Universidad Técnica Particular de Loja

Devops

Documento de Arquitectura de Software

Fecha: 02/02/2025

Aprobador(es) del Documento:

Nombre del Aprobador	Rol
Armando Cabrera	Docente tutor

Revisor(es) del Documento:

Nombre del Revisor	Rol
Pablo José Reyes Mosquera	Líder del grupo, Desarrollador Frontend
Carlos Enrique Morocho Carrión	Desarrollador UX/UI
Hermin Leonardo Chuquimarca Jaramillo	Arquitecto de Software
Jorge Enrique Armijos Cabrera	Desarrollador Backend
Jhon Josué Calle Castillo	Gerente de producto

Resumen de cambios:

Versión	Fecha	Hecho por	Descripción del cambio
0.1	05/12/2024	Carlos Morocho	Actualización del README.md
0.1	05/12/2024	Jorge Armijos	Actualización del modelo de datos
0.2	05/12/2024	Carlos Morocho	Pruebas del README.md
0.3	05/12/2024	Leonardo Chuquimarca	Actualización de la Vista Lógica
0.4	06/12/2024	Pablo Reyes	Actualización de las especificaciones de los casos de uso.
0.5	06/12/2024	Leonardo Chuquimarca	Actualización de la Vista Lógica
0.6	06/12/2024	Leonardo Chuquimarca	Actualización del Mapa de Capacidades
0.7	06/12/2024	Carlos Morocho	Actualización de los casos de uso
0.7	06/12/2024	Carlos Morocho	Ingreso de imágenes de los casos de uso
0.8	06/12/2024	Carlos Morocho	Actualización en diagramas de casos de uso
0.9	06/12/2024	Carlos Morocho	Actualización de las especificaciones
1.0	06/12/2024	Carlos Morocho	Actualización del flujo de la especificación
1.1	06/12/2024	Pablo Reyes	Creación de una rama “carlos-design-app” en el repositorio para el diseño de la aplicación
1.2	06/12/2024	Leonardo Chuquimarca	Actualización del Mapa de Capacidades
1.3	08/12/2024	Leonardo Chuquimarca	Actualización de la vista lógica
1.4	08/12/2024	Carlos Morocho	Actualización de los diagramas de casos de uso
1.5	08/12/2024	Jorge Armijos	Actualización Modelo de datos
1.6	08/12/2024	Pablo Reyes	Actualización Modelo de datos
1.7	08/12/2024	Pablo Reyes	Eliminación de la rama “carlos-design-app” en el repositorio para el diseño de la aplicación
1.8	08/12/2024	Pablo Reyes	Creación de una rama “diseño” en el repositorio para el diseño de la aplicación

1.9	08/12/2024	Pablo Reyes	Actualización del README.md
2.0	08/12/2024	Pablo Reyes	Solicitud de extracción abierta en README.md (12 veces)
2.1	08/12/2024	Pablo Reyes	Solicitud de extracción cerrada en README.md (12 veces)
2.2	08/12/2024	Pablo Reyes	Eliminación de la rama “diseño”
2.3	08/12/2024	Pablo Reyes	Creación de una nueva rama llamada “Pablo-Reyes—Diseño-frontend”
2.4	08/12/2024	Pablo Reyes	Actualización del README.md
2.5	08/12/2024	Pablo Reyes	Creación de una nueva rama llamada “Jorge-Armijos-Desarrollo-backEnd”
2.6	08/12/2024	Leonardo Chuquimarca	Actualización Vista de Procesos
2.7	09/12/2024	Leonardo Chuquimarca	Actualización Vista de Procesos
2.8	09/12/2024	Pablo Reyes	Actualización README.md
2.9	09/12/2024	Leonardo Chuquimarca	Actualización Mapa de Capacidades
3.0	09/12/2024	Carlos Morocho	Actualización Diagramas de Casos de uso
3.1	11/12/2024	Leonardo Chuquimarca	Actualización Vista Lógica
3.2	11/12/2024	Pablo Reyes	Creó un aoartado “Vista de Desarrollo”
3.3	11/12/2024	Carlos Morocho	Actualización Tabla Pipeline.md
3.4	11/12/2024	Carlos Morocho	Actualización Vista de Desarrollo
3.5	11/12/2024	Leonardo Chuquimarca	Actualización Vista de Procesos
3.6	12/12/2024	Leonardo Chuquimarca	Actualización de las especificaciones de los casos de uso
3.7	17/12/2024	Jorge Armijos	Actualización Modelo de datos
3.8	17/12/2024	Pablo Reyes	Actualización Modelo de datos
3.9	04/01/2025	Leonardo Chuquimarca	Actualización Vista de Despliegue
4.0	04/01/2025	Jorge Armijos	Desarrollo Login UTPLBox
4.1	21/01/2025	Pablo Reyes	Actualización README.md
4.2	22/01/2025	Jorge Armijos	Api Backend
4.3	26/01/2025	Jorge Armijos	Cargar y Agregar Auditorias backend
4.4	28/01/2025	Pablo Reyes	Actualización del repositorio

Tabla de contenidos

Contenido

1. INTRODUCCIÓN

1.1. Propósito

Este documento proporciona una visión arquitectónica integral del sistema de gestión de inventario para la Universidad Técnica Particular de Loja (UTPL). Se utilizan diversas vistas arquitectónicas para representar aspectos clave del sistema. Su propósito es capturar y transmitir las decisiones arquitectónicas significativas tomadas en el desarrollo de la solución.

1.2. Alcance

La arquitectura descrita en este documento se centra en el desarrollo de una aplicación móvil para la gestión de inventario de la bodega de la UTPL. El sistema estará basado en servicios en la nube para permitir la automatización del control del inventario, mejorar la eficiencia operativa y optimizar el uso de los recursos. La solución proporcionará monitoreo en tiempo real, acceso remoto y mejor colaboración entre departamentos.

1.3. Referencias

#	Documento	Contenido
[REF1]	Arquitectura de Aplicaciones en la Nube	Principios y mejores prácticas para el desarrollo de sistemas basados en la nube.
[REF2]	Firebase Documentation	Guía oficial sobre la implementación de Firebase como backend para la aplicación móvil.
[REF3]	REST API Design Best Practices	Mejores prácticas para la creación de APIs REST eficientes y seguras.
[REF4]	ISO 27001	Estándares de seguridad de la información aplicables a la gestión de datos en la nube.
[REF5]	Google Material Design	Principios y guías de diseño para la interfaz de usuario de la aplicación.
[REF6]	Metodología Ágil (Scrum)	Enfoque ágil para el desarrollo del sistema de gestión de inventario.
[REF7]	Documentación de Firebase	Firebase es una plataforma Backend-as-a-Service (BaaS) que proporciona bases de datos en tiempo real, autenticación y funciones en la nube.
[REF8]	Documentación de Node.js	Node.js es un entorno de ejecución de JavaScript construido sobre el motor V8 de Chrome, utilizado para construir aplicaciones de red escalables.
[REF9]	Documentación de React Native	React Native es una biblioteca de JavaScript para construir interfaces de usuario, particularmente aplicaciones multiplataforma.
[REF10]	Documentación de Expo	Expo es un marco y plataforma para aplicaciones universales de React, que simplifica el desarrollo móvil.
[REF11]	Directrices para desarrolladores de Google Play	Directrices para publicar y gestionar aplicaciones en Google Play Store.
[REF12]	Documentación de Git	Sistema de control de versiones distribuido.

[REF13]	Documentación de GitLab CI/CD	Servicio de integración y despliegue continuo.
[REF14]	Discord para Desarrolladores	API y herramientas para integrar Discord.
[REF15]	(eDelivery)(AP)(SAD)(3.4) (1	Guía arquitectónica para la aplicación de gestión de inventarios.

1.4. Resumen del Contenido del Documento

El presente documento describe el Desarrollo de la aplicación para gestión de inventarios utilizando las consideraciones sobre tamaño, rendimiento y calidad y las vistas arquitectónicas las cuales son:

- **Vista de Caso de Uso**: Describe los requisitos funcionales y las interacciones entre los usuarios y el sistema.
- **Vista Lógica**: Ilustra la estructura de alto nivel del sistema, incluyendo componentes y sus relaciones.
- **Vista de Procesos**: Muestra el comportamiento en tiempo de ejecución del sistema, incluyendo procesos e hilos.
- **Vista de Despliegue**: Detalla cómo se despliega el sistema en la infraestructura física y en la nube.
- **Vista de Implementación**: Se enfoca en la organización del código y las herramientas de desarrollo.
- **Vista de Datos**: Describe el modelo de datos y el esquema de la base de datos.

2. REPRESENTACIÓN ARQUITECTONICA

En las siguientes secciones se describen los objetivos y restricciones de la arquitectura del sistema de gestión de inventario para la Universidad Técnica Particular de Loja (UTPL).

