****

**UNIVERSIDAD PRIVADA DE TACNA**

**FACULTAD DE INGENIERIA**

**Escuela Profesional de Ingeniería de Sistemas**

**Proyecto *Loppify***

Curso: *Programación Web 2*

Docente: Patrick Cuadros Quiroga

Integrantes:

Condori Vargas Tomas Yoel – (2018000487)

Salinas Condori Erick Javier – (2020069046)

**Tacna – Perú**

***2025***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| CONTROL DE VERSIONES | | | | | |
| Versión | Hecha por | Revisada por | Aprobada por | Fecha | Motivo |
| 1.0 | Erick Javier Salinas Condori  Tomas Yoel Condori Vargas | Erick Javier Salinas Condori  Tomas Yoel Condori Vargas | ARV | 2/11/2025 | Versión Original |

Sistema *Loppify*

Documento de Arquitectura de Software

Versión *{1.0}*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| CONTROL DE VERSIONES | | | | | |
| Versión | Hecha por | Revisada por | Aprobada por | Fecha | Motivo |
| 1.0 | MPV | ELV | ARV | 10/10/2020 | Versión Original |

INDICE GENERAL

Indice

[1. INTRODUCCIÓN 1](#_Toc190184971)

[1.1. Propósito (Diagrama 4+1) 1](#_Toc190184972)

[1.2. Alcance 1](#_Toc190184973)

[1.3. Definición, siglas y abreviaturas 1](#_Toc190184974)

[1.4. Organización del documento 1](#_Toc190184975)

[2. OBJETIVOS Y RESTRICCIONES ARQUITECTONICAS 1](#_Toc190184976)

[2.1.1. Requerimientos Funcionales 1](#_Toc190184977)

[2.1.2. Requerimientos No Funcionales – Atributos de Calidad 1](#_Toc190184978)

[3. REPRESENTACIÓN DE LA ARQUITECTURA DEL SISTEMA 1](#_Toc190184979)

[3.1. Vista de Caso de uso 1](#_Toc190184980)

[3.1.1. Diagramas de Casos de uso 1](#_Toc190184981)

[3.2. Vista Lógica 1](#_Toc190184982)

[3.2.1. Diagrama de Subsistemas (paquetes) 1](#_Toc190184983)

[3.2.2. Diagrama de Secuencia (vista de diseño) 1](#_Toc190184984)

[3.2.3. Diagrama de Colaboración (vista de diseño) 1](#_Toc190184985)

[3.2.4. Diagrama de Objetos 1](#_Toc190184986)

[3.2.5. Diagrama de Clases 1](#_Toc190184987)

[3.2.6. Diagrama de Base de datos (relacional o no relacional) 1](#_Toc190184988)

[3.3. Vista de Implementación (vista de desarrollo) 1](#_Toc190184989)

[3.3.1. Diagrama de arquitectura software (paquetes) 1](#_Toc190184990)

[3.3.2. Diagrama de arquitectura del sistema (Diagrama de componentes) 1](#_Toc190184991)

[3.4. Vista de procesos 1](#_Toc190184992)

[3.4.1. Diagrama de Procesos del sistema (diagrama de actividad) 1](#_Toc190184993)

[3.5. Vista de Despliegue (vista física) 1](#_Toc190184994)

[3.5.1. Diagrama de despliegue 1](#_Toc190184995)

[4. ATRIBUTOS DE CALIDAD DEL SOFTWARE 1](#_Toc190184996)

[Escenario de Funcionalidad 1](#_Toc190184997)

[Escenario de Usabilidad 1](#_Toc190184998)

[Escenario de confiabilidad 1](#_Toc190184999)

[Escenario de rendimiento 1](#_Toc190185000)

[Escenario de mantenibilidad 1](#_Toc190185001)

[Otros Escenarios 1](#_Toc190185002)

1. INTRODUCCIÓN
   1. Propósito (Diagrama 4+1)

Este documento presenta una visión global y resumida de la arquitectura de Loppify, un sistema web basado en ASP.NET Core y Angular, que permite la venta de comidas con excedentes. La arquitectura sigue el modelo 4+1, abarcando vistas esenciales como la lógica del negocio, la implementación, la distribución y el desarrollo.

