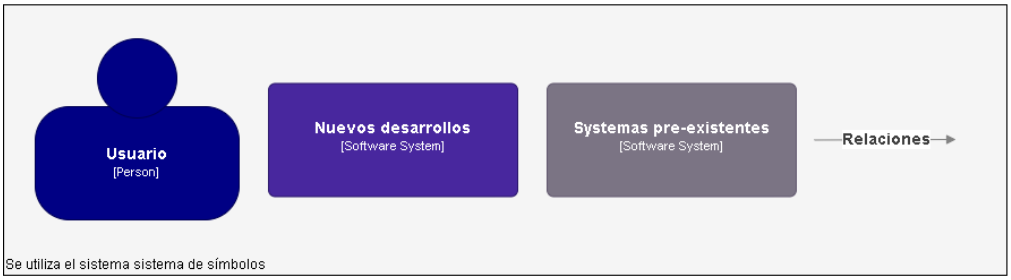
| | |
|---|---|
| **Usabilidad** | Dado que el sistema será utilizado por público en general, tiene que ser diseñado para ser fácilmente utilizable por cualquier persona. |
| | |

## Arquitectura de contexto del sistema

*<<En esta sección se debe agregar un diagrama de arquitectura general del sistema, que represente los elementos más importantes, aquí es recomendable poner el diagrama de contexto del sistema, así como una breve descripción de los elementos que aparecen en el diagrama>>*




**[System Context] ACME Store**
Arquitectura del sistema de venta en línea AMCE Store



*1 - Diagrama de contexto del sistema*

*<< Describa los elementos que aparecen en el diagrama con la finalidad de que las personas externas al sistema puedan comprender la relevancia y cada uno de los elementos y el papel que juegan dentro de la arquitectura.>>*

<<Ejemplo:

ACME Store es la aplicación web de comercio electrónico de ACME Corp, la cual puedes visitar en acmestore.com, o descargando la aplicación desde la App Store.

ACME Store puede parecer una simple aplicación, sin embargo, es en realidad una serie de componentes que le dan vida al sistema, que van desde la aplicación web, la app para móviles, el API REST de backend y procesos de sincronización de productos y existencias. Además de estos, se hacen uso de otros sistemas, tanto externos como internos, los cuales son:

- ERP: Sistema desde el cual se lleva a cabo el seguimiento de los pedidos, productos e inventarios.
- PaymentService: Es un servicio para procesar pagos, el cual simplifica la integración con los diferentes proveedores de pago.
- Keycloak: Sistema para gestionar la autenticación y la autorización utilizando los estándares de OAuth y OpenID Connect.
- EmailSender: Servicio de infraestructura para enviar correos electrónicos, simplificando la comunicación con los proveedores de Email Marketing, como es el caso de MailChimp.
- Fedex: Representa el API de Fedex para cotizar los envíos.

Más adelante en este mismo documento vamos a profundizar en los componentes que conforman el sistema ACME Store así como el componente central de este documento que es ACME Store API.

