* Introducción

Estimado equipo de Ingenio,

Es un honor participar en el proceso de selección para el puesto de Desarrollador Java Senior en su destacada empresa. Agradezco la oportunidad de demostrar mis habilidades y conocimientos técnicos a través de esta prueba, y estoy entusiasmado por la posibilidad de contribuir al éxito continuo de Ingenio con mi experiencia en el desarrollo de software Java.

* Entregables esperados
* Diagrama E-R del modelo de datos, deberá diagramarse con alguna herramienta a elección del candidato.
* Script de base de datos de la solución.
* URL del repositorio GIT.
* Artefactos de despliegue de la solución.
* Definición de buenas prácticas.
* Justificación de las tecnologías empleadas y patrones de diseño empleados.
* Enunciado

Una empresa de gestión logística terrestre la cual administra el transporte en camiones, y al ser bastante exitosa ha obtenido en los últimos meses peticiones de empresas de transporte marítimo para que agregue la logística marítima a sus servicios.

Por ello la empresa lo ha contratado a usted para que inicie con el desarrollo de una nueva aplicación con la funcionalidad logística terrestre y logística marítima.

El sistema debe registrar los datos básicos del cliente, los tipos de productos que transporta tanto para la logística terrestre como para la logística marítima y contar con el registro de las bodegas de almacenamiento terrestre y puertos marítimos a su disposición a nivel nacional o internacional.

La logística de camiones cuenta con un plan de entrega, donde se debe obtener los siguientes datos.

1. Tipo de producto
2. Cantidad del producto
3. Fecha de registro
4. Fecha de entrega
5. Bodega de entrega
6. Precio de envió
7. Placa del vehículo (El formato debe corresponder a 3 letras iniciales y 3 números finales)
8. Numero de guía (Numero único alfanumérico de 10 dígitos)

La logística marítima cuenta con el siguiente plan de entrega, donde se debe obtener los siguientes datos.

1. Tipo de producto
2. Cantidad del producto
3. Fecha de registro
4. Fecha de entrega
5. Puerto de entrega
6. Precio de envío
7. Numero de flota (El formato debe corresponder a 3 letras iniciales, seguidas de 4 números y finalizando con una letra)
8. de guía (Numero único alfanumérico de 10 dígitos)

Cada envió debe quedar relacionado a un cliente para poder hacer el seguimiento correcto del servicio.

* **Descuentos:** Si la cantidad de productos es superior a 10 unidades, se deberá otorgar un descuento del 5% al precio de envío para la logística terrestre y un 3% para la logística marítima y dejar registrado el precio normal y el descuento otorgado.

Para cumplir con el requerimiento deberá desarrollar una API RESTful (CRUD) tanto para la logística marítima como para la logística terrestre con su debido proceso, además de poder

consultar los datos con un sistema de filtros de búsqueda.

* **Seguridad:** Implemente la seguridad de los servicios por medio de token del tipo bearer "solo implemente la validación del token".

**Nota:** los campos de API RESTful deben ser validados y en caso de no cumplir con los valores y tipos de datos, este deberá retornar un mensaje de error (401, 422, 400, 500).

* **Persistencia de datos:** Todos los datos deberán ser almacenados en alguno de los siguientes motores de bases de datos (Seleccione el de su preferencia):

1. MySQL
2. SQL Server
3. PostgreSQL
4. MongoDB

**Nota:** deberá crear el diagrama E-R de la solución

* **Pruebas unitarias:** Deberá crear las pruebas necesarias de las funcionalidades implementadas en el desarrollo con el fin de garantizar la calidad del código desarrollado.
* Resumen de la Solución
  1. Tecnologías y Herramientas Utilizadas

|  |  |
| --- | --- |
| Tecnologías | |
| Editor de código | Visual Studio Code |
| Back-end | Java: 17.10, Spring boot |
| Gestor de dependencias | Maven |
| Front-end | HTML5. CSS3, JavaScript, Tailwind, thymeleaf |
| Base de datos | PostgreSQL, Supabase |
| Framework de pruebas | Playwright 1.42.0 |

