# E-COMMERCE

## Documento de Arquitectura de Software

Versión 1.0


**- Aldo Ponce de la Cruz A01651119**
**- Ignacio Alvarado Reyes A01656149**
**- José Antonio Berrón Gutiérrez A01377269**
**- Arturo Efrén Jiménez Garibaldi A00824428**
**- Manuel Herrasti González A01657028**


**Historial de Revisiones**

| Fecha | Versión | Descripción | Autor |
|---|---|---|---|
| 30/03/2022 | 1 | Primera Versión | El equipo |
| | | | |
| | | | |
| | | | |

# Contenido

# Documento de Arquitectura de Software

# 1. Introducción

## 1.1 Propósito

La creación de una webapp que permita a un grupo de administradores utilizar un sistema crud de una tienda online, para la gestión de productos y usuarios. Y por otra parte, una webapp para los usuarios que permita realizar compras y pedidos de productos electrónicos.

El sistema permitirá a los usuarios comprar productos que se ofrezcan en la tienda online, y de la misma manera los usuarios administradores pueden añadir sus propios productos para ser vendidos dentro de la plataforma.

Los administradores de la misma manera deben ser capaces de administrar los usuarios para un funcionamiento adecuado del sistema. En caso de ser necesario los administradores tendrán la capacidad de borrar usuarios inactivos y de la misma manera actualizar información de productos que hayan cambiado en algún ámbito.

El sistema de ventas debe tener implementado un módulo de seguridad de pago y entrega de productos, con esto asegurando el rendimiento de la página y la escalabilidad de la misma. Todo esto será gracias al uso de servicios o módulos añadidos a nuestro sistema principal.

## 1.2 Alcance

Crear un sistema CRUD para la administración de una tienda online y sistema de compras para usuarios.

Realizar un sistema donde toda la funcionalidad sea correcta como la autenticación de los distintos usuarios en nuestro sistema como: cliente, admin y superadmin. Este sistema debe de operar de una manera correcta dependiendo del usuario que use la página web.

Implementar una arquitectura de software que nos guíe en el proceso y documentación de una tienda en línea. A través de esta arquitectura de microservicios añadiremos paquetes que amplíen la funcionalidad del sistema como: ventas, administración bancaria, pagos seguros, etc.

## 1.3 Definiciones, Siglas y Abreviaturas

CRUD: de las siglas en inglés Create Read Update Delete, se refiere a la serie de operaciones estándar que se realizan en el uso de una base de datos donde se crean, consultan, cambian y borran datos.

API: de las siglas en inglés Application Programming Interface (Interfaz de Programación de Aplicaciones) consiste básicamente en un intermediario de software que permite la comunicación entre dos programas.

Software: Datos o programas utilizados para completar tareas de cómputo.

Hardware: Parte física de una computadora.

PC: de las siglas en inglés Personal Computer (Computadora Personal). Hardware computacional designado para uso personal.

Módulos: Paquetes destinados al incremento de la funcionalidad del sistema teniendo en cuenta el sistema principal.

Customer: Persona que navega la página, este puede buscar productos por diferentes categorías y agregar los productos a un carrito. Sus productos y ajustes serán guardados en el perfil del cliente.

Admin: Usuario que administra los productos y clientes del sistema de ventas. Este tiene la posibilidad de modificar tanto los productos como los clientes.

SuperAdmin: Usuario con el nivel de acceso más profundo que tiene las características de un Admin, además de tener la posibilidad de modificar los Admins existentes.

REST: Es una interfaz de programación de aplicaciones (API o API web) que se ajustan a los límites de la arquitectura REST y permite la interacción con los servicios web de RESTful.

## 1.4 Referencias

What is CRUD? Explaining CRUD Operations | Sumo Logic. (2022). 30 Marzo 2022, de Sumo Logic Sitio Web: https://www.sumologic.com/glossary/crud/

What is an API? (Application Programming Interface) | MuleSoft. (2022). 30 Marzo 2022, de MuleSoft Sitio web: https://www.mulesoft.com/resources/api/what-is-an-api

What is Software? Definition, Types and Examples. (2022). 30 Marzo 2022, de techtarget Sitio Web: https://www.techtarget.com/searchapparchitecture/definition/software

IBM Cloud Education. (2021). API REST. 30 Marzo 2022, de IBM Sitio web: https://www.ibm.com/mx-es/cloud/learn/rest-apis

## 1.5 Resumen

## 2. Representación Arquitectónica

### 2.1.1 Objetivos Arquitectónicos

El objetivo principal de esta arquitectura es crear un sistema completamente modular, donde cada servicio es independiente de los otros, y existen interfaces que comunican los componentes. Cada componente puede ser desplegado por separado, los componentes pueden combinarse entre sí para representar un sólo servicio o función y el acceso a estos es a través de protocolos de acceso remoto. Todas estas ventajas y características están alineadas a ofrecer un servicio web con altos estándares de calidad.

Respecto a los drivers del proyecto, describimos los puntos más importantes.

Atributos de calidad: En el sistema se hará uso de métricas de calidad del sistema, como lo son el desempeño del mismo, la seguridad para los usuarios, la facilidad de modificar o realizar cambios al sistema y finalmente la utilización de pruebas y hacer que estas se realicen de manera eficaz y sencilla. Con estos podemos asegurar la calidad de nuestro sistema para cualquier usuario que quiera usar la plataforma.

Funcionalidades del sistema: El sistema mostrará un servicio eficaz para el acceso a los productos y carrito de compras de cada uno de los usuarios. De la misma manera debe de poder tener un manejo de sesión integrado para que el usuario pueda administrar la sesión como él vea conveniente.

Business Drivers: Entre los puntos más importantes para ofrecer un buen servicio de negocio está la eficiencia de respuesta de la página web, porque esto permite que más clientes puedan acceder al sistema y realizar compras de forma simultánea. También se ofrece un dashboard de administración que permite categorizar los productos y registrar sus respectivos precios, así se tiene un orden en cuanto a las finanzas disponibles del negocio.

Constraints: Estos están especificados en el siguiente apartado de restricciones arquitectónicas.

### 2.1.2 Restricciones arquitectónicas

Algunas desventajas considerables de esta arquitectura son los siguientes:

Mantenimiento: Los sistemas complejos y distribuidos suelen tener algunos problemas de mantenimiento, porque hay que conocer a detalle toda la estructura para tener las relaciones bien definidas. Algunos ejemplos de mantenimiento son implementación de comunicación interna entre todos los servicios del sistema y solicitudes que se extienden a otros servidores.

Disponibilidad del sistema: El sistema web puede estar comprometido si existen picos de usuarios en consultas, ya que se supera la capacidad de procesamiento de la base de datos y el host.

Mayor consumo de recursos: Puesto que cada microservicio tiene su propio sistema, es más costoso realizar procesos dentro de estas. Es importante tener un conteo y un limite de servicios que el usuario puede hacer uso en el programa. Al no limitar esto los recursos que el sistema puede utilizar son más grandes que un sistema sin servicios externos.

| | |
|---|---|
| Agilidad | High |
| Facilidad de despliegue | High |
| Pruebas | High |
| Rendimiento | Low |
| Escalabilidad | High |
| Facilidad de desarrollo | High |

## 2.2 Estilo Arquitectónico y rationale

Microservicios es un estilo arquitectónico que divide a la aplicación en una colección de servicios independientes.

Es un modelo de arquitectura que usa conceptos de:

- Unidades de despliegue: Cada componente es desplegado como una unidad separada.
- Componente de servicio: Clases que representan una función del sistema.
- Arquitectura distribuida: Los componentes están separados y se requieren protocolos de acceso para comunicarse.

En particular, seleccionamos la topología de API-REST, donde tendremos componentes granulares de servicio para las diferentes funcionalidades de nuestra tienda que se comunicaran con APIs externas.

Los componentes que tenemos planeados para nuestra aplicación son:

- Carrito de Compras: Agregación de productos y estimación de precios.
- Orden: Contenidos de ordenes, precios y envíos.
- Productos: Descripción de productos en almacén, precios y cantidad en inventario.
- Usuarios
    - Cliente: Capaz de poder seleccionar productos y guardarlos en un carrito de compras. Posteriormente puede borrar y comprar los productos.
    - Administrador: Capaz de crear productos y usuarios.
    - Superadministrador: Mismas habiliades del Administrador, con la funcionalidad extra de poder crear otros Administradores.
- Autentificador: Validación de las credenciales de los usuarios para otorgar acceso a las diferentes funcionalidades del sistema.
- Landing Page
    - Para Clientes: Presentación de productos y direcciones solamente a componentes donde estos puedan hacer consultas y compras.
    - Para Administradores: Opción de poder crear usuarios o productos que se mostrarán en la landing page de los clientes.

