Hiring

Objetivo

El objetivo de la siguiente prueba consiste en la construcción de un microservicio encargado de la generación del cálculo de rutas para el envío de mercancía de forma optimizada. Para ello, se provee de una estructura base de microservicio siguiendo **arquitectura hexagonal** que nos debe permitir:

- Ingestar información de envíos pendientes.
- Extraer información de envíos.
- Orquestar el cálculo de rutas optimas para los diferentes envíos.
- Purgar los envíos no pendientes.

Retos

- Se exigirá el uso de Git como herramienta de control de cambios. Cada tarea (anotada con #1,2,3..) deberá ser resuelta en una entrada de control de versiones independiente (commit) con el formato *"[Tarea #?]: <comentario>"*
- Se provee de una API REST parcialmente codificada usando OpenAPI v3 para la definición de los mecanismos de entrada y salida de datos. El desarrollador deberá completarla en base a las especificaciones acordadas y establecer los mecanismos de transformación entre modelos que fuesen necesarios.
- Se provee un Adapter de comunicación con una base de datos relacional en memoria (h2) y otra parte in-memory. El desarrollador deberá completar el modelo de bbdd haciendo los cambios necesarios.
- El algoritmo de cálculo de rutas se basa en el algoritmo de Dijkstra: [Dijkstra - Introducción gráfica]. El algoritmo se provee **COMPLETAMENTE** implementado pero **[Opcional]**, el desarrollador deberá introducir modificaciones y aplicar optimizaciones si las hubiera.
- Se solicitará la creación de test unitarios para la validación de código.
- Se plantearán desafíos para demostrar el dominio de SQL, JPA y Spring
- **[Opcional]** Se plantearán desafíos para demostrar el dominio del manejo de eventos
- **[Opcional]** Se planteará un desafío para habilitar la consulta de información en multiples formatos: json,xml, csv.
- Los retos opcionales pueden ser resueltos en cualquier orden.

> NO OLVIDE resolver cada tarea en un commit independiente.

Arquitectura

- El proyecto se ha compilado usando OpenJdk 17.0.8 y Maven 3.8.4
- Está basado en SpringBoot 3.2.4 usando una arquitectura hexagonal simplificada a 3 capas, dominio, aplicación e infraestructuras. Adicionalmente el proyecto contiene el módulo boot de aplicación, el módulo de definición de api (api-definition) y un módulo de jacoco para la obtención de cobertura.
- En el proyecto se emplean herramientas de soporte para la generación de código repetitivo: Project Lombok y MapStruct.
- El framework de testing está basado en Junit5, Mockito 5.7, Hamcrest 2.2 e Instancio. No obstante, el desarrollador es libre de cambiar las herramientas que considere oportunas.

Entregables

Código fuente

- Enlace a un repositorio Git con la solución al proyecto propuesto. Preferiblemente GitHub.

Respuestas teóricas

- Añada al final de este documento las respuestas que deba dejar por escrito o todas aquellas anotaciones que considere interesantes e inclúyalo en el repositorio anterior.

Tareas

Ingesta de información de envíos pendientes (Obligatorio) [#1]

- El servicio de ingesta de envíos debe recibir un DTO parcialmente definido por el esquema **schema:/component/ShipmentInput** en el fichero *itx-iop_tech-transport-hiring-openapi.yaml* en formato Json. Completa la especificación para poder cargar los envíos en base de datos. Presta atención a las transformaciones que fuesen necesarias
- El servicio debe atender a peticiones de tipo POST sin autenticación a /shipment
- El caso de uso asociado a este flujo es: LoadShipmentUseCase
- Para facilitar el escenario **use** el operationID : **loadShipment**
- El nombre de los objetos del esquema DEBEN mantenerse según la especificación.
- El formato de ShipmentInput DEBE ser el siguiente:

Campo	Tipo	Restricciones	Obligatorio
shipmentId	string	formato UUID	Sí
originCityCode	string	Tamaño fijo de 3 caracteres	Sí
destinationCityCode	string	Tamaño fijo de 3 caracteres	Sí
departureDate	string	Date/Time	No
expectedArrivalDate	string	Date/Time	No

- Añada un test unitario que valide la lógica implementada en el caso de uso.
- Añada un test unitario que valide el controlador MVC.
- NO olvide revisar los **TODO #1** en el código fuente.
- Compruebe que puede ejecutar la aplicación y cargar envíos en base de datos.