>>

# ARQUITECTURA LÓGICA

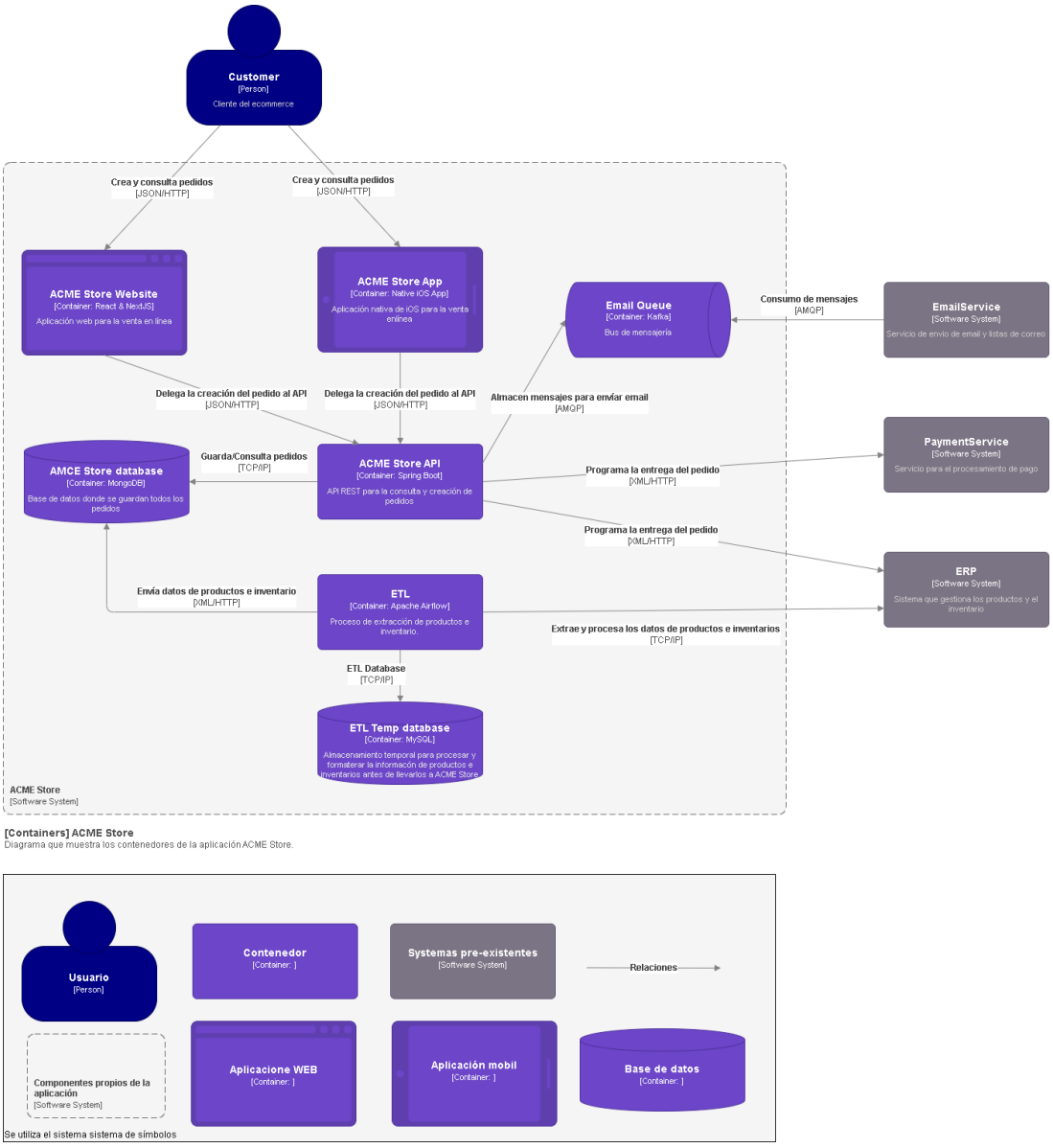
## Arquitectura de contenedores

*<< En esta sección, describiremos la arquitectura lógica del sistema ACME Store, es decir, los elementos más significativos y las relaciones que muestran su interacción. En este punto, es un buen momento para agregar el diagrama de contenedores. En esta sección, nos centraremos en cómo el sistema interactúa con el resto de los componentes >>*

<< Ejemplo:

ACME Store es una aplicación de comercio electrónico, compuesto por diferentes componentes que juegan un papel importante dentro de la arquitectura. Si bien ACME Store tiene la parte visible para el usuario que es la versión web y móvil de la aplicación, existen otros componentes de backend que tiene una importante relevancia para poder procesar los pedidos, realizar los pagos, envío de email, sincronización de productos e inventarios, etc. En este sentido ACME Store API es el componente de backend que expone los servicios y la lógica de negocio que le da vida a la aplicación y en el que nos centraremos en este documento.

>>




**[Containers] ACME Store**
Diagrama que muestra los contenedores de la aplicación ACME Store.




*2 - Diagrama de contenedores del sistema ACME Store.*

<< Ejemplo:

En esta vista de la arquitectura podemos apreciar los siguientes sistemas:

- ACME Store Website: Aplicación web (React) que se renderiza desde el navegador y que es la que los usuarios utilizan para consultar el catálogo de productos y realizar sus pedidos.
- ACME Store App: Aplicación nativa para iOS (Swift) que los usuarios pueden descargar desde la App Store para consultar el catálogo de productos y realizar pedidos.
- ACME Store API: Corresponde al componente de Backend (Spring Boot) que da soporte tanto a la versión web como mobile mediante la exposición de servicios REST.

- ACME Store database: Base de datos (MongoDB) de ACME Store API donde se almacena la información de usuarios, pedidos, productos e inventarios, etc.
- ETL: Proceso (Airflow) de extracción que sincroniza el catálogo de productos e inventarios desde el ERP, hasta la base de datos de ACME Store API. Este proceso se ejecuta todas las noches.
- ETL Temp database: Base de dato (MySQL) utilizada por el proceso ETL para procesar temporalmente el catálogo de productos e inventarios.
- Email Queue: Es la cola (queue en Kafka) que se utiliza para almacenar los mensajes de envío de correo electrónicos, con la finalidad de perder los mensajes y tener bajo acoplamiento con el componente de EmailService.
- EmilService: Es el servicio de infraestructura (reutilizable) utilizado para el envío de correos electrónicos, de tal forma que oculta al proveedor real de email marketing para hacer más simple y estándar el envío de email en toda la compañía.
- PaymentService: Servicio de infraestructura utilizado para el procesamiento de pagos.

ACME Store API es un API REST desarrollado en Spring Boot bajo la arquitectura de microservicios, lo que implica que cuenta con su propio runtime basado en Apache Tomcat y puede ejecutarse por sí mismo sin necesidad de un servidor de aplicaciones. Además, se ejecuta dentro de un contenedor en Kubernetes.