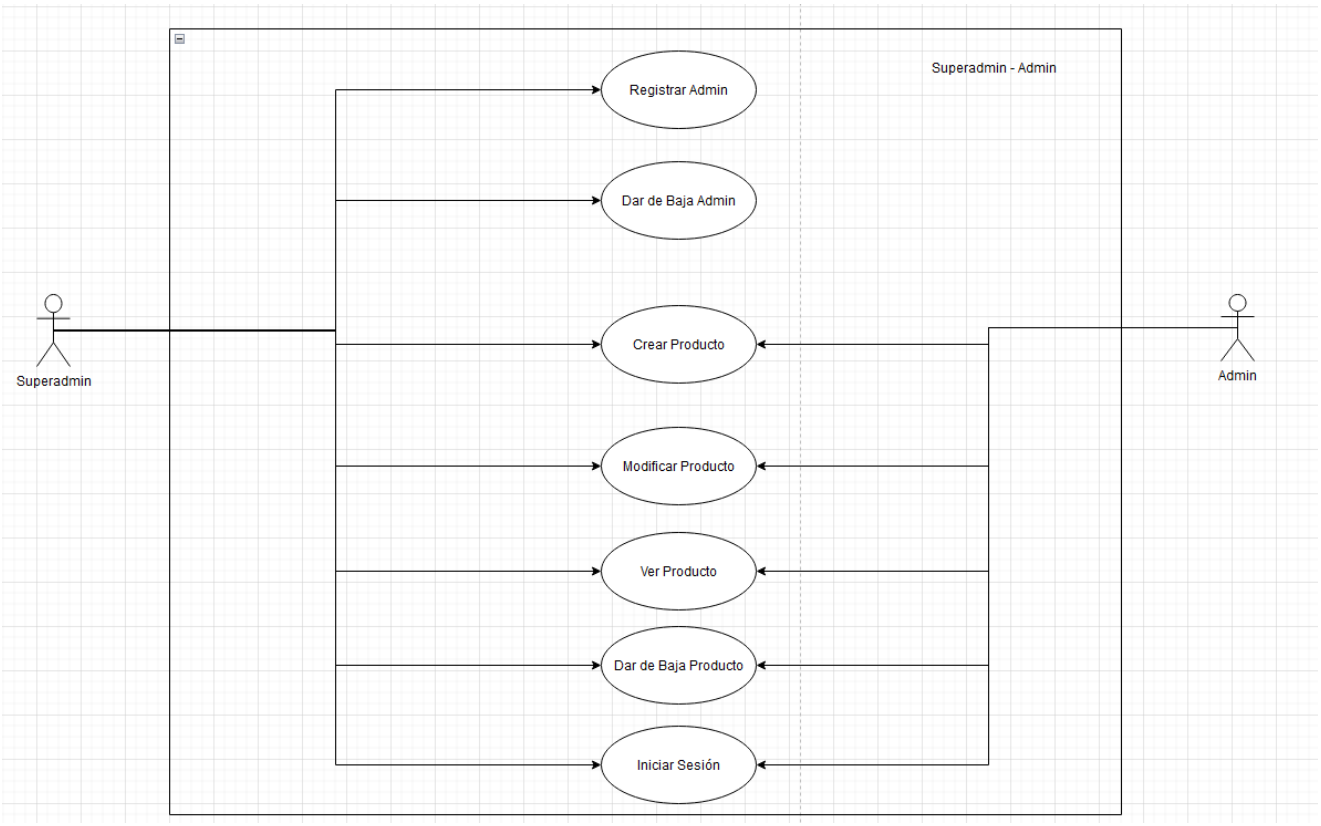
Respecto al motivo de nuestra selección, podemos decir que esta arquitectura nos da las siguientes ventajas:

- Altamente modificable y testable.
- Cada componente y view es independiente.

- Buena escalabilidad.
- Fácil implementación de servicios y APIs externas.

Por otra parte, es una arquitectura comúnmente utilizada en servicios de web apps porque es la misma estructura que siguen las metodologías sugeridas por frameworks de front-end como React, dónde creamos cada componente de forma aislada y las comunicaciones son a través de protocolos de promps.

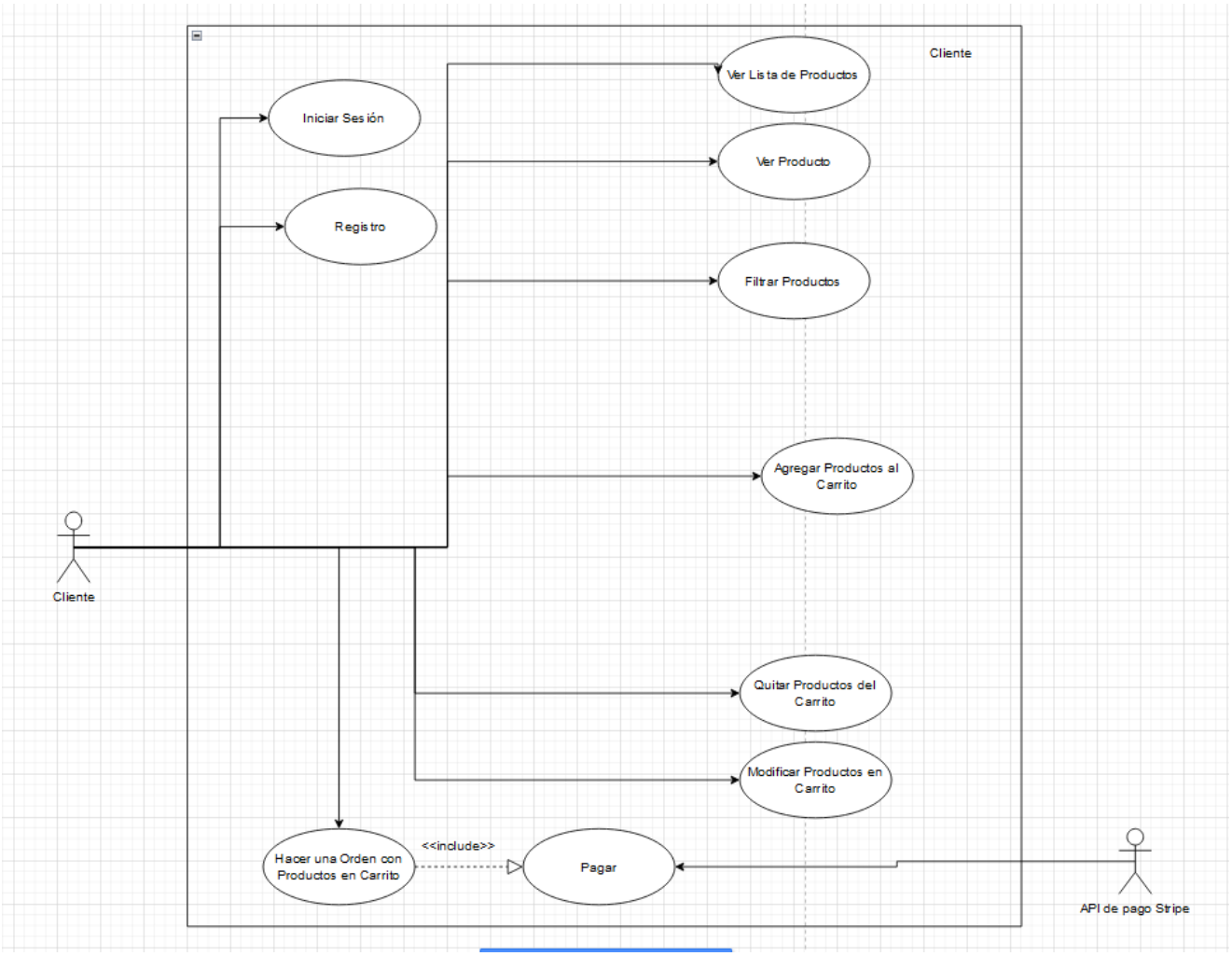
## 2.2 Vista de casos de uso



Casos de uso para los administradores

El superadmin puede realizar un registro de administrador y eliminar administradores, en un apartado del sistema donde tenga un listado para visualizarlos, esta sección solamente puede ser accedida por el superadmin al dar click en 'Users' en el dashboard. Por otra parte, tanto el admin o superadmin pueden crear, modificar, ver y cambiar estatus de los productos de la tienda a través de un listado, está lista aparece cuando el admin da clic en 'products' dentro de su dashboard.

Tanto el admin como el superadmin pueden hacer login usando sus respectivas credenciales registradas en el sistema.



Caso de uso para los clientes

Si el usuario quiere registrarse, el sistema le enviará una página dónde le pedirá introducir las credenciales que desea introducir, las que serían nombre de usuario, e-mail y contraseña, o también le da la opción de registrarse con sus cuentas de redes sociales. Una vez introducidas las credenciales deseadas por el cliente, el sistema le confirma que su registro fue exitoso y que ya puede iniciar sesión en el sistema.

Cuando el usuario quiere acceder a la página, el sistema le proporciona una página dónde introducirá sus credenciales registradas o acceda con su cuenta de red social previamente registrada en el sistema. El sistema, después de confirmar que las credenciales de acceso del cliente son correctas, le confirma al usuario que estas son correctas y le proporciona acceso a la parte del sistema que le corresponde.

De igual manera, un cliente tiene la posibilidad de ver un listado de los productos disponibles y cómo se encuentran registrados en el sistema. Tendrá la opción de filtrar los productos ya sea por categoría, color y precio, pudiendo ser de manera ascendente o descendente. Una vez que el cliente ve un producto de su agrado, puede darle click dónde entrará a una página con más información sobre el producto seleccionado.

Podrá consultar los diferentes colores en los que el producto está disponible e igualmente se mostrará el precio del mismo. En la misma página estará un botón para agregar el producto al carrito de compras, este carrito de compras llevará un registro de los productos seleccionados.

Cuando el cliente quiere finalizar su compra, tendrá que ir a la página del carrito de compras en donde se mostrará en detalle qué productos tiene dentro del carrito y el precio por producto. De igual manera se mostrará el precio total de todos los productos junto con los gastos de envío. Si el cliente está de acuerdo con el contenido de su carrito, podrá proceder con el pago de los productos.

## 2.3 Escenarios Arquitectónicos y Requerimientos No Funcionales Asociados

### 2.3.1 Escenarios Arquitectónicos

# Autenticación

**Escenario #1 :**

Usuario registra una nueva cuenta usando su correo electrónico o mediante los servicios disponibles de Google y Facebook.

