CODERHOUSE

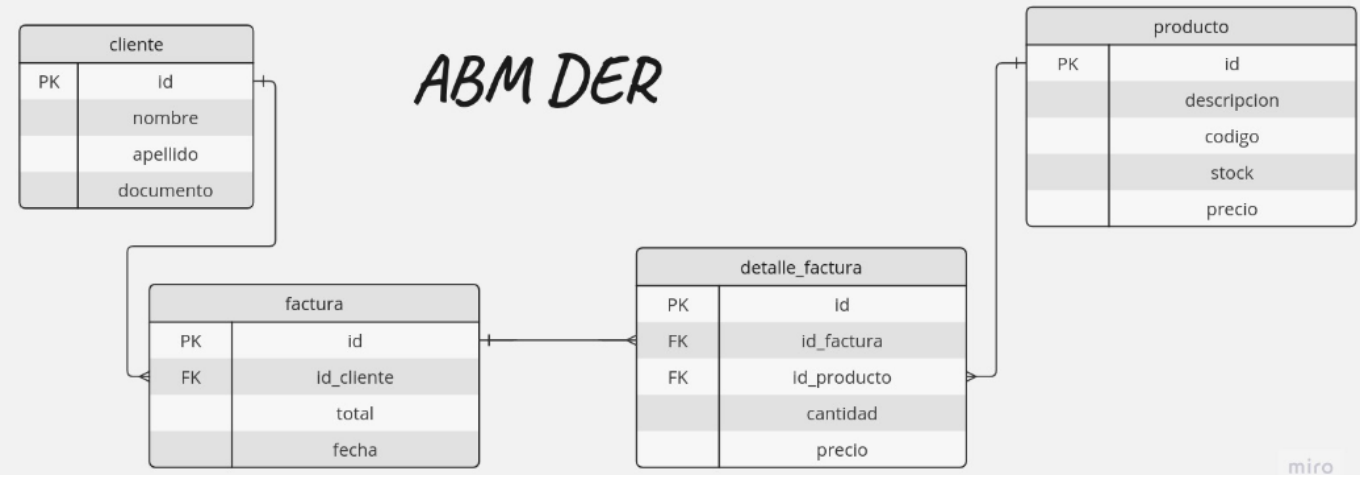
# Java Inicial: Proyecto Final

## ABM (Alta - baja - modificación)

Desarrollarás un sistema de facturación ABM para poder generar facturas utilizando la arquitectura API RESTful con Spring Boot como framework. El sistema debe ser capaz de dar de alta, baja y modificar clientes y productos. Una vez generados se pueden generar comprobantes (invoices) cuya información se detalla a continuación.

## DER

A continuación se adjunta el diagrama entidad relación que se espera del ABM:

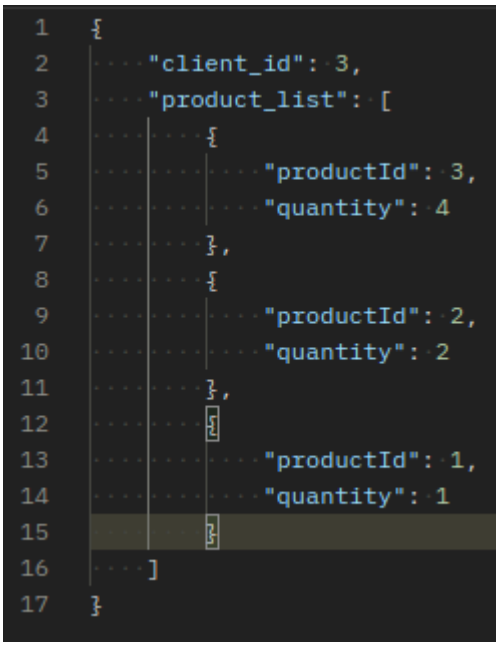



## Features esperados:

● Los datos del usuario a guardar serán: **nombre** (string), **apellido** (string), y **número de documento**(number). Si alguno de los campos viene vacío o no coincide con el tipo descrito se debe enviar el error correspondiente en la respuesta.

● Los datos del producto a guardar serán: **título** (string), **descripción** (string), **stock** (number), **precio** (double) y **código interno** (string). Si alguno de los campos viene vacío o no coincide con el tipo descrito se debe enviar el error correspondiente en la respuesta.

- Para generar los comprobantes, se requiere la siguiente información:



```
{
    "client_id": 3,
    "product_list": [
        {
            "productId": 3,
            "quantity": 4
        },
        {
            "productId": 2,
            "quantity": 2
        },
        {
            "productId": 1,
            "quantity": 1
        }
    ]
}
```