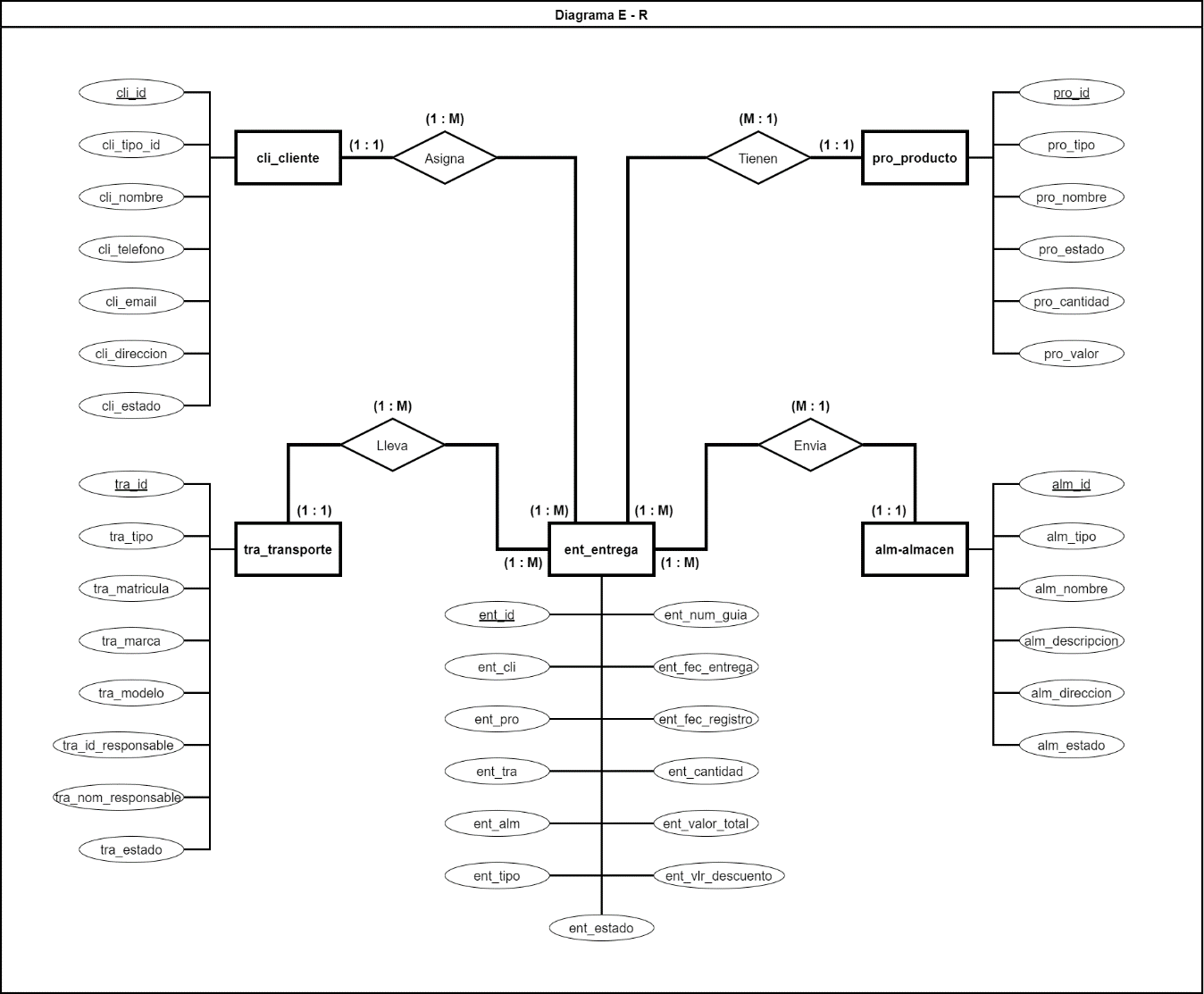
Para abordar el problema planteado, se propone desarrollar una aplicación de gestión logística que incluya tanto la logística terrestre como la logística marítima. Aquí está un resumen de la solución propuesta utilizando las tecnologías especificadas:

* Editor de código: Visual Studio Code: Se utilizará Visual Studio Code como el entorno de desarrollo integrado (IDE) para escribir y depurar el código de la aplicación. Visual Studio Code es una herramienta popular y potente que ofrece una amplia gama de extensiones para mejorar la productividad y la experiencia de desarrollo.
* Back-end: Java 17.10, Spring Boot: se utilizará Java 17.10 para el desarrollo del backend de la aplicación. Spring Boot se utilizará como el framework principal para la creación de la API RESTful (CRUD) para la gestión de la logística terrestre y marítima. Spring Boot facilita la creación de aplicaciones Java de manera rápida y sencilla, proporcionando un conjunto de herramientas y librerías para el desarrollo ágil.
* Gestor de dependencias: Maven: será utilizado como gestor de dependencias para gestionar las dependencias del proyecto y facilitar la construcción y gestión del mismo. Maven es una herramienta ampliamente utilizada en el ecosistema Java para la gestión de proyectos.
* Front-end: HTML5, CSS3, JavaScript, Tailwind, Thymeleaf: para el desarrollo del frontend de la aplicación, se utilizarán HTML5, CSS3 y JavaScript para la estructura, estilo y funcionalidad de la interfaz de usuario. Tailwind CSS se utilizará como framework de diseño para facilitar la creación de interfaces de usuario modernas y receptivas. Thymeleaf se utilizará como motor de plantillas para generar HTML dinámicamente en el servidor, integrándose con Spring Boot para la generación de vistas.
* Base de datos: PostgreSQL: se utilizará como el sistema de gestión de bases de datos relacional para almacenar los datos de la aplicación. PostgreSQL es un sistema de gestión de bases de datos robusto y de código abierto que ofrece características avanzadas de rendimiento, escalabilidad y seguridad.
* Framework de pruebas: Playwright 1.42.0: Se utilizará como framework de pruebas para automatizar las pruebas funcionales y de regresión de la aplicación. Playwright proporciona una API moderna y fácil de usar para interactuar con navegadores web y realizar acciones como clics, escritura de texto y navegación por páginas, lo que facilita la escritura y ejecución de pruebas automatizadas.

1. Entregables finales

|  |  |
| --- | --- |
| Tecnologías | |
| Editor de código | Visual Studio Code |
| Back-end | Java: 17.10, Spring boot |
| Gestor de Dependencias | Maven |
| Front-end | HTML5. CSS3, JavaScript, Tailwind, thymeleaf |
| Base de datos | PostgreSQL, Supabase |
| Framework de pruebas | Playwright 1.42.0 |
| GitHub URL | https://github.com/ztevenx100/Ingeneo-PruebaTecnica |

* 1. Diagrama entidad - relación



* 1. Scripts de base de datos

-- Creacion de tablas

create table public.cli\_cliente (

cli\_id text not null,

cli\_email text not null,

cli\_nombre text not null default ''::text,

cli\_telefono text null,

cli\_tipo\_id text not null default 'C'::text,

cli\_direccion text null,

cli\_estado text not null default 'A'::text,

constraint cli\_cliente\_pkey primary key (cli\_id)

) tablespace pg\_default;

create table public.pro\_producto (

pro\_id text not null,

pro\_tipo text not null default 'T'::text,

pro\_nombre text not null,

pro\_cantidad numeric not null default '0'::numeric,

pro\_valor\_unitario numeric not null default '0'::numeric,

pro\_estado text not null default 'A'::text,

constraint pro\_producto\_pkey primary key (pro\_id)