El diseño de Loppify prioriza la eficiencia en el manejo de transacciones y la optimización de la base de datos, sobre la portabilidad, dado que está diseñado específicamente para entornos basados en Azure. Se han tomado decisiones estratégicas en cuanto a seguridad, autenticación, escalabilidad y mantenibilidad.

El sistema está alineado con los requisitos funcionales (gestión de usuarios, productos y ventas) y no funcionales (rendimiento, seguridad y disponibilidad).

* 1. Alcance

El presente documento se enfoca en la vista lógica del sistema, detallando su estructura y relación entre componentes. Se describen las interacciones entre las capas del sistema (frontend, backend y base de datos) y los principales módulos funcionales.

Además, se incluyen aspectos relevantes de otras vistas, como la vista de implementación (estructura de archivos y despliegue en Azure) y la vista de desarrollo (tecnologías y herramientas utilizadas). No se detallará la vista de procesos, dado que Loppify no maneja flujos de procesamiento en tiempo real críticos.

* 1. Definición, siglas y abreviaturas

Término Definición

Loppify Plataforma web para la venta de comidas con excedentes.

ASP.NET Core Framework para la construcción del backend del sistema.

Angular Framework utilizado para el desarrollo del frontend.

PostgreSQL Base de datos utilizada para el almacenamiento de datos.

Entity Framework Core (EF Core) ORM utilizado para interactuar con la base de datos.

Azure Plataforma en la nube donde se despliega la aplicación.

CI/CD Integración y despliegue continuo mediante GitHub Actions.

ORM Mapeo objeto-relacional utilizado para manejar datos en la base de datos.

JWT (JSON Web Token) Mecanismo de autenticación usado en Loppify.

MVC Patrón de diseño utilizado en la arquitectura del sistema.

* 1. Organización del documento

El proyecto Loppify está organizado en una arquitectura modular basada en MVC (Modelo-Vista-Controlador), separando claramente las responsabilidades entre sus componentes:

Backend (ASP.NET Core)

Capa de Modelo: Define las entidades del dominio utilizando Entity Framework Core.

Capa de Controladores: Gestiona las solicitudes HTTP y la lógica de negocio.

Capa de Servicios: Implementa la lógica de aplicación y se comunica con la base de datos.

Capa de Repositorios: Maneja las consultas a la base de datos mediante el ORM EF Core.

Frontend (Angular)

Módulos: Separación en componentes reutilizables para gestionar vistas y funcionalidades.

Servicios: Comunicación con el backend a través de HTTPClient.

Rutas: Implementación de la navegación y redirección según el rol del usuario.

Base de Datos (PostgreSQL)

Esquema Relacional: Modelado de entidades como Usuarios, Productos, Pedidos y Negocios.

Migraciones: Uso de EF Core Migrations para actualizar la estructura de la base de datos.

Infraestructura y DevOps

Despliegue en Azure: Uso de Terraform para definir la infraestructura y despliegue automatizado con GitHub Actions.

Contenerización con Docker: Uso de imágenes de .NET y Angular para facilitar el despliegue.

Manejo de Seguridad: Implementación de JWT para autenticación y protección de rutas.

# **OBJETIVOS Y RESTRICCIONES ARQUITECTONICAS**

A continuación, se presentan los requerimientos funcionales y no funcionales priorizados, determinando así el orden de implementación.

* 1. Priorización de requerimientos

### Requerimientos Funcionales

*[Definir la prioridad de los requerimientos funcionales.]*

|  |  |  |
| --- | --- | --- |
| ID | Descripción | Prioridad |
| RF01 | Un usuario debe poder registrarse en la plataforma con su correo electrónico. | Alta |
| RF02 | Un usuario debe poder iniciar sesión con su correo y contraseña. | Alta |
| RF03 | Un administrador debe poder gestionar los negocios registrados en la plataforma. | Alta |
| RF04 | Un vendedor debe poder registrar productos y administrar su inventario. | Media |
| RF05 | Un cliente debe poder visualizar los productos disponibles y realizar compras. | Alta |
| RF06 | El sistema debe permitir diferentes roles: Administrador, Vendedor y Cliente. | Alta |
| RF07 | El sistema debe enviar notificaciones de pedidos a los vendedores y clientes. | Media |
| RF08 | Los clientes deben poder realizar pagos en línea mediante una pasarela de pagos. | Baja |
| RF09 | El sistema debe mostrar recomendaciones personalizadas de productos a los clientes. | Baja |