*Invoice request*

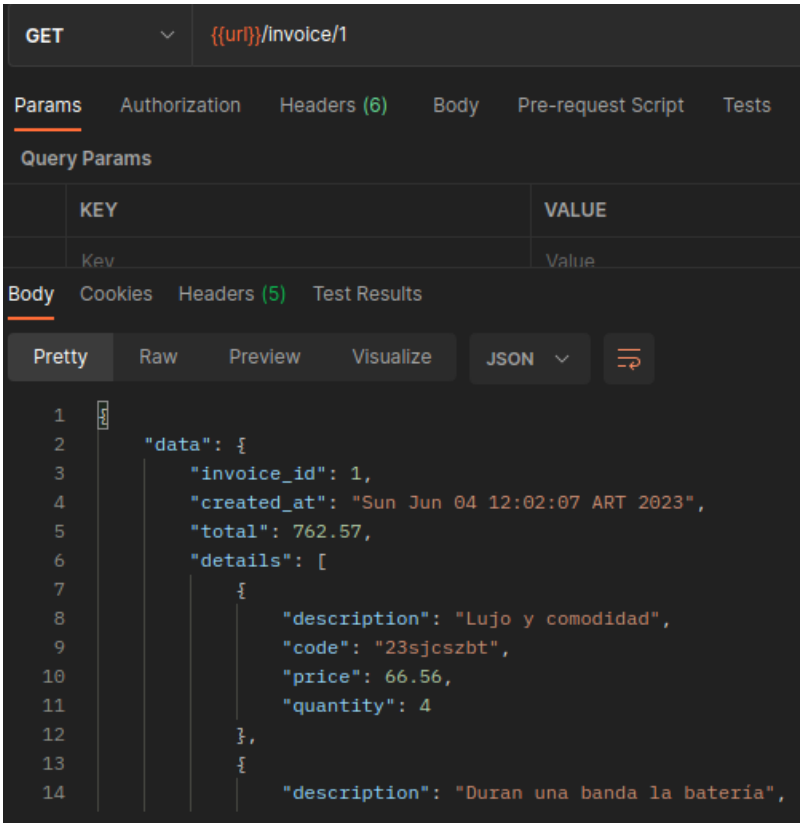
*Validaciones*: debe validarse que exista el cliente y el producto, además verificar que la quantity sea menor al stock del producto a la hora de comprar, sino informar. Una vez realizada la compra, se debe **disminuir** el stock del producto de la tabla de productos.

Por cada línea del comprobante se debe guardar un registro en la tabla **“invoice_detail”**, esta tabla es inmutable, no debe poder modificarse, al igual que la tabla comprobante.

En el invoice_detail o detalle de factura debe figurar:

- ➔ id_factura
- ➔ id_producto
- ➔ precio
- ➔ cantidad

- Para el caso del modelado de la respuesta al GET de un comprobante se adjunta un ejemplo:




*Faltaría el client_id en el nivel superior, al mismo nivel que invoice_id y también el title del product a nivel del "details"*

- Para todas las tablas, los id's deben ser **autogenerados** implementando la estrategia que vean más conveniente (autoincremental, uuid, etc)

- Para los métodos GET de obtención hay distintos requisitos según las entidades:
  - ➔ Cliente: por id.
  - ➔ Producto: listado completo y por id.
  - ➔ Comprobante: listado de comprobantes por id de cliente, y obtención por id.
  - ➔ Detalle: N/A.

**Nota**: en caso de no existir debe informarse en la respuesta de la solicitud, enviando un "null" en la data, con status 200.

- Por último los métodos PUT y DELETE sólo se solicitarán para las entidades de **Client y Product** mediante su id (excluyendo Invoice e InvoiceDetail)

## Piezas sugeridas

**Te recomendamos incluir:**

- Model:

1. Para cliente: Client, ClientDTO (opcional)
2. Para producto: Product, ProductDTO (opcional)
3. Para el comprobante: Invoice, InvoiceDTO (obligatorio), InvoiceRequest (obligatorio)
4. Para el detale del comprobante: InvoiceDetail, InvoiceDetailDTO (opcional), InvoiceDetailRequest (opcional)

- Controller:
  - ClientController
  - ProductController
  - InvoiceController
- Service:
  - ClientService
  - ProductService
  - InvoiceService
  - InvoiceDetailService
- Repository:
  - ClientRepository
  - ProductRepository
  - InvoiceRepository
  - InvoiceDetailRepository
- Middleware (opcional):
  - ResponseHandler

## Requisitos adicionales

**Los requisitos adicionales no son parte del código, pero serán parte de los criterios de evaluación para aprobar el proyecto.**

- Adjuntar el archivo readme.MD que es la puerta de entrada para comprender el proyecto. No realizar una introducción de más de 3 o 4 líneas en dicho archivo.
- Una carpeta “doc” en el directorio raíz con todas las imágenes, apuntes y todo lo que consideren pertinente a la hora de instalar y correr el proyecto.
- Una carpeta “collection” donde estará exportada la colección de Postman para realizar las peticiones. Colocarle **NOMBRE Y APELLIDO** a la collection para identificarla fácilmente.
- Script de creación de tablas (no importa si se utiliza o no, debe estar igual)
- De manera OBLIGATORIA se debe adjuntar el link al repositorio de Github para poder descargar y corregir el código.

## RECORDAR

**En caso que el proyecto arroje errores al ejecutarlo se considera como incorrecto y no estará en condiciones de aprobar.**

## Dont's

**No realizar las siguientes acciones**

- Agregar dependencias que no apliquen al proyecto, por ejemplo Spring Boot Security.
- Agregar funcionalidades que no se piden en el proyecto (principio YAGNI)
- Utilizar un framework distinto al visto en clase.

*Cualquier falta cometida en los dont's generará desaprobación inmediata.*