>>

## Proceso de extracción ETL

*<< Se puede agregar una sección dedica para cada contenedor donde sea necesario realizar una explicación más detallada que una simple viñeta >>*

<< Ejemplo:

El proceso de extracción ETL no es parte del contenedor ACME Store API, pero es sumamente relevante para su correcto funcionamiento, ya que los productos y los inventarios son administrados y actualizados desde el ERP. Por tal motivo, el API requiere de un proceso que sincronice los datos todos los días. El proceso ETL es un desarrollo construido exclusivamente con esta finalidad y se trata de un desarrollo en Airflow que extrae los datos de los productos y los inventarios, los almacena temporalmente en su propia base de datos llamada ETL Temp Database para finalmente convertir la información en el formato esperado por la base de datos del API.

Dado que es un proceso tardado por la gran cantidad de productos, se ha tomado la decisión de que se ejecute una sola vez a medianoche. Este proceso está automatizado, sin embargo, si se requiere, puede ser forzado para correr en cualquier hora del día. La importancia de este componente es tal

que, si no funcionara correctamente, no se verían reflejados los cambios en los productos y sus inventarios, por lo que es de suma importancia para el correcto funcionamiento de ACME Store API y de la tienda en general.

>>

## Arquitectura de componentes

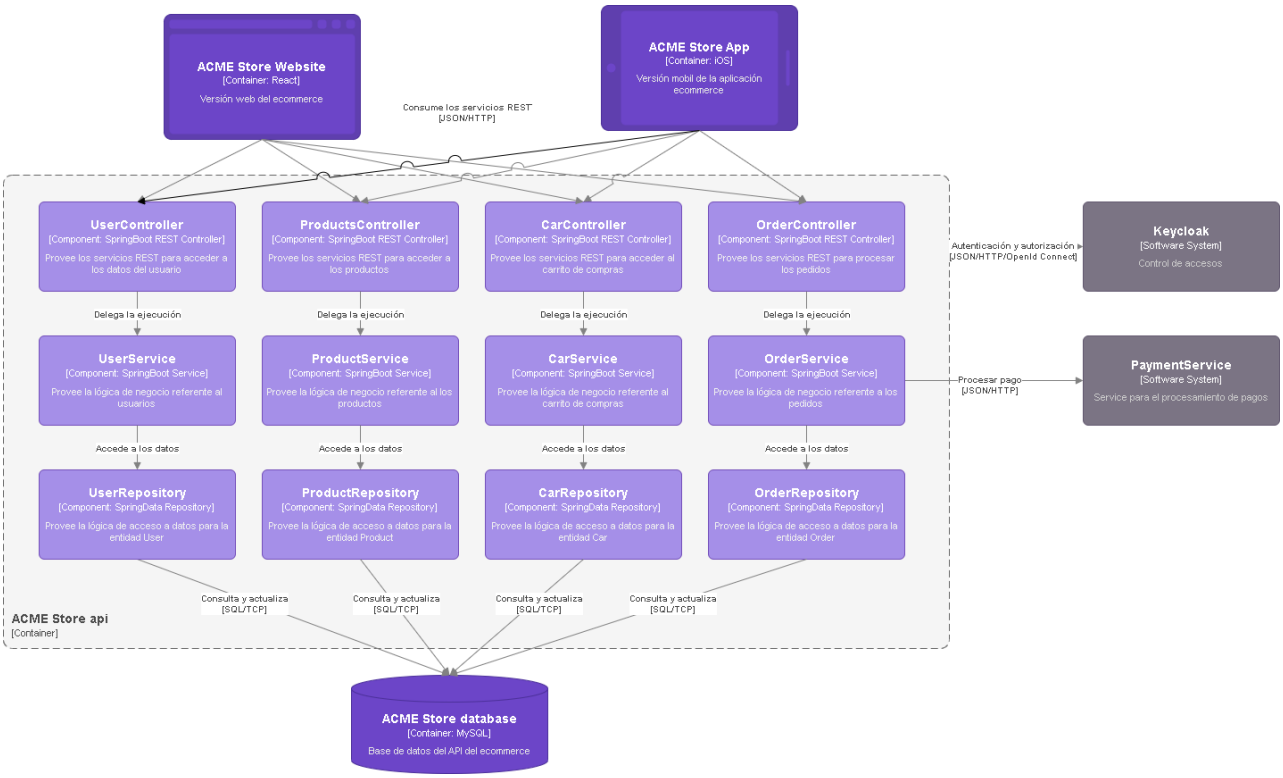
*<< Esta sección se enfoca únicamente en el contenedor de la aplicación en cuestión. Se espera que se muestren los componentes más importantes que conforman el contenedor con su correspondiente explicación. En esta sección ya no es necesario explicar los sistemas externos, aunque siempre se puede mencionar si es que aporta algo a la explicación del contenedor. >>*

<< Ejemplo:

ACME Store API es construido en capas, donde cada una de ellas cubre una responsabilidad del sistema, estas capas sirven para organizar mejor el proyecto y estandarizar la forma en que estos trabajan. Las capas definidas en el servicio son:

- Controllers: representa la capa de servicios y en esta se definen todos los métodos que serán expuestos como servicio REST. Esta capa se encarga de recibir las peticiones y delegarlas a la capa de servicios. Esta capa no procesa nada de lógica de negocios, solo se limita al procesamiento de las solicitudes y las transformaciones para las respuestas en formato JSON.
- Services: capa transaccional de la arquitectura, en este se lleva a cabo toda la lógica de negocio y el procesamiento de peticiones al API. Esta capa no se preocupa por el acceso a datos o los formatos de entrada/salida esperados por el API.
- Repository: capa de acceso a datos, ha esta capa se le delega la responsabilidad de consultas, guardado, actualizaciones y borrado a la base de datos. Esta capa no se preocupa por lógica de negocio, solo la comunicación con la base de datos.

