UNIVERSIDAD DEL BÍO-BÍO

Facultad de Ciencias Empresariales

Departamento de Ciencias de la Computación y Tecnologías de Información

Ingeniería Civil Informática.

Tarea Ingeniería de software 3

Integrante: Diego Alamos Vallejos

Docente: Gastón Márquez Ortega

Roberto Anabalón Suazo

Asignatura: Ingeniería de Software

Carrera: Ingeniería Civil en Informática

Fecha: 08/09/2025

Introducción sobre evaluación

Se presenta la entrega 3 de la correspondiente evaluación de la asignatura de *Ingenieria de software II .* El objetivo principal fue desarrollar una plataforma web completa que permita administrar un catálogo de muebles, gestionar sus variantes y revisar cotizaciones.

Para lograr esto, diseñe el sistema siguiendo una arquitectura moderna, separando claramente la "lógica" (Backend) de la "parte visual" (Frontend). Además, me enfoqué en que fuera fácil de instalar y ejecutar para cualquier persona, utilizando herramientas de automatización como Docker.

Primero que nada descargar archivos o clonar repositorio a continuación:

REPOSITORIO: https://github.com/Dibox12/AlamosDiego-Secci-n1-Evaluaci-n-3.git

1. Requisitos previos y Dependencias

El desarrollo de este sistema se fundamenta en una arquitectura moderna basada en microservicios dockerizados en containers, separando claramente la lógica de negocio (Backend), la interfaz de usuario (Frontend) y la persistencia de datos (Base de Datos). A continuación, se detallan las tecnologías y dependencias clave seleccionadas para cada capa (Frontend y Backend)

Backend: Spring Boot (Java) Para el servidor use Spring Boot 3.5.6 con Java 17 Elegí esto porque facilita mucho la configuración y se pide en la evaluación

Frontend: Angular (TypeScript) Decidí ocupar este framework por que también lo ocupamos para nuestro proyecto semestral use Angular 17.3.0. Me decidí por este framework porque es muy ordenado para trabajar y permite crear sitios que no necesitan recargar la página todo el tiempo.

@angular/core y @angular/common: Son las partes básicas de Angular. Tienen las herramientas para que funcionen los componentes y cosas como el *ngIf (para ocultar cosas) o *ngFor (para mostrar listas de cosas).

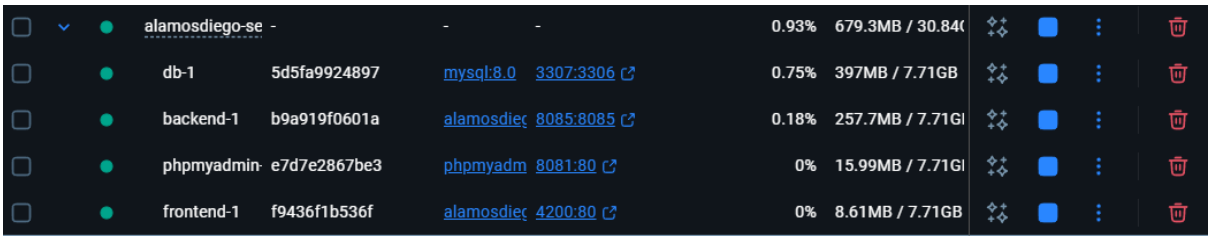
@angular/router: Lo uso para crear la navegación. Gracias a esto, puedo tener direcciones como ”/admin/catalog ”/public para que la página cambie su contenido sin tener que recargar el navegador completo

rxjs: Angular usa esto para manejar las respuestas del servidor. Permite que la página no se quede pegada esperando datos, sino que reaccione cuando la información llega desde el backend

2. Cómo funciona el proyecto

Arquitectura general:

El sistema funciona usando Docker, lo que me permite tener todo encapsulado para que sea fácil de correr en cualquier computador sin instalar tantas cosas. Tiene 4 contenedores:



1. Base de Datos (MySQL): Aquí es donde se guarda toda la información de forma permanente.
2. Backend: Es la API que recibe las órdenes. Procesa los datos y los guarda en la base de datos.
3. Frontend: Es la página web que ven los usuarios. Se conecta al Backend para pedir y mostrar la información.
4. phpMyAdmin: para mantener una gestión de la base de datos de forma más simple

3. Cómo funciona el proyecto

Para desplegar el sistema en un entorno local , se ha automatizado el proceso con Docker Compose. A continuación se describen los pasos necesarios:

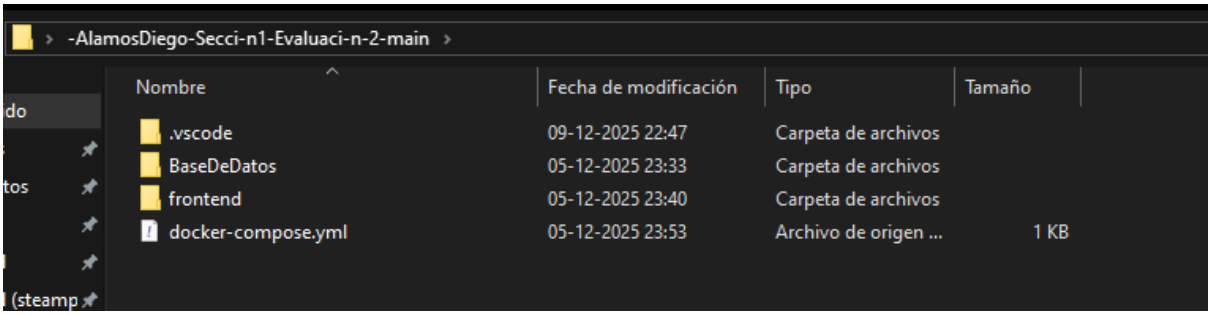
Requisitos Previos

Sistema Operativo: Windows 10/11 o Linux.

Docker Desktop instalado y en ejecución

Lo que se necesita es tener instalado Docker Desktop y que esté abierto.

Pasos 1: Abrir la carpeta: Primero, hay que abrir una terminal y entrar a la carpeta donde está el proyecto.

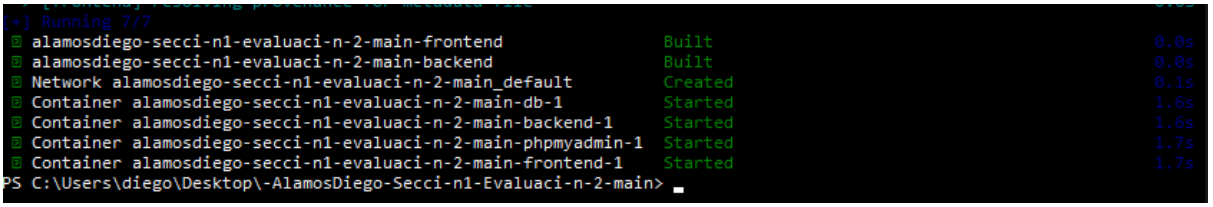


cd carpeta_del_proyecto

```
PS C:\Users\diego\Desktop\-AlamosDiego-Secci-n1-Evaluaci-n-2-main>
```

Paso 2: Solo hay que correr este comando:

docker compose up --build -d



Esto va a descargar todo lo necesario y levantar los servicios automáticamente. La primera vez se demora un poco

Paso 3: Listo: Cuando en la terminal diga que el backend arrancó y MySQL está listo, ya se puede usar.

Con todo corriendo se podrian acceder a los siguientes links o apretando en las direcciones de docker

Frontend (La página web) : http://localhost:4200

Backend (La API): http://localhost:8085

Ver la Base de Datos (PhpMyAdmin): http://localhost:8081

para entrar en phpmyadmin:

Conclusión

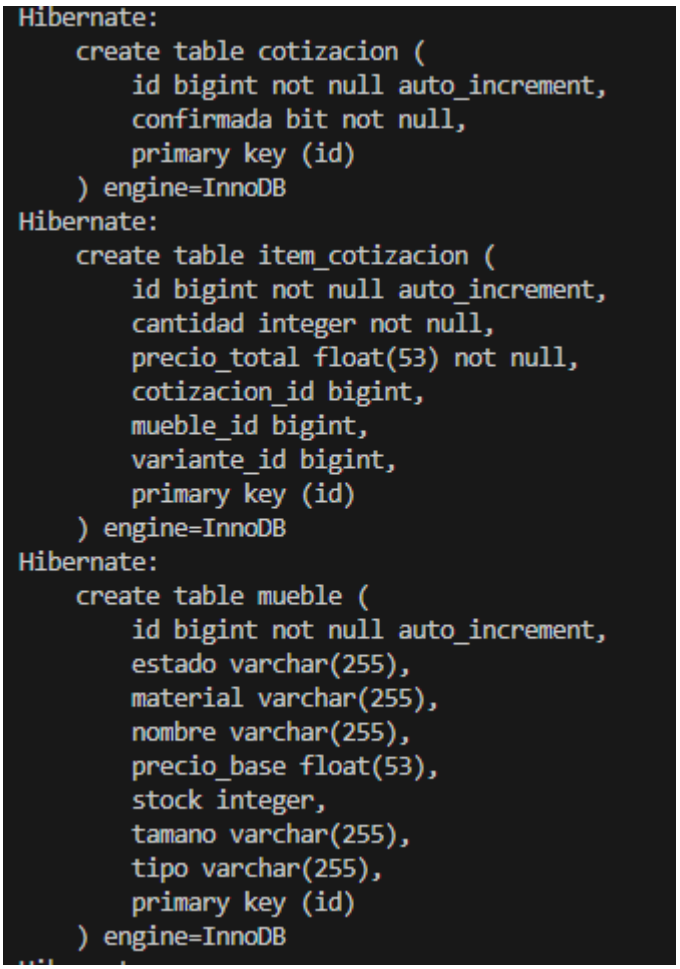
En conclusión, con este trabajo pude integrar varias tecnologías para crear un sistema completo y funcional. Lo más importante fue lograr que Angular y Spring Boot se comunicaran bien entre sí y que la base de datos guardará toda la información correctamente.