2.1. Objetivos Arquitectónicos

1. **Escalabilidad:** Garantizar que la aplicación pueda manejar un aumento en el número de usuarios, solicitudes y datos sin afectar el rendimiento.

2. **Disponibilidad:** Asegurar que el sistema esté accesible en todo momento para los usuarios encargados de la gestión del inventario.

3. **Seguridad:** Proteger los datos del inventario y los accesos de los usuarios mediante autenticación segura y cifrado de datos.

4. **Mantenibilidad:** Diseñar una arquitectura modular y bien documentada para facilitar futuras actualizaciones y mejoras del sistema.

5. **Interoperabilidad:** Permitir la integración con otros sistemas de la UTPL, como los sistemas de mantenimiento y gestión de activos.

6. **Accesibilidad Remota:** Garantizar que el sistema pueda ser utilizado desde dispositivos móviles y estaciones de trabajo en la nube para una gestión eficiente del inventario.

2.2. Restricciones Arquitectónicas

1. **Tecnologías específicas**: Backend desarrollado con Django para la gestión de datos y lógica de negocio; base de datos en Firebase o PostgreSQL para almacenamiento seguro en la nube y aplicación móvil basada en React Native para compatibilidad con Android e iOS.

2. **Despliegue en la nube**: El sistema debe alojarse en una infraestructura en la nube para garantizar accesibilidad, escalabilidad y disponibilidad; se utilizarán servicios en la nube para autenticación, almacenamiento de datos y procesamiento de notificaciones.

3. **Compatibilidad móvil**: La aplicación debe ser accesible desde dispositivos móviles mediante una aplicación optimizada. También debe contar con una interfaz web para usuarios administrativos.

4. **Cumplimiento de normativas**: El sistema debe cumplir con las políticas de seguridad y privacidad de la UTPL. Se implementarán buenas prácticas de seguridad como autenticación de dos factores y cifrado de datos sensibles.

3. OBJETIVOS Y RESTRICCIONES ARQUITECTÓNICAS

Se han identificado los siguientes requisitos no funcionales que afectan a la solución arquitectónica:

Requisitos no funcionales	**Descripción**
Escalabilidad	La aplicación debe ser capaz de gestionar el crecimiento del inventario y la cantidad de usuarios sin afectar el rendimiento.
Portabilidad	La solución debe poder ser implementada en diversas plataformas móviles (Android e iOS) y navegadores web.
Interoperabilidad	El sistema debe integrarse con otros servicios existentes de la universidad, como la gestión académica y administrativa.
Seguridad	Debe garantizarse el acceso seguro a la información mediante autenticación con credenciales institucionales y cifrado de datos.
Disponibilidad	El sistema debe estar accesible 24/7 con mínima interrupción del servicio.

4. SEGURIDAD

4.1. Introducción

El sistema de gestión de inventario de la Universidad Técnica Particular de Loja (UTPL) incorpora medidas de seguridad basadas en buenas prácticas de la industria y en estándares de seguridad en la nube. Se garantiza la protección de los datos almacenados y transmitidos mediante autenticación segura, cifrado de información y controles de acceso. Además, el sistema puede integrarse fácilmente con el esquema de autenticación existente de la universidad.

4.2. Comunicación entre Usuarios y Servidor en la Nube

Dado que los usuarios acceden al sistema desde distintos dispositivos y ubicaciones, la seguridad en la transmisión de datos se implementa a través de protocolos seguros como HTTPS. Toda la comunicación entre la aplicación móvil y los servicios en la nube está cifrada, evitando accesos no autorizados o manipulaciones de los datos.

4.3. Autenticación y Control de Acceso

La autenticación de los usuarios se gestiona mediante el sistema de credenciales institucionales de la UTPL. Se implementa autenticación basada en OAuth 2.0 o Firebase Authentication, asegurando que solo usuarios autorizados puedan acceder al sistema.

4.4. Seguridad de Datos en la Nube

Toda la información del inventario se almacena en bases de datos en la nube, con cifrado de datos en reposo y en tránsito. Se implementan copias de seguridad automáticas para evitar pérdidas de información en caso de fallos.

4.5. Seguridad en la Administración del Sistema

El acceso a la plataforma administrativa está restringido a personal autorizado mediante autenticación con las respectivas credenciales institucionales de la Universidad Técnica Particular de Loja.

DECLARACIÓN DE SEGURIDAD

Además de las medidas de seguridad implementadas en el sistema, se recomienda que los usuarios sigan buenas prácticas de seguridad, tales como:

- Utilizar redes seguras al acceder a la aplicación.
- No compartir credenciales con terceros.
- Configurar listas blancas de IP y cifrado en el sistema de almacenamiento.

.

5. VISTA DE CASOS DE USO

Esta sección proporciona una representación de los casos de uso relevantes para la arquitectura.

5.1. Justificación de la selección

Los casos de uso relevantes para la arquitectura se han seleccionado basándose en los siguientes criterios:

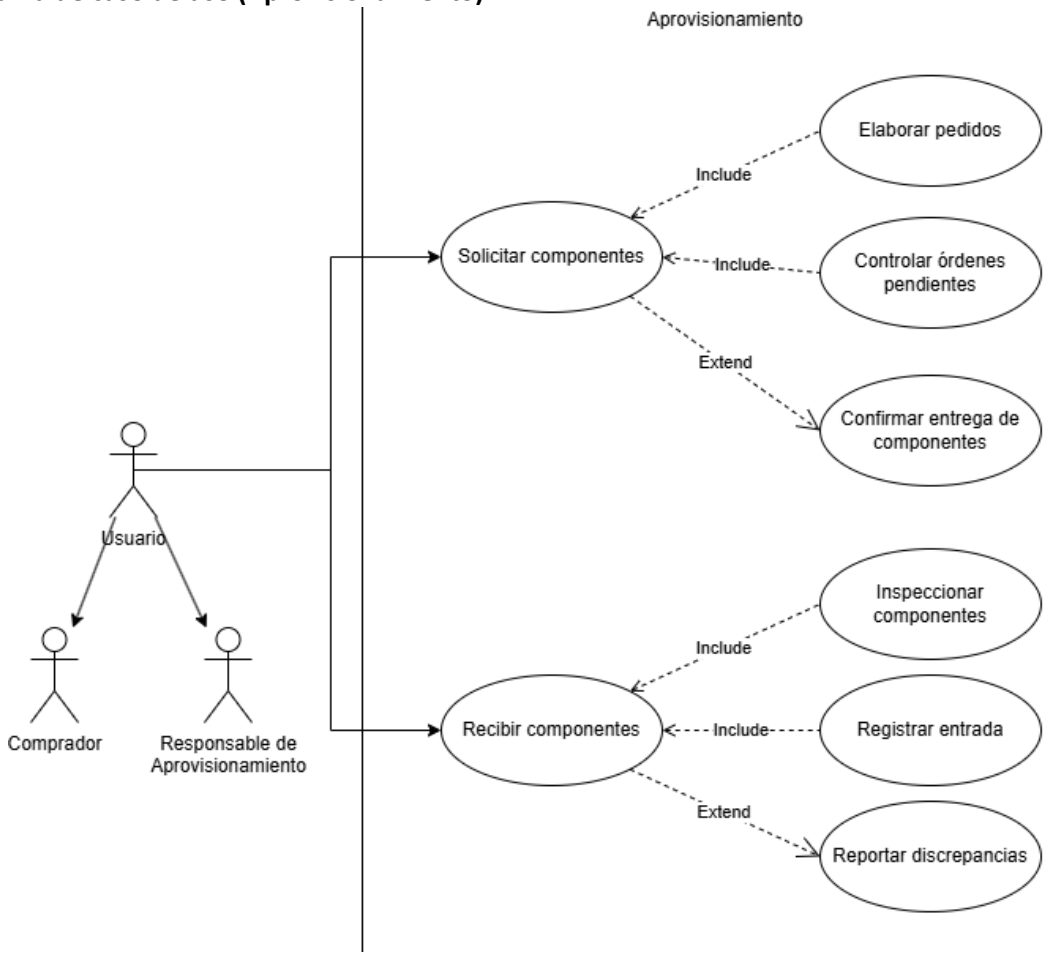
- Casos de uso que afectan la interacción entre los usuarios y la aplicación.
- Casos de uso que puede representar riesgos de la aplicación basándose en la arquitectura.

