Forma

Descripción generada automáticamente con confianza media

**DOCUMENTO DE ARQUITECTURA DE SOFTWARE PARA EL SISTEMA DE COMERCIO ELECTRÓNICO LAMBDA STORE (SCELS)**

Forma

Descripción generada automáticamente con confianza media

|  |  |  |
| --- | --- | --- |
| **HISTÓRICO DE CAMBIOS** | | |
| **Versión** | **Fecha** | **Cambios realizados** |
| 01 | 01/06/2022 | Emisión Inicial |
| 02 | 02/06/2022 | Definición del documento |
| 03 | 04/06/2022 | Alcance del documento |
| 04 | 08/06/2022 | Ajustes en la definición del documento |
| 05 | 09/06/2022 | Especificación de atributos de calidad |
| 06 | 10/06/2022 | Definición de arquitecturas |
| 07 | 10/06/2022 | Agrega decisión arquitectónica final |

|  |  |  |
| --- | --- | --- |
| **Elaboró** | **Revisó** | **Aprobó** |
| * Araccelli Zevallos * Guillermo Savero | * Leonardo Ormeño | * Duany Lirio |

**Contenido**

[**Introducción 4**](#_Toc105750583)

[**Propósito 4**](#_Toc105750584)

[**Alcance 4**](#_Toc105750585)

[**Atributos de calidad 4**](#_Toc105750586)

[**Estilos arquitectónicos 5**](#_Toc105750587)

[**Control de subsistemas 5**](#_Toc105750588)

[**Diagrama de Contexto Arquitectónico 5**](#_Toc105750589)

[**Estructura arquitectónica general 6**](#_Toc105750590)

[**Decisión arquitectónica 6**](#_Toc105750591)

# Introducción

El presente documento provee una vista de alto nivel de la arquitectura del Sistema de Comercio Electrónico Lambda Store (SCELS) y da una vista general del proceso de desarrollo.

# Propósito

El Documento de Arquitectura de Software presenta la arquitectura a través de ilustraciones, cada una de las cuales ilustra un aspecto en particular del software desarrollado. Se pretende de esta forma que el documento brinde al lector una visión global y comprensible del diseño general del sistema desarrollado.

# Alcance

En el documento se incluyen los aspectos fundamentales de la arquitectura y se omiten aquellos puntos que no se consideren pertinentes como es el caso de los procesos específicos del sistema.

# Atributos de calidad

* **Fiabilidad**

El sistema debe ser robusto: El software debe mostrar una alta tolerancia a fallos. El sistema debe tardar un máximo de 10 minutos para la recuperación de un fallo de caída total, en el 95% de las ocasiones.

* **Usabilidad**

El tiempo de aprendizaje del sistema por un usuario debe ser menor a cuatro horas. El sistema debe contar con manuales de usuario debidamente estructurados.

* **Eficiencia**

El sistema debe ser capaz de operar adecuadamente con hasta 1000 usuarios con sesiones concurrentes. El sistema debe ser capaz de procesar 900 transacciones por segundo

* **Escalabilidad**

La aplicación mantendrá su buen funcionamiento y rendimiento tras los cambios o crecimiento que pueda aplicarse a lo largo de su ciclo de vida.

* **Portabilidad**

El sistema podrá ser ejecutado de manera transparente en diferentes plataformas en caso fuera necesaria la migración de sistema operativo, base de datos o tecnología de equipos.

* **Seguridad**

El sistema guardará las contraseñas de los usuarios encriptándolas en la base de datos.

# Estilos arquitectónicos

Las aplicaciones web son un tipo especial de aplicación cliente/servidor, donde el cliente (el navegador o explorador) como el servidor (servidor web) y el protocolo de comunicación (HTTP) están estandarizados y/o han de ser creados por el programador de la aplicación.