CRUD System

High - High

| Portion of Scenario | Possible Values |
|---|---|
| Source | El usuario |
| Stimulus | El usuario ingresa datos como el nombre de la cuenta, contraseña y correo electrónico en caso de hacer un registro de usuario sin el uso de una red social externa, y posteriormente dar click en el botón 'Registrar'. En caso de usar Google o Facebook se sigue el método de registro implementado por el servicio. |
| Artifact | Módulo RegisterView.tsx |
| Environment | Operación normal |
| Response | RegisterView hace un POST request al endpoint "api/users/register" con los parámetros recibidos del usuario, ya sea por registro normal o por los APIs de Google o Facebook. Una vez que se valida, se redirecciona al usuario al homepage con su sesión iniciada y su respectivo token. |
| Measure response | Se recibe un status response 201 si el proceso terminó satisfactoriamente, o un error status 500 si hubo un problema al iniciar registro. |

**Escenario #2 :**

El usuario puede hacer login con su cuenta asociada y es redireccionado al Homepage.

CRUD System

High - High

| Portion of Scenario | Possible Values |
|---|---|
| Source | El usuario |
| Stimulus | El usuario ingresa datos como el nombre de la cuenta y contraseña en caso de hacer un registro de usuario sin el uso de una red social externa, y posteriormente dar click en el botón 'Login'. En caso de usar Google o Facebook se sigue el método de registro implementado por el servicio. |
| Artifact | Módulo LoginView.tsx |
| Environment | Operación normal |
| Response | LoginView hace un POST request al endpoint "api/users/login" con los parámetros recibidos del usuario, ya sea por registro normal o por los APIs de Google o Facebook. Una vez que se valida, se redirecciona al usuario al homepage con su sesión iniciada y su respectivo token. |
| Measure response | Se recibe un status response 200 si el proceso terminó satisfactoriamente, o un error status 500 si hubo un problema al iniciar login. |

# Carrito

**Escenario #3:**

Usuario agrega un producto al carrito de compras.

CRUD Cart

High-High

| Portion of Scenario | Possible Values |
|---|---|
| Source | El usuario |
| Stimulus | Al darle click al botón de “Add to cart” dentro de ProductView.tsx |
| Artifact | ProductView.tsx |
| Environment | Operación normal |
| Response | Al darle clic al botón, se guarda el producto y la cantidad de este hacia el carrito. En el símbolo del carrito que se encuentra en el componente de la Navbar.tsx, se verá actualizado con la nueva cantidad de productos agregados. |
| Measure response | Número actualizado de productos segundos después de darle clic al botón de “Add to cart” |

**Escenario #4:**

Usuario elimina un producto de su carrito de compras.

CRUD Cart

High - High

| Portion of Scenario | Possible Values |
|---|---|
| Source | Usuario |
| Stimulus | El usuario selecciona la opción de eliminar un artículo de su carrito |
| Artifact | CartView.tsx |
| Environment | Operación Normal |
| Response | Cuando se registra el click del botón se elimina el componente visual del producto en el carrito, se quita del registro de la orden y se reduce el total del precio del carrito en la cantidad del producto eliminado. |
| Measure response | Se actualiza la vista de los productos y el precio de la orden. |

## Seguridad - Administración de tokens

**Escenario #5:**

Se genera el token cuando el usuario ha hecho login satisfactoriamente y se mantiene durante la sesión.

Generate Token

Medium - High

| Portion of Scenario | Possible Values |
|---|---|

| | |
|---|---|
| Source | Sistema |
| Stimulus | Un usuario hace login satisfactoriamente |
| Artifact | En el módulo LoginView.tsx |
| Environment | Operación normal |
| Response | Se crea un token de la sesión que servirá para las peticiones de las sesiones. |
| Measure response | En caso de hacer login satisfactoriamente, se redirecciona a la view de Homepage con el token generado. De lo contrario se enviara un código de estatus 500 sin tener un token de sesión. |

**Escenario #6:**

User realiza una acción (ya sea user, admin o superadmin) y se verifica el token para validar el nivel de privilegio de la sesión.

Verify Token

Medium - High

| Portion of Scenario | Possible Values |
|---|---|
| Source | Sistema |
| Stimulus | Al hacer una operación la cual solo cierto tipo de usuario tenga permiso a realizarla |
| Artifact | En el view actual |

| | |
|---|---|
| Environment | Operación normal |
| Response | Se valida o invalida la operación del usuario dependiendo del rol que tenga y de la operación que realice. |
| Measure response | Si es una operación inválida se envía un mensaje con la razón por la cual no pueda hacerla. Si es válida se envía un status con un código 200. |

**Escenario #7:**
El usuario puede visualizar los productos en su carrito de compras durante el checkout, y antes de confirmar puede remover productos del carrito.

Método Pago

High - Medium

| Portion of Scenario | Possible Values |
|---|---|
| Source | El usuario |
| Stimulus | Al darle click al icono carrito para tener una vista detallada del mismo. |
| Artifact | En la view que se encuentre el usuario |
| Environment | Operación normal |
| Response | Se redirecciona a la view de CartView en donde se mostrarán los productos y el desglose del precio total. |
| Measure response | Se hace la redirección máximo 1 segundo. |

# Usuario - navegación

**Escenario #8:**
El usuario puede navegar y filtrar productos por categoría (color, tipo).

User Views

Medium - High

| Portion of Scenario | Possible Values |
|---|---|
| Source | El usuario |
| Stimulus | El usuario selecciona múltiples filtros en los filtros ‘color’ y ‘type’ ProductListView.tsx |
| Artifact | ProductListView.tsx |
| Environment | Operación normal |
| Response | ProductListView pasa los params de filters al component Products.tsx, Products.tsx hace un GET request a “api/products?$params” y este se encarga de actualizar la lista de productos. |
| Measure response | Se recibe un status response 200 si el proceso terminó satisfactoriamente, o un error status 500. |

**Escenario #9:**
El usuario podrá ver más detalles de un producto al darle click en su imagen.

User Views

Medium - High

| Portion of Scenario | Possible Values |
|---|---|
| Source | El usuario |
| Stimulus | Al darle click a la imágen del producto en ProductListView.tsx |
| Artifact | ProductListView.tsx |
| Environment | Operación normal |
| Response | El component Products.tsx se encarga de hacer un redirect al user hacia el path /product/:id y se carga el view ProductView.tsx |
| Measure response | El usuario es redireccionado sin errores. |

**Escenario #10:**
El usuario puede reordenar la lista de productos por precio y fecha de publicación de producto.

User Views

Medium - High

| Portion of Scenario | Possible Values |
|---|---|
| Source | El usuario |
| Stimulus | El usuario selecciona una opción de filtro en “Sort Products” en ProductListView.tsx |

| | |
|---|---|
| Artifact | ProductListView.tsx |
| Environment | Operación normal |
| Response | ProductListView pasa los params de filters al component Products.tsx, Products.tsx hace un GET request a “api/products?$params” y este se encarga de actualizar la lista de productos. |
| Measure response | Se recibe un status response 200 si el proceso terminó satisfactoriamente, o un error status 500. |

## Usuario - admin

**Escenario #11:**
El superadmin y admin puede crear productos que se van a persistir en la base de datos.
CRUD Admin
High - High

| Portion of Scenario | Possible Values |
|---|---|
| Source | El admin o superadmin |
| Stimulus | Ingresa la información válida para dar de alta a un nuevo producto y da click en ‘create’. |
| Artifact | NewProduct.tsx |
| Environment | Operación normal |
| Response | NewProduct.tsx hace un POST request al endpoint “api/products/” con los |

| | |
|---|---|
| | parámetros recibidos. Una vez que se validan, se redirecciona al view ProductList.tsx donde puede ver el producto creado. |
| Measure response | Se recibe un status response 200 si el proceso terminó satisfactoriamente, o un error status 500 si hubo un problema al crear el producto. |

**Escenario #12:**
El superadmin y admin puede modificar productos, y las modificaciones son actualizadas en la base de datos.

CRUD Admin

High - High

| Portion of Scenario | Possible Values |
|---|---|
| Source | El admin o superadmin |
| Stimulus | Ingresa la información válida para modificar un producto y da click en 'Update'. |
| Artifact | Product.tsx |
| Environment | Operación normal |
| Response | Product.tsx hace un PUT request al endpoint "api/products/:id" con los parámetros recibidos. Una vez que se valida, se redirecciona al view ProductList.tsx donde puede ver el producto actualizado. |
| Measure response | Se recibe un status response 200 si el |

| | |
|---|---|
| | proceso terminó satisfactoriamente, o un error status 500 si hubo un problema al modificar el producto. |

**Escenario #13:**
El superadmin y admin puede ver todos los productos disponibles en la base de datos.

CRUD Admin

High - High

| Portion of Scenario | Possible Values |
|---|---|
| Source | El admin o superadmin |
| Stimulus | El superadmin da click en 'Products' desde el Homepage. |
| Artifact | ProductList.tsx |
| Environment | Operación normal |
| Response | ProductList.tsx hace un GET request al endpoint "api/products/". Una vez que se valida, se cargan todos los productos en el componente Product. |
| Measure response | Se recibe un status response 200 si el proceso terminó satisfactoriamente, o un error status 500 si hubo un problema al modificar el producto. |

**Escenario #14:**
El superadmin y admin pueden deshabilitar (sin borrar de base de datos) productos.

CRUD Admin

High - High

| Portion of Scenario | Possible Values |
|---|---|
| Source | El admin o superadmin |
| Stimulus | Da click en el icono de deshabilitar a un producto seleccionado |
| Artifact | ProductList.tsx |
| Environment | Operación normal |
| Response | ProductList hace un UPDATE request al endpoint “api/products/:id” y cambia el parámetro “available” a ‘False’. |
| Measure response | Se recibe un status response 200 un mensaje que dice “Product unsubscribed” si el proceso terminó satisfactoriamente, o un error status 500 si hubo un problema. |

