Bootcamp Crisalis

Orange Team

Faccipieri, Federico

Fleitas, Gonzalo

Flores, Elio

Haoys, Sebastián

Pepa Degani, Julián

Estrategias del Backend

Responsable: Pepa Degani, Julián

JIRA: SCRUM-5 Back End

Descripción

Las estrategias y la organización de la estructura del backend en el desarrollo de software es esencial para construir sistemas eficientes, mantenibles y escalables. Es por ello que para lograr un sistema con dichas características, se llevaron a cabo las siguientes estrategias de trabajo.

Requerimientos

- Se separó la aplicación en **dos grandes capas**: una referente al frontend y otra al backend.

- Se implementó el principio de **Separación de Responsabilidades**, es decir, se diseñó y aplicó una arquitectura de software donde se divide el sistema en componentes y/o módulos más pequeños en donde cada uno posee una responsabilidad específica.

- **Patrón de arquitectura utilizado**. Se utilizaron diferentes patrones de arquitectura. Uno de ellos es el conocido **patrón MVC**, el cual se caracteriza por tener separado una capa referida al modelo (M), en donde se representan los datos y la lógica de negocio; una capa referida a la vista (V), la cual refiere a la interfaz de usuario; y, finalmente, un controlador (C) que básicamente se encarga de intermediar entre las peticiones del usuario y el modelo. Otro de los patrones utilizados fue el de **Factory**, el cual tiene como objetivo principal abstraer el proceso de creación de objetos. Se decidió implementar este patrón ya que nos permite delegar la responsabilidad de la creación de objetos a un componente específico (la fábrica), lo cual nos garantiza que la creación de objetos siga un proceso consistente.

FALTARÍA AGREGAR MÁS PATRONES PERO DE MOMENTO NO UTILIZAMOS OTRO (CREO).

- **Estructura general de carpetas**: el código se separa en una estructura de tres carpetas lógicas: la primera de ellas alude a los modelos (carpeta “model”); la segunda refiere a los repositorios (carpeta “repository”); y la restante se basa en la seguridad (carpeta “security”).

- **Seguridad**: se implementaron prácticas de autenticación y autorización de usuarios y validación de datos de entrada, utilizando JSON Web Token (JWT).

- **Docker & Azure**: en primera medida se decidió utilizar Docker ya que es una plataforma de contenerización que permite empaquetar aplicaciones y sus dependencias en entornos aislados llamados contenedores, lo cual proporciona portabilidad, aislamiento y eficiencia al momento de la implementación de la aplicación. Asimismo, se decidió utilizar Microsoft Azure ya que es una plataforma en la nube de Microsoft que ofrece, entre otras cosas, servicios de **base de datos SQL.** La razón por la que se decidió implementar la combinación Docker-Azure es que esta es muy sólida y popular para trabajar en entornos de la nube y aplicaciones contenerizadas.

- **Testing**: para garantizar la calidad, confiabilidad y eficiencia de la aplicación se realizaron diferentes pruebas y depuración del código. Entre los diferentes test realizados, cabe mencionar: **pruebas de integración**. **pruebas unitarias**, y **pruebas de usabilidad**. Las pruebas de integración se centraron en asegurar que todos los componentes se comuniquen de manera efectiva y que los datos fluyan correctamente entre ellos. Las pruebas unitarias fueron realizadas para evaluar individualmente las partes más pequeñas de la aplicación back-office de modo tal de asegurarnos que cada unidad del software funcione según lo previsto. Por último, las pruebas de usabilidad refirieron a comprobar la usabilidad de la interfaz de usuario de manera tal que no haya incoherencias entre el front-end y el back-end de la aplicación.

.

Diagrama entidad relación

Diagrama de tablas Usuario y Rol

La tabla Usuario corresponde a los usuarios que tendrán acceso a la aplicación. Estos tienen uno o más roles con diferentes permisos.
Los roles son “ROLE_ADMIN” y “ROLE_USER”.
ROLE_USER: tiene los permisos necesarios para la atención al cliente, y configuración de su propio usuario.
ROLE_ADMIN: tiene permisos más generales, como poder crear, ver, editar y eliminar usuarios, productos, servicios.