Los siguientes casos de uso han sido seleccionados:

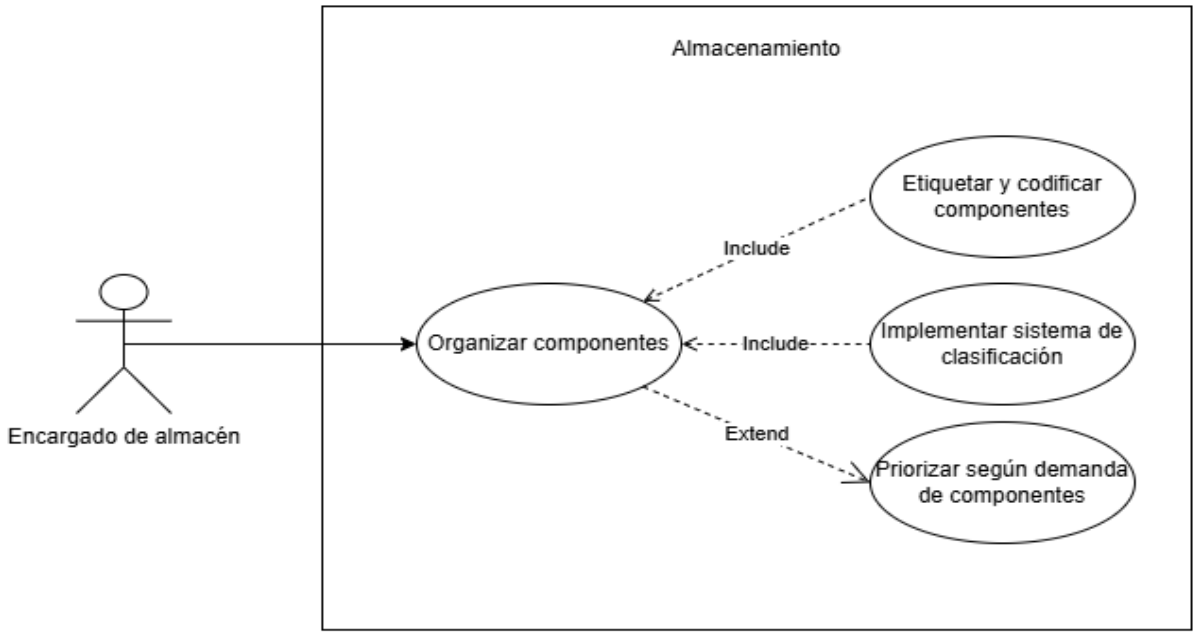
#	Nombre del Caso de Uso	Actores	Flujo Normal
1	Aprovisionamiento	- Responsable de aprovisonamiento - Comprador	1. Usuario identifica los componentes necesarios según las políticas de inventario. 2. Usuario elabora los pedidos de los componentes requeridos. 3. Usuario controla las órdenes pendientes para garantizar entregas a tiempo. 4. Usuario confirma la recepción de los componentes solicitados. 5. Usuario verifica físicamente los componentes recibidos contra los pedidos realizados. 6. Usuario inspecciona los componentes para garantizar su calidad y cantidad. 7. Usuario registra las entradas de los componentes al Inventario. 8. Usuario reporta cualquier discrepancia o daño identificado.
2	Almacenamiento	- Encargado de almacén.	1. Encargado de almacén etiqueta y codifica los componentes para facilitar su identificación. 2. Encargado de almacén clasifica los componentes según el sistema de categorización definido. 3. Encargado de almacén organiza los componentes priorizando aquellos con mayor demanda o rotación.

3	Gestión Operativa	- Coordinador de operaciones de inventario	1. Coordinador de operaciones de inventario revisa los niveles actuales de existencias. 2. Coordinador de operaciones de inventario realiza auditorías regulares para verificar discrepancias. 3. Coordinador de operaciones de inventario actualiza los registros con los cambios identificados. 4. Coordinador de operaciones de inventario controla los componentes obsoletos o caducados. 5. Coordinador de operaciones de inventario identifica componentes que alcanzan el punto de reorden. 6. Coordinador de operaciones de inventario activa las órdenes de compra para reabastecer dichos componentes. 7. Coordinador de operaciones de inventario calcula el stock de seguridad necesario según la demanda actual.

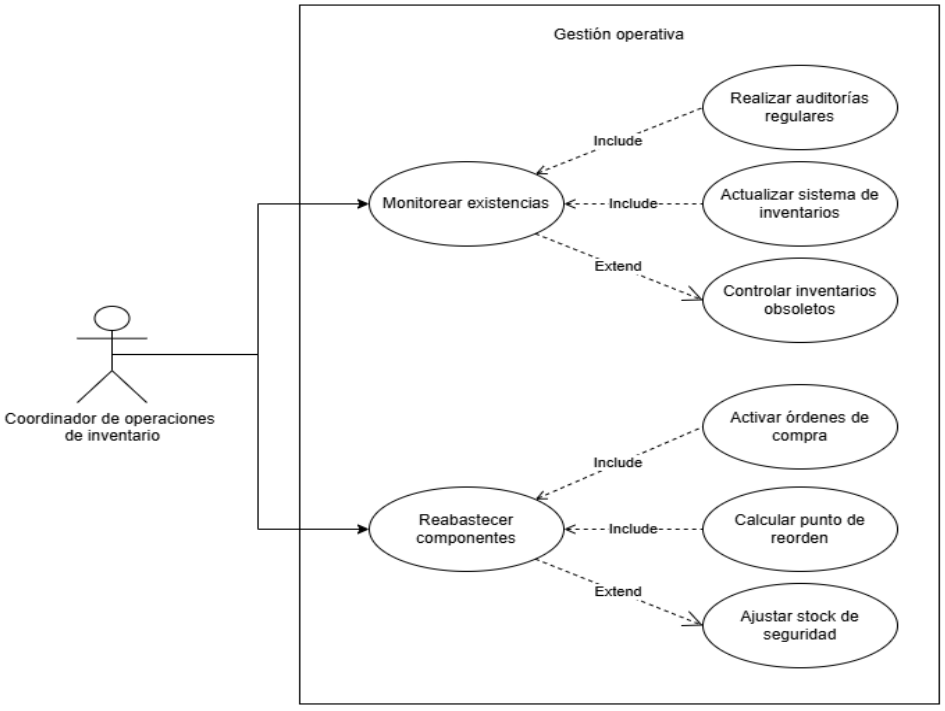
1. Diagrama de caso de uso (Aprovisionamiento)



2. Diagrama de caso de uso (Almacenamiento)



3. Diagrama de caso de uso (Gestión Operativa)



6. VISTA LÓGICA

6.1. Visión general

Este capítulo describe el diagrama de clases que modela la estructura que está compuesta la aplicación para la gestión de inventarios.

6.2. Diagrama de Clases

Un diagrama de clases es un tipo de diagrama estático en la Programación Orientada a Objetos (POO) que modela la estructura de un sistema mostrando:

- Clases: Representan las entidades o conceptos del sistema.
- Atributos: Definen las características o propiedades de cada clase.
- Métodos: Especifican los comportamientos o acciones que puede realizar la clase.
- Relaciones: Muestran las conexiones entre las clases, como asociaciones, composiciones o herencias.

Es ampliamente utilizado en la fase de diseño de software para definir cómo interactúan las diferentes partes del sistema.

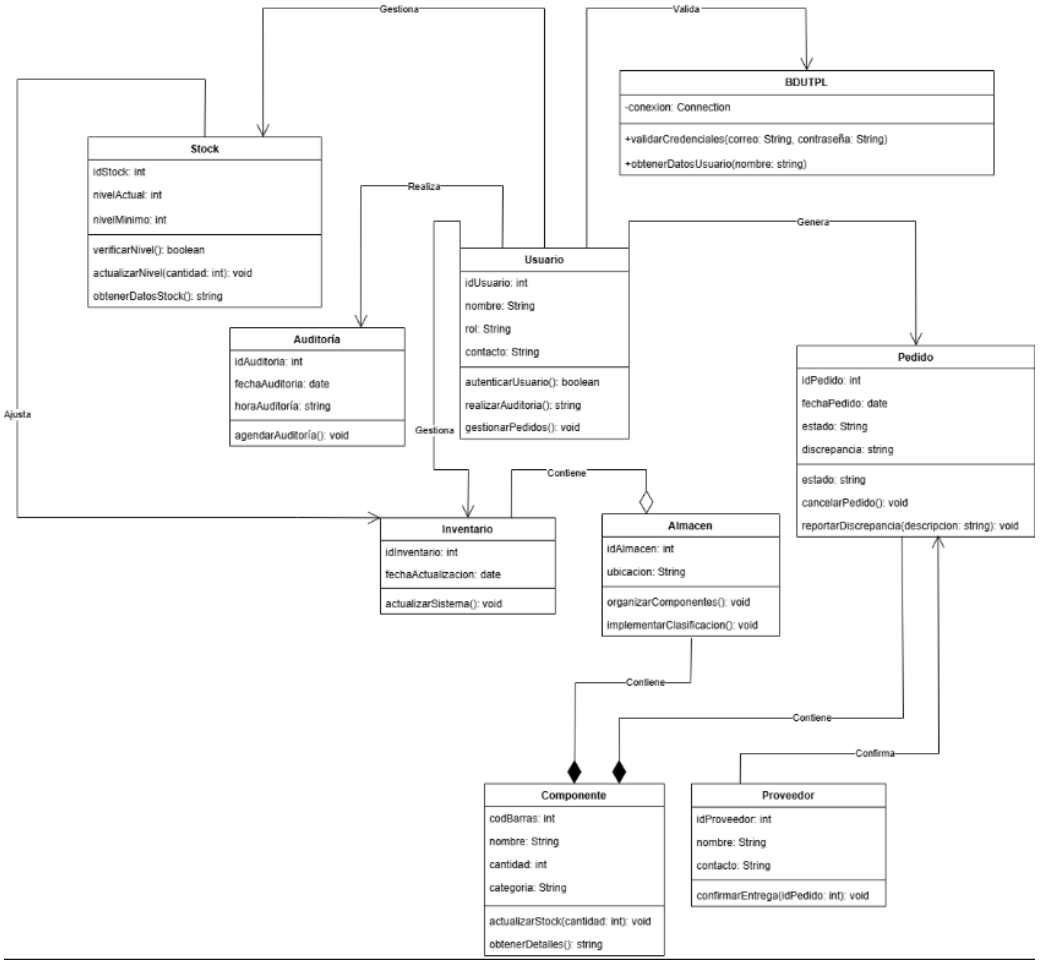
Estructura general de un diagrama de clases