) tablespace pg\_default;

create table public.alm\_almacen (

alm\_id text not null,

alm\_tipo text not null default 'T'::text,

alm\_nombre text not null,

alm\_descripcion text null,

alm\_direccion text not null,

alm\_estado text not null default 'A'::text,

constraint alm\_almacen\_pkey primary key (alm\_id)

) tablespace pg\_default;

create table public.tra\_transporte (

tra\_id text not null,

tra\_tipo text not null default 'T'::text,

tra\_matricula text not null,

tra\_marca text not null,

tra\_modelo text not null,

tra\_id\_responsable text not null,

tra\_nom\_responsable text null,

tra\_estado text not null,

constraint tra\_transporte\_pkey primary key (tra\_id)

constraint tra\_transporte\_tra\_matricula\_key unique (tra\_matricula)

) tablespace pg\_default;

create table public.ent\_entrega (

ent\_id text not null,

ent\_cli text not null,

ent\_pro text not null,

ent\_alm text not null,

ent\_tra text not null,

ent\_num\_guia text not null,

ent\_fec\_registro timestamp with time zone not null,

ent\_fec\_entrega timestamp with time zone null,

ent\_estado text not null default 'A'::text,

ent\_tipo text not null default 'T'::text,

ent\_cantidad numeric not null default '0'::numeric,

ent\_valor\_total numeric not null default '0'::numeric,

ent\_valor\_descuento numeric null default '0'::numeric,

constraint ent\_entrega\_pkey primary key (ent\_id),

constraint ent\_entrega\_ent\_num\_guia\_key unique (ent\_num\_guia),

constraint public\_ent\_entrega\_ent\_alm\_fkey foreign key (ent\_alm) references alm\_almacen (alm\_id),

constraint public\_ent\_entrega\_ent\_cli\_fkey foreign key (ent\_cli) references cli\_cliente (cli\_id),

constraint public\_ent\_entrega\_ent\_pro\_fkey foreign key (ent\_pro) references pro\_producto (pro\_id),

constraint public\_ent\_entrega\_ent\_tra\_fkey foreign key (ent\_tra) references tra\_transporte (tra\_id)

) tablespace pg\_default;

-- Index

CREATE UNIQUE INDEX cli\_cliente\_pkey ON public.cli\_cliente USING btree (cli\_id);

CREATE UNIQUE INDEX pro\_producto\_pkey ON public.pro\_producto USING btree (pro\_id);

CREATE UNIQUE INDEX alm\_almacen\_pkey ON public.alm\_almacen USING btree (alm\_id);

CREATE UNIQUE INDEX tra\_transporte\_pkey ON public.tra\_transporte USING btree (tra\_id);

CREATE UNIQUE INDEX tra\_transporte\_tra\_matricula\_key ON public.tra\_transporte USING btree (tra\_matricula);

CREATE UNIQUE INDEX ent\_entrega\_pkey ON public.ent\_entrega USING btree (ent\_id);

CREATE UNIQUE INDEX ent\_entrega\_ent\_num\_guia\_key ON public.ent\_entrega USING btree (ent\_num\_guia);

* 1. Definición de buenas prácticas

Para definir buenas prácticas en el desarrollo de la aplicación de gestión logística, puedes seguir estas pautas:

1. **Convención de nombres**: Utiliza nombres descriptivos y significativos para las clases, métodos y variables. Sigue una convención de nombres consistente, como camelCase para variables y métodos, y UpperCamelCase para clases.
2. **Organización del código**: Estructura tu código de manera lógica y coherente. Utiliza paquetes para organizar y agrupar clases relacionadas. Divide el código en módulos y capas claramente definidas, como controladores, servicios, modelos y repositorios.
3. **Documentación del código**: Documenta tu código de manera clara y concisa. Incluye comentarios descriptivos en el código para explicar su funcionalidad, propósito y cualquier consideración importante.
4. **Principios de diseño SOLID**: Aplica los principios de diseño SOLID (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation y Dependency Inversion) en el diseño y la implementación de tu aplicación. Diseña clases y componentes que sean cohesivos, con responsabilidades únicas y que puedan ser fácilmente extendidos y modificados.
5. **Seguridad**: Implementa medidas de seguridad adecuadas para proteger los datos sensibles y prevenir vulnerabilidades de seguridad. Utiliza técnicas como la validación de entrada, la autenticación y la autorización para garantizar la seguridad de la aplicación.
6. **Pruebas unitarias**: Escribe pruebas unitarias para verificar el comportamiento individual de los componentes de tu aplicación. Utiliza marcos de pruebas como JUnit para escribir y ejecutar pruebas de manera automatizada.
7. **Control de versiones**: Utiliza un sistema de control de versiones como Git para gestionar el código fuente de tu aplicación. Organiza el repositorio de manera estructurada y sigue prácticas de ramificación y fusión para colaborar de manera efectiva con otros desarrolladores.
8. **Revisión de código**: Realiza revisiones de código periódicas para garantizar la calidad y consistencia del código. Solicita a otros miembros del equipo que revisen tu código y proporcionen comentarios constructivos para mejorar la calidad del mismo.
9. **Optimización del rendimiento**: Optimiza el rendimiento de la aplicación identificando y resolviendo cuellos de botella y puntos críticos de rendimiento. Realiza pruebas de rendimiento y perfilado de código para identificar áreas de mejora y optimización.
10. **Mantenibilidad y escalabilidad**: Diseña la aplicación con la mantenibilidad y escalabilidad en mente. Utiliza patrones de diseño y arquitecturas que faciliten la evolución y el crecimiento de la aplicación a medida que los requisitos y la complejidad cambien con el tiempo.
    1. Artefactos de despliegue de la solución

Artefactos:

* Empaquetado del proyecto: target/empresa\_gestion\_logistica-0.0.1-SNAPSHOT.jar
* Docker: Dockerfile
* Base de datos: supabase
  1. Justificación
     1. Tecnologías

Editor de código: Visual Studio Code

* Visual Studio Code es un editor de código ligero pero potente que ofrece una amplia gama de extensiones para mejorar la productividad y la experiencia de desarrollo. Su integración con Git, depurador incorporado, resaltado de sintaxis avanzado y soporte para múltiples lenguajes de programación lo hacen ideal para proyectos de desarrollo de software.

Backend: Java

* Java es un lenguaje de programación robusto y maduro que es ampliamente utilizado en el desarrollo de aplicaciones empresariales y sistemas backend. Su amplia comunidad de desarrolladores, la gran cantidad de bibliotecas y frameworks disponibles, y su capacidad multiplataforma hacen que sea una elección sólida para construir la lógica del servidor en tu proyecto.

Spring Boot

* Spring Boot es un framework de desarrollo de aplicaciones Java que simplifica el proceso de creación de aplicaciones empresariales mediante el uso de convenciones sobre configuración. Proporciona un conjunto de herramientas y librerías que facilitan la creación de aplicaciones web escalables y de alto rendimiento. Al aprovechar la arquitectura de Spring y sus módulos, Spring Boot simplifica tareas comunes como la configuración, la gestión de dependencias y la creación de puntos finales RESTful. Además, su integración con Spring Data JPA facilita el acceso y la manipulación de datos en la base de datos, mientras que Spring Security proporciona características de autenticación y autorización robustas para proteger tu aplicación. En resumen, Spring Boot es una excelente opción para el desarrollo del backend en tu proyecto, ya que te permite crear aplicaciones Java robustas y fáciles de mantener en poco tiempo.