>>




*3 - Diagrama de componentes*

<< Ejemplo:

Para comprender mejor el funcionamiento de la aplicación, hemos dividido el contenedor en “dominios”, donde cada domino agrupa una serie de clases relacionadas entre sí y que responde en función de una entidad.

*<< Muchas veces la explicación de los componentes es repetitiva, ya que se componen de una mista estructura de clases, como es el caso de las Entidades, Controladores, Servicios y Repositorios, por tal motivo, podemos dividir la explicación por Dominios >>*

## Dominio de usuarios

El dominio de usuarios esta encabezado por la entidad User, clase que se omite en el diagrama de componentes para simplificarlo. El dominio de usuarios está conformado por las clases UsuarioController, UserService, UserRepository y la entidad User. Este dominio se centra en la creación de servicios relacionados con los usuarios, como la creación de nuevas cuentas, actualizaciones de perfiles, etc.

En este dominio pueden entrar en juego otras Entidades secundarias y relacionadas como el usuario, como puede ser las Direcciones (Address), métodos de pago (PaymentMethod) y la entidad del perfil del usuario (UserProfile)

### Dominio de Productos

Este dominio esta encabezado por la entidad Producto, clase que se omite del diagrama de componentes para simplificarlo. El dominio de productos está conformado por las clases ProductController, ProductService, ProductRepository y la entidad Producto. Este dominio se centra en los servicios para la consulta del catálogo de productos y los inventarios.

### Dominio de Carrito de compras

Este dominio está encabezado por la entidad Car, clase que se omite del diagrama de componentes para simplificarlo. El dominio de carrito de compras está conformado por las clases CarController, CarService, CarRepository y la entidad Car. Este dominio se centra en los servicios relacionados con el carrito de compra, como limpiar el carrito, agregar/eliminar productos al carrito, etc.

En este domino interactúa entidades secundarias como la CarItem, que representa un producto dentro del carrito.

### Dominio de Ordenes

Este dominio esta encabezado por la entidad Orden, clase que se omite del diagrama de componentes para simplificarlo. El dominio de carrito de compras está conformado por las clases OrderController, OrderService, OrderRepository y la entidad Order. Este dominio se centra en las ordenes y los servicios relacionados con esta, como es la creación de pedidos, pago de ordenes, actualizaciones de inventario, cancelación de pedidos, etc.

>>

## Arquitectura de bajo nivel

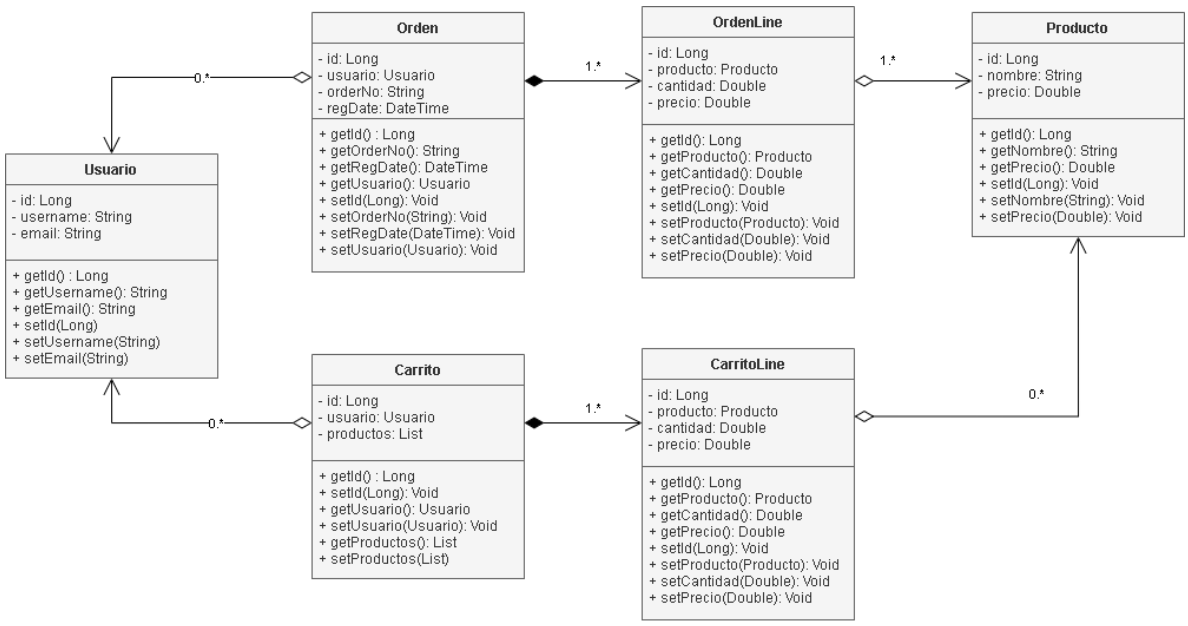
*<< Esta sección se reserva solo para aquellos proceso que requieren un nivel de detalle de bajo nivel y que son fundamentales para comprender el componente. Esta sección es opcional y no siempre será requerida, sobre todo en componentes más simples que no tiene procesos complejos.*

*En esta sección es aconsejable incluir diagramas de UML o código y diagramas dinámicos de C4. >>*

En esta sección describiremos aquellos procesos que, por su complejidad o criticidad, son candidatos a diagramar con un detalle más fino.

