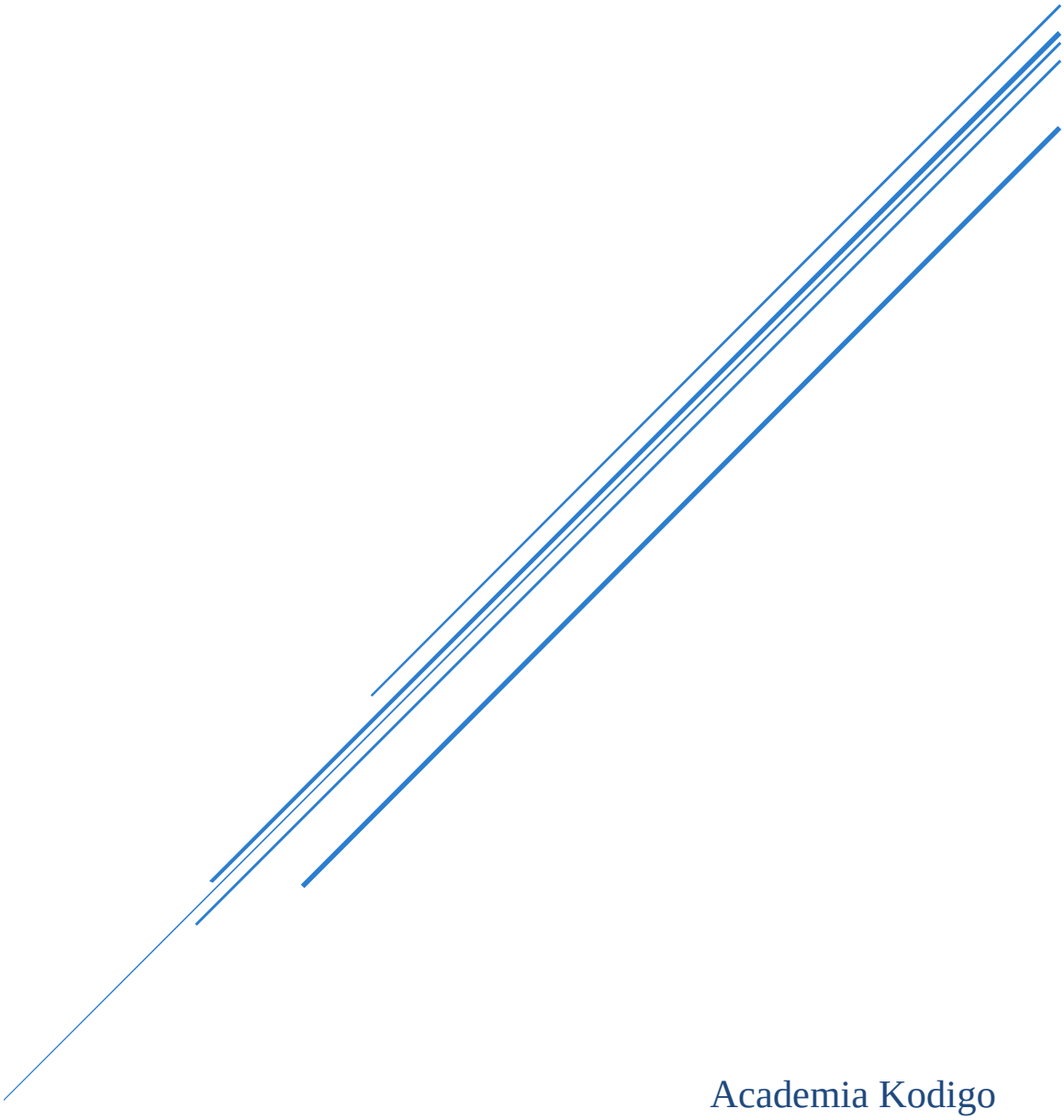
GESTIÓN DE INVENTARIO
ESQUEMA MOVIMIENTOS

Carlos Alfredo Puente Murillo & Cristian Alfredo Alas Castellanos



Academia Kodigo
Java Developer 16

Índice

Introducción I

Arquitectura 2

Diagrama Entidad-Relación: 2

Arquitectura en Capas 2

1. Capa de Controlador (Controller Layer) 2

2. Capa de Servicio (Service Layer) 3

3. Capa de Repositorio (Repository Layer) 3

4. Capa de Entidades (Entity Layer) 4

5. Capa de DTOs y Mappers (DTO and Mapper Layer) 5

Principios SOLID Aplicados al Sistema 7

1. Single Responsibility Principle (SRP) - Principio de Responsabilidad Única 7

2. Open/Closed Principle (OCP) - Principio Abierto/Cerrado 7

3. Liskov Substitution Principle (LSP) - Principio de Sustitución de Liskov 8

4. Interface Segregation Principle (ISP) - Principio de Segregación de Interfaces 8

5. Dependency Inversion Principle (DIP) - Principio de Inversión de Dependencias 8

Patrones de Diseño Aplicados al Sistema 9

1. Patrón de Diseño Factory 9

2. Patrón de Diseño Facade 9

3. Patrón de Diseño Decorator 9

4. Patrón de Diseño Observer 10

5. Patrón de Diseño Repository 10

6. Patrón de Diseño DTO (Data Transfer Object) 10

Detalles de Implementación 11

Descripción de las Principales Funcionalidades 11

Esquema Individual 11

1. Category (Categoría) 11
2. Brand (Marca) 11
3. Supplier (Proveedor) 12
4. Store (Tienda) 12
5. Aisle (Pasillo) 13
6. Shelf (Estante) 13
Explicación del Flujo de Datos y Procesos 13
Procesos: 14
Esquema Movimientos 15
1. Purchase (Compra a Proveedor) 15
2. Receive (Recepción de Productos en Tienda) 15
3. Sale (Venta de Productos en Tienda) 16
Explicación del Flujo de Datos y Procesos 16
Lista de Endpoints Disponibles 18
Purchase (Compra a Proveedor) 18
POST /api/v1/purchases 18
PUT /api/v1/purchases/{id} 19
DELETE /api/v1/purchases/{id} 19
Receive (Recepción de Productos en Tienda) 20
POST /api/v1/receives 20
PUT /api/v1/receives/{id} 20
DELETE /api/v1/receives/{id} 20
Sale (Ventas) 21
POST /api/v1/sales 21
PUT /api/v1/sales/{id} 21
DELETE /api/v1/sales/{id} 21
Esquema Individual. 22