**Escenario #15:**
El superadmin es capaz de crear cuentas de administrador.

CRUD Admin

High - High

| Portion of Scenario | Possible Values |
|---|---|
| Source | El superadmin |
| Stimulus | Ingresa la información válida para dar de alta a un nuevo administrador y da click en registrar. |

| | |
|---|---|
| Artifact | NewUser.tsx |
| Environment | Operación normal |
| Response | UserView.tsx hace un POST request al endpoint “api/auth/register” con los parámetros recibidos del superadmin. Una vez que se validan, se redirecciona al view UserList.tsx donde puede ver el admin creado. |
| Measure response | Se recibe un status response 201 si el proceso terminó satisfactoriamente, o un error status 500 si hubo un problema al crear el usuario. |

**Escenario #16:**
El superadmin es capaz de deshabilitar (sin borrar de base de datos) cuentas de administrador.

CRUD Admin

High - High

| Portion of Scenario | Possible Values |
|---|---|
| Source | El superadmin |
| Stimulus | Da click en el icono de deshabilitar a un usuario seleccionado |
| Artifact | UserList.tsx |
| Environment | Operación normal |
| Response | UserList hace un UPDATE request al endpoint “api/users/:id” y cambia el parámetro “available” a ‘False’. |

| | |
|---|---|
| Measure response | Se recibe un status response 200 un mensaje que dice “User unsubscribed” si el proceso terminó satisfactoriamente, o un error status 500 si hubo un problema. |

## 2.3.1 Requerimientos No Funcionales Asociados

**Tamaño y Rendimiento**

| | |
|---|---|
| **RNF 0001** | **Las solicitudes por el usuario se deben de completar en máximo 10 segundos** |
| **Tiempo de solución de solicitudes** | |
| La aplicación debe de resolver todas las solicitudes realizadas por el usuario en un máximo de 10 segundos, en caso de no ser así se debe de advertir al usuario que se tiene un problema con el rendimiento del backend de la aplicación | |
| **RNF 0002** | **El sistema deberá soportar al menos 100 usuarios concurrentes en la página** |
| **Número de usuarios simultáneos.** | |
| La interfaz permite que sea ejecutada por más de 100 usuarios concurrentes en la página. Con esto se debe de alcanzar una continuidad en el servicio otorgado por el sistema para los usuarios. En caso de no ser posible soportar la cantidad de usuarios especificada, se debe de notificar al usuario que se están teniendo problemas de conexión. | |

| | |
|---|---|
| **RNF 0003** | **El sistema debe ser capaz de correr de forma eficiente en dispositivos móviles** |
| **Eficiencia del sistema en dispositivos móviles** | |

| |
|---|
| La interfaz permite que sea ejecutada por dispositivos móviles sin problema alguno, debe ser ágil en la transición de pantallas para un funcionamiento fluido. De la misma manera esta debe ser adaptada para que el usuario se le haga posible navegar de manera natural la plataforma. |

| | |
|---|---|
| **RNF 0004** | **El sistema debe ser capaz de manejar 100 transacciones por segundo.** |
| **Manejo de transacciones concurrentes** | |
| La interfaz debe ser capaz de manejar de manera ágil y eficiente 100 transacciones por segundo y actualizar en tiempo real estas transacciones en la base de datos en máximo 10 seg por cada una. Esto para otorgar una respuesta al usuario que su transacción fue correctamente realizada. | |

### Usabilidad

| | |
|---|---|
| **RNF 0005** | **El tiempo de aprendizaje del sistema por un usuario deberá ser menor a 4 horas** |
| **Aprendizaje del sistema por usuario** | |
| El sistema debe ser capaz de enseñar al usuario a utilizar el sistema en menos de 4 horas. De la misma manera el sistema debe ser intuitivo para que el usuario se adapte a las funcionalidades del mismo de manera ágil. | |

| | |
|---|---|
| **RNF 0006** | **El sistema debe de poseer interfaces gráficas bien estructuradas y bien implementadas.** |
| **Estructuralización e implementación correcta** | |

| |
|---|
| El sistema debe de ser estructurado de manera correcta para que el usuario pueda conocer y acostumbrarse a la navegación dentro de la página añadiendo funciones necesarias a la página. Al implementar de manera correcta esta estructuración el mantenimiento del sistema es más ágil. |

| | |
|---|---|
| **RNF 0007** | **La página debe ser responsiva para múltiples dispositivos.** |
| **Responsividad en todos los dispositivos** | |
| El sistema debe ser capaz de adaptarse a cualquier resolución de dispositivo, ya sea algún dispositivo móvil o una PC. Esta responsividad debe de mostrarse de manera adecuada y adaptada a las necesidades del usuario. | |

### Fiabilidad

| | |
|---|---|
| **RNF 0008** | **La interfaz debe estar disponible el 99.9%** |
| **Alta disponibilidad** | |
| La dependencia de conectividad a través del proveedor de servicios de Internet es inminente, aun cuando la aplicación tenga la capacidad de recuperarse ante una caída y continuar con su proceso. | |

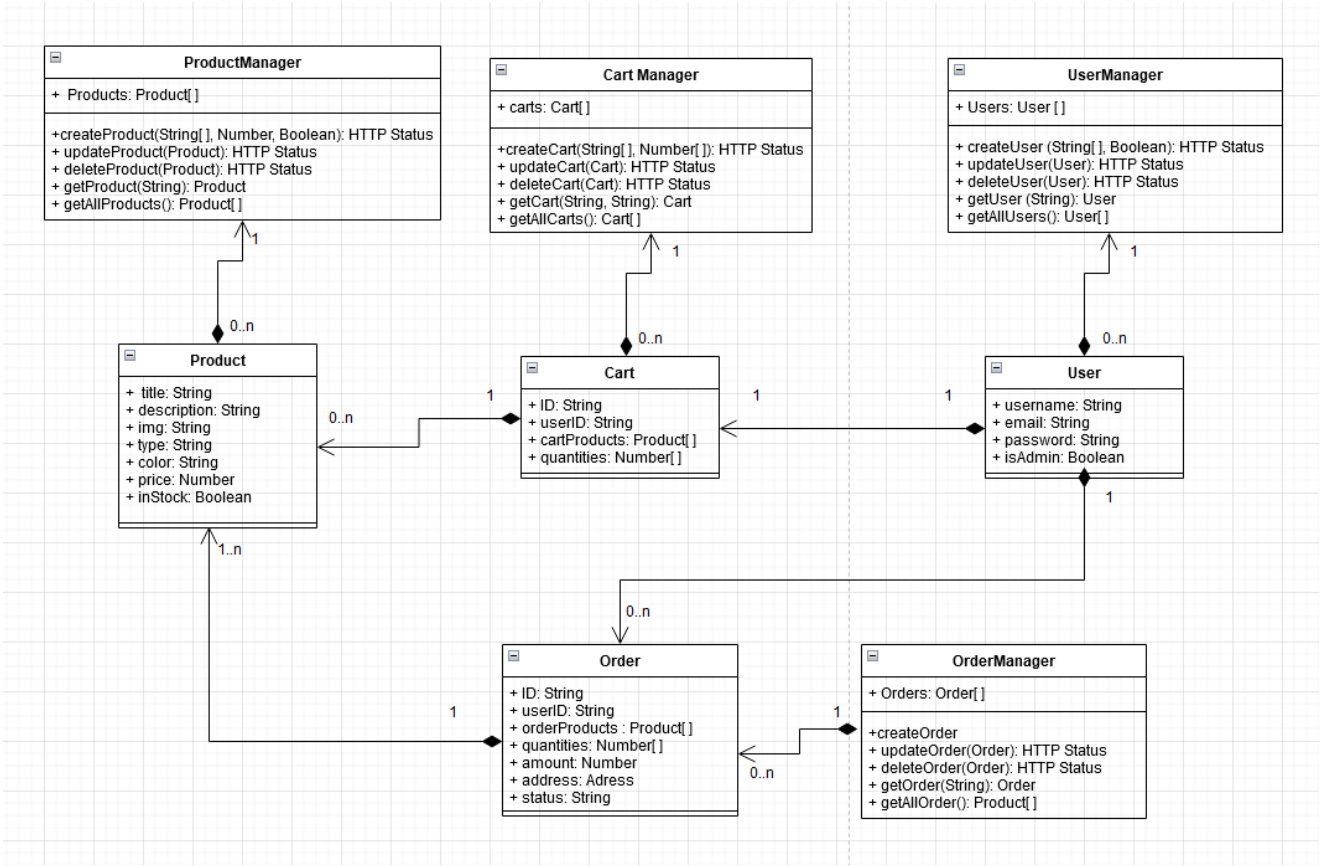
### Seguridad

| | |
|---|---|
| **RNF 0009** | **Los pagos y su información deben ser encriptados mediante la API de Stripe.** |
| **Encriptación de información mediante Stripe** | |

| |
|---|
| Al realizar algún pago o movimiento que tenga que usar datos del usuario, esta debe ser asegurada con el API de Stripe para un funcionamiento adecuado y seguro de estos pagos. Esta herramienta nos otorga formas de encriptar la información del usuario y debe ser adaptada a las necesidades de la página. |