## Dominio de la aplicación

El siguiente diagrama muestra las Entidades administradas por este componente, las cuales tiene una relación directa con la base de datos.

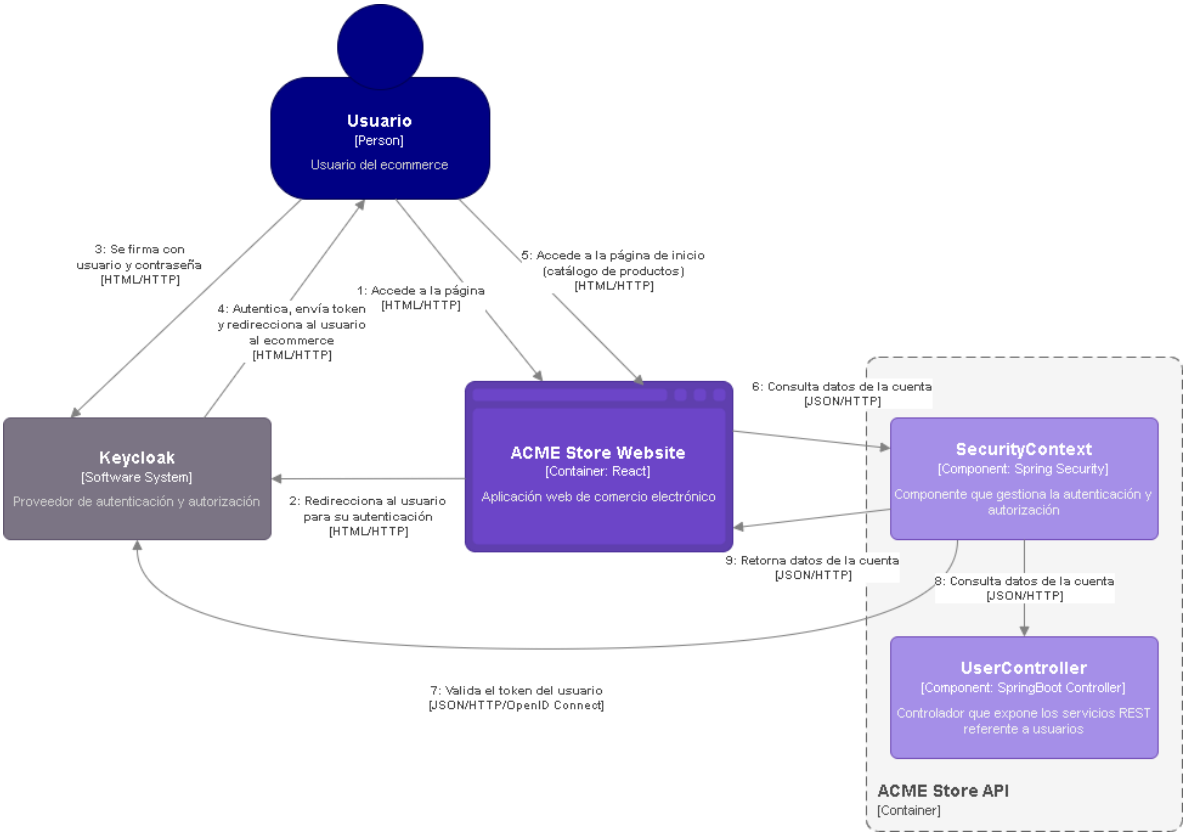



*4 - Diagrama de Entidades o dominio de la aplicación.*

## Proceso de autenticación

El proceso de autenticación puede resultar complejo en primera instancia, sobre todo si se desconocen los estándares de OAuth y OpenID Connect, por lo que el siguiente diagrama intenta

dar una idea más clara del proceso por medio del cuál se lleva a acabo la autenticación de los usuarios.




*5 - Diagrama dinámico del proceso a autenticación.*

Keycloak es un proyecto de código abierto que implementa la autenticación por medio del estándar de OAuth y OpenID Connect y es la base sobre la que funciona el sistema de autenticación de ACME Store.