POST /api/v1/products 22
PUT /api/v1/products/{code} 22
DELETE /api/v1/products/{code} 23
Category (Categoría) 23
GET /api/v1/categories 23
POST /api/v1/categories 23
PUT /api/v1/categories/{categoryCode} 23
DELETE /api/v1/categories/{categoryCode} 23
Brand (Marca) 24
GET /api/v1/brands 24
POST /api/v1/ brands 24
PUT /api/v1/ brands /{brandCode} 24
DELETE /api/v1/ brands /{brandCode} 24
Supplier (Proveedor) 25
GET /api/v1/suppliers 25
POST /api/v1/suppliers 25
PUT /api/v1/suppliers/{supplierCode} 25
DELETE /api/v1/ suppliers/{supplierCode} 25
Store (Tienda) 26
POST /api/v1/stores 26
PUT /api/v1/stores/{storeCode} 26
GET /api/v1/stores 27
Descripción de las Pruebas Realizadas 27
Ejemplos de Casos de Prueba 27
Resumen 31
Posibles Mejores Futuras: 32
Conclusión 34

Introducción

Descripción General del Sistema

El Sistema Gestor de Inventario es una aplicación integral desarrollada para optimizar la gestión de inventarios en una tienda. Este sistema permite realizar compras a proveedores, recibir productos en tiendas específicas y gestionar las ventas de manera eficiente. La aplicación está diseñada para asegurar que cada operación se registre y actualice correctamente, proporcionando una herramienta robusta y fiable para la administración de inventarios.

Este documento proporciona una descripción exhaustiva de la arquitectura del sistema, los principios SOLID aplicados, los patrones de diseño utilizados, los detalles de implementación, los endpoints de la API y las pruebas realizadas. Cada sección está diseñada para ofrecer una visión clara y detallada del funcionamiento interno del sistema y su desarrollo.

Propósito del Documento

El propósito de este documento es proporcionar una visión detallada del diseño y la implementación del Sistema Gestor de Inventario. Su objetivo es facilitar la comprensión y el mantenimiento del sistema, ofreciendo información clara y estructurada sobre los componentes clave, la lógica de negocio y los procesos involucrados en la gestión del inventario. Además, este documento servirá como guía para desarrolladores y otros interesados en entender cómo funciona el sistema y cómo se puede ampliar o modificar en el futuro.

Arquitectura

Diagrama Entidad-Relación:

Para poder ver el diagrama de la base de datos, hacer clic en el vínculo:

Link: DiagramaER.pdf

Arquitectura en Capas

El sistema gestor de inventario ha sido desarrollado utilizando una arquitectura en capas. Este enfoque permite una separación clara de responsabilidades y facilita el mantenimiento y la escalabilidad del sistema. A continuación, se detallan las capas que componen el sistema:

1. Capa de Controlador (Controller Layer)
   - Función

     Esta capa gestiona las solicitudes HTTP entrantes. Los controladores actúan como una interfaz entre el cliente y la lógica de negocio, direccionando las solicitudes hacia los servicios adecuados.
   - Implementación

     Los controladores están anotados con @RestController y definen los endpoints del sistema.
   - Ejemplo de un controlador:

```java
@Tag(name = "Purchase Management", description = "APIs for managing purchases")
@RestController
@RequestMapping("/api/v1/purchases")
@RequiredArgsConstructor
public class PurchaseController {

    private final PurchaseService purchaseService;

    @Operation(summary = "Create a new purchase", description = "Creates a new purchase")
    @PostMapping
    public ResponseEntity<PurchaseResponse> createPurchase(@Valid @RequestBody PurchaseRequest request) {
        PurchaseResponse response = purchaseService.createPurchase(request);
        return ResponseEntity.created(URI.create("/api/v1/purchases/" + response.getId())).body(response);
    }
```

2. Capa de Servicio (Service Layer)

- Función

  Contiene la lógica de negocio del sistema. Los servicios son responsables de ejecutar las operaciones de negocio y se comunican con los repositorios para acceder a la base de datos.

- Implementación

  Los servicios están anotados con @Service y encapsulan la lógica de negocio.

- Ejemplo de un servicio:

```java
@Service
@RequiredArgsConstructor
public class PurchaseServiceImpl implements PurchaseService {

    private final PurchaseRepository purchaseRepository;
    private final ProductRepository productRepository;
    private final CategoryRepository categoryRepository;
    private final BrandRepository brandRepository;
    private final ModelProductRepository modelProductRepository;
    private final SupplierRepository supplierRepository;
    private final PurchaseMapper purchaseMapper;

    private static final Logger logger = LoggerFactory.getLogger(PurchaseServiceImpl.class);

    @Override 1 usage
    @Transactional
    public PurchaseResponse createPurchase(PurchaseRequest request) {
        Supplier supplier = supplierRepository.findById(request.getSupplierId())
                .orElseThrow(SupplierNotFoundException::new);

        Purchase purchase = new Purchase();
```

3. Capa de Repositorio (Repository Layer)

- Función

  Esta capa maneja la persistencia de datos utilizando JPA (Java Persistence API). Los repositorios son responsables de las operaciones CRUD (Create, Read, Update, Delete) y las consultas específicas.

- Implementación

  Los repositorios están anotados con @Repository y extienden JpaRepository u otras interfaces de Spring Data JPA.

- Ejemplo de un repositorio:

```java
public interface PurchaseRepository extends JpaRepository<Purchase, Long> {

    Optional<Purchase> findById();

}
```

4. Capa de Entidades (Entity Layer)

- Función

  Define las entidades del dominio del sistema, que representan las tablas de la base de datos y sus relaciones.

- Implementación

  Las entidades están anotadas con @Entity y mapean las tablas de la base de datos.

- Ejemplo de una entidad:

```java
@Getter  18 usages
@Setter
@Entity
@Table(name = "purchases")
public class Purchase extends Auditable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true, nullable = false)
    private String code;

    @ManyToOne
    @JoinColumn(name = "supplier_id", nullable = false)
    private Supplier supplier;

    @OneToMany(mappedBy = "purchase", cascade = CascadeType.ALL, orphanRemoval = true)
    private Set<PurchaseDetail> details = new HashSet<>();
}
```

5. Capa de DTOs y Mappers (DTO and Mapper Layer)

- Función

  Utiliza DTOs (Data Transfer Objects) para transferir datos entre las distintas capas. Los mappers convierten entre entidades y DTOs.