Un diagrama de clases sigue la siguiente estructura:

- Clases:
  - Representadas como rectángulos divididos en tres secciones:
    - Nombre de la clase (parte superior).
    - Atributos (parte media).
    - Métodos (parte inferior).
- Relaciones entre clases:
  - Asociaciones: Indican conexiones entre clases.
  - Composición: Una clase contiene a otra como parte esencial.
  - Agregación: Una clase contiene a otra, pero no de manera esencial.
  - Herencia: Una clase hereda propiedades y métodos de otra.
- Visibilidad de atributos y métodos:
  - + Público: Accesible desde cualquier lugar.
  - - Privado: Accesible solo dentro de la clase.
  - # Protegido: Accesible dentro de la clase y sus subclases.
  - Cardinalidad: Define la cantidad de objetos relacionados, como 1:1, 1:N o N:M.

El diagrama muestra la estructura principal del sistema, compuesta por los siguientes módulos principales:

Diagrama de clases



6.2.1. Clases principals del sistema

El diagrama de clases determinado mediante el diccionario de clases desarrollado, corresponde a un sistema de gestión de inventario en una organización. Aquí se describen las principales clases y sus relaciones:

Clases principales:

- Stock:
  - Atributos:
    - idStock: Identificador único del stock.
    - nivelActual: Nivel actual del stock.
    - nivelMinimo: Nivel mínimo permitido.
  - Métodos:
    - verificarNivel(): Verifica si el nivel de stock está por debajo del mínimo.
    - actualizarNivel(): Permite ajustar el nivel del stock.
    - obtenerDatosStock(): Devuelve detalles del stock.
- Auditoría:
  - Atributos:
    - idAuditoria: Identificador único de la auditoría.
    - fechaAuditoria: Fecha en que se realizará la auditoría.
    - horaAuditoria: Hora de la auditoría.
  - Métodos:
    - agendarAuditoria(): Programa una auditoría.

- Usuario:
  - Atributos:
    - idUsuario: Identificador del usuario.
    - nombre: Nombre del usuario.
    - rol: Rol que desempeña el usuario.
    - contacto: Información de contacto del usuario.
  - Métodos:
    - autenticarUsuario(): Valida las credenciales del usuario.
    - realizarAuditoria(): Permite al usuario realizar auditorías.
    - gestionarPedidos(): Maneja la creación o modificación de pedidos.
- Inventario:
  - Atributos:
    - idInventario: Identificador del inventario.
    - fechaActualizacion: Fecha de la última actualización del inventario.
  - Métodos:
    - actualizarSistema(): Actualiza el sistema de inventario.
- BDUTPL (Base de Datos):
  - Atributos:
    - conexion: Representa la conexión a la base de datos.
  - Métodos:
    - validarCredenciales(): Comprueba las credenciales del usuario.
    - obtenerDatosUsuario(): Obtiene los datos de un usuario en base a su nombre.
- Pedido:
  - Atributos:
    - idPedido: Identificador único del pedido.
    - fechaPedido: Fecha en la que se generó el pedido.
    - estado: Estado actual del pedido (e.g., en proceso, completado).
    - discrepancia: Información sobre posibles problemas en el pedido.
  - Métodos:
    - cancelarPedido(): Cancela un pedido existente.
    - reportarDiscrepancia(): Reporta problemas asociados al pedido.
- Almacén:
  - Atributos:
    - idAlmacen: Identificador único del almacén.
    - ubicacion: Ubicación del almacén.
  - Métodos:
    - organizarComponentes(): Organiza los componentes dentro del almacén.
    - implementarClasificacion(): Clasifica los componentes según ciertas reglas.
- Componente:
  - Atributos:
    - codBarras: Código de barras del componente.
    - nombre: Nombre del componente.
    - cantidad: Cantidad en stock.
    - categoria: Categoría a la que pertenece.
  - Métodos:
    - actualizarStock(): Actualiza el stock de componentes.
    - obtenerDetalles(): Devuelve detalles del componente.

- Proveedor:
  - Atributos:
    - idProveedor: Identificador único del proveedor.
    - nombre: Nombre del proveedor.

        - contacto: Información de contacto del proveedor.
    - Métodos:
        - confirmarEntrega(): Confirma la entrega de un pedido.

Relaciones entre clases:

- Stock se ajusta por medio de Auditoría.
- Usuario realiza auditorías y gestiona pedidos.
- Inventario contiene múltiples componentes y se actualiza con la interacción de los almacenes.
- BDUTPL valida las credenciales y gestiona la información de los usuarios.
- Pedido se genera en base al inventario y está relacionado con el proveedor para confirmar entregas.
- Proveedor confirma la entrega de componentes.
- Componente está contenido en inventarios y almacenes.

7. VISTA DE PROCESOS

7.1. Visión general

La vista de procesos describe el comportamiento dinámico del sistema de gestión de inventario ante la interacción con el usuario.

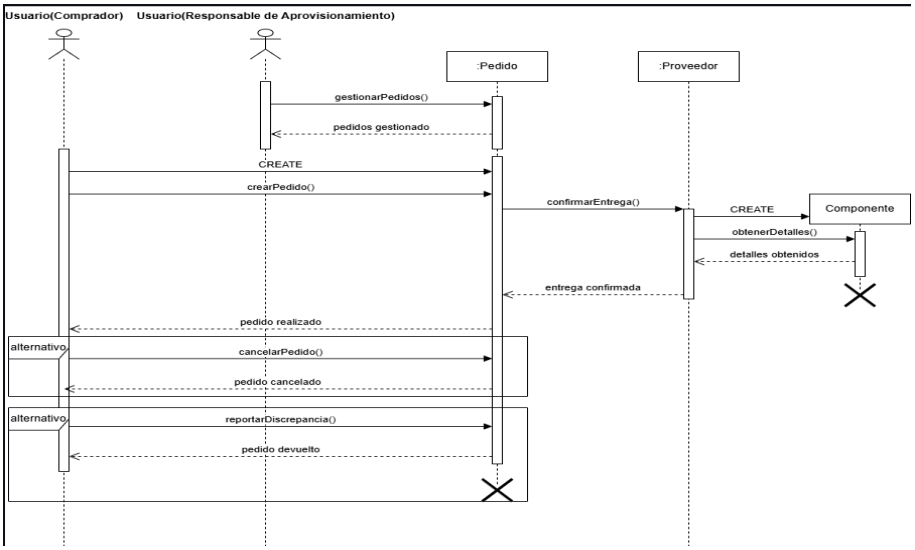
7.2. Diagrama de Secuencia

Los diagramas de secuencia son una herramienta de modelado en la Ingeniería de Software que forma parte del Lenguaje Unificado de Modelado (UML). Se utilizan para describir cómo los objetos de un sistema interactúan entre sí a través del tiempo, mostrando el flujo de mensajes o llamadas entre ellos. Estos diagramas son especialmente útiles para representar procesos y casos de uso en sistemas orientados a objetos.

Estructura que siguen los diagramas de secuencia

- Actores y Objetos: Representan las entidades que participan en la interacción. Los actores se representan como figuras humanas, y los objetos como rectángulos.
- Líneas de vida: Indican la existencia de un actor u objeto durante el proceso. Son líneas verticales que comienzan debajo de cada actor u objeto.
- Mensajes: Representan la comunicación entre actores y objetos mediante flechas horizontales.
- Bloques de ejecución: Rectángulos sobre las líneas de vida que representan la ejecución de un proceso o método.
- Tiempo: Se representa de forma implícita en la dirección descendente del diagrama.