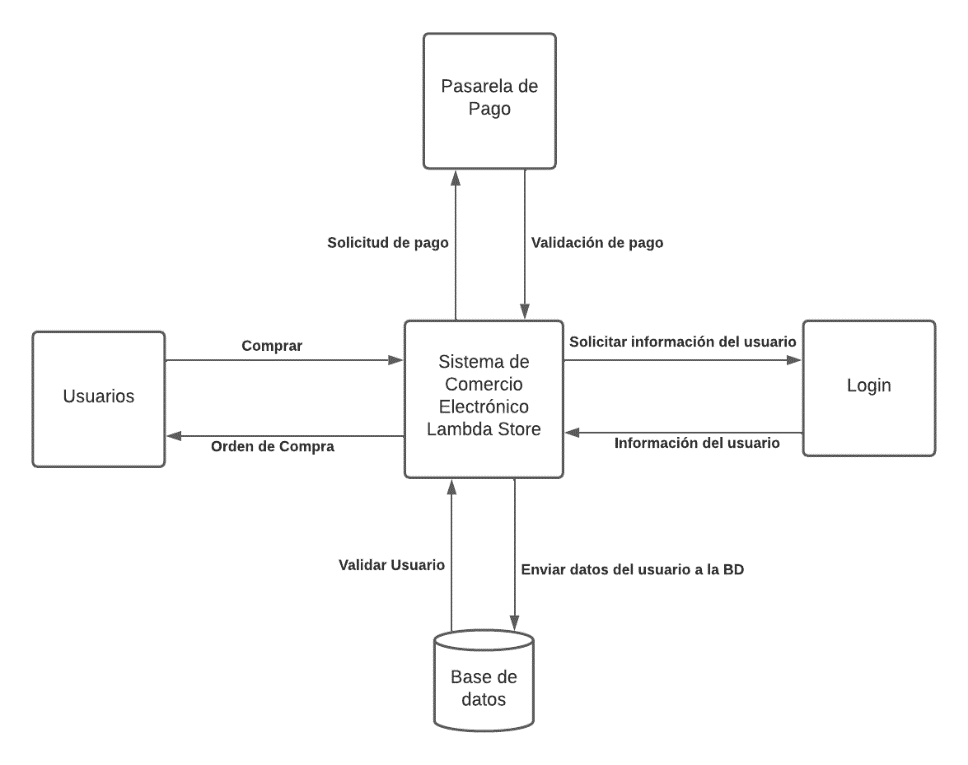
La idea principal de este modelo es una arquitectura de dos capas (la capa cliente y la capa de servidor); sin embargo, este modelo se puede extender a una arquitectura de tres capas, donde la interfaz del usuario, la lógica de negocio y las bases de datos se separan por componentes.

## Control de subsistemas

En esta arquitectura, la capacidad de proceso está repartida entre los clientes y los servidores, aunque son más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema.

Por lo tanto, se acordó que el estilo de control de subsistemas más adecuado sería el centralizado ya que el subsistema usuario necesita la capacidad de controlar los otros subsistemas.

# Diagrama de Contexto Arquitectónico



# Estructura arquitectónica general

Diagrama

Descripción generada automáticamente

# Decisión arquitectónica

|  |  |
| --- | --- |
| **Problema** | Se requiere un sistema de comercio electrónico en el que los clientes de Lambda Store puedan comprar productos desde cualquier navegador web. Además, se debe permitir gestionar este e-commerce desde otro sistema web de uso interno y exclusivo para Lambda Store. |
| **Decisión** | Arquitectura de tres capas |
| **Estado** | Aprobado |
| **Supuestos** | Calendario:   * Revisar cronograma del proyecto (SCELS-CP)   Herramientas usadas (**MEAN STACK**):   * HTML/CSS * Angular * NodeJS * ExpressJS * MongoDB * Visual Studio Code |
| **Restricciones** | Es mucho más difícil programar y probar el software que en una arquitectura de dos niveles porque tienen que comunicarse más dispositivos para terminar la transacción de un usuario, y esto supone también una mayor cantidad de tráfico en la red. |
| **Otras propuestas** | 1. Cliente / Servidor   Client-Server Architecture | EN.601.421: Object-Oriented Software  Engineering (OOSE)   1. Multicapa de 4 capas   Alt Text   1. Multinivel   Three-tier architecture overview - AWS Serverless Multi-Tier Architectures  with Amazon API Gateway and AWS Lambda |
| **Argumento** | La ventaja fundamental de una arquitectura n-capas comparado con una arquitectura de dos niveles (o una tres-capas con una de dos niveles) es que separa hacia fuera el proceso, eso ocurre para mejorar el balance la carga en los diversos servidores.  Centralización del control: los accesos, recursos y la integridad de los datos son controlados por el servidor de forma que un programa cliente defectuoso o no autorizado no pueda dañar el sistema.  Escalabilidad: se puede aumentar la capacidad de clientes y servidores por separado. Cualquier elemento puede ser aumentado (o mejorado) en cualquier momento, o se pueden añadir nuevos nodos a la red (clientes y/o servidores).  Fácil mantenimiento: al estar distribuidas las funciones y responsabilidades entre varios ordenadores independientes, es posible reemplazar, reparar, actualizar, o incluso trasladar un servidor, mientras que sus clientes no se verán afectados por ese cambio (o se afectarán mínimamente). Esta independencia de los cambios también se conoce como encapsulación. |
| **Principios relacionados** | Las decisiones anteriormente planteadas son de la creación de los siete integrantes del grupo, estamos dispuestos a abordar el SCELS cumpliendo con los principios que consideramos más importantes. De esta forma garantizamos la eficiencia y calidad adecuada, interfaces agradables al usuario y costos accesibles para el cliente.  El sistema y las decisiones que hemos planteado van acorde con ello. |