Entre Usuario y Rol se genera una tabla intermedia “Rol_usuario”, ya que un usuario puede tener varios roles, y un tipo de rol puede estar en varios usuarios.

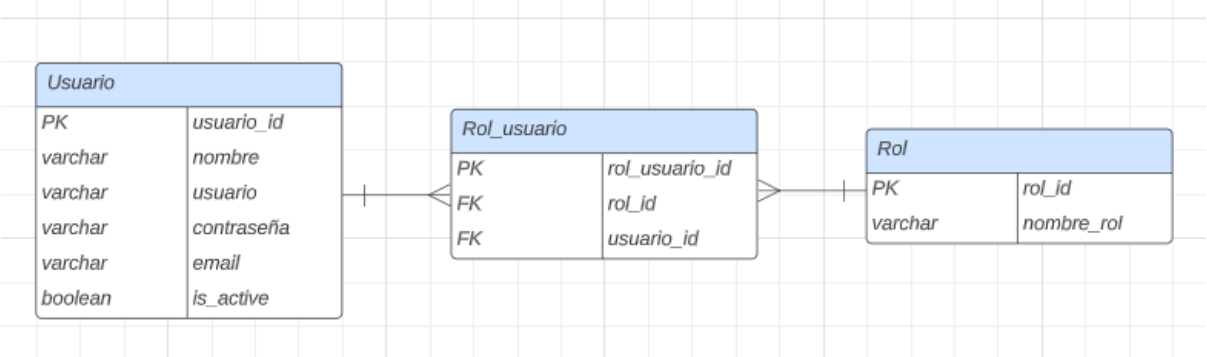


Diagrama de tabla Cliente

Existen dos tipos de clientes, estos pueden ser de tipo Persona o de tipo Empresa, que heredan de la clase principal Cliente.
Mediante una clase factory, se va a crear un objeto Persona o Empresa, dependiendo el parámetro que se pase.