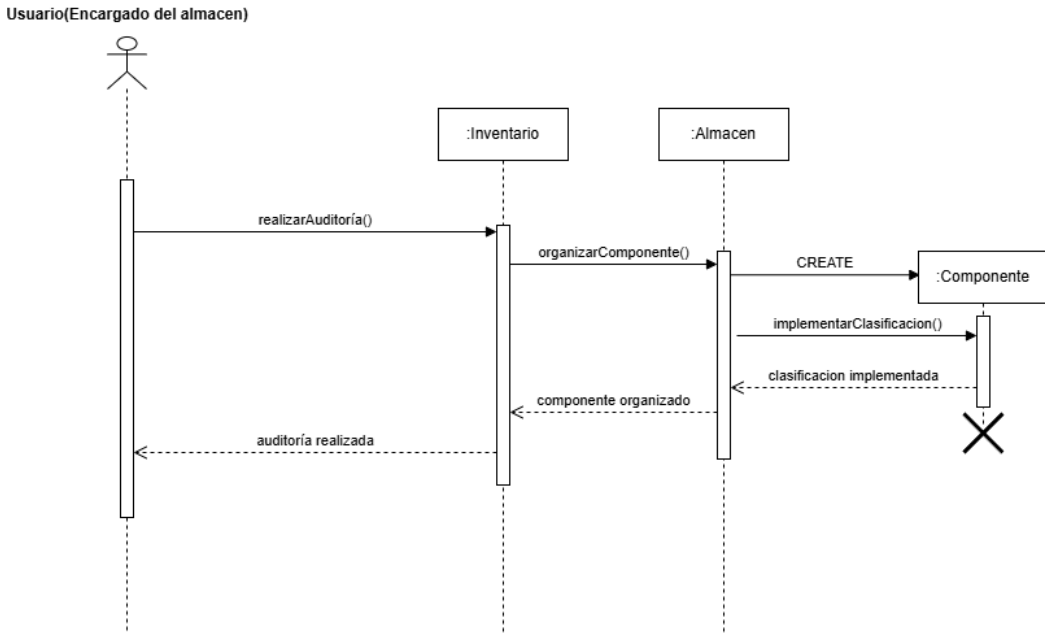
1. Diagrama de Aprovisionamiento



Explicación:
Este diagrama describe cómo el Personal Administrativo inicia sesión en el sistema de la UTPL. Una vez autenticado, puede agregar componentes al inventario tras validar su existencia. Los pasos clave incluyen:

- Verificar credenciales.
- Obtener los datos del usuario.
- Agregar componentes tras verificar su disponibilidad.

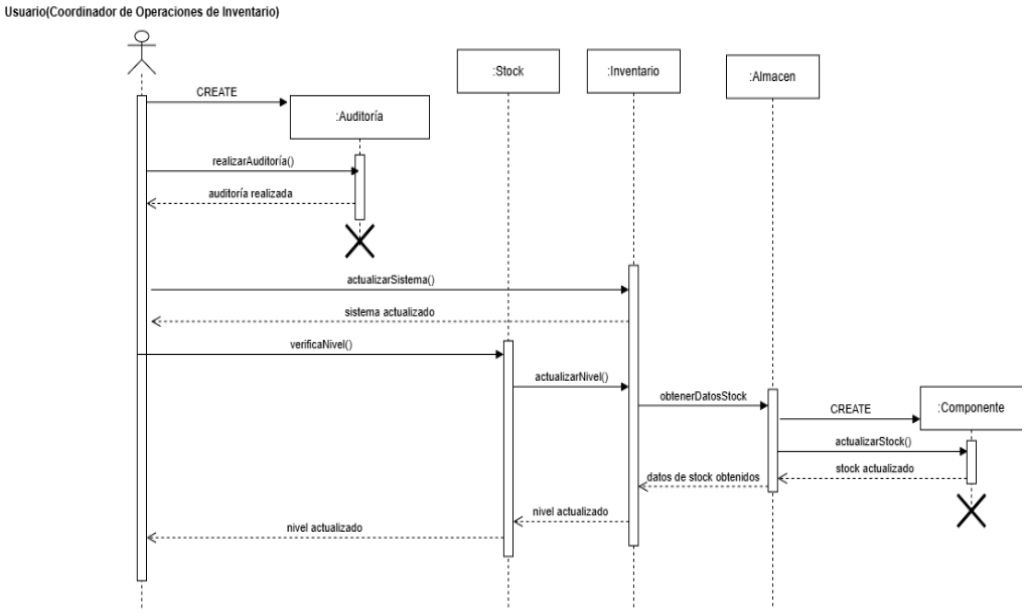
2. Diagrama de Almacenamiento



Explicación:
Este diagrama muestra el proceso de verificación y clasificación de componentes. El flujo incluye:

- Verificar si un componente existe en el inventario.
- Verificar el estado del componente.
- Clasificar los componentes en diferentes categorías basadas en su tipo.

3. Diagrama de Gestión Operativa



Explicación:
Aquí se describe el proceso de monitoreo de movimientos e irregularidades en el inventario, incluyendo la generación de alertas y registros. Las acciones clave son:

- Monitorear movimientos.
- Generar informes de irregularidades.
- Registrar movimientos específicos y enviar alertas al sistema.

7.3. Diagrama de Robustez

Los diagramas de robustez son una herramienta de modelado utilizada en el diseño de software para detallar cómo los actores, objetos, y procesos interactúan dentro de un sistema. Sirven como un puente entre los diagramas de casos de uso y los diagramas de diseño detallado, permitiendo identificar elementos clave del sistema como actores, controladores y entidades.

Estructura que siguen los diagramas de robustez

- Actores: Representan los usuarios u otros sistemas que interactúan con el sistema principal. Se dibujan como figuras humanas o íconos.

- Entidades: Son los elementos de datos o conceptos principales del dominio, representados como rectángulos.

- Controladores: Representan la lógica de la aplicación y la interacción entre actores y entidades. Se muestran como círculos u óvalos.

- Relaciones: Representan las conexiones entre actores, controladores y entidades mediante líneas con etiquetas que describen la interacción.