### Requerimientos No Funcionales – Atributos de Calidad

*[Definir la prioridad de los requerimientos NO funcionales.]*

|  |  |  |
| --- | --- | --- |
| ID | Descripción | Prioridad |
| RNF01 | El sistema debe estar disponible el 99.9% del tiempo. | Alta |
| RNF02 | El tiempo de respuesta de cualquier consulta debe ser menor a 2 segundos. | Media |
| RNF03 | La información de los usuarios y transacciones debe estar cifrada y protegida. | Alta |
| RNF04 | La arquitectura del sistema debe permitir escalabilidad horizontal en Azure. | Alta |
| RNF05 | La plataforma debe ser accesible desde dispositivos móviles y de escritorio. | Media |
| RNF06 | El sistema debe soportar al menos 1,000 usuarios concurrentes. | Media |
| RNF07 | La autenticación debe implementarse mediante JWT para seguridad. | Alta |
| RNF08 | Los datos deben ser respaldados automáticamente en la nube cada 24 horas. | Media |

* 1. Restricciones

Las siguientes restricciones han sido identificadas para el desarrollo y despliegue de Loppify:

* Presupuesto: El sistema debe ser desarrollado y alojado en servicios cloud de bajo costo.
* Plazo de entrega: El desarrollo del sistema debe completarse en un período máximo de 6 meses.
* Infraestructura: El sistema debe ser desplegado en Azure utilizando Terraform para su gestión.
* Tecnologías: Se deben utilizar ASP.NET Core para el backend, Angular para el frontend y PostgreSQL como base de datos.
* Regulaciones: La aplicación debe cumplir con normativas de protección de datos como GDPR o similares.
* Compatibilidad: La plataforma debe ser compatible con los navegadores más utilizados (Chrome, Firefox, Edge).
* Mantenimiento: El código debe seguir buenas prácticas de desarrollo para facilitar futuras mejoras

# **REPRESENTACIÓN DE LA ARQUITECTURA DEL SISTEMA**

* 1. Vista de Caso de uso

### Diagramas de Casos de uso

Caso de uso: Registro de Usuario

Diagrama

El contenido generado por IA puede ser incorrecto.

Caso de uso: Inicio de Sesion

Diagrama

El contenido generado por IA puede ser incorrecto.

Caso de uso: Gestion de Vendedores

Diagrama

El contenido generado por IA puede ser incorrecto.

Caso de uso: Gestión de negocios

Diagrama

El contenido generado por IA puede ser incorrecto.

Caso de uso: Publicación de productos

Diagrama

El contenido generado por IA puede ser incorrecto.

Caso de uso: Recuperacion de contraseña

Diagrama

El contenido generado por IA puede ser incorrecto.

Caso de uso: Visualizacion de Negocios

Diagrama

El contenido generado por IA puede ser incorrecto.

Caso de uso: Compra de Productos

Diagrama

El contenido generado por IA puede ser incorrecto.

Caso de uso: Reportes

Diagrama

El contenido generado por IA puede ser incorrecto.

Caso de uso: Notificaciones

Diagrama

El contenido generado por IA puede ser incorrecto.

* 1. Vista Lógica

### 

### Diagrama de Subsistemas (paquetes)

*Interfaz de usuario gráfica, Aplicación, Word

El contenido generado por IA puede ser incorrecto.*

### Diagrama de Secuencia (vista de diseño)

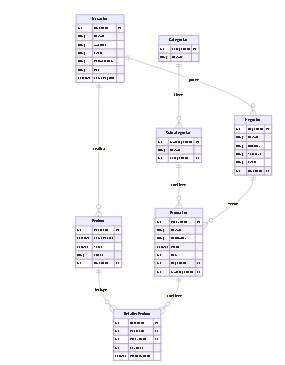
### Diagrama de Objetos

### Diagrama de Clases

Diagrama

El contenido generado por IA puede ser incorrecto.