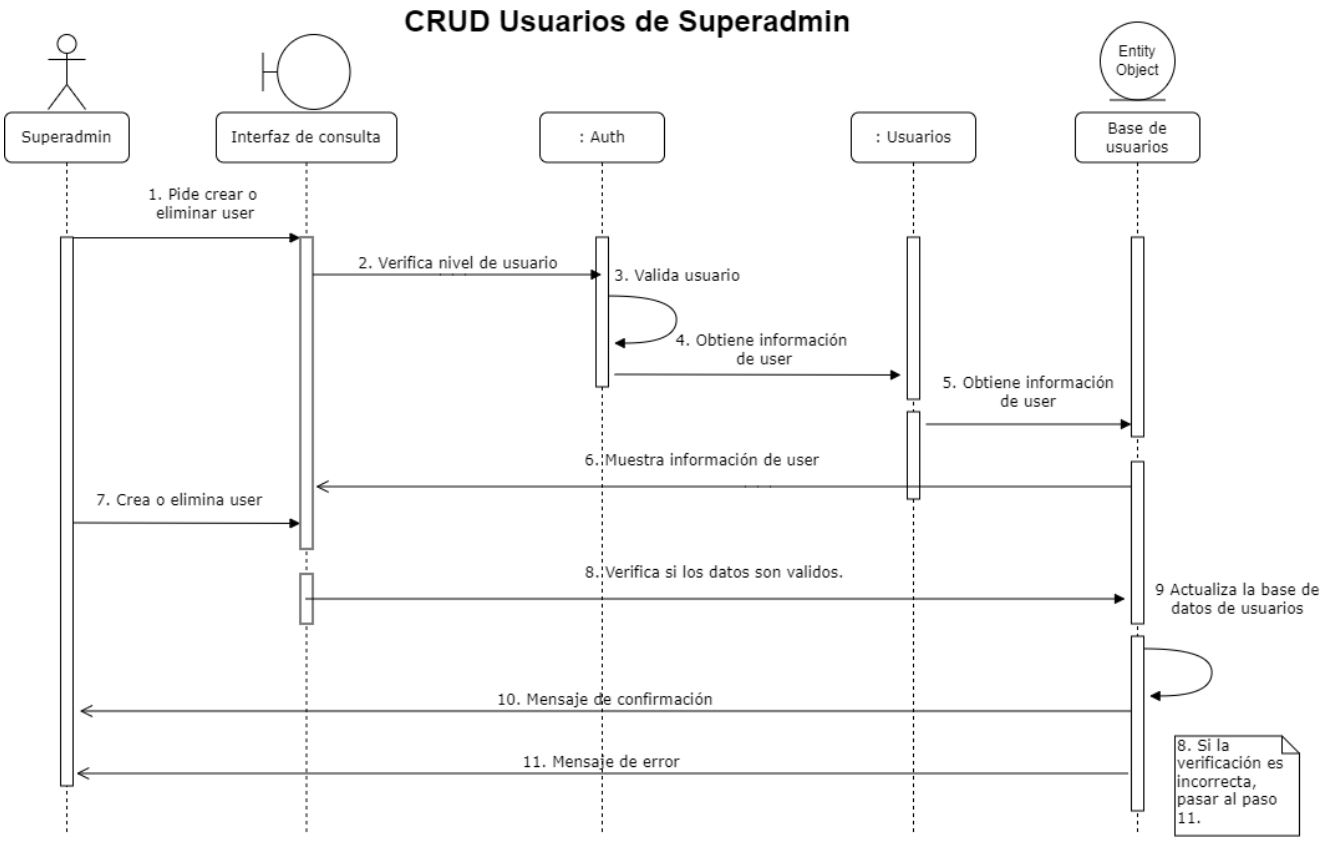
| | |
|---|---|
| **RNF 0010** | **Las contraseñas para la creación y autenticación de usuarios deberá ser encriptada.** |
| **Encriptación de contraseñas al iniciar sesión** | |
| Al realizar alguna creación o modificación de contraseña del usuario, esta debe ser encriptada para que el sistema tenga la seguridad de evitar filtraciones de información. De la misma manera al autenticar un token debe ser acoplado al usuario para el uso seguro de la plataforma. | |