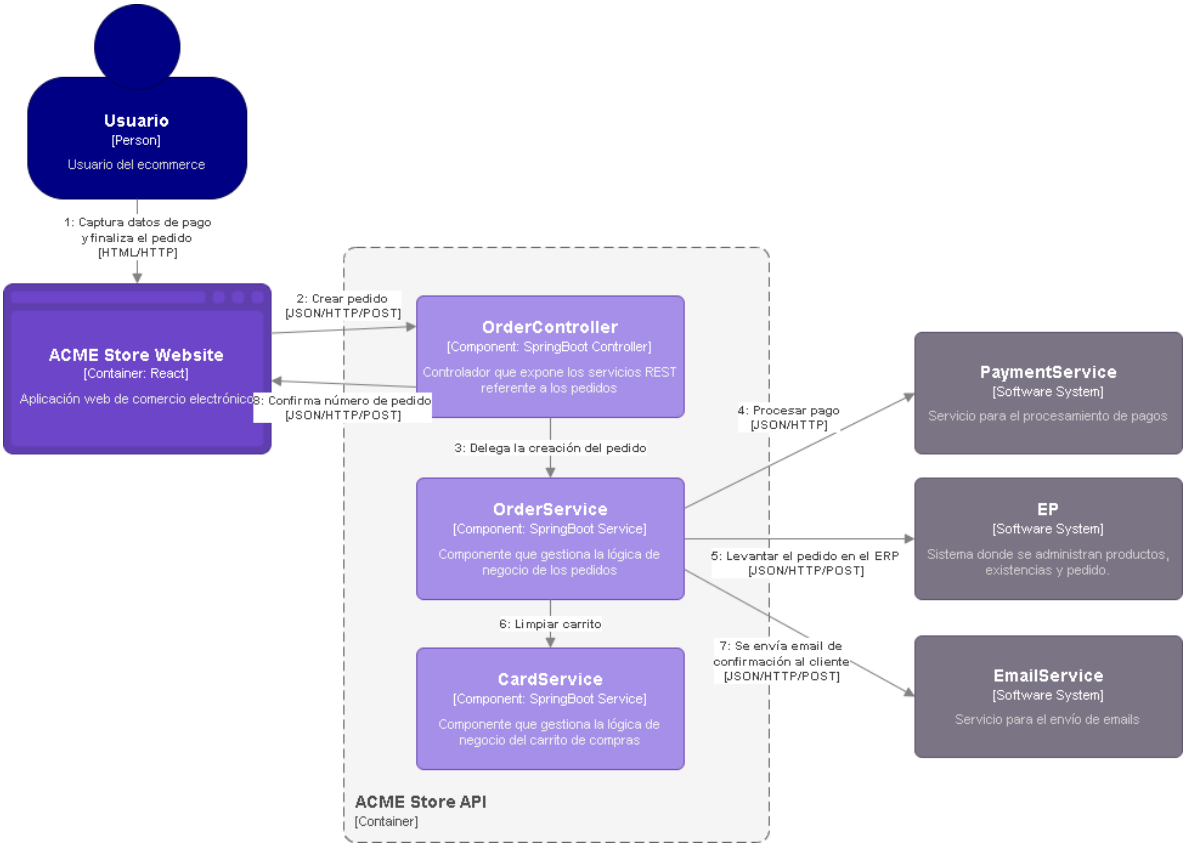
Cuando un usuario accede a la página de ACME Store, tendrá la opción de iniciar sesión en la página. En primera instancia, el usuario será redirigido a la página de autenticación alojada en Keycloak, donde tendrá que ingresar su usuario y contraseña. Una vez que introduzca sus credenciales, Keycloak las validará y, de ser correctas, redireccionará al usuario de regreso a la página de ACME Store.

Una vez de regreso en la página de ACME Store, la página intentará autenticar al usuario con el token generado por Keycloak, por lo que enviará ese token de regreso a Keyclok para su validación. Si Keyclok lo validar, entonces la página de ACME Store autentica al usuario y regresa los datos del cliente usuario.

De allí en adelante, cada petición que realice el usuario sobre el sistema, tendrá que incluir el token, y ACME Store API lo tendrá que validar nuevamente a fin de validar que sea autentico y vigente.

## Proceso de creación de pedidos

La generación de pedidos es el proceso más importante de la aplicación, pues de este depende que la tienda tenga ventas y pueda ser rentable, además, es un proceso complejo, ya que interviene varios servicios o sistemas externos para su correcto funcionamiento.