- Implementación

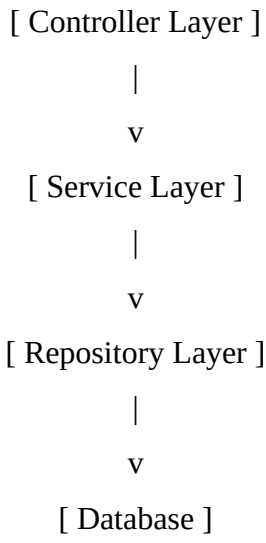
  Los DTOs son clases simples que contienen los datos necesarios para las operaciones del sistema. Los mappers utilizan MapStruct para convertir entre entidades y DTOs.

- Ejemplo de un DTO y un mapper:

```java
@Getter  11 usages
@Setter
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class PurchaseRequest {

    @NotNull(message = "The field supplierId cannot be null")
    private Long supplierId;

    @NotEmpty(message = "The details cannot be empty or null")
    private Set<PurchaseDetailRequest> details;
}
```

```java
@Mapper(componentModel = "spring")  3 usages  1 implementation
public interface PurchaseMapper {

    PurchaseResponse toResponse(Purchase purchase);  1 implementation

    @Mapping(target = "id", ignore = true)  1 implementation
    @Mapping(target = "details", ignore = true)
    Purchase toEntity(PurchaseRequest request);

    @Mapping(target = "id", ignore = true)  no usages  1 implementation
    @Mapping(target = "product", ignore = true)
    PurchaseDetail toDetailEntity(PurchaseDetailRequest detailRequest);

}
```

Diagrama de la Arquitectura en Capas



Beneficios de la Arquitectura en Capas

- Separación de Responsabilidades: Cada capa tiene una responsabilidad específica, lo que facilita la comprensión y el mantenimiento del código.
- Escalabilidad: Permite añadir nuevas funcionalidades y mejorar las existentes sin afectar otras partes del sistema.
- Reutilización de Código: La lógica de negocio y los accesos a datos se pueden reutilizar en diferentes partes de la aplicación.
- Mantenibilidad: La estructura clara y organizada del código facilita la identificación y resolución de problemas.

Principios SOLID Aplicados al Sistema

Para abarcar de forma general los principios SOLID, decidí profundizar su explicación en los 3 movimientos más importantes del sistemta, Purchase, Receive, y Sale.

1. Single Responsibility Principle (SRP) - Principio de Responsabilidad Única
Cada clase y módulo de tu sistema tiene una única responsabilidad. Esto significa que cada clase se encarga de una parte específica de la lógica del negocio, lo que facilita la comprensión y el mantenimiento del código.

Ejemplo en el sistema:

- ProductServiceImpl: Se encarga exclusivamente de la lógica relacionada con los productos, como la creación, actualización y eliminación de productos.
- PurchaseServiceImpl: Maneja la lógica relacionada con las compras, como la creación y actualización de compras.
- SaleServiceImpl: Administra la lógica de las ventas, gestionando la creación y actualización de las ventas.

Cada uno de estos servicios tiene su responsabilidad claramente definida y no se mezclan con otras partes del sistema.

2. Open/Closed Principle (OCP) - Principio Abierto/Cerrado
Las clases, módulos y funciones están abiertas para extensión, pero cerradas para modificación. Esto significa que se pueden agregar nuevas funcionalidades sin modificar el código existente, minimizando el riesgo de introducir errores.

Ejemplo en el sistema:
Uso de interfaces y herencia: Las interfaces como ProductService, PurchaseService y SaleService permiten la extensión de la funcionalidad a través de la implementación de nuevas clases que las implementen. Esto facilita agregar nuevas operaciones sin cambiar las clases existentes.
Enum MovementType: Se puede extender fácilmente para incluir nuevos tipos de movimientos sin modificar la lógica existente en las clases que ya utilizan este enum.

3. Liskov Substitution Principle (LSP) - Principio de Sustitución de Liskov
Los objetos de una clase derivada deben ser sustituibles por objetos de la clase base sin alterar el funcionamiento del sistema. Esto asegura que las clases derivadas puedan ser usadas en lugar de sus clases base sin problemas.

Ejemplo en el sistema:
Uso de polimorfismo con interfaces: Los servicios como ProductService, PurchaseService y SaleService implementan métodos definidos en sus interfaces respectivas, garantizando que cualquier implementación de estas interfaces pueda sustituirse sin afectar al sistema.

4. Interface Segregation Principle (ISP) - Principio de Segregación de Interfaces
Los clientes no deberían verse obligados a depender de interfaces que no utilizan. Esto significa que las interfaces deben ser específicas y limitadas a métodos que sean relevantes para la implementación concreta.

Ejemplo en el sistema:
Interfaces segregadas: En lugar de tener una interfaz gigante con métodos para todas las operaciones posibles, tu sistema tiene interfaces específicas como ProductService, PurchaseService y SaleService, cada una con métodos relevantes solo para su contexto.

5. Dependency Inversion Principle (DIP) - Principio de Inversión de Dependencias
Las clases de alto nivel no deben depender de clases de bajo nivel, sino de abstracciones. Esto se consigue mediante la inversión de dependencias, donde las dependencias son inyectadas a través de constructores o inyecciones de dependencia.

Ejemplo en el sistema:
Inyección de dependencias mediante @Autowired y @RequiredArgsConstructor: Las dependencias como repositorios y mappers son inyectadas en los servicios a través de la inyección de dependencias. Esto asegura que los servicios dependen de abstracciones (interfaces) y no de implementaciones concretas. La configuración y el manejo de dependencias se realizan a través del framework de Spring, facilitando la implementación del DIP y promoviendo la inversión de control.

Patrones de Diseño Aplicados al Sistema

1. Patrón de Diseño Factory
El patrón Factory abstrae la creación de objetos y permite que las subclases decidan qué clase instanciar. Esto facilita la creación de objetos complejos y promueve la reutilización de código.