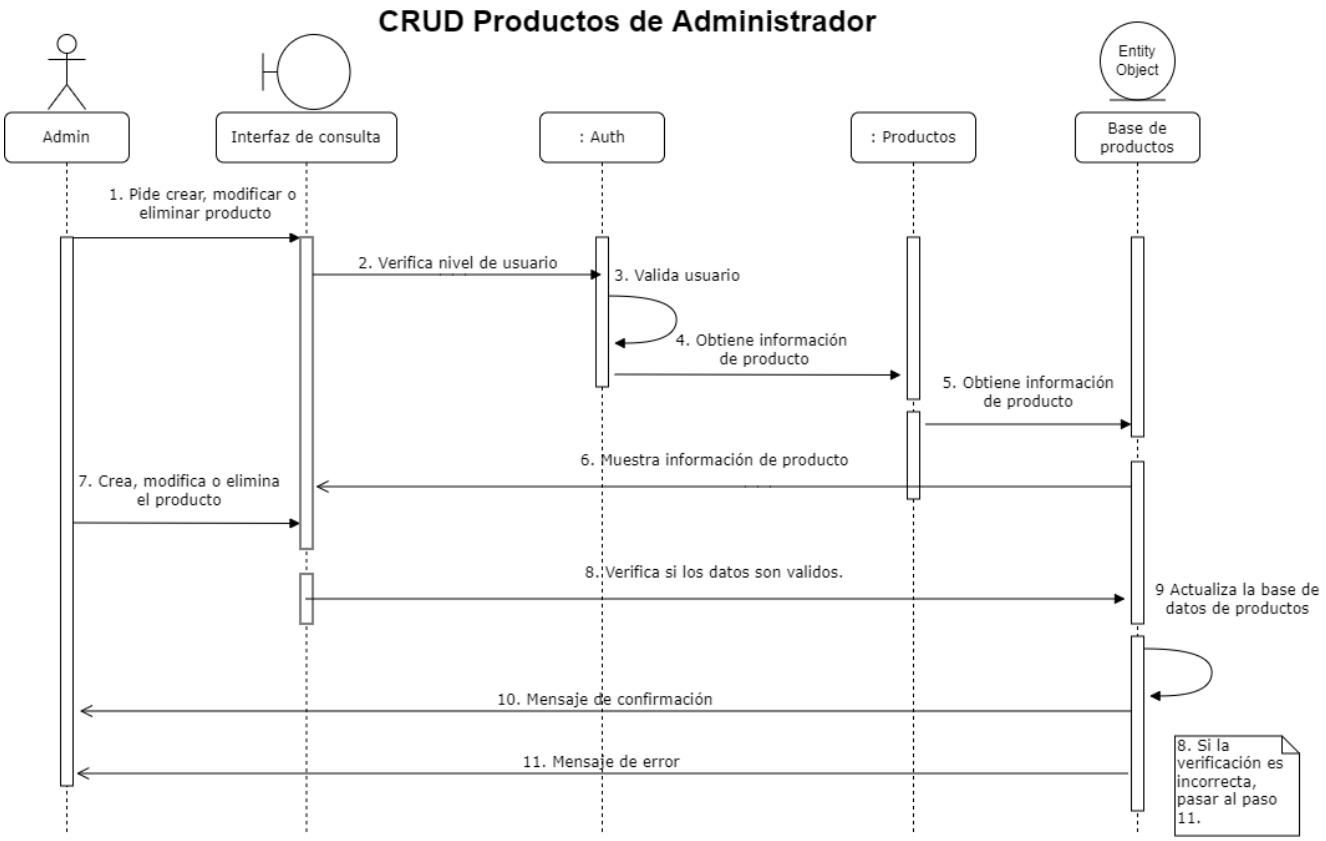
## 2.3 Vista lógica

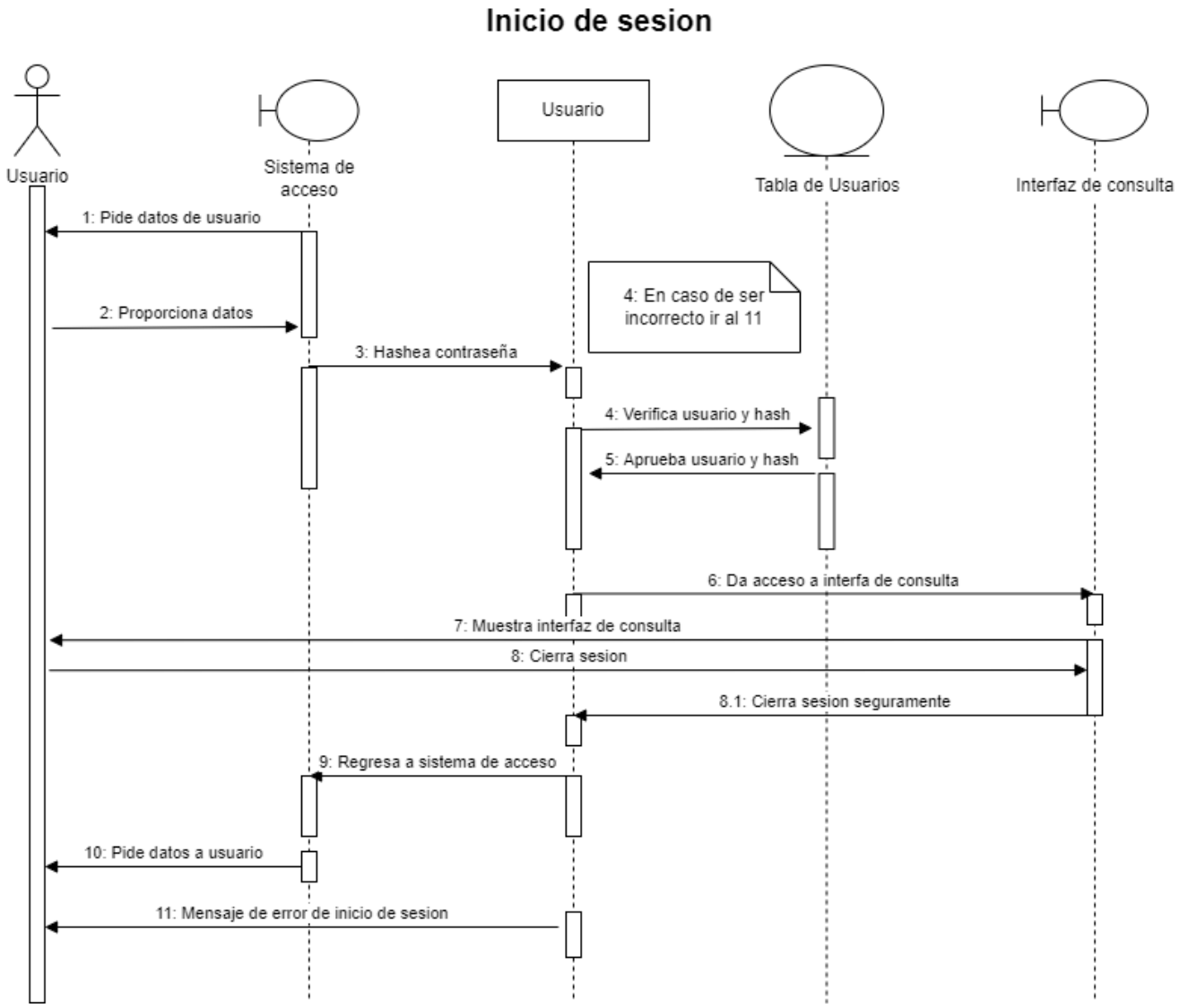


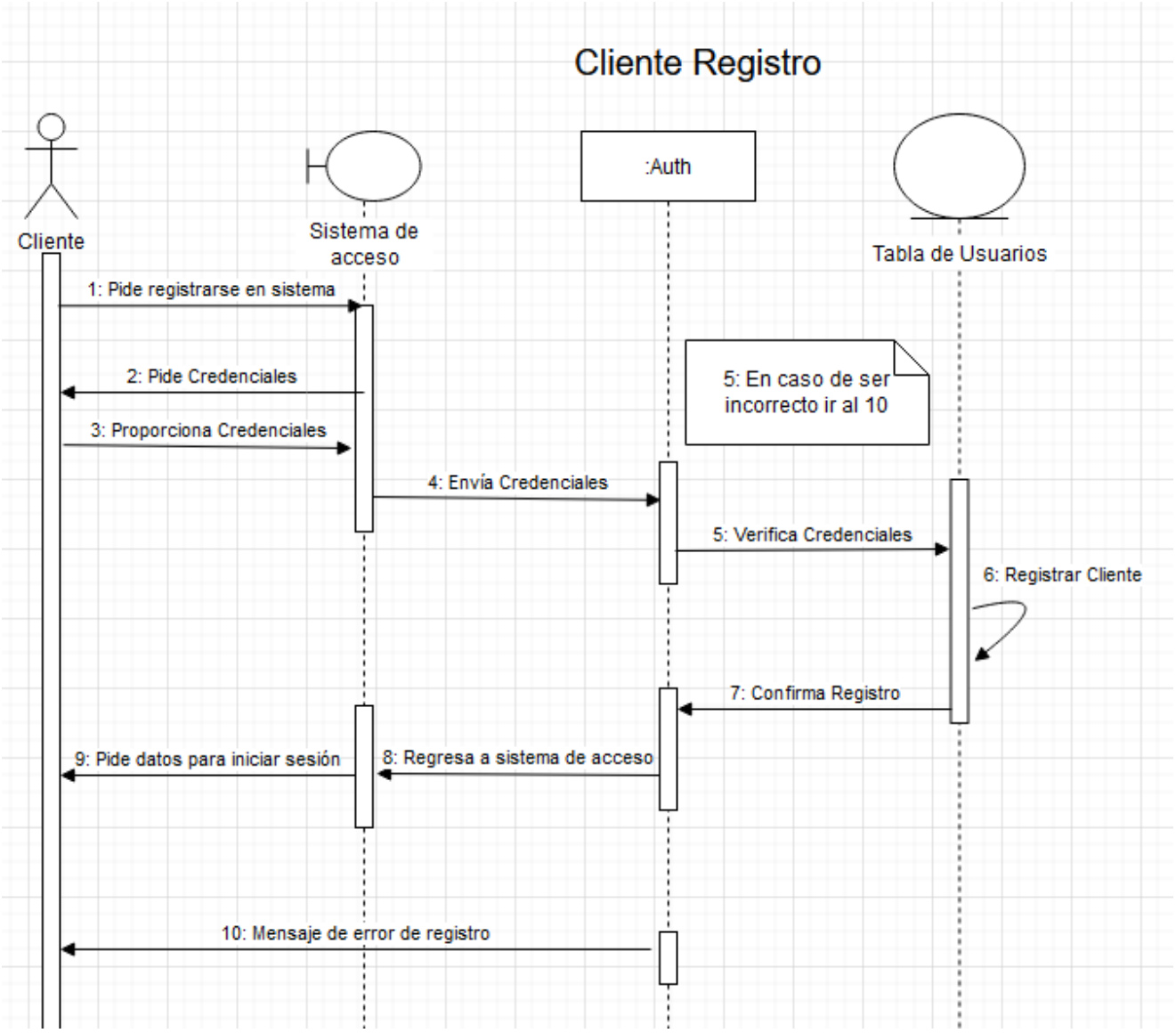

## 2.4 Vista de proceso

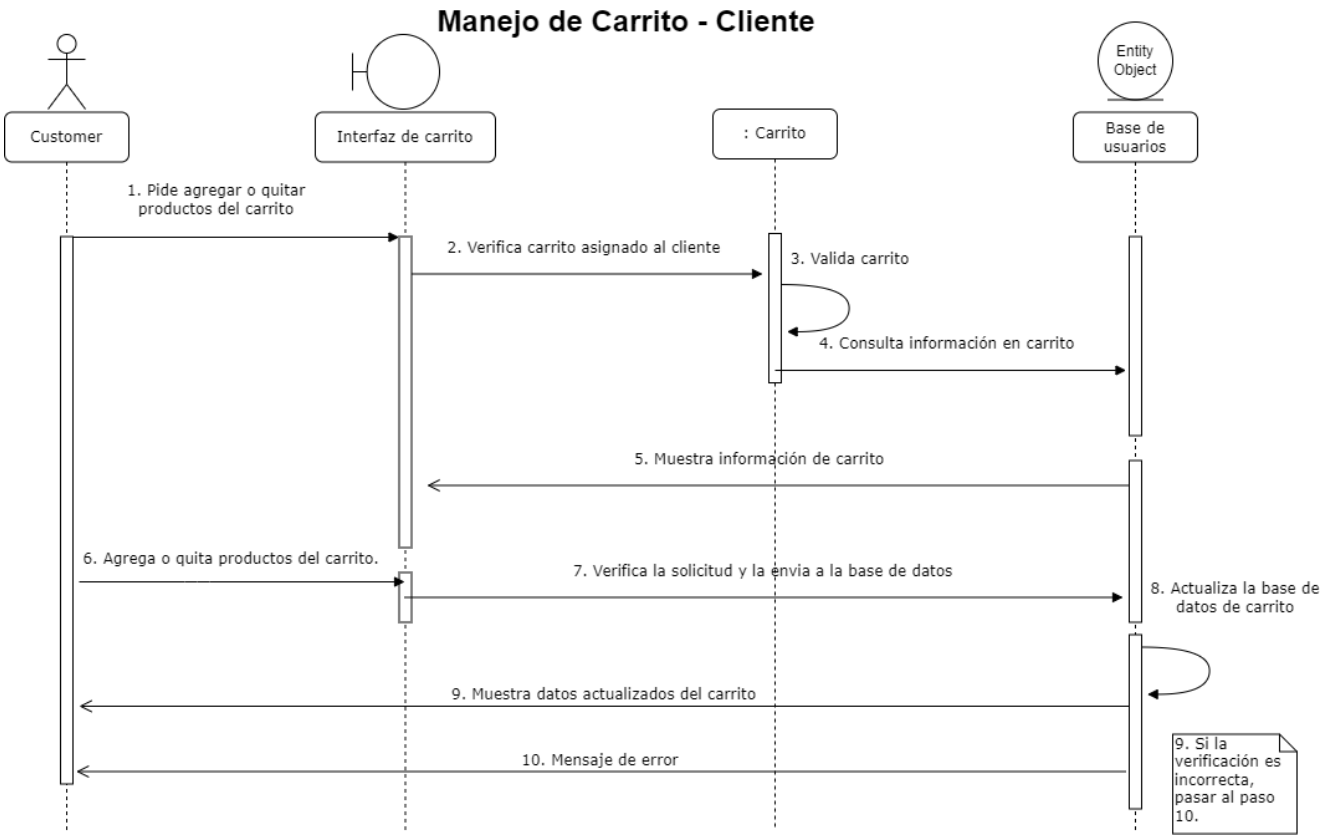
Utilizar un diagrama de secuencia o de actividad para mostrar las interacciones

