Implementar una solución de Backend para una problemática típica de venta de productos en línea que implemente las siguientes características:

- ➔ Descripción del modelo de datos

  Product: represente los artículos que están a la venta
  - **id**: PK
  - **nombre**: nombre del producto
  - **price**: precio del producto
  - **stock**: cantidad de productos en existencias

  Cliente: representa los clientes quienes compran los productos
  - **id**: PK
  - **nombre**: nombre completo del cliente
  - **email**: email del cliente
  - **dirección**: dirección física del cliente

  Pedido: representa un pedido de compra realizado por un cliente.
  - **id**: PK
  - **clienteId**: FK apuntando a Cliente
  - **fechaPedido**: Fecha y hora cuando el pedido fue realizado
  - **status**: estado actual del pedido, los valores podrán ser, PENDIENTE, ENVIADO y ENTREGADO

  ItemPedido: Representa los artículos individuales dentro de un pedido.
  - **id**: PK
  - **pedidoId**: FK a Pedido
  - **productoId**: FK a Producto
  - **cantidad**: cantidad de productos solicitados
  - **precioUnitario**: precio del producto

  Pago: Representa los datos de pago de un pedido.
  - Id: PK
  - pedidoId: FK a pedido
  - totalPago: total cantidad a pagar
  - fechaPago: Fecha cuando se realiza el pago
  - metodoPago: método de pago, estos podrás ser EFECTIVO, TARJETA_CREDITO, PAYPAL, NEQUI, DAVIPLATA, PSE

  DetalleEnvio: representa la información de envío de un pedido.
  - **id**: PK
  - **pedidoId**: FK a pedido
  - **direccion**: dirección de envío
  - **trasnportadora**: Servicio de transporte utilizado para el envío
  - **numeroGuia**: Número de seguimiento del envío

  Las relaciones entre las entidades son:

- Un cliente puede tener varios pedidos.
- Un pedido puede tener varios OrderItems y está vinculado a un único cliente.
- Un OrderItem está vinculado a un único Producto y a un único Pedido.
- Un producto puede formar parte de varios artículos de pedido.
- Un pago está vinculado a un único pedido.
- Los detalles de envío están vinculados a un único pedido.

➔ Implementar la capa de persistencia utilizando el motor de base de datos Postgresql

➔ Para la capa de resistencia aparte de los métodos típcos del CRUD, se implementarán los siguientes métodos de búsqueda de datos:

ProductoRepository

- buscar productos según un término de búsqueda
- Buscar los productos que están en stock.
- Buscar los productos que no superen un precio y un stock determinado

ClienteRepository

- encontrar clientes por email
- Encontrar clientes por dirección
- Encontrar clientes por todos los clientes que comiencen por un nombre

PedidoRepository

- buscar pedidos entre dos fecha
- Buscar pedidos por cliente y un estado
- recuperar pedidos con sus artículos usando JOIN fetch para evitar el problema N+1, para un cliente específico

PagoRepository

- Recuperar pagos dentro de un rango de fecha
- Recuperar pagos por un identificador de una orden y método de pago

ItemPedidoRepository

- Buscar Items del pedido por Pedido Id
- Buscar items del pedido para un producto específico
- Calcular la suma del total de ventas para un producto, utilice la agregación SUM

DetalleEnvioRepository

- Buscar los detalles del envío por pedido Id

- Buscar los detalles de envío para un transportadora
- Buscar los detalles de envio por estado

➔ Para la capa de persistencia se deberán implementar las pruebas de integración usando **spring boot test**, **junit** y **testcontainer**

➔ Se deberán implementar pruebas de integración para todos las clases Repository implementadas

➔ Se deberán implementar casos de pruebas para las operaciones básicas del CRUD y para todo los métodos de búsquedas agreagos en los repositorios
➔ Implementar una capa de servicios con lógica de negoció para todos los métodos del CRUD y búsquedas definidos en el apartado del repositorio
➔ Para la implementación de los DTO Mapper se debe utilizar **MapStruct**
➔ Implementar las pruebas de unidad para todas las clases de la capa de repositorio cumpliendo con un 100% del coverage de esta capa
➔ Utilizar **mockito** para la implementación de los test de unidad para la capa de servicios
➔ Implementar la capa de controlador para los siguientes endpoints