También aprendí que usar contenedores con Docker es muy útil. Aunque al principio parece complejo, al final hace que "instalar" el proyecto sea solo correr un comando, lo que evita muchos problemas de configuración en diferentes computadores.

El sistema final cumple con todo lo pedido en la rúbrica: permite ver el catálogo, administrar productos y gestionar cotizaciones, todo documentado y listo para usar.

Bueno pasando a la ejecución del código:

No será necesario CREAR LAS TABLAS ya que al ejecutar el código Main



```
Hibernate:
    create table cotizacion (
        id bigint not null auto_increment,
        confirmada bit not null,
        primary key (id)
    ) engine=InnoDB
Hibernate:
    create table item_cotizacion (
        id bigint not null auto_increment,
        cantidad integer not null,
        precio_total float(53) not null,
        cotizacion_id bigint,
        mueble_id bigint,
        variante_id bigint,
        primary key (id)
    ) engine=InnoDB
Hibernate:
    create table mueble (
        id bigint not null auto_increment,
        estado varchar(255),
        material varchar(255),
        nombre varchar(255),
        precio_base float(53),
        stock integer,
        tamano varchar(255),
        tipo varchar(255),
        primary key (id)
    ) engine=InnoDB
```

etc. (y muchas mas tablas)

hibernate se encargará de crear esta tabla con todos los atributos correspondientes con estos deberíamos verlos en la base de datos de phpMyAdmin

MODELO

Entidades

Mueble :Representa un producto del catálogo. Contiene atributos como nombre, tipo, material, tamaño, precio base, stock y estado.

Variante: Describe modificaciones o complementos del mueble que alteran su precio base (por ejemplo, “Barniz Premium”).

Cotización: Agrupa uno o más ítems que representan una posible venta. Contiene un estado confirmado para indicar si se ha concretado.

ItemCotización: Une un mueble con una variante y cantidad específica dentro de una cotización. Permite calcular subtotales y precios finales.

Controladores

Los controladores exponen la funcionalidad del sistema mediante endpoints REST bajo el prefijo /api/

MuebleControlador: /api/muebles Permite crear, listar, actualizar y eliminar muebles.

VarianteControlador: /api/variantes Gestiona variantes asociadas a muebles.

CotizacionControlador: /api/cotizaciones Crea y confirma cotizaciones, controlando precios y stock.

Servicios

Los servicios contienen la lógica de negocio del sistema y actúan como intermediarios entre los controladores y repositorios es un componente de Spring anotado con @Service y utiliza la inyección de dependencias (@Autowired):

MuebleServicio: Maneja el CRUD del catálogo, actualiza o desactiva muebles.

CotizacionServicio: Gestiona cotizaciones y ventas: crea cotizaciones, calcula precios totales, válida stock y confirma ventas.

Repositorios

Cada entidad cuenta con su propio repositorio que extiende JpaRepository, lo que permite ejecutar operaciones CRUD sin escribir SQL manualmente.

Esto permite Simplificación del acceso a datos e integración directa con Spring Data JPA

Testing

Se puede realizar pruebas mediante el Comando **.\mvnw.cmd test** o tambien dandole al boton que se encuentra al lado de los testing

Caso de prueba	Descripción	Resultado esperado
testCrearYActualizarMueble()	Verificar el correcto funcionamiento del CRUD de muebles	El mueble se crea con ID válido y us nombre actualizado coincide con el valor modificado
testVentaConStockInsuficiente()	Validar el control de stock al intentar confirmar una venta	Se lanza una excepción con el mensaje “Stock insuficiente” y la venta no se realiza
testCalcculoPrecioConVariante()	Probar el cálculo del precio total de una cotización considerando variantes o cantidades	El total acumulado coincide con el esperado, demostrando el correcto funcionamiento del servicio de precios

Salida que se debería conseguir