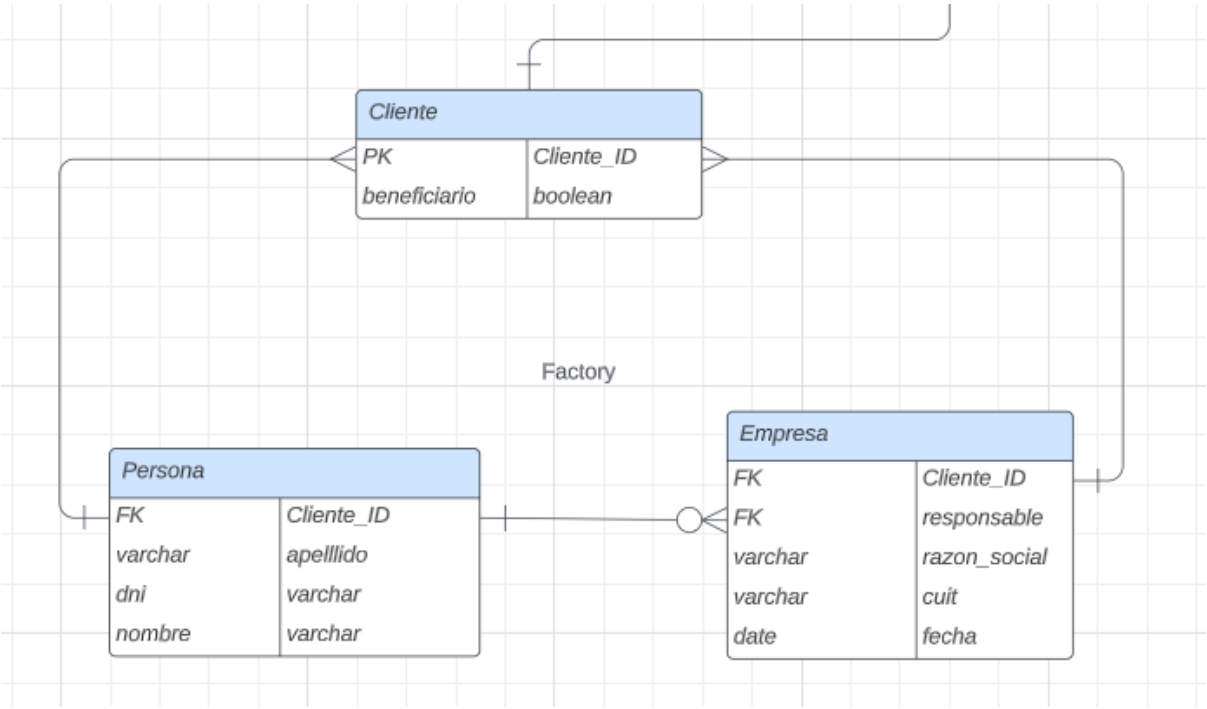
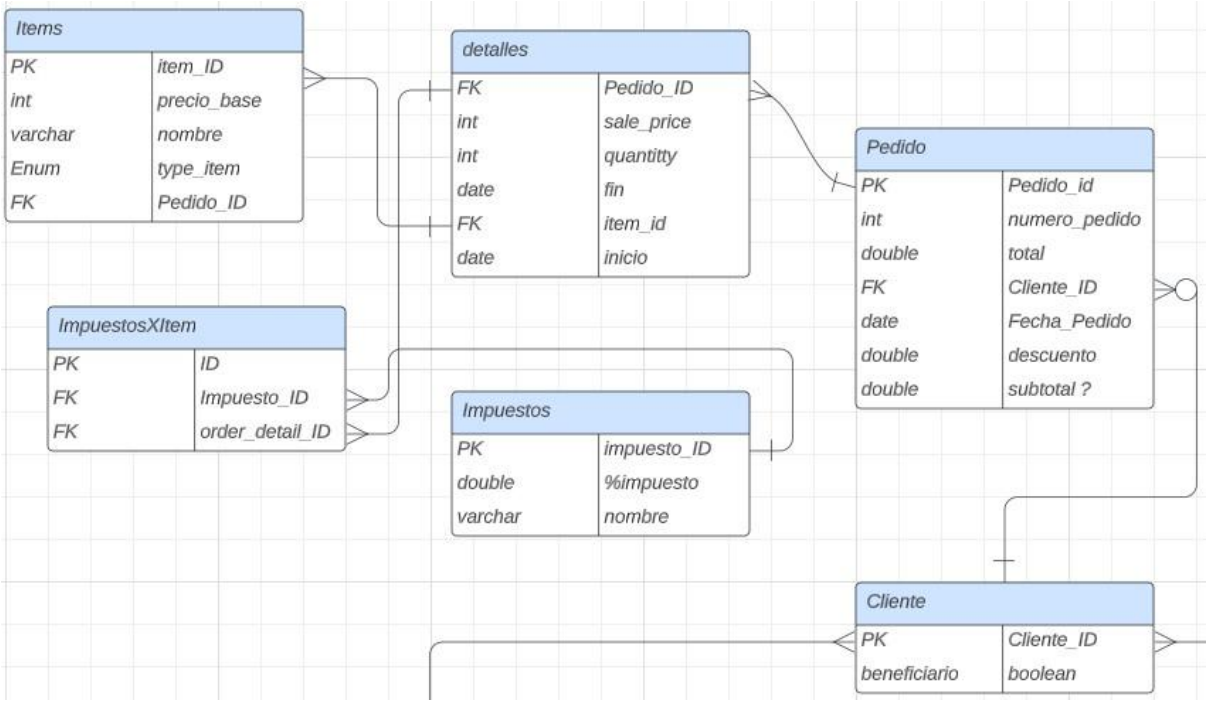


Diagrama general de Cliente, Items y Pedido

Items representa a un producto o un servicio, definido en el atributo type_item.
Detalles es una tabla que se genera al realizar un pedido, en donde se guardan datos como el item que se suma al pedido, los impuestos que se le carga a través de una tabla intermedia ImpuestosXItem, entre la tabla Impuestos y Detalles.

Por cada item que se agregue a un pedido, se genera una objeto Detalles que se asocia con Pedido. Este Pedido puede tener muchos Detalles.
Un Cliente puede tener muchos Pedidos.