ProductoController

Endpoint	Método	Path Variable	Request Param	Request Body
/api/v1/products/{id}	GET	id		
/api/v1/products/search	GET		searchTerm	
/api/v1/products/instock	GET			
/api/v1/products	POST			Objeto Producto
/api/v1/products/{id}	PUT	id		Objeto Producto
/api/v1/products/{id}	DELETE			

ClienteController

Endpoint	Método	Path Variable	Request Param	Request Body

/api/v1/customers/{id}	GET	id		
/api/v1/customers	GET all productos			
/api/v1/customers/email/{email}	GET	email		
/api/v1/customers/city?cityName=	GET		cityName	
/api/v1/customers	POST			Objeto Cliente
/api/v1/customers/{id}	PUT	id		Objeto Ciente
/api/v1/customers/{id}	DELETE			

PedidoController

Endpoint	Método	Path Variable	Request Param	Request Body
/api/v1/orders/{id}	GET	id		
/api/v1/orders	GET all productos			
/api/v1/orders/customer/{customerId}	GET pedido por cliente Id	customerId		
/api/v1/orders/date-range?startDate= & endDate=	GET		startDate endDate	
/api/v1/orders	POST			Objeto Cliente
/api/v1/orders/{id}	PUT	id		Objeto

				Ciente
/api/v1/orders/{id}	DELETE			

PagoController

Endpoint	Método	Path Variable	Request Param	Request Body
/api/v1/payments/{id}	GET	id		
/api/v1/payments	GET all productos			
/api/v1/payments/order/{orderId}	GET	orderId		
/api/v1/payments/date-range?startDate= & endDate=	GET		startDate endDate	
/api/v1/payments	POST			Objeto Payment
/api/v1/payments/{id}	PUT	id		Objeto Payment
/api/v1/payments/{id}	DELETE			

ItemPedidoController

Endpoint	Método	Path Variable	Request Param	Request Body
/api/v1/order-items/{id}	GET	id		
/api/v1/order-items	GET all			

	items			
/api/v1/order-items/order/{orderId}	GET	orderId		
/api/v1/order-items/product/{productId}	GET	productId		
/api/v1/order-items	POST			Objeto ItemPedido
/api/v1/order-items/{id}	PUT	id		Objeto ItemPedido
/api/v1/order-items/{id}	DELETE			

DetalleEnvioController

Endpoint	Método	Path Variable	Request Param	Request Body
/api/v1/shipping/{id}	GET	id		
/api/v1/shipping	GET all items			
/api/v1/shipping/order/{orderId}	GET	orderId		
/api/v1/shipping/carrier?name=	GET		name	
/api/v1/shipping	POST			Objeto ItemPedido
/api/v1/shipping/{id}	PUT	id		Objeto ItemPedido
/api/v1/shipping/{id}	DELETE			

REFERENCIAS

JPA
https://blog.stackademic.com/the-complete-guide-to-spring-data-jpa-building-a-bookstore-application-from-scratch-part-i-3f8ba9d13b10

https://blog.stackademic.com/the-complete-guide-to-spring-data-jpa-building-a-bookstore-application-from-scratch-part-ii-28f70149fa9

https://blog.stackademic.com/the-complete-guide-to-spring-data-jpa-building-a-bookstore-application-from-scratch-part-iii-8a1de3bc9949

DTO, Mapstruct y Spring Boot
https://dzone.com/articles/thats-why-you-need-to-use-mapstruct-in-your-spring

https://blog.stackademic.com/efficient-data-transfer-in-rest-apis-a-deep-dive-into-the-dto-pattern-with-spring-boot-and-mysql-df2bdf1ece74

https://mayankposts.medium.com/simplify-model-mapping-in-spring-boot-with-mapstruct-and-lombok-65d93b56a76b

Test Unitarios y de Integración en spring
https://blog.stackademic.com/a-comprehensive-guide-to-testing-spring-boot-with-junit-and-mockito-1b3d7bf00154

https://reflectoring.io/unit-testing-spring-boot/

https://medium.com/javarevisited/restful-api-testing-in-java-with-mockito-controller-layer-f4605f8ffaf3

https://medium.com/javarevisited/restful-api-testing-in-java-with-mockito-service-layer-4d0e5dc58023

https://refactorizando.com/ejemplos-testing-spring-boot/