Diagrama 1: Aprovisionamiento

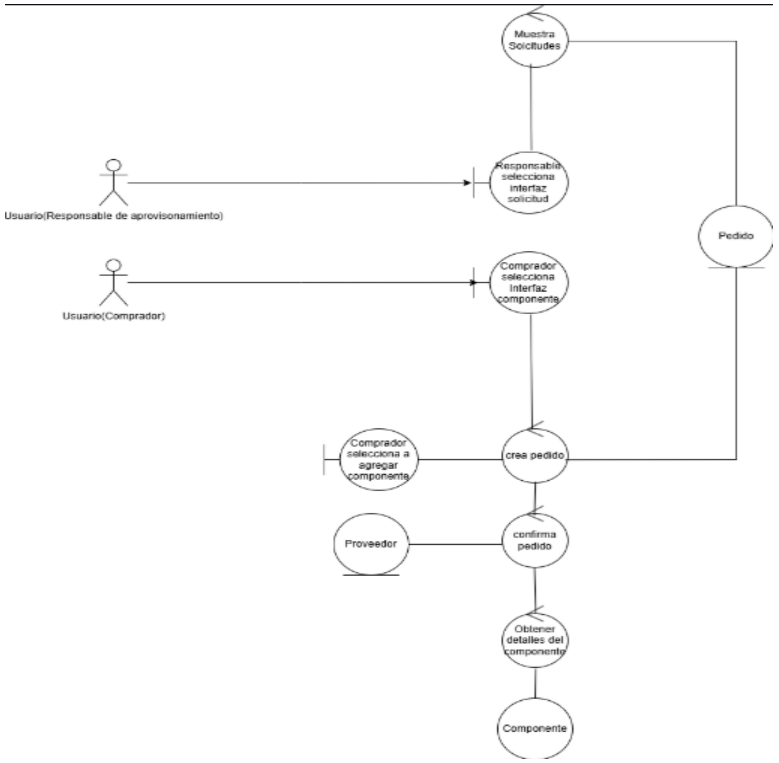


Diagrama 2: Almacenamiento

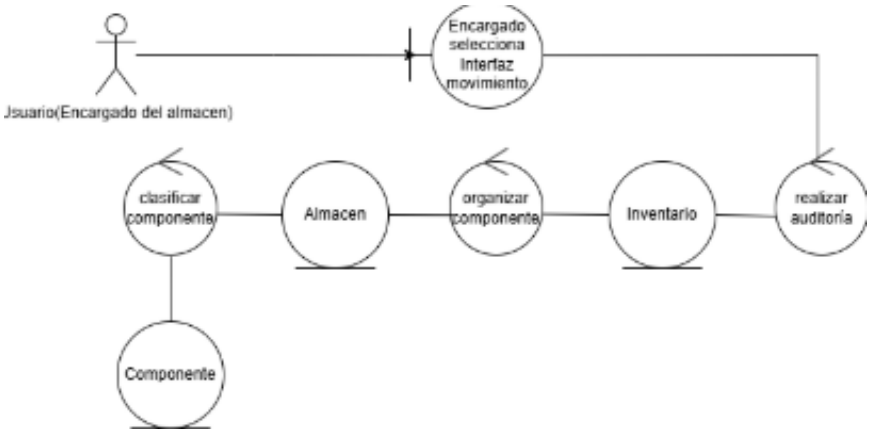
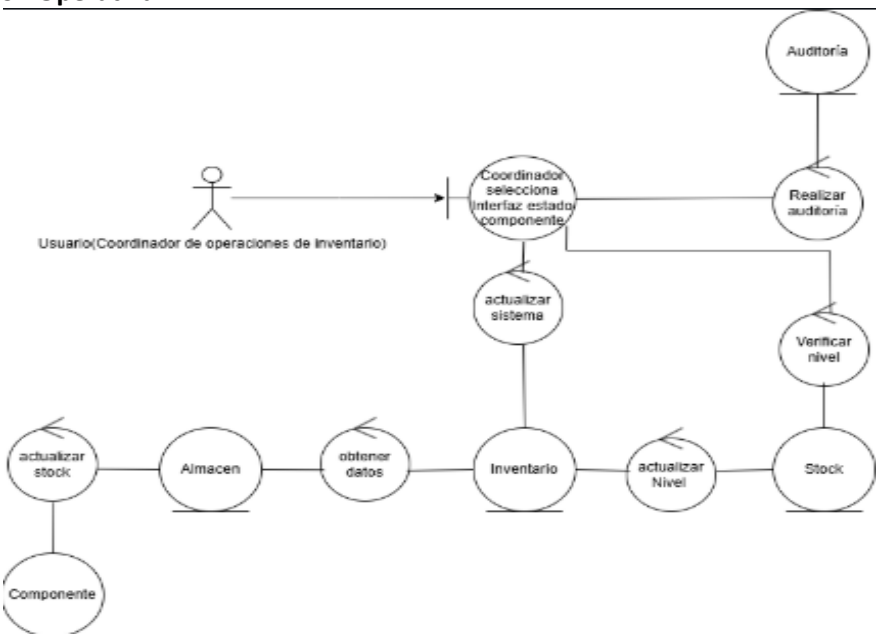


Diagrama 3: Gestión Operativa



8. VISTA DE DESARROLLO

8.1. Visión general

La vista de desarrollo describe la implementación física del sistema, detallando la arquitectura tecnológica, herramientas, procesos y estrategias de desarrollo utilizadas para construir la aplicación.

8.2. Arquitectura de Desarrollo

El proyecto se estructura en dos componentes principales:

1. **Backend**
    - Tecnologías: Node.js, Firebase
    - Bases de datos: Firebase
    - Enfoque: Desarrollo de servicios y gestión de datos
2. **Frontend**
    - Tecnologías: React
    - Plataforma: Aplicación móvil multiplataforma
    - Enfoque: Interfaz de usuario, experiencia del usuario, integración de APIs

.

8.3. Metodología del Desarrollo

Características principales:

- **CI/CD**: Integración y despliegue continuos
- **DevOps**: Integración de desarrollo y operaciones

¿Qué es CI/CD?

CI/CD, que significa Integración Continua y Entrega/Despliegue Continuo, es un conjunto de prácticas que automatizan las etapas de desarrollo, prueba y despliegue del software. Su objetivo principal es mejorar la velocidad, calidad y confiabilidad de las entregas de software.

Integración Continua (CI):

- Es el proceso de integrar regularmente el código de diferentes desarrolladores en un repositorio compartido.
- Incluye la ejecución automática de pruebas para identificar errores rápidamente.

Entrega Continua (CD - Continuous Delivery):

- Extiende la CI al automatizar la preparación de entregas del software en cualquier momento.
- Garantiza que el código esté siempre en un estado listo para producción.
- Requiere pruebas adicionales y validaciones antes del despliegue.

Despliegue Continuo (CD - Continuous Deployment):

- Va un paso más allá y automatiza también el proceso de despliegue en producción.
- Cada cambio que pasa las pruebas se despliega automáticamente.
- El enfoque CI/CD fomenta ciclos de desarrollo cortos, iterativos y más seguros.

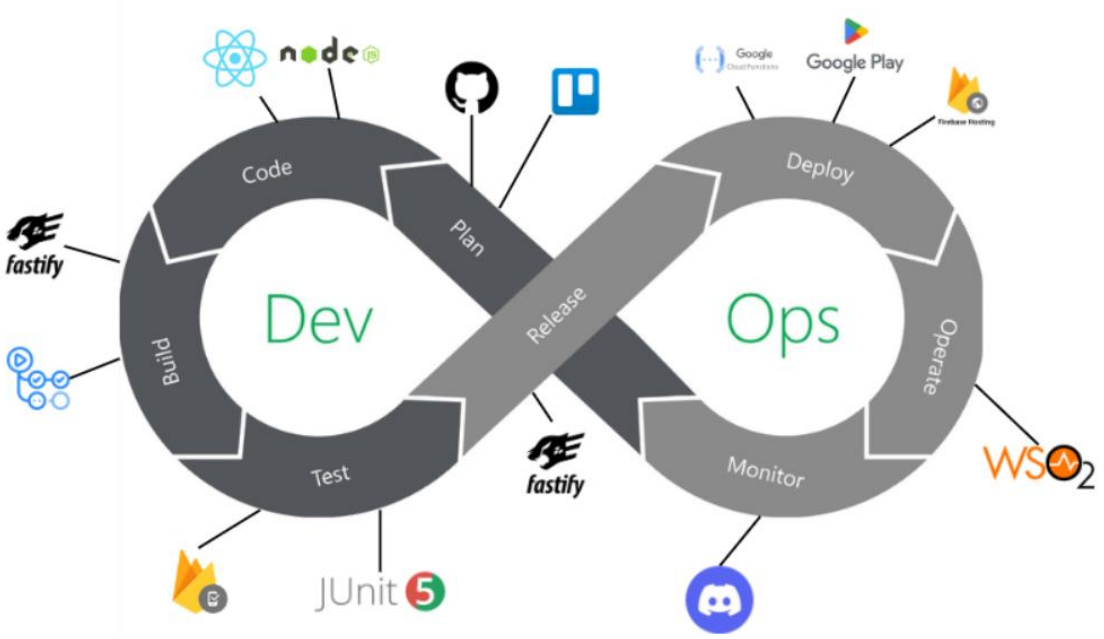
¿Qué es DevOps?

DevOps es una cultura, metodología y conjunto de prácticas que busca integrar los equipos de desarrollo (Development) y operaciones (Operations) para mejorar la colaboración, automatización y entrega continua de software. DevOps no es solo una herramienta, sino una filosofía que combina personas, procesos y tecnologías.

Beneficios de DevOps:

- Ciclos de entrega más rápidos.
- Mayor estabilidad y calidad del software.
- Mejor alineación entre objetivos técnicos y empresariales.

8.4. Herramientas y Tecnologías Clave



Fase	Herramientas	Descripción
Plan	Github , Trello	**Github:** Se utilizará para gestionar el desarrollo colaborativo, organizar tareas mediante branches **Trello:** Se empleará para el seguimiento de tareas y actualizaciones del estado de los incidentes detectados.
Code	React, Node.js	**React:** Será empleado para desarrollar la interfaz móvil, garantizando una experiencia de usuario eficiente. **Node.js:** Se utiliza Node.js para escribir el código del backend, aprovechando su capacidad para manejar múltiples conexiones simultáneas y su ecosistema de módulos para integrar funcionalidades necesarias en el proyecto.
Build	Github Actions, Fastify	**Github Actions:** Se utilizará para la automatización del testeo de cada parte del código. **Fastify:** Su arquitectura modular facilita la integración de diferentes componentes del sistema.
Test	Firebase TestLab, JUnit	**Firebase:** Permite realizar pruebas unitarias y de integración, asegurando que las funciones del backend interactúan correctamente con Firestore y que los datos se gestionan adecuadamente en el sistema de inventario. **Junit:** Se utilizará para realizar pruebas unitarias automatizadas.
Release	Fastify	Permite preparar la aplicación para su lanzamiento mediante la configuración de rutas y middleware, asegurando que todos los endpoints estén listos para ser utilizados en producción.
Deploy	Firebase Hosting, Cloud Functions, Google Play Store	**Firebase Hsoting**: Permite implementar el backend sin preocuparse por la infraestructura subyacente, asegurando una alta disponibilidad y escalabilidad del sistema de gestión de inventario. **Cloud Functions:** Permite implementar el backend sin preocuparse por la infraestructura subyacente, asegurando una alta disponibilidad y escalabilidad del sistema de gestión de inventario. **Google Play Store:** Se utilizará como plataforma para publicar y distribuir la aplicación móvil.