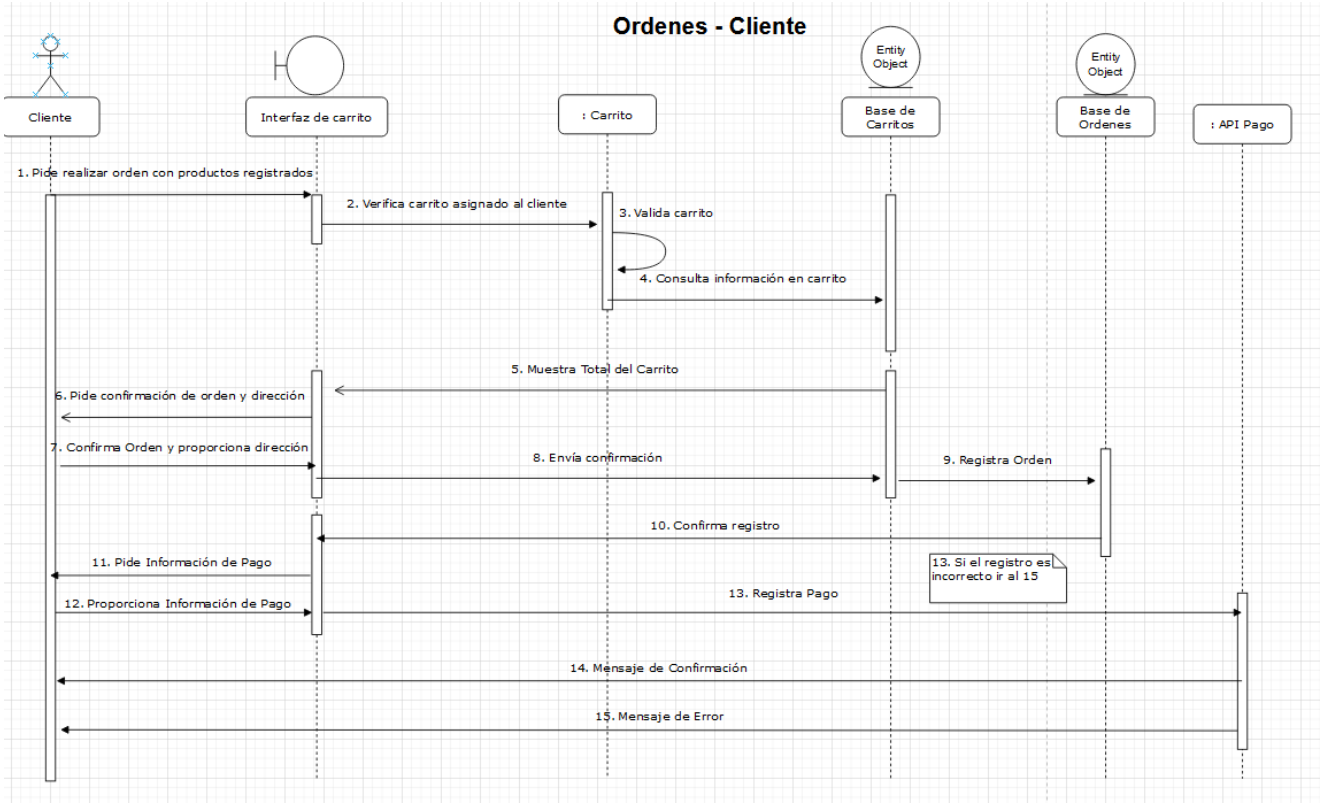

CRUD Usuarios de Superadmin
Superadmin
Interfaz de consulta
: Auth
: Usuarios
Entity Object
Base de usuarios
1. Pide crear o eliminar user
2. Verifica nivel de usuario
3. Valida usuario
4. Obtiene información de user
5. Obtiene información de user
6. Muestra información de user
7. Crea o elimina user
8. Verifica si los datos son validos.
9 Actualiza la base de datos de usuarios
10. Mensaje de confirmación
11. Mensaje de error
8. Si la verificación es incorrecta, pasar al paso 11.


CRUD Productos de Administrador
Admin
Interfaz de consulta
: Auth
: Productos
Entity Object
Base de productos
1. Pide crear, modificar o eliminar producto
2. Verifica nivel de usuario
3. Valida usuario
4. Obtiene información de producto
5. Obtiene información de producto
6. Muestra información de producto
7. Crea, modifica o elimina el producto
8. Verifica si los datos son validos.
9 Actualiza la base de datos de productos
10. Mensaje de confirmación
11. Mensaje de error
8. Si la verificación es incorrecta, pasar al paso 11.


Inicio de sesion
Usuario
Sistema de acceso
Usuario
Tabla de Usuarios
Interfaz de consulta
1: Pide datos de usuario
2: Proporciona datos
3: Hashea contraseña
4: En caso de ser incorrecto ir al 11
4: Verifica usuario y hash
5: Aprueba usuario y hash
6: Da acceso a interfa de consulta
7: Muestra interfaz de consulta
8: Cierra sesion
8.1: Cierra sesion seguramente
9: Regresa a sistema de acceso
10: Pide datos a usuario
11: Mensaje de error de inicio de sesion

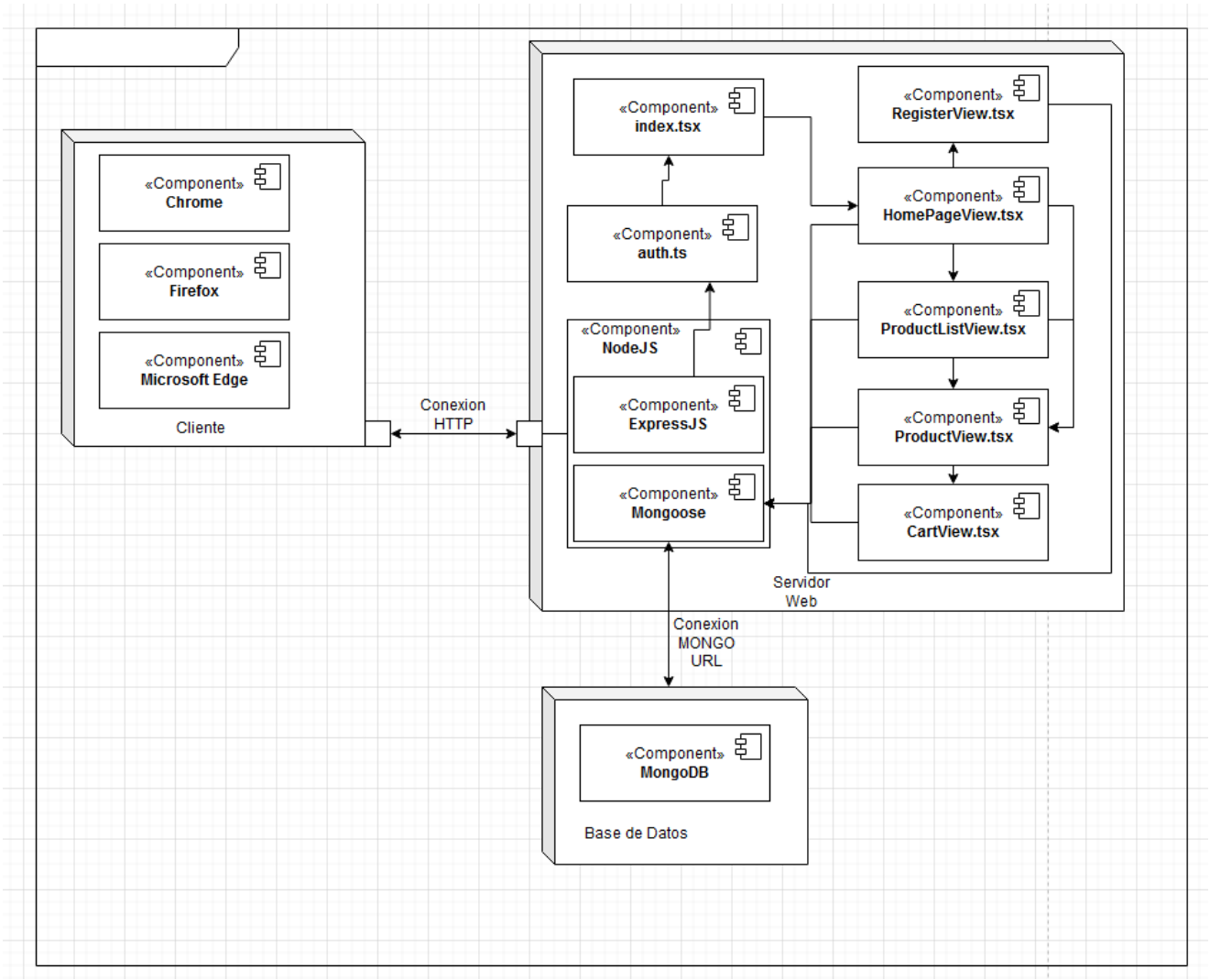

Cliente Registro
Cliente
Sistema de acceso
:Auth
Tabla de Usuarios
1: Pide registrarse en sistema
2: Pide Credenciales
3: Proporciona Credenciales
4: Envía Credenciales
5: En caso de ser incorrecto ir al 10
5: Verifica Credenciales
6: Registrar Cliente
7: Confirma Registro
8: Regresa a sistema de acceso
9: Pide datos para iniciar sesión
10: Mensaje de error de registro


Manejo de Carrito - Cliente
Customer
Interfaz de carrito
: Carrito
Entity Object
Base de usuarios
1. Pide agregar o quitar productos del carrito
2. Verifica carrito asignado al cliente
3. Valida carrito
4. Consulta información en carrito
5. Muestra información de carrito
6. Agrega o quita productos del carrito.
7. Verifica la solicitud y la envia a la base de datos
8. Actualiza la base de datos de carrito
9. Muestra datos actualizados del carrito
10. Mensaje de error
9. Si la verificación es incorrecta, pasar al paso 10.