Ejemplo en el sistema:
Creación de Productos y Modelos: En el método createOrUpdateProduct del servicio PurchaseServiceImpl, se utiliza una lógica similar a una fábrica para crear productos y modelos, asegurando que se manejen correctamente las dependencias y la inicialización de estos objetos.

2. Patrón de Diseño Facade
El patrón Facade proporciona una interfaz simplificada para un conjunto de interfaces en un subsistema. Facilita el uso de un sistema complejo ocultando su complejidad a través de una interfaz más sencilla.

Ejemplo en el sistema:
Servicios de Movimientos: Los servicios como PurchaseService, ReceiveService y SaleService actúan como fachadas que ocultan la complejidad de las operaciones subyacentes, proporcionando métodos simples para realizar compras, recepciones y ventas, respectivamente.

3. Patrón de Diseño Decorator
El patrón Decorator permite añadir funcionalidad a un objeto de manera dinámica. Proporciona una alternativa flexible a la subclase para extender la funcionalidad.

Ejemplo en el sistema:
Auditoria: Si se implementa la auditoría de cambios en las entidades (por ejemplo, utilizando Spring Data JPA Auditing), se podría usar el patrón Decorator para añadir funcionalidad de auditoría a las entidades sin modificar las clases base.

4. Patrón de Diseño Observer
El patrón Observer define una dependencia de uno a muchos entre objetos, de manera que cuando un objeto cambia de estado, todos sus dependientes son notificados y actualizados automáticamente.

Ejemplo en el sistema:
Actualización de Stock: Cuando se realiza una venta o una recepción, el sistema puede notificar automáticamente a los servicios de inventario para actualizar el stock disponible. Esto asegura que todas las partes del sistema estén sincronizadas con los cambios en el inventario.

5. Patrón de Diseño Repository
El patrón Repository encapsula el acceso a la capa de persistencia y permite a la lógica del negocio interactuar con los datos sin conocer los detalles de acceso a la base de datos.

Ejemplo en el sistema:
Repositorios de JPA: Los repositorios como ProductRepository, PurchaseRepository, SaleRepository, y otros, proporcionan una abstracción para el acceso a los datos, permitiendo a los servicios de negocio realizar operaciones CRUD sin preocuparse por los detalles de la implementación de la base de datos.

6. Patrón de Diseño DTO (Data Transfer Object)
El patrón DTO se utiliza para transferir datos entre las capas de un sistema. Los DTOs son objetos simples que no contienen lógica de negocio, pero encapsulan el estado necesario para las operaciones.

Ejemplo en el sistema:
DTOs para Solicitudes y Respuestas: En el sistema, se utilizan DTOs como PurchaseRequest, PurchaseResponse, SaleRequest, SaleResponse, etc., para transferir datos entre la capa de presentación y la capa de servicios. Esto asegura que los datos estén bien estructurados y facilita la validación y transformación de datos.
Estos patrones de diseño aseguran que el sistema sea flexible, mantenible y escalable, permitiendo la adición de nuevas funcionalidades y la modificación de las existentes con un mínimo impacto en el código base.

Detalles de Implementación

Descripción de las Principales Funcionalidades

Esquema Individual

1. Category (Categoría)

Descripción:

Las categorías permiten organizar los productos en grupos específicos. Cada categoría tiene un nombre único y puede tener múltiples productos asociados.

Estructura:

- ID: Identificador único de la categoría.
- Code: Código único de la categoría.
- Name: Nombre de la categoría.

Proceso:

- Creación: Los usuarios pueden crear nuevas categorías proporcionando un nombre y un código.
- Asociación: Los productos pueden ser asociados a una o más categorías.
- Consulta: Se puede consultar la lista de categorías y los productos asociados a cada una.

2. Brand (Marca)

Descripción:

Las marcas permiten identificar el fabricante o proveedor de los productos. Cada marca tiene un nombre único y puede tener múltiples productos asociados.

Estructura:

- ID: Identificador único de la marca.
- Code: Código único de la marca.
- Name: Nombre de la marca.

Proceso:

- Creación: Los usuarios pueden crear nuevas marcas proporcionando un nombre y un código.
- Asociación: Los productos pueden ser asociados a una o más marcas.
- Consulta: Se puede consultar la lista de marcas y los productos asociados a cada una.

3. Supplier (Proveedor)
Descripción:
Los proveedores suministran los productos al sistema. Cada proveedor tiene información de contacto y puede tener múltiples compras asociadas.
Estructura:

- ID: Identificador único del proveedor.
- Code: Código único del proveedor.
- Name: Nombre del proveedor.
- Address: Dirección del proveedor.
- Phone: Teléfono de contacto del proveedor.

Proceso:

- Creación: Los usuarios pueden crear nuevos proveedores proporcionando la información de contacto.
- Asociación: Las compras (Purchases) pueden ser asociadas a proveedores específicos.
- Consulta: Se puede consultar la lista de proveedores y las compras asociadas a cada uno.

4. Store (Tienda)
Descripción:
Las tiendas son puntos de venta o almacenamiento de productos. Cada tienda tiene información de ubicación y puede tener múltiples recepciones (Receives) y ventas (Sales) asociadas.

Estructura:

- ID: Identificador único de la tienda.
- Code: Código único de la tienda.
- Name: Nombre de la tienda.
- Address: Dirección de la tienda.
- Phone: Teléfono de contacto de la tienda.
- Aisles: Lista de pasillos dentro de la tienda.
- Shelves: Lista de estantes dentro de cada pasillo.

Proceso:

- Creación: Los usuarios pueden crear nuevas tiendas proporcionando la información.
- Asociación: Las recepciones (Receives) y ventas (Sales) pueden ser asociadas a tiendas.
- Consulta: Se puede consultar la lista de tiendas y los productos almacenados en cada una.

5. Aisle (Pasillo)
Descripción:
Los pasillos son secciones dentro de las tiendas que contienen estantes para organizar los productos.

Estructura:

- ID: Identificador único del pasillo.
- Name: Nombre del pasillo.
- Shelves: Lista de estantes dentro del pasillo.

Proceso:

- Creación: Los usuarios pueden crear nuevos pasillos dentro de una tienda.
- Asociación: Los productos pueden ser asociados a pasillos específicos dentro de una tienda.
- Consulta: Se puede consultar la lista de pasillos y los estantes asociados a cada uno.