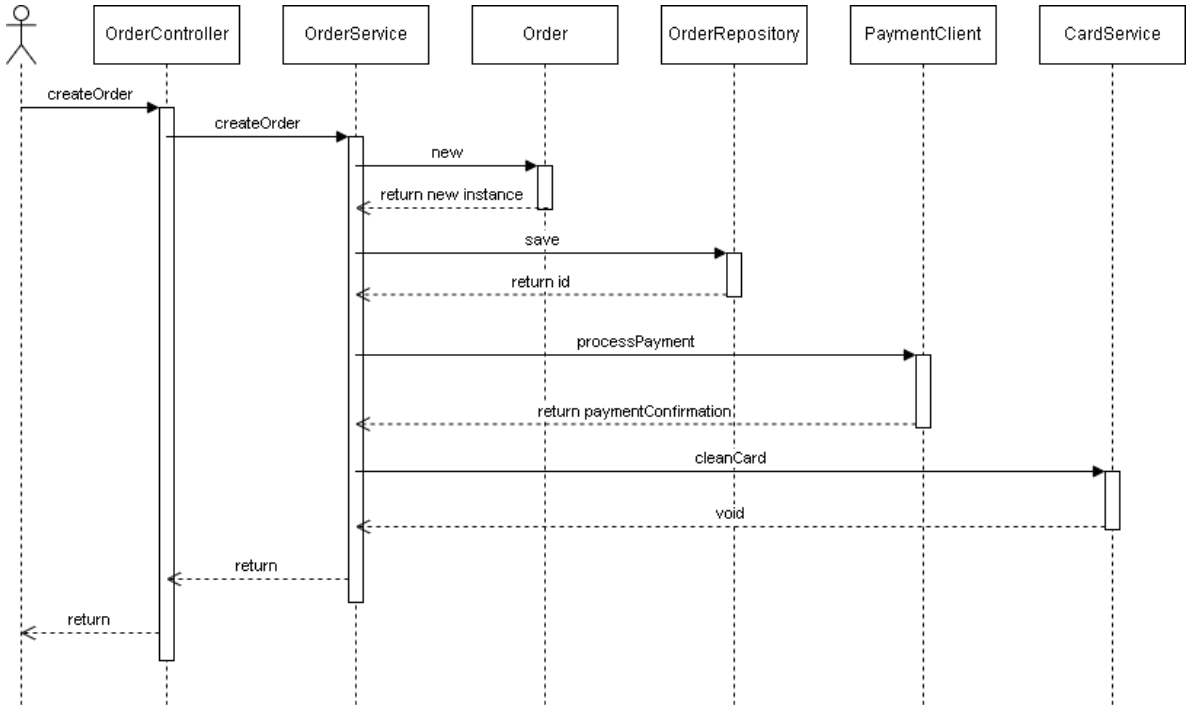

*6 - Diagrama dinámico del proceso de generación de pedidos.*

El proceso de creación de ordenes se puede disparar desde diferentes medios, ya que al ser un servicio REST, solo basta que un consumidor ejecute el servicio para que se detone la creación, sin embargo, por el momento, los dos únicos actores conocidos que pueden hacer esto son los contenedores ACME Store App y ACME Store Website.

Dicho lo anterior, el proceso de creación se dispara desde el servicio REST de creación de órdenes que responde en el endpoint/orders sobre el método POST. Este recibe la petición y realiza validaciones simples de formato y delega la creación del pedido al componente OrderService, el cual guarda el pedido en la base de datos por medio de la capa de persistencia, realiza el cargo a la tarjeta

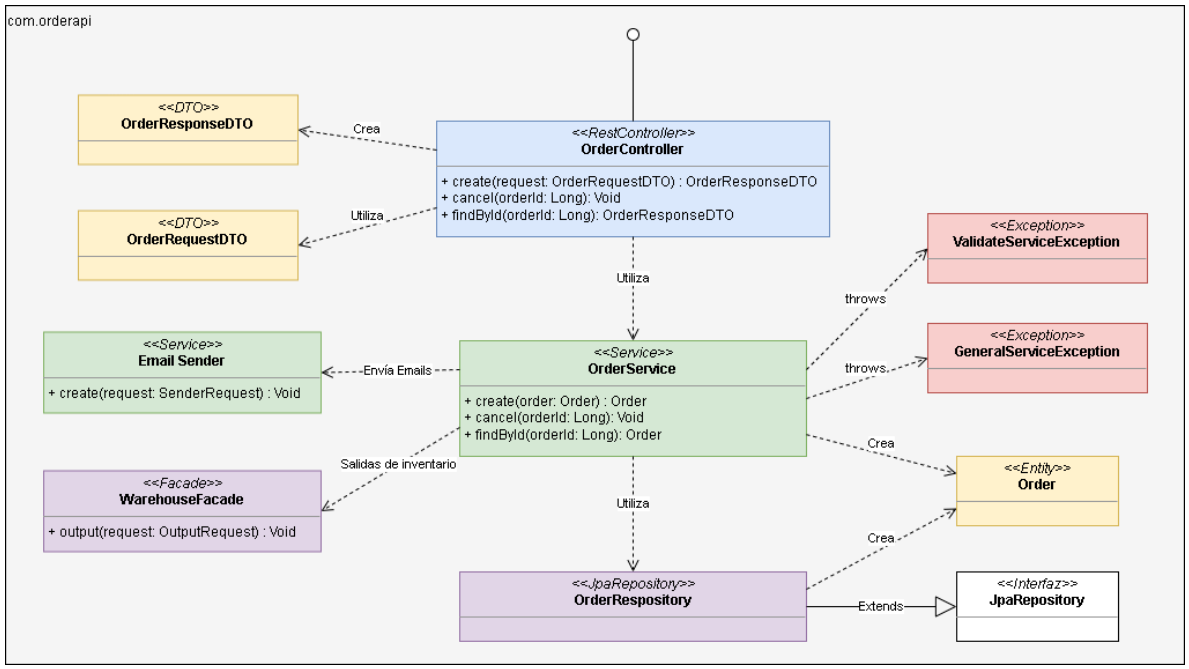
del cliente mediante el sistema PaymentService, sincroniza el pedido con el ERP y envía un correo electrónico con la confirmación del pedido.