Ordenes - Cliente
Cliente
Interfaz de carrito
: Carrito
Entity Object
Base de Carritos
Entity Object
Base de Ordenes
: API Pago
1. Pide realizar orden con productos registrados
2. Verifica carrito asignado al cliente
3. Valida carrito
4. Consulta información en carrito
5. Muestra Total del Carrito
6. Pide confirmación de orden y dirección
7. Confirma Orden y proporciona dirección
8. Envía confirmación
9. Registra Orden
10. Confirma registro
11. Pide Información de Pago
12. Proporciona Información de Pago
13. Si el registro es incorrecto ir al 15
13. Registra Pago
14. Mensaje de Confirmación
15. Mensaje de Error

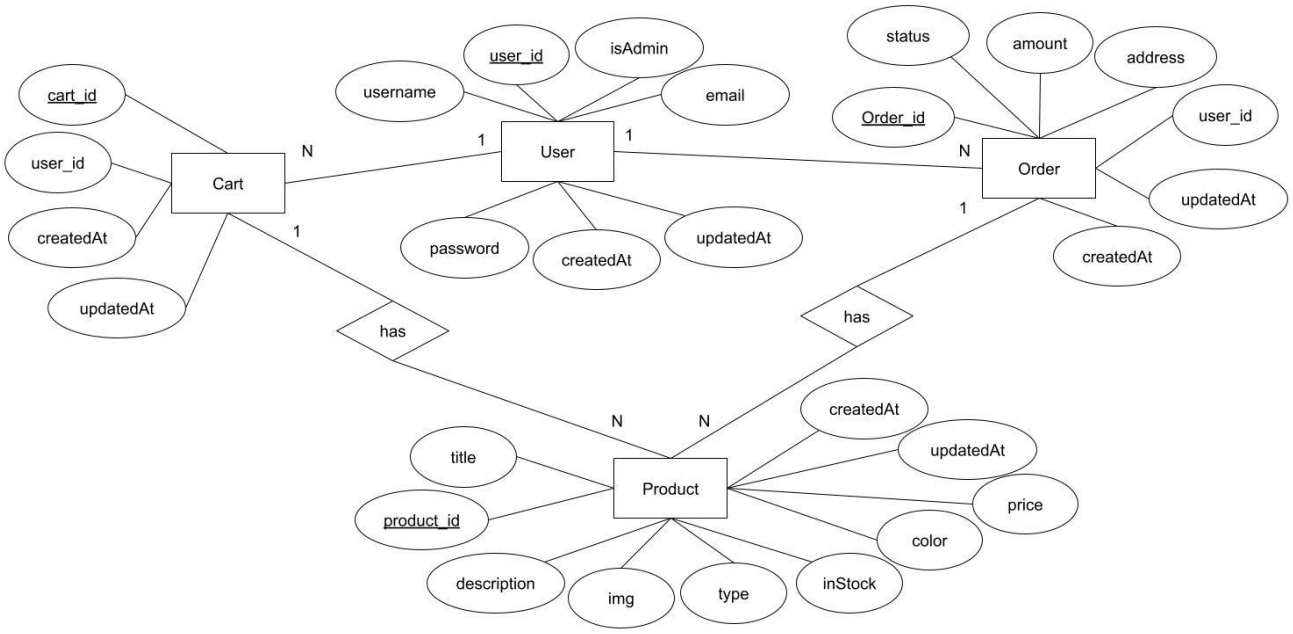
## 2.5 Vista de implementación

Descripción del diagrama de despliegue




## 2.6 Vista de datos

Incluir el diagrama entidad-relación




## 2.6.1 Base de datos

El back-end utiliza una base de datos NoSQL - MongoDB.

User

| Nombre de Columna | Tipo de Dato | Permite Null? | Descripción |
|---|---|---|---|
| username | String | No | Nombre del usuario registrado en el sistema. |
| email | String | No | Correo Electrónico del usuario registrado en el sistema. |
| password | String | No | Contraseña del usuario registrado en el sistema. |
| isAdmin | Boolean | Sí | Indica si el usuario |

| | | | |
|---|---|---|---|
| | | | está registrado como administrador si es verdadero, y si es falso indica que es un usuario regular. |

Product

| Nombre de Columna | Tipo de Dato | Permite Null? | Descripción |
|---|---|---|---|
| title | String | no | Título o nombre del producto. |
| description | String | no | Descripción del producto. |
| img | String | no | URL al path de la imágen. |
| type | String | yes | Tipo de producto, puede ser “pc-parts”, “pc-cases”, “pc-add-ons” |
| color | String | yes | Color del producto, puede ser “rgb”, “neutrals”, “pastel-colours” |
| price | Number | no | Precio del producto en dls. |
| inStock | Boolean | no | Booleano para declarar si el producto está disponible o no. |

Order

| Nombre de Columna | Tipo de Dato | Permite Null? | Descripción |
|---|---|---|---|
| userID | String | No | ID en base de datos del usuario registrado en el sistema al que corresponde esta orden. |
| products | Array de <String, Number> | Sí | Lista de <product_id, amount>, donde 'amount' es la cantidad de productos con cierto ID agregados a la orden. |
| amount | Number | No | Precio total de la orden a pagar por el usuario. |
| address | Objeto | No | Dirección a la que se enviará la orden. |
| status | String | Sí | Estado de la orden en el sistema, puede ser: "Aceptada", "En proceso" o "Cancelada" |

Cart

| Nombre de Columna | Tipo de Dato | Permite Null? | Descripción |
|---|---|---|---|
| userID | String | No | ID del usuario asignado al carrito. |
| products | Array de <String, Number> | Sí | Lista de <product_id, amount>, donde 'amount' es la |

| | | | |
|---|---|---|---|
| | | | cantidad de productos con cierto ID agregados al carrito. |

## 3. Lineamientos Arquitectónicos

## 3.1 Diseño

El diseño arquitectónico por el que optamos usar en nuestro proyecto fue el de Modelo-Vista-Controlador (MVC). Se caracteriza por contar con 3 capas o componentes distintos. Esto se hace con el objetivo de poder reutilizar código, además de poder separar las tareas. Esto hace que el desarrollo de un proyecto sea ordenado y también de fácil mantenimiento.

La estructura y fundamento es la separación del código en tres capas diferentes, acotadas por su responsabilidad, en lo que se llaman Modelos, Vistas y Controladores (Models, Views & Controllers).

En la capa de **Modelos** es donde se encuentra la lógica de negocios, la persistencia en la base de datos y en donde se manipulan los datos que se estarán utilizando en la aplicación.

En esta capa de nuestra aplicación se pueden encontrar métodos o funciones permite agregar productos al carrito de compras, así como también poder calcular el precio total de un carrito tomando en cuenta los precios individuales de los productos. Igualmente contamos con las queries que se realizan a la base de datos para recuperar los productos que se encuentren en inventario y mostrarlos en la aplicación, de la misma manera tenemos las funcionalidades de poder cambiar el estatus de un producto para indicar que no se encuentra en el inventario. En la parte de los usuarios contamos con los métodos de creación, modificación y suspension de cualquier usuario.

En la capa de **Vistas** es donde se presenta la interfaz la cual el usuario podrá interactuar y proveer de datos los cuales podrán ser usados en el modelo de la aplicación. Estas vistas pueden llegar a ser desde líneas de comando en una consola, hasta interfaces gráficas más complejas y completas en donde es más intuitivo para el usuario el manejo del sistema.

En nuestra aplicación la capa de **Vistas** fue realizada con Typescript en conjunto de React.js para poder mostrar los productos que se encuentran dados de alta en la base de datos

La capa de **Controladores** sirve como un intermediario entre la capa de **Vistas** y **Modelos.** Se encarga de regular el flujo de la información y adapta la información para que tanto el Modelo como la Vista puedan funcionar correctamente.

En nuestro proyecto esto se puede ver cómo React hace un enlace entre las vistas y los modelos, respondiendo a los mecanismos que puedan requerirse para implementar las necesidades de nuestra

aplicación. Estos controladores sirven solamente de enlace entre los modelos y las vistas para implementar las diversas necesidades del desarrollo.

Al utilizar un diseño arquitectónico como lo es MVC obtenemos varias ventajas como lo es el orden en nuestro código, así como también facilidad de mantenimiento cuando se requiere modificar o agregar alguna nueva funcionalidad.

## 3.1 Codificación

Para nuestra codificación estamos usando la metodología de notación Camell de código fuente, que se basa en que el primer carácter de todas las palabras, excepto de la primera palabra se escribe en mayúsculas y los otros caracteres en minúsculas. Intentamos usar nombres de variables entendibles y descriptivas, sin importar que tan largo sea el nombre de la variable. Esto nos ayuda a saber el uso de la variable que asignamos o el método que llamamos.

Al momento de utilizar prefijos como boolean debemos de escribir “es” en inglés “is” para entender que esta variable o método regresa un booleano. Intentamos que cada uno de nuestros métodos o clases que utilizamos deban coincidir con el nombre del archivo que asignamos y no añadir métodos de más que no tengan que ver con la clase o archivo. Esto nos ayuda a tener un control para cuando se vayan a realizar las pruebas de la funcionalidad del sistema.

En caso de los inputs de los usuarios intentamos siempre validar si el campo no está vacío y tiene los campos necesarios para continuar la operación. Debemos de asumir que cualquier cosa puede salir mal por lo que en nuestro código tenemos condiciones para asegurar el funcionamiento correcto del sistema.

En nuestra implementación de los métodos y componentes de nuestra aplicación evitamos completamente la utilización de palabras reservadas de javascript y typescript para evitar la confusión al momento de buscar y obtener los componentes que nosotros escribimos, estos están estrictamente definidos con una relación a su representación en la vida real.

## 3.2 Reutilización de Software

- **React**

React nos permite definir diferentes componentes gráficos con funcionalidad asignada que podemos codificar en archivos propios e importarlos a las diferentes páginas de nuestra aplicación que los requieran y así agilizar nuestra creación del front-end.

- **Yarn**

Esta aplicación nos permite descargar e instalar diferentes paquetes programáticos que se utilizarán para añadir funcionalidad externa a nuestra aplicación.

- **Axios**

Este paquete nos permite hacer peticiones de http desde node js de forma sencilla para poder conectarnos a nuestro Back-end con mayor facilidad y fluidez.

- **Mongoose**

Este paquete nos ayuda a traducir la funcionalidad de MongoDB a un entorno web. Esto nos permitirá crear modelos para las diferentes colecciones de este sistema de bases de datos que representarán a los datos que guardamos y también nos facilitará realizar las diferentes operaciones CRUD para manipular los datos guardados como lo necesitemos en la aplicación.

- **Express**

Este paquete nos permite manejar las peticiones dentro de nuestra aplicación y el ruteo del mismo como middleware. Es algo esencial para poder realizar nuestra aplicación y funcione de manera correcta para el cliente con las rutas establecidas.

- **Jsonwebtoken**

Este paquete nos sirve para manejar nuestras sesiones con más seguridad y encriptar nuestros datos del usuario cuando inicia sesión o realiza alguna modificación para mandar a nuestra base de datos.