6. Shelf (Estante)
Descripción:
Los estantes son unidades dentro de los pasillos que contienen productos específicos.

Estructura:

- ID: Identificador único del estante.
- Name: Nombre del estante.
- Products: Lista de productos almacenados en el estante.

Proceso:

- Creación: Los usuarios pueden crear nuevos estantes dentro de un pasillo.
- Asociación: Los productos pueden ser almacenados en estantes específicos dentro de un pasillo.
- Consulta: Se puede consultar la lista de estantes y los productos almacenados en cada uno.

Explicación del Flujo de Datos y Procesos
Flujo de Datos:
Recepción de Solicitudes:

- Todas las solicitudes (Purchase, Receive, Sale) son recibidas a través de los controladores REST correspondientes.
- Los controladores validan las solicitudes y las transforman en objetos de transferencia de datos (DTOs).

Lógica de Negocio:

- Los servicios correspondientes (PurchaseService, ReceiveService, SaleService) manejan la lógica de negocio.

- Los servicios utilizan repositorios para interactuar con la base de datos y realizar operaciones CRUD.

Persistencia:

- Los repositorios gestionan la persistencia de datos en la base de datos utilizando JPA.
- Las entidades son mapeadas a tablas de base de datos, y las operaciones de persistencia son manejadas automáticamente por JPA.

Generación de Respuestas:

- Los servicios generan respuestas basadas en los resultados de las operaciones de negocio.
- Los controladores transforman estas respuestas en objetos JSON y los devuelven al cliente.

Procesos:

Validación:
Las solicitudes son validadas para asegurar que contienen toda la información necesaria y que los datos referenciados existen en el sistema.

Transformación:
Los DTOs son transformados en entidades para realizar operaciones de negocio.
Las entidades se transforman nuevamente en DTOs para enviar respuestas al cliente.

Transacciones:
Las operaciones de negocio que afectan múltiples entidades se manejan dentro de transacciones para asegurar la consistencia de los datos.
Si alguna operación dentro de una transacción falla, toda la transacción se revierte para mantener la integridad de los datos.

Esquema Movimientos

1. Purchase (Compra a Proveedor)

Descripción:

La funcionalidad de "Purchase" permite registrar la compra de productos a proveedores. Esto incluye la creación de productos y sus modelos, la asociación con categorías y marcas, y la actualización del inventario con las cantidades compradas.

Proceso:

- Solicitud de Compra: El cliente envía una solicitud de compra que incluye detalles del proveedor y productos a comprar.
- Validación: Se valida la existencia del proveedor y las categorías, marcas y productos mencionados.
- Creación de Productos y Modelos: Se crean los productos y sus modelos según la solicitud.
- Actualización del Inventario: Se actualiza el inventario con las cantidades de productos compradas.
- Persistencia: Los datos de la compra se guardan en la base de datos.
- Respuesta: Se devuelve una respuesta con los detalles de la compra registrada.

2. Receive (Recepción de Productos en Tienda)

Descripción:

La funcionalidad de "Receive" permite registrar la recepción de productos en una tienda específica. Esto incluye la asignación de productos a estantes y pasillos dentro de la tienda y la actualización del inventario de la tienda.

Proceso:

- Solicitud de Recepción: El cliente envía una solicitud de recepción que incluye detalles de la tienda y los productos recibidos.
- Validación: Se valida la existencia de la tienda, pasillos y estantes mencionados en la solicitud.
- Asignación de Productos: Los productos se asignan a los pasillos y estantes especificados en la tienda.

- Actualización del Inventario de Tienda: Se actualiza el inventario de la tienda con las

cantidades de productos recibidos.

- Persistencia: Los datos de la recepción se guardan en la base de datos.
- Respuesta: Se devuelve una respuesta con los detalles de la recepción registrada.

3. Sale (Venta de Productos en Tienda)

Descripción:
La funcionalidad de "Sale" permite registrar la venta de productos desde una tienda específica. Esto incluye la disminución de las cantidades de productos en el inventario de la tienda y el registro de detalles de la venta.

Proceso:

- Solicitud de Venta: El cliente envía una solicitud de venta que incluye detalles de la tienda y los productos vendidos.
- Validación: Se valida la existencia de la tienda y los productos en el inventario de la tienda.
- Actualización del Inventario de Tienda: Se disminuye la cantidad de productos vendidos del inventario de la tienda.
- Registro de Detalles de Venta: Se registran los detalles de la venta, incluyendo precios y cantidades.
- Persistencia: Los datos de la venta se guardan en la base de datos.
- Respuesta: Se devuelve una respuesta con los detalles de la venta registrada.

Explicación del Flujo de Datos y Procesos
Flujo de Datos:
Recepción de Solicitudes:

- Todas las solicitudes (Purchase, Receive, Sale) son recibidas a través de los controladores REST correspondientes.
- Los controladores validan las solicitudes y las transforman en objetos de transferencia de datos (DTOs).

Lógica de Negocio:

- Los servicios correspondientes (PurchaseService, ReceiveService, SaleService) manejan la lógica de negocio.
- Los servicios utilizan repositorios para interactuar con la base de datos y realizar operaciones CRUD.

Persistencia:

- Los repositorios gestionan la persistencia de datos en la base de datos utilizando JPA.
- Las entidades son mapeadas a tablas de base de datos, y las operaciones de persistencia son manejadas automáticamente por JPA.

Generación de Respuestas:
Los servicios generan respuestas basadas en los resultados de las operaciones de negocio.

Los controladores transforman estas respuestas en objetos JSON y los devuelven al cliente.

Procesos:

Validación:
Las solicitudes son validadas para asegurar que contienen toda la información necesaria y que los datos referenciados existen en el sistema.

Transformación:
Los DTOs son transformados en entidades para realizar operaciones de negocio.

Las entidades se transforman nuevamente en DTOs para enviar respuestas al cliente.

Transacciones:
Las operaciones de negocio que afectan múltiples entidades se manejan dentro de transacciones para asegurar la consistencia de los datos.

Si alguna operación dentro de una transacción falla, toda la transacción se revierte para mantener la integridad de los datos.

Actualización de Inventarios:
Las operaciones de Purchase y Receive incrementan los inventarios de productos en el sistema.

Las operaciones de Sale disminuyen los inventarios de productos en la tienda específica.