Extraer información de envíos (Obligatorio) [#2]

- El servicio de consulta permitirá obtener la información compuesta por la información del envío y su cálculo de ruta.
- Se ofrecerán dos servicios de consulta:
  - Complete el endpoint para la consulta de envíos por identificador: GET /shipment/*_identificadorUUID_*
  - Añada un endpoint para la Consulta de envíos: GET /shipments
  - Los casos de uso asociados son: FindShipmentUseCase y FindShipmentsUseCase
- El servicio se entrega con una implementación de respuesta parcialmente codificada.
- Se deberá completar el esquema schema:/component/Shipment en el fichero *itx-iop_tech-transport-hiring-openapi.yaml*. El formato debe Shipment debe ser:

Campo	Tipo	Restricciones	Obligatorio
shipmentId	string	formato UUID	Sí
originCity	components/schemas/City		Sí
destinationCity	components/schemas/City		Sí

Campo	Tipo	Restricciones	Obligatorio
departureDate	string	Date/Time	No
expectedArrivalDate	string	Date/Time	No
status	enum	Valores posibles: PENDING, DISCARDED, PLANNED	Sí
routePlan	Array of components/schemas/Route		No

- Se deberá proveer de la lógica necesaria para completar los distintos campos que componen la respuesta:

1. Información del envío cargada desde base de datos
2. Ciudades de origin y destino obtenidos del adapter de ciudades (inicialmente in-memory)
3. Ruta planificada y coste mínimo (si estuviese calculado)

Los siguientos ejemplos muestran una respuesta válida de consulta de envíos **ANTES** y **DESPUÉS** de procesar mediante una llamada a POST /shipments/process:


```
{
    "shipmentId": "13b96d13-40d9-4ef9-9fe1-1b92a309d92f",
    "originCity": {
        "code": "BER",
        "name": "Berlin (JSON)",
        "handlingCost": 5.0
    },
    "destinationCity": {
        "code": "REY",
        "name": "Reykjavik (JSON)",
        "handlingCost": 2.0
    },
    "departureDate": "2024-02-02T03:00:00Z",
    "expectedArrivalDate": "2024-02-03T11:00:00Z",
    "status": "PENDING",
    "routePlan": null
}
```

```
{
    "shipmentId": "13b96d13-40d9-4ef9-9fe1-1b92a309d92f",
    "originCity": {
        "code": "BER",
        "name": "Berlin (JSON)",
        "handlingCost": 5.0
    },
    "destinationCity": {
        "code": "REY",
        "name": "Reykjavik (JSON)",
        "handlingCost": 2.0
    },
    "departureDate": "2024-02-02T03:00:00Z",
    "expectedArrivalDate": "2024-02-03T11:00:00Z",
    "status": "DISCARDED",
```

```
    "routePlan": [
        {
            "id": "cee586a1-e73b-404c-9c09-b2e0727e7879",
            "origin": {
                "code": "PAR",
                "name": "París (JSON)",
                "handlingCost": 5.0
            },
            "destination": {
                "code": "BER",
                "name": "Berlin (JSON)",
                "handlingCost": 5.0
            },
            "transportType": "AIR",
            "cost": "PT1H",
            "handlingCost": null
        },
        {
            "id": "5401e337-16ce-4733-a05f-72e89a0567a7",
            "origin": {
                "code": "LON",
                "name": "Londres (JSON)",
                "handlingCost": 6.0
            },
            "destination": {
                "code": "PAR",
                "name": "París (JSON)",
                "handlingCost": 5.0
            },
            "transportType": "AIR",
            "cost": "PT1H",
            "handlingCost": null
        },
        {
            "id": "27566132-bd16-4841-8145-99455d65ad7f",
            "origin": {
                "code": "LON",
                "name": "Londres (JSON)",
                "handlingCost": 6.0
            },
            "destination": {
                "code": "REY",
                "name": "Reykjavik (JSON)",
                "handlingCost": 2.0
            },
            "transportType": "SEA",
            "cost": "PT40H",
            "handlingCost": null
        }
    ]
}
```