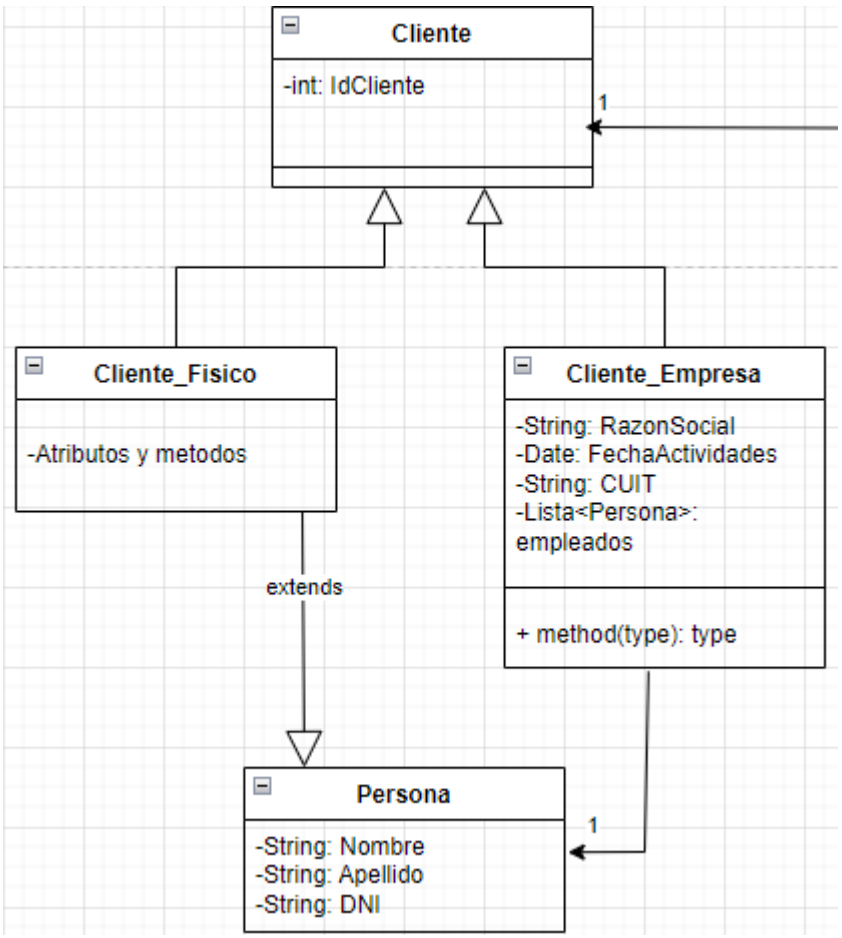


UML de Usuario y Rol

Usuario y Rol tienen una relación muchos a muchos, la tabla Rol_usuario funciona como tabla intermedia.