Registro de Movimientos:

Cada operación de Purchase, Receive, y Sale registra un movimiento correspondiente en la base de datos. Los detalles de los movimientos se almacenan para proporcionar un historial de transacciones y facilitar auditorías. Estos procesos y flujos de datos aseguran que el sistema de gestión de inventario sea robusto, confiable y fácil de mantener, permitiendo una gestión eficiente de inventarios, compras, recepciones y ventas en diferentes tiendas.

Lista de Endpoints Disponibles

Purchase (Compra a Proveedor)

POST /api/v1/purchases

- **Descripción:** Crea una nueva compra a un proveedor.
- **Ejemplo de Uso:**

```
{
  "supplier_id": 1,
  "details": [
    {
      "name": "Martillo",
      "description": "Descripción para Martillo",
      "price": 25,
      "category_ids": [1],
      "brand_ids": [1],
      "models": [
        {
          "name": "Modelo A"
        },
        {
          "name": "Modelo B"
        },
        {
          "name": "Modelo C"
        }
      ],
      "quantity": 30,
      "status": "ACTIVO"
    }
  ]
}
```

PUT /api/v1/purchases/{id}

- **Descripción:** Actualiza una compra existente.
- **Ejemplo de Uso:**

```
{
  "supplier_id": 2,
  "details": [
    {
      "name": "Luces Led",
      "description": "Descripción para luces led",
      "price": 150,
      "category_ids": [2],
      "brand_ids": [1, 2],
      "models": [
        {
          "name": "Modelo SA"
        },
        {
          "name": "Modelo SB"
        },
        {
          "name": "Modelo SC"
        }
      ],
      "quantity": 40,
      "status": "ACTIVO"
    }
  ]
}
```

DELETE /api/v1/purchases/{id}

- **Descripción:** Elimina una compra existente.
- **Ejemplo de Uso:**

```
DELETE /api/v1/purchases/1
```

Receive (Recepción de Productos en Tienda)

POST /api/v1/receives

- **Descripción:** Registra la recepción de productos en una tienda.
- **Ejemplo de Uso:**

```
{
  "store_id": 1,
  "supplier_id": 1,
  "details": [
    {
      "product_id": 1,
      "aisle_id": 1,
      "shelf_id": 1,
      "quantity": 10
    }
  ]
}
```

PUT /api/v1/receives/{id}

- **Descripción:** Actualiza una recepción de productos en una tienda.
- **Ejemplo de Uso:**

```
{
  "store_id": 1,
  "supplier_id": 2,
  "details": [
    {
      "product_id": 2,
      "aisle_id": 1,
      "shelf_id": 1,
      "quantity": 15
    }
  ]
}
```

DELETE /api/v1/receives/{id}

- **Descripción:** Elimina una recepción de productos en una tienda.
- **Ejemplo de Uso:**

```
DELETE /api/v1/receives/1
```

Sale (Ventas)

POST /api/v1/sales

- **Descripción:** Registra una nueva venta de productos desde una tienda.
- **Ejemplo de Uso:**

```
{
  "store_id": 1,
  "type": "SALE",
  "details": [
    {
      "product_id": 1,
      "quantity": 5,
      "sale_price": 20.00
    }
  ]
}
```

PUT /api/v1/sales/{id}

- **Descripción:** Actualiza una venta existente.
- **Ejemplo de Uso:**

```
{
  "store_id": 1,
  "type": "SALE",
  "details": [
    {
      "product_id": 1,
      "quantity": 3,
      "sale_price": 22.00
    }
  ]
}
```

DELETE /api/v1/sales/{id}

- **Descripción:** Elimina una venta existente.
- **Ejemplo de Uso:**

```
DELETE /api/v1/sales/1
```

Esquema Individual.
A pesar de que a la hora crear una nueva compra, se crea un producto desde cero para poder insertarlo al sistema. También existen endpoint para manejar el esquema de productos de forma individual.

POST /api/v1/products

Descripción: Crea un nuevo producto.

Ejemplo de Uso:

```
{
  "name": "Luces Led",
  "description": "Descripción para luces led",
  "price": 27.15,
  "category_ids": [1, 2],
  "brand_ids": [1, 2],
  "models": [
    {
      "name": "Modelo A"
    },
    {
      "name": "Modelo B"
    },
    {
      "name": "Modelo C"
    }
  ],
  "status": "ACTIVO"
}
```

PUT /api/v1/products/{code}

Descripción: Actualiza un producto existente.

Ejemplo de Uso:

```
{
  "name": "Fresa",
  "description": "Descripción para luces Fresa",
  "price": 30.00,
  "category_ids": [1, 2],
  "brand_ids": [1, 2],
  "models": [
    {
      "name": "Modelo D"
    },
    {
      "name": "Modelo E"
    },
    {
      "name": "Modelo F"
    }
  ],
```

```
  "status": "ACTIVO"
}
```

DELETE /api/v1/products/{code}

Descripción: Elimina un producto existente.

Ejemplo de Uso:

```
DELETE /api/v1/products/PRD0001
```

Category (Categoría)
GET /api/v1/categories

- **Descripción:** Obtiene la lista de todas las categorías.
- **Ejemplo de Uso:**

```
GET /api/v1/categories
```

POST /api/v1/categories

- **Descripción:** Crea una nueva categoría.
- **Ejemplo de Uso:**

```
{
  "name": "Engine"
}
```

PUT /api/v1/categories/{categoryCode}

- **Descripción:** Modifica una categoría.
- **Ejemplo de Uso:**

```
{
  "name": "Filters"
}
```

DELETE /api/v1/categories/{categoryCode}

- **Descripción:** Elimina una categoría.
- **Ejemplo de Uso:**

```
DELETE /api/v1/categories/CATG0001
```

Brand (Marca)

GET /api/v1/brands

- **Descripción:** Obtiene la lista de todas las marcas.
- **Ejemplo de Uso:**

```
GET /api/v1/brands
```

POST /api/v1/ brands

- **Descripción:** Crea una nueva marca.
- **Ejemplo de Uso:**

```
{
  "name": "Toyota"
}
```

PUT /api/v1/ brands /{brandCode}

- **Descripción:** Modifica una marca.
- **Ejemplo de Uso:**

```
{
  "name": "Mazda"
}
```