El orden en que se realizan estos pasos se puede apreciar en el siguiente diagrama de secuencia.




*7 - Diagrama de secuencia del proceso de creación del pedido.*

El siguiente diagrama ilustra perfectamente como esta conformado el contenedor ACME Store API respecto al dominio de pedidos:

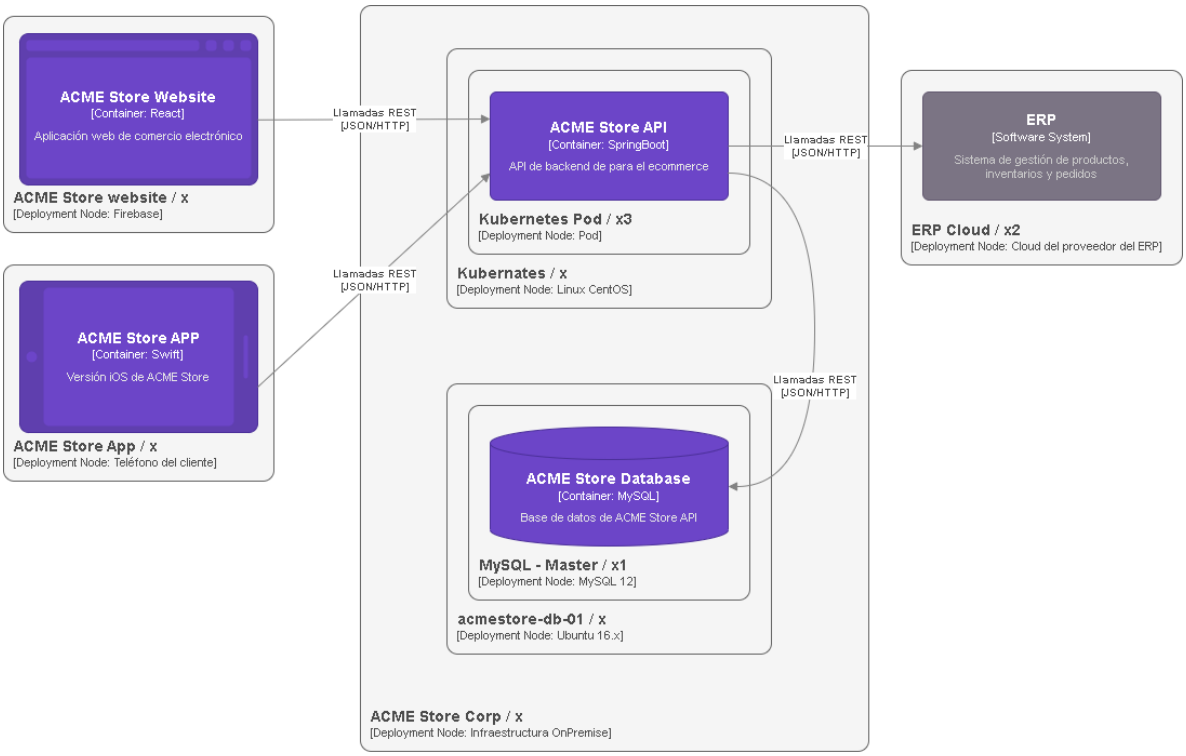



*8 - Diagrama de clases del dominio de Ordenes*

# ARQUITECTURA FÍSICA

La arquitectura física tiene como objetivo ilustrar cómo los contenedores se desplegarán físicamente dentro de los servidores de nuestra infraestructura o de la nube.

*<< Agrega una descripción que ayude a comprender el diagrama, donde se expliquen los contenedores y sistemas, así como las cajas sobre las que están desplegados los contenedores >>*




*9 - Diagrama de despliegue*

<< Ejemplo:

ACME Corp cuenta con una robusta infraestructura on-premise que le permite desplegar sus aplicaciones de forma loca con infraestructura propia. Para esto, ACME Corp cuenta un cluster de Kubernetes para desplegar aplicaciones y también cuenta con la posibilidad de desplegar aplicaciones en maquinas virtuales o físicas.

Los elementos que aparecen en este diagrama son:

- ACME Store Website: Esta desplegado sobre Firebase, por lo que es una plataforma totalmente administrada por Google y no se requiere acciones para escalar la aplicación.
- ACME Store App: Al ser una aplicación nativa de iOS, la aplicación de sube a la App Store y se ejecuta en el smarthpone de cada cliente.
- ACME Store API: El API REST de la aplicación está desplegado sobre un cluster de Kubernetes alojado en la infraestructura de ACME Corp, por lo que la gestión y el escalamiento de este componente esta a cargo de ACME Corp. Inicialmente la aplicación funciona en 3 instancias de contenedor.
- ACME Store Database: Base de datos MySQL donde se almacena toda la información del API. Esta base de datos se ejecuta desde una máquina virtual llamada “acmestore-db-01”.
- ERP: Al ser un sistema de un tercero, es ejecutado y administrado en su totalidad por el proveedor, por lo que vive en la nube del vendor.

>>