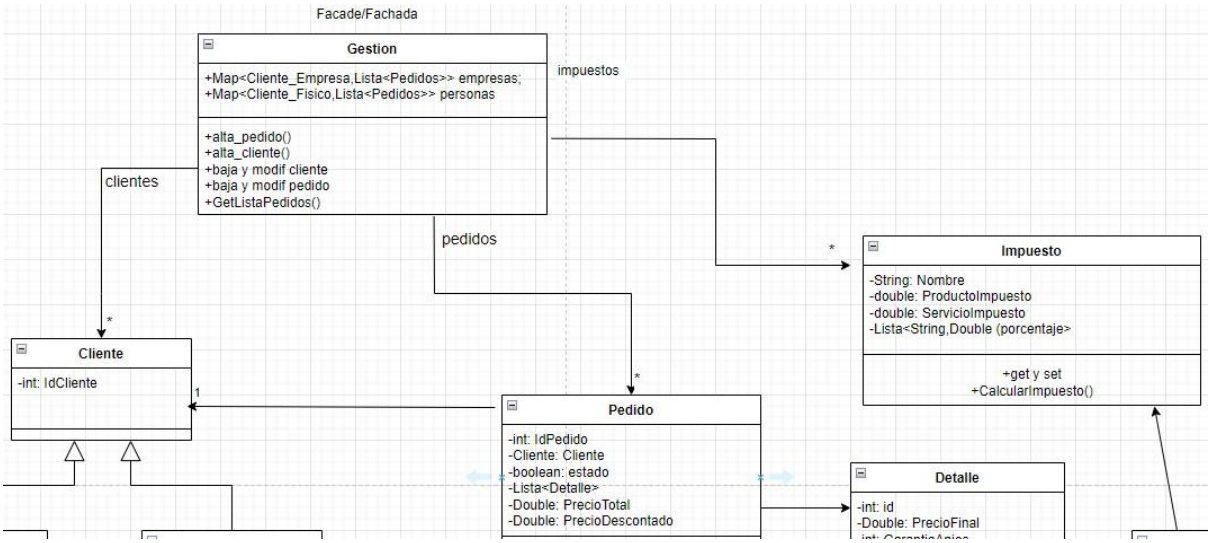
UML de Cliente

Cliente_Fisico y Cliente_Empresa heredan de Cliente.
Persona hereda de Cliente_Fisico, a su vez, esta persona puede cumplir la función de la persona responsable de la adquisición de un producto o servicio de un Cliente_Empresa.



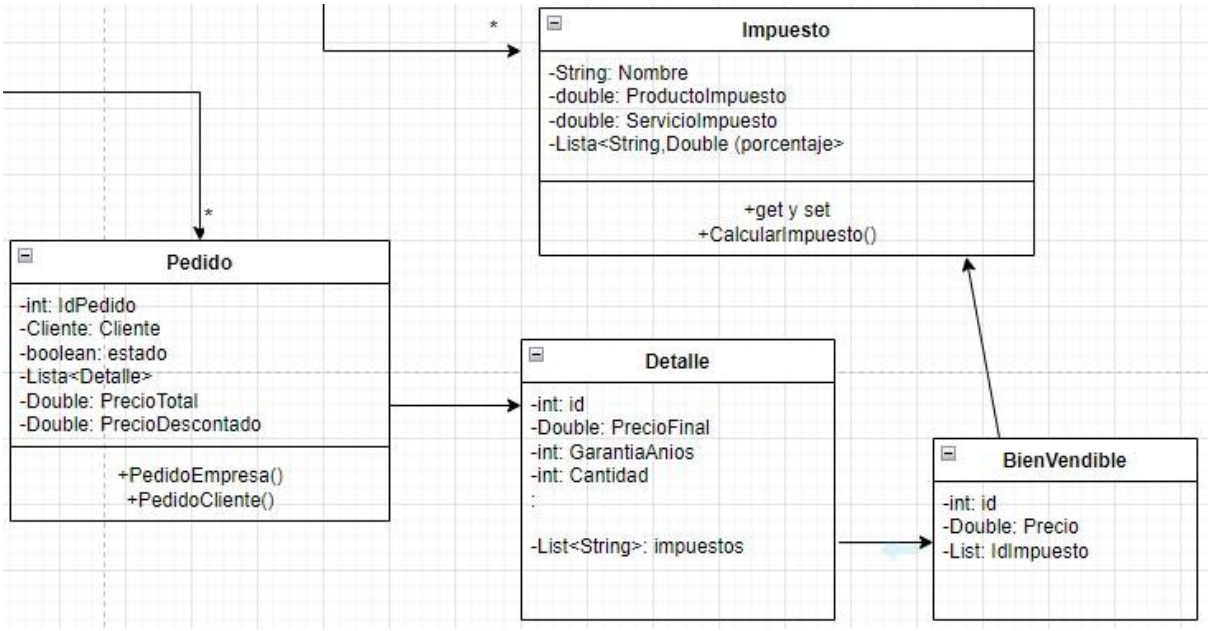
UML de Gestión

La clase Gestión es una fachada que sirve para los pedidos de un cliente y los impuestos de los items.



UML de Pedido, Impuesto, Detalles, Bienes

Conjunto de entidades que participan a la hora de hacer un Pedido. La clase Gestión se encarga de utilizar estas entidades para conformar el pedido del cliente.



UML de Cliente y Pedido

Relación entre Cliente y Pedido. Un Cliente puede tener muchos Pedidos.