Operate	WSO2	**WSO2:** Ayudará a gestionar, publicar, monitorear y asegurar APIs, facilitando el control y la gobernanza de interfaces de comunicación.
Monitor	Discord	Se utilizará para la comunicación en tiempo real y notificaciones de eventos relevantes.

9. VISTA DE DESPLIEGUE

9.1. Visión general

En esta siguiente vista del diseño arquitectónico, se presenta un diagrama de arquitectura de software basado en una estructura de microservicios para una aplicación móvil que interactúa con una infraestructura en la nube. A continuación, se describe cada componente y su función:

1.Device

Mobile App:

- Es la interfaz principal de usuario (User Interface), que actúa como el punto de interacción para los usuarios.
- Realiza peticiones mediante protocolos HTTP/HTTPS al servidor API a través del puerto 443.

2. APIServer

API Gateway Service:

- Sirve como puerta de enlace entre la aplicación móvil y los microservicios del backend.
- Recibe las solicitudes de la aplicación y las redirige a los microservicios correspondientes a través de los puertos 80/443.

3. Backend Microservices

Contiene una serie de microservicios especializados que operan de forma independiente pero conectados entre sí:

- Microservice Component Request: Gestiona las solicitudes de componentes realizadas desde la aplicación.
- Microservice Component Receive: Procesa la recepción de componentes en el sistema.
- Microservice Component Organize: Administra la organización y clasificación de los componentes en el inventario.
- Microservice Monitor Stocks: Monitorea los niveles de stock y envía alertas cuando se requiere reabastecimiento.
- Microservice Component Replenish: Maneja el reabastecimiento del inventario cuando los niveles son bajos.

Todos los microservicios se comunican a través de HTTP/HTTPS usando los puertos 80/443.

4. Cloud Server

Contiene las bases de datos y servicios en la nube para gestionar la información:

- Component Data: Almacena los datos relacionados con los componentes, utilizando Firestore

Database como sistema de almacenamiento.

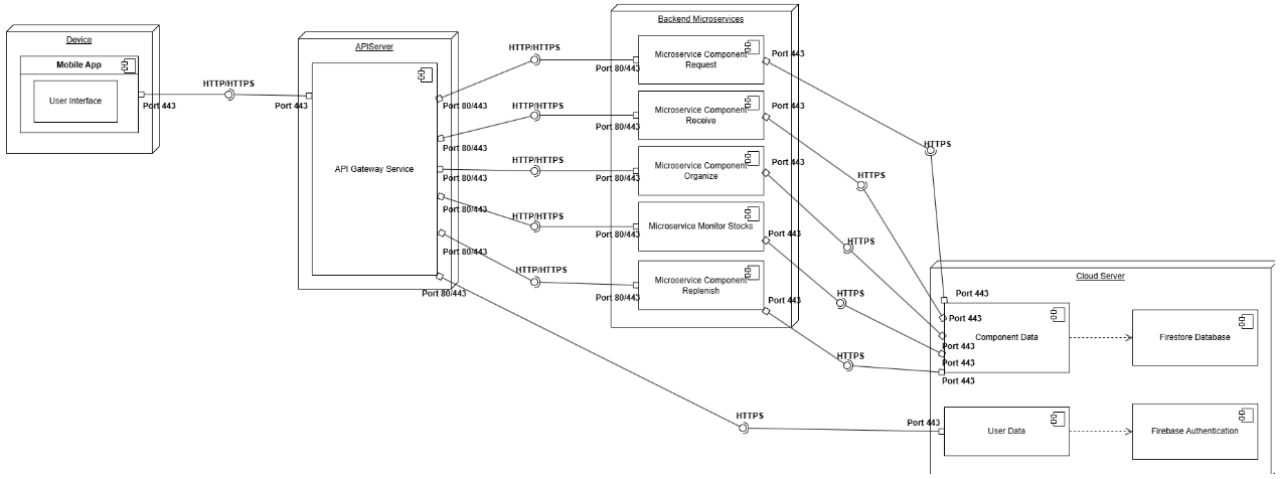
- User Data: Contiene información de los usuarios, autenticada mediante Firebase Authentication.

- La comunicación con el servidor en la nube ocurre mediante el puerto 443, asegurando que todas las interacciones son seguras (HTTPS).

Conexiones:

- Todas las comunicaciones utilizan HTTP/HTTPS, asegurando el intercambio de datos a través de conexiones seguras.

- Cada componente interactúa de manera específica, siguiendo los principios de la arquitectura basada en microservicios, con un enfoque en la escalabilidad y modularidad.



10. VISTA DE DATOS

10.1. Modelo de Datos

Un **modelo de datos** es una representación estructurada que define cómo se organizan, almacenan y manipulan los datos dentro de un sistema o base de datos. Es un diseño conceptual que describe:

- **Entidades**: Objetos o conceptos del mundo real, como "Clientes", "Productos" o "Órdenes".
- **Atributos**: Propiedades o características de las entidades, como el "nombre" de un cliente o el "precio" de un producto.
- **Relaciones**: Conexiones o asociaciones entre entidades, como la relación entre "Clientes" y "Órdenes".

El propósito principal de un modelo de datos es facilitar la comprensión y gestión de los datos en sistemas complejos, asegurando que estén organizados de manera lógica y eficiente

10.2. Estructura general de un modelo de datos

Un modelo de datos se compone de los siguientes elementos clave:

1. Entidades

Representan los objetos principales que necesitan ser modelados dentro del sistema. Ejemplo: Un cliente puede ser una entidad en un sistema de ventas.

2. Atributos

Son las características o propiedades que describen una entidad. Ejemplo: Los atributos de un cliente pueden incluir:

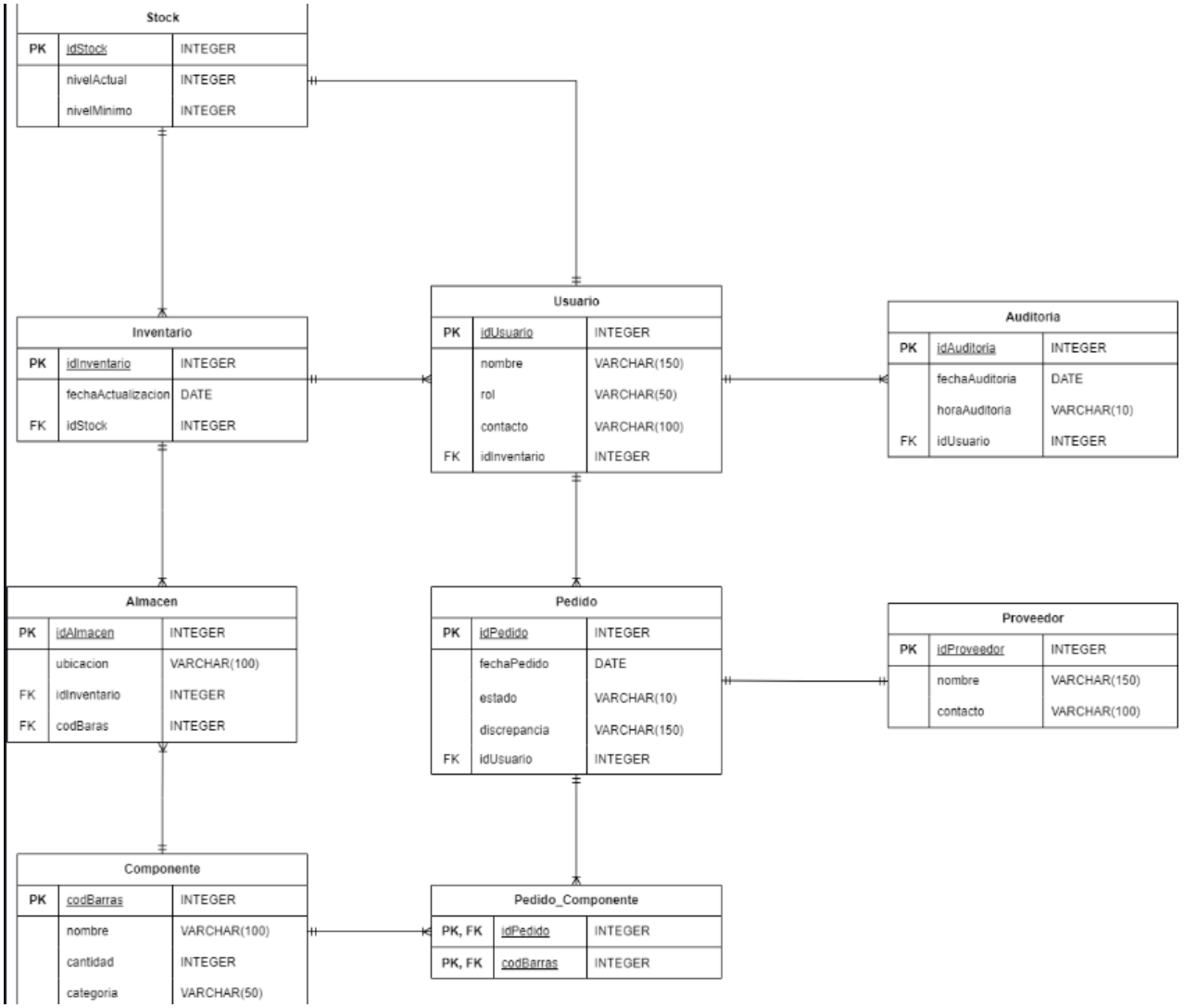
- Nombre
- Dirección
- Teléfono

3. Relaciones

Describen cómo las entidades están conectadas entre sí y cómo interactúan. Ejemplo: Un cliente puede tener una relación de "realiza" con varias órdenes en un sistema de ventas.