Frontend: HTML5, CSS3, JavaScript, Tailwind y Thymeleaf

* Estas tecnologías son fundamentales para el desarrollo de la interfaz de usuario de tu aplicación web. HTML5 proporciona la estructura básica de la página web, CSS3 permite el diseño y la estilización, y JavaScript permite la interactividad del usuario y la manipulación del DOM. Tailwind CSS es un framework de CSS utility-first que facilita la creación de interfaces de usuario modernas y receptivas. Thymeleaf es un motor de plantillas para Java que te permite generar HTML dinámicamente en el servidor. En conjunto, estas tecnologías proporcionan las herramientas necesarias para crear una experiencia de usuario atractiva y funcional.

API RESTful: Supabase

* es una plataforma completa y potente que ofrece una solución integral para el desarrollo de aplicaciones modernas. Su facilidad de uso, base de datos PostgreSQL gestionada, API RESTful automatizada, capacidades de autenticación y autorización integradas, eventos en tiempo real, escalabilidad y flexibilidad lo convierten en una opción atractiva para proyectos de desarrollo de software.

Base de datos: PostgreSQL

* PostgreSQL es un sistema de gestión de bases de datos relacional de código abierto ampliamente utilizado en proyectos empresariales y de código abierto. Ofrece una sólida combinación de características, incluida la capacidad de manejar grandes volúmenes de datos, la compatibilidad con transacciones, la extensibilidad mediante la creación de funciones y tipos de datos personalizados, y el soporte para lenguajes de programación adicionales a través de extensiones. Además, es conocido por su fiabilidad, rendimiento y seguridad.

Framework de pruebas: Playwright

* Playwright es un poderoso framework de automatización de pruebas que te permite realizar pruebas funcionales y de regresión en aplicaciones web. Su capacidad para automatizar múltiples navegadores, incluidos Chrome, Firefox y Safari, lo hace ideal para proyectos que requieren pruebas en diferentes entornos de navegador. Además, su API fácil de usar y su capacidad para simular interacciones de usuario complejas, como clics, escritura de texto y navegación por páginas, lo convierten en una excelente opción para garantizar la calidad y la estabilidad de tu aplicación web.
  + 1. Patrones de diseño

Patrón Factory (Factory Pattern)

* El patrón Factory es útil cuando creas objetos de un tipo específico sin exponer la lógica de creación directamente en tu código. Proporciona un método centralizado para la creación de objetos, lo que facilita la gestión de la creación de objetos en un solo lugar y permite la flexibilidad para cambiar la implementación de creación sin afectar al resto del código. Esto es especialmente útil cuando necesitas instanciar diferentes subclases de un objeto en función de ciertas condiciones o configuraciones. El patrón Factory promueve el principio de diseño **SOLID** de la responsabilidad única al separar la lógica de creación de la lógica de negocio principal, lo que facilita la mantenibilidad y extensibilidad del código.

Patrón Strategy (Strategy Pattern)

* El patrón Strategy es útil cuando necesitas definir un conjunto de algoritmos intercambiables y encapsularlos en objetos separados. Permite que los algoritmos puedan variar independientemente de los clientes que los utilizan, lo que promueve la flexibilidad y la reutilización del código. Al utilizar el patrón Strategy, puedes definir múltiples estrategias para resolver un problema particular y seleccionar la estrategia adecuada en tiempo de ejecución según las necesidades del cliente. Esto facilita la gestión de la complejidad al dividir el código en componentes más pequeños y fácilmente mantenibles. Además, el patrón Strategy promueve el principio de diseño **SOLID** de la inversión de dependencia al permitir que las estrategias sean inyectadas en los objetos clientes, en lugar de depender directamente de implementaciones concretas, lo que hace que el código sea más flexible y fácil de probar.

1. Agradecimiento

Agradezco la oportunidad brindada y estoy comprometido a proporcionar una solución que refleje no solo mis habilidades técnicas, sino también mi capacidad para integrarme exitosamente en el equipo de Ingenio.

Espero con entusiasmo la evaluación de mi trabajo y estoy disponible para cualquier aclaración o consulta adicional.

Atentamente,

Pedro Steven Castiblanco