DELETE /api/v1/ brands /{brandCode}

- **Descripción:** Elimina una marca.
- **Ejemplo de Uso:**

```
DELETE /api/v1/brands/ BRN0001
```

Supplier (Proveedor)

GET /api/v1/suppliers

- **Descripción:** Obtiene la lista de todos los proveedores.
- **Ejemplo de Uso:**

```
GET /api/v1/suppliers
```

POST /api/v1/suppliers

- **Descripción:** Crea un nuevo proveedor.
- **Ejemplo de Uso:**

```
{
  "name": "Proveedor General",
  "address": "123 Calle Principal",
  "phone": "123-456-7890"
}
```

PUT /api/v1/suppliers/{supplierCode}

- **Descripción:** Modifica un proveedor.
- **Ejemplo de Uso:**

```
{
  "name": "Proveedor Actualziado",
  "address": "456 Calle Principal",
  "phone": "456-456-7890"
}
```

DELETE /api/v1/ suppliers/{supplierCode}

- **Descripción:** Elimina una marca.
- **Ejemplo de Uso:**

```
DELETE /api/v1/suppliers/SUPL0001
```

Store (Tienda)
POST /api/v1/stores

- **Descripción:** Crea una nueva tienda.
- **Ejemplo de Uso:**

```
{
        "name": "Tienda Express",
        "address": "Carretera Principal 5566",
        "phone": "555-567891",
        "aisles": [
            {
                "name": "Aisle 3",
                "shelves": [
                    {"name": "Shelf E"},
                    {"name": "Shelf F"}
                ]
            },
            {
                "name": "Aisle 4",
                "shelves": [
                    {"name": "Shelf G"},
                    {"name": "Shelf H"}
                ]
            }
        ]
    }
```

PUT /api/v1/stores/{storeCode}

- **Descripción:** Modifica una tienda.
- **Ejemplo de Uso:**

```
{
        "name": "Tienda Express",
        "address": "Carretera Principal 5566",
        "phone": "555-567891",
        "aisles": [
            {
                "name": "Aisle 3",
                "shelves": [
                    {"name": "Shelf E"},
                    {"name": "Shelf F"}
                ]
            },
            {
                "name": "Aisle 4",
                "shelves": [
                    {"name": "Shelf G"},
                    {"name": "Shelf H"}
                ]
            }
        ]
    }
```

GET /api/v1/stores

- **Descripción:** Obtiene la lista de todas las tiendas.
- **Ejemplo de Uso:**

```
GET /api/v1/stores
```

Descripción de las Pruebas Realizadas

Para asegurar la correcta implementación y funcionalidad del Sistema Gestor de Inventario, se llevaron a cabo pruebas exhaustivas en cada una de las funcionalidades clave del sistema. Las pruebas fueron diseñadas para verificar tanto el comportamiento esperado como el manejo de situaciones excepcionales. Para poder probar correctamente el funcionamiento del sistema y sus 3 movimientos, se necesita seguir las siguientes indicaciones:

1. Verificar el uso del Esquema 1. Crear un nuevo usuario, luego hacer login, y copiar ese JWT para poder copiarlo en el Bearer en cada enpoint, ya que esto permitirá poder probar las funcionalidades del sistema. Obviamente todo esto está mejor detallado y documentado en el Esquema 1.
2. Hacer por lo menos 1 insert de Category, Brand, Supplier y Store. Esto quiere decir que necesita hacer uso del Endpoint POST para cada uno de los esquemas; de este modo pordemos asegurarnos de utilizar correctamente estos datos después en los movimientos.

Ejemplos de Casos de Prueba
Caso de Prueba 1: Creación de un Producto
Descripción: Verificar que un producto con múltiples modelos pueda ser creado correctamente.

Entrada Esperada:

```
{
  "name": "Mango",
  "description": "Descripción para Mango",
  "price": 27.15,
  "category_ids": [1, 2],
  "brand_ids": [1, 2],
  "models": [
    {
      "name": "Modelo A"
    },
    {
```

```
      "name": "Modelo B"
    },
    {
      "name": "Modelo C"
    }
  ],
  "quantity": 100,
  "status": "ACTIVO"
}
```

Salida Esperada:

```
{
  "id": 1,
  "code": "PRD0001",
  "name": "Mango",
  "description": "Descripción para Mango",
  "price": 27.15,
  "quantity": 100,
  "status": "ACTIVO"
  "categories": [
    {
      "id": 1,
      "code": "CATG0001",
      "name": "Engine"
    }
  ],
  "brands": [
    {
      "id": 1,
      "code": "BRN0001",
      "name": "Toyota"
    }
  ],
  "models": [
    {
      "id": 1,
      "name": "Modelo A"
    },
    {
      "id": 2,
      "name": "Modelo B"
    },
    {
      "id": 3,
      "name": "Modelo C"
    }
  ]
}
```

Caso de Prueba 2: Recepción de Productos en Tienda

- **Descripción:** Verificar que la recepción de productos en una tienda se registre correctamente.
- **Entrada:**

```
{
  "store_id": 1,
  "supplier_id": 1,
  "details": [
    {
      "product_id": 1,
      "aisle_id": 1,
      "shelf_id": 1,
      "quantity": 10
    }
  ]
}
```

- **Salida Esperada:**

```
{
  "id": 1,
  "code": "REC0001",
  "store": {
    "id": 1,
    "code": "STOR0001",
    "name": "Tienda Central",
    "address": "Avenida Principal 12345",
    "phone": "555-123457"
  },
  "supplier": {
    "id": 1,
    "code": "SUPL0001",
    "name": "Proveedor General",
    "address": "123 Calle Principal",
    "phone": "123-456-7890"
  },
  "details": [
    {
      "id": 1,
      "product": {
        "id": 1,
        "code": "PRD0001",
        "name": "Mango",
        "description": "Descripción para Mango",
        "price": 27.15,
        "status": "ACTIVO"
      },
      "aisle": {
        "id": 1,
        "name": "Pasillo A"
      },
      "shelf": {
        "id": 1,
        "name": "Estante A"
      },
      "quantity": 10
    }
  ]
}
```

Caso de Prueba 3: Venta de Productos

- **Descripción:** Verificar que una venta de productos desde una tienda se registre correctamente y las existencias se actualicen.
- **Entrada:**

```
{
  "store_id": 1,
  "type": "SALE",
  "details": [
    {
      "product_id": 1,
      "quantity": 5,
      "sale_price": 20.00
    }
  ]
}
```