Tipos comunes de modelos de datos:

- **Modelo conceptual**: Representación abstracta, como diagramas Entidad-Relación (ER).
- **Modelo lógico**: Especifica estructuras detalladas como tablas y columnas.
- **Modelo físico**: Implementación en un sistema de base de datos específico.


Stock
PK idStock INTEGER
nivelActual INTEGER
nivelMinimo INTEGER
Usuario
PK idUsuario INTEGER
nombre VARCHAR(150)
rol VARCHAR(50)
contacto VARCHAR(100)
FK idInventario INTEGER
Auditoria
PK idAuditoria INTEGER
fechaAuditoria DATE
horaAuditoria VARCHAR(10)
FK idUsuario INTEGER
Inventario
PK idInventario INTEGER
fechaActualizacion DATE
FK idStock INTEGER
Almacen
PK idAlmacen INTEGER
ubicacion VARCHAR(100)
FK idInventario INTEGER
FK codBaras INTEGER
Pedido
PK idPedido INTEGER
fechaPedido DATE
estado VARCHAR(10)
discrepancia VARCHAR(150)
FK idUsuario INTEGER
Proveedor
PK idProveedor INTEGER
nombre VARCHAR(150)
contacto VARCHAR(100)
Componente
PK codBarras INTEGER
nombre VARCHAR(100)
cantidad INTEGER
categoria VARCHAR(50)
Pedido_Componente
PK, FK idPedido INTEGER
PK, FK codBarras INTEGER

11. Tamaño y rendimiento

11.1. Visión general

Las restricciones de tamaño y rendimiento en la gestión de inventarios de la Universidad Técnica Particular de Loja (UTPL) afectan la arquitectura del sistema y su configuración. Se busca optimizar el almacenamiento, la rapidez de acceso a los datos y la eficiencia en el procesamiento de solicitudes.
Manejo del Tamaño de Datos

- Inventario de Componentes y Activos: El sistema manejará información de los productos almacenados, su ubicación, estado y disponibilidad.
- Imágenes y Documentos Adjuntos: Para auditorías y control de inventario, se podrán adjuntar imágenes de productos o documentos en Firebase Storage, evitando sobrecargar la base de datos.
- Registros Históricos: Se almacenarán registros históricos de movimientos de inventario para análisis futuros. Se implementará archivado automático de datos antiguos para optimizar el rendimiento.
- Tamaño Máximo de Archivos: Se limita el tamaño de los documentos e imágenes a 5 MB por archivo para garantizar rapidez en la consulta.

11.2. Rendimiento

El sistema de gestión de inventarios de la UTPL debe manejar un alto volumen de datos y usuarios sin degradar su rendimiento. Para ello, se adoptan las siguientes estrategias:

1. Escalabilidad y Distribución de Carga

- Uso de Firebase Firestore como base de datos en la nube con replicación automática para garantizar acceso rápido y disponibilidad.
- Implementación de balanceadores de carga en el backend para distribuir solicitudes entre múltiples instancias.
- Soporte para escalado automático en función de la demanda, utilizando Google Cloud Functions para procesamiento de eventos en tiempo real.

2. Optimización de Consultas

- Indexación de datos en Firestore para acelerar búsquedas.
- Uso de almacenamiento en caché para reducir la carga en la base de datos en consultas frecuentes.
- Implementación de consultas paginadas para manejar grandes volúmenes de información sin afectar la velocidad de respuesta.

3. Procesamiento Asíncrono

- Manejo de tareas pesadas en segundo plano mediante Cloud Tasks o Firebase Cloud Functions.

- Implementación de mensajería asíncrona para el procesamiento de eventos, evitando bloqueos en la interfaz de usuario.

4. Pruebas de Rendimiento y Monitoreo

- Se realizarán pruebas de carga con herramientas como JMeter para evaluar el comportamiento del sistema bajo alta demanda.

- Integración con Google Cloud Logging y Firebase Performance Monitoring para registrar métricas de uso y tiempos de respuesta.

- Análisis de rendimiento de la interfaz con Google Lighthouse para optimizar la experiencia del usuario.

11.3. Consideraciones adicionales

- **Disponibilidad:** Uso de servicios en la nube con 99.9% de uptime garantizado.
- **Seguridad:** Autenticación mediante credenciales institucionales y cifrado de datos en tránsito y en reposo.
- **Mantenimiento y actualización:** La arquitectura modular facilita la aplicación de mejoras sin afectar el rendimiento del sistema.

12. CALIDAD

El sistema de gestión de inventarios de la Universidad Técnica Particular de Loja (UTPL) ha sido diseñado bajo principios de extensibilidad, fiabilidad y portabilidad, garantizando un funcionamiento eficiente y adaptable a futuras mejoras.

12.1. Extensibilidad

El sistema sigue una arquitectura modular, lo que facilita la incorporación de nuevas funcionalidades sin afectar el rendimiento.

- **Arquitectura basada en microservicios:** Permite agregar nuevas funcionalidades sin modificar el núcleo del sistema.
- **APIs RESTful bien definidas:** Facilitan la integración con otros sistemas de la UTPL, como el sistema de mantenimiento y activos.
- **Soporte para nuevas tecnologías:** El diseño permite la migración a nuevas tecnologías sin afectar la operatividad actual.

12.2. Fiabilidad

El sistema garantiza la disponibilidad y seguridad de los datos mediante diversas estrategias:

1. **Almacenamiento en la nube:** Se utilizan Google Cloud Firestore y Firebase Storage para una alta disponibilidad y replicación automática.
2. **Mecanismo de reintento automático:** Si una operación falla, se reintenta para garantizar su correcta ejecución.
3. **Monitoreo en tiempo real:** Uso de Google Cloud Monitoring para detectar anomalías en el sistema.
4. **Backups automáticos:** Copias de seguridad regulares en la nube para prevenir pérdidas de información.

12.3. Portabilidad

El sistema ha sido diseñado para ser fácilmente implementado en diferentes entornos.

1. **Uso de contenedores Docker:** Permite su despliegue en cualquier infraestructura compatible con Docker.

2. **Compatibilidad con múltiples plataformas:** Funciona en Android, iOS y web mediante React Native.

3. **Independencia de base de datos:** Aunque se usa Firestore, el sistema puede migrar a PostgreSQL o MongoDB con cambios mínimos.

13. REGISTRO DE LOGS (LOGGING)

El sistema cuenta con un sistema de registro de logs para rastrear eventos y mejorar la seguridad.

13.1. Categoría de Logs

Los registros se dividen en:

- **Seguridad:** Eventos de autenticación y acceso.
- **Errores:** Fallos en el sistema o excepciones inesperadas.
- **Negocio:** Registros de operaciones de inventario (movimientos, auditorías, pedidos).
- **Rendimiento:** Métricas de tiempo de respuesta y uso de recursos.

13.2. Estructura de Logs

Formato general de los registros:

[Fecha y Hora] [Nivel] [Categoría] [Usuario] [Módulo] [Mensaje]

Ejemplo:

[2025-02-06 12:45:32] [ERROR] [Seguridad] [Admin-UTPL] [AuthModule] "Intento fallido de inicio de sesión"

- **Fecha y Hora**: Marca de tiempo del evento.
- **Nivel de Log**: Registran eventos relacionados con las funcionalidades principales de la aplicación.
- **Categoría**: Tipo de log (Seguridad, Negocio, Errores, Rendimiento).
- **Usuario**: Usuario asociado al evento (si aplica).
- **Módulo:** Módulo o componente del sistema que generó el log.
- **Mensaje:** Descripción detallada del evento.

14. MULTITENANCIA

14.1. General

El sistema está diseñado para múltiples unidades y departamentos dentro de la UTPL, asegurando que cada usuario solo tenga acceso a los datos correspondientes a su área.

14.2. Identificación del dominio

- Usuarios autenticados con credenciales UTPL.
- Cada solicitud HTTP incluye un identificador de departamento.

14.3. Interfaz de Usuario (UI)

1. Cada usuario solo ve datos de su departamento.
2. Superadministradores pueden gestionar todos los departamentos.
3. Los accesos están controlados mediante autenticación y roles de usuario.

15. INFORMACIÓN DE CONTACTO

Checklist Support Team