### Diagrama de Base de datos (relacional o no relacional)



* 1. Vista de Implementación (vista de desarrollo)

### Diagrama de arquitectura software (paquetes)

*Interfaz de usuario gráfica, Aplicación, Word

El contenido generado por IA puede ser incorrecto.*

### Diagrama de arquitectura del sistema (Diagrama de componentes)

*[Se detalla la manera como fue implementado el sistema propuesto, se describe visualmente las capas que tiene el sistema, como están distribuidas y sus principales funciones]*

* 1. Vista de procesos

*[Describe la descomposición del sistema procesos pesados. Indica que procesos o grupos de procesos se comunican o interactúan entre sí y los modos en que estos se comunican.]*

### Diagrama de Procesos del sistema (diagrama de actividad)

*[Se realizará un diagrama del o los procesos del sistema donde se exponga las actividades donde interviene el sistema propuesto, adicionando diagramas que definan el detalle la descomposición del sistema en procesos pesados. Indica que procesos o grupos de procesos se comunican o interactúan entre sí y los modos en que estos se comunican]*

* 1. Vista de Despliegue (vista física)

### Diagrama de despliegue

*[un diagrama de despliegue, amplía el sistema de software y muestra los contenedores (aplicaciones, almacenamiento de datos, microservicios, etc.) que componen este sistema de software]*

# **ATRIBUTOS DE CALIDAD DEL SOFTWARE**

Los atributos de calidad (QAs) del sistema Loppify garantizan que la plataforma cumpla con los requisitos esperados por los stakeholders, asegurando funcionalidad, usabilidad, confiabilidad, rendimiento y mantenibilidad.

Escenario de Funcionalidad

Loppify proporciona un conjunto de características esenciales para la gestión de alimentos excedentes, permitiendo la interacción entre consumidores y vendedores. Entre sus funcionalidades clave se encuentran el registro y autenticación de usuarios, gestión de negocios, visualización y compra de productos, procesamiento de pagos y notificaciones automáticas. Además, la plataforma implementa medidas de seguridad para proteger la información de los usuarios y las transacciones realizadas.

Escenario de Usabilidad

El sistema está diseñado para ofrecer una interfaz intuitiva y amigable, accesible desde dispositivos móviles y computadoras. Loppify permite a los usuarios registrarse y comenzar a utilizar la plataforma de manera sencilla, minimizando la curva de aprendizaje. Se han implementado principios de diseño UX/UI para mejorar la experiencia de usuario, incluyendo navegación clara, mensajes de error comprensibles y opciones de personalización según el rol del usuario (Administrador, Vendedor o Cliente).

Escenario de confiabilidad

Loppify garantiza la integridad y seguridad de los datos mediante cifrado en la autenticación y almacenamiento seguro en PostgreSQL. Se implementan técnicas de mitigación de ataques como SQL Injection y Cross-Site Scripting (XSS). La disponibilidad del sistema es prioritaria, asegurando un tiempo de actividad del 99.9% y respaldos automáticos en la nube cada 24 horas para evitar pérdida de datos.

Escenario de rendimiento

La plataforma está optimizada para responder en menos de 2 segundos a cualquier consulta dentro del sistema. Se emplea caching para mejorar la velocidad de carga y reducir la carga en el servidor. Además, Loppify soporta hasta 1,000 usuarios concurrentes sin degradación en la experiencia de uso. La base de datos está optimizada para garantizar tiempos de respuesta eficientes en la búsqueda de productos y procesamiento de pedidos.

Escenario de mantenibilidad

El código de Loppify sigue principios de desarrollo limpio y modular, facilitando futuras actualizaciones y mejoras. Se emplean buenas prácticas en ASP.NET Core y Angular para garantizar una arquitectura escalable y mantenible. El sistema está diseñado para ser extensible, permitiendo la integración de nuevas funciones sin afectar la estabilidad del software.

Otros Escenarios

Performance:

El sistema de Loppify ha sido diseñado para responder eficientemente a los eventos generados por los usuarios, asegurando tiempos de carga mínimos y procesamiento optimizado de las operaciones clave. Se monitorean métricas de rendimiento en Azure para detectar y corregir posibles cuellos de botella, asegurando una experiencia fluida para los usuarios.