- **Salida Esperada:**

```
{
  "id": 1,
  "code": "SAL0001",
  "store": {
    "id": 1,
    "code": "STOR0001",
    "name": "Tienda Central",
    "address": "Avenida Principal 12345",
    "phone": "555-123457"
  },
  "type": "SALE",
  "details": [
    {
      "id": 1,
      "product": {
        "id": 1,
        "code": "PRD0001",
        "name": "Mango",
        "description": "Descripción para Mango",
        "price": 27.15,
        "status": "ACTIVO"
      },
      "quantity": 3,
      "sale_price": 30.00,
      "total": 90.00
    }
  ]
}
```

Resumen

El desarrollo del Sistema Gestor de Inventario ha sido un esfuerzo integral que involucró la implementación de diversas funcionalidades clave para la gestión eficiente de inventarios.

Definición de la Arquitectura del Sistema:

- Se adoptó una arquitectura en capas que separa la lógica de negocio, la lógica de acceso a datos y la presentación.
- Se aseguraron las mejores prácticas de diseño de software mediante la aplicación de los principios SOLID y patrones de diseño.

Implementación de Funcionalidades Principales:

- Purchase (Compra a Proveedor): Se desarrolló la funcionalidad para registrar la compra de productos a proveedores, incluyendo la creación y actualización de productos y la gestión de inventarios.
- Receive (Recepción de Productos en Tienda): Se implementó la recepción de productos en tiendas específicas, gestionando su ubicación en pasillos y estantes.
- Sale (Ventas): Se desarrolló la funcionalidad para registrar las ventas de productos desde tiendas, actualizando las existencias y registrando los detalles de la venta.

Desarrollo de Entidades y Relaciones:

- Se definieron y implementaron entidades fundamentales como Product, Category, Brand, Supplier, Store, Aisle, Shelf, Movement, y sus relaciones.
- Se aseguraron las relaciones Many-to-Many, One-to-Many, y Many-to-One según las necesidades del negocio.

Creación de Endpoints RESTful:

- Se diseñaron y documentaron endpoints RESTful para interactuar con el sistema.
- Se implementaron pruebas exhaustivas utilizando Postman para verificar la correcta funcionalidad de cada endpoint.

Gestión de Errores y Validaciones:

- Se desarrollaron mecanismos para la gestión de errores y excepciones, asegurando respuestas claras y útiles para el usuario.
- Se implementaron validaciones a nivel de DTOs para asegurar la integridad de los datos.

Pruebas y Validación:

Se llevaron a cabo pruebas unitarias e integrales para validar cada funcionalidad del sistema.

Se verificó la correcta integración entre diferentes componentes del sistema y la coherencia de las operaciones.

Posibles Mejores Futuras:

Aunque el sistema desarrollado cumple con los requisitos y proporciona una gestión eficiente del inventario, existen varias mejoras futuras que podrían implementarse para aumentar su funcionalidad y eficiencia:

Mejoras en la Interfaz de Usuario:

- Desarrollo de una interfaz gráfica de usuario (GUI) más intuitiva y fácil de usar para mejorar la experiencia del usuario final.
- Implementación de dashboards interactivos para una mejor visualización y análisis de datos.

Optimización del Rendimiento:

- Implementación de técnicas de optimización de consultas y uso de índices en la base de datos para mejorar el rendimiento del sistema.
- Utilización de caché para reducir el tiempo de respuesta en operaciones frecuentes.

Funcionalidades Adicionales:

- Gestión de Devoluciones: Desarrollo de una funcionalidad para gestionar las devoluciones de productos por parte de los clientes.

Alertas y Notificaciones:

- Implementación de un sistema de alertas y notificaciones para avisar sobre niveles bajos de inventario, pedidos pendientes, etc.

Integración con Otros Sistemas:

- Integración con sistemas de facturación y contabilidad para una gestión completa del ciclo de ventas y compras.
- Conexión con plataformas de comercio electrónico para sincronizar el inventario en tiempo real.

Automatización y Tareas Programadas:

- Implementación de tareas programadas para la actualización automática de precios, generación de informes periódicos, etc.
- Utilización de servicios en la nube para la escalabilidad y disponibilidad del sistema.

Conclusión

El desarrollo del Sistema Gestor de Inventario ha sido una experiencia enriquecedora y desafiante que ha resultado en una herramienta robusta y eficiente para la gestión de inventarios. A lo largo de este proyecto, se han implementado diversas funcionalidades clave que permiten la gestión integral de compras, recepciones de productos en tienda y ventas, asegurando que cada operación sea registrada y actualizada correctamente en el sistema.

La adopción de una arquitectura en capas, junto con la aplicación de principios SOLID y patrones de diseño, ha permitido crear un sistema mantenible, escalable y fácil de entender. Cada capa del sistema cumple un propósito específico, lo que facilita la incorporación de nuevas funcionalidades y la modificación de las existentes sin afectar otras partes del sistema.

Las pruebas exhaustivas realizadas aseguran que cada componente del sistema funcione como se espera y que la integración entre ellos sea fluida. La documentación detallada y la clara definición de endpoints RESTful proporcionan una guía completa para los desarrolladores y usuarios del sistema, permitiendo una fácil interacción y ampliación de este.

El sistema, aunque completo en su estado actual, tiene un gran potencial para futuras mejoras. La implementación de funcionalidades adicionales, optimización del rendimiento, y la integración con otros sistemas de gestión empresarial, asegurarán que el Sistema Gestor de Inventario siga siendo una herramienta valiosa y adaptable a las necesidades cambiantes del negocio.

Este proyecto no solo ha proporcionado una solución práctica para la gestión de inventarios, sino que también ha sido una oportunidad para aplicar y consolidar conocimientos en desarrollo de software, arquitectura de sistemas y buenas prácticas de programación. Estoy convencido de que el Sistema Gestor de Inventario será una herramienta útil y eficiente para cualquier organización que busque mejorar su gestión de inventarios. Con este proyecto, he demostrado mi capacidad para diseñar, desarrollar y entregar soluciones de software de alta calidad, y estoy preparado para enfrentar nuevos desafíos y oportunidades en el campo del desarrollo de software.