Complete la implementación de la respuesta e incluiya la información necesaria. Tenga en cuenta que:

- Dispone de un Adapter para la consulta de envíos en base de datos.
- Dispone de un Adapter para la consulta de rutas en memoria.

- Dispone de dos Adapters para la consulta de ciudades tanto en base de datos como en memoria. [En este punto estará habilitado únicamente el adapter en memoria].

- Añada un test unitario que valide la lógica implementada del caso de uso.

- NO olvide revisar los **TODO #2** en el código fuente.

- Compruebe que puede ejecuar la aplicación y las consultas de envío devuelven el formato esperado.

Optimización (Opcional) [#3]

- Si implementase la consulta de envíos usando JPA y se encontrase con este escenario:

```
Hibernate: select
se1_0.shipment_id,se1_0.departure_date,se1_0.city_destination_fk,se1_0.expected_arrival_
date,se1_0.city_origin_fk,se1_0.status from public.shipment se1_0
Hibernate: select ce1_0.city_code,ce1_0.HANDLING_COST,ce1_0.name from public.city ce1_0
where ce1_0.city_code=?
Hibernate: select ce1_0.city_code,ce1_0.HANDLING_COST,ce1_0.name from public.city ce1_0
where ce1_0.city_code=?
Hibernate: select ce1_0.city_code,ce1_0.HANDLING_COST,ce1_0.name from public.city ce1_0
where ce1_0.city_code=?
Hibernate: select ce1_0.city_code,ce1_0.HANDLING_COST,ce1_0.name from public.city ce1_0
where ce1_0.city_code=?
Hibernate: select ce1_0.city_code,ce1_0.HANDLING_COST,ce1_0.name from public.city ce1_0
where ce1_0.city_code=?
Hibernate: select ce1_0.city_code,ce1_0.HANDLING_COST,ce1_0.name from public.city ce1_0
where ce1_0.city_code=?
```

¿Sabrías indicar a qué se debe? ¿Cómo podría resolverse? ¿Podría implementar una alternativa haciendo uso de ShipmentH2Repository.findShipmentDetails?

Purgar envíos procesados (Obligatorio) [#4]

La interfaz de purgado se ofrece parcialmente completa. Es necesario implementar la lógica del adaptador de base de datos correspondiente. Puede emplear SQL, HQL o la API de SpringData para la eliminación de los registros. (Estado distinto de PENDING)

- NO olvide revisar los **TODO #4** en el código fuente.
- Compruebe que puede ejecutar la aplicación y que se eliminan SÓLO los envíos no pendientes. Puede forzar el procesamiento de envíos con una solicitud POST a /shipments/process

Migrar el repositorio de Ciudades a base de datos (Obligatorio) [#5]

El proyecto viene configurado con un adaptador de ciudades en memoria por defecto (CityRepositoryFileAdapter). Complete la implementación del adaptador de ciudades en base de datos (CityRepositoryH2Adapter). Si surgen conflictos con la presencia de ambos beans, establezca una estragia para priorizar el de base de datos.

> Se espera que ambos adaptadores sean cargados

- Realice los cambios en las entidades para que estas puedan ser consultadas / persistidas en base de datos (CityEntity).
- La base de datos H2 embebida ya se encuentra precargada con datos de ciudades. Tendrá que mapear los campos de la entidad con los campos de la tabla.

- Guarde la relación entre los envíos y las ciudades usando los campos **origin** y **destination**.

El modelo E/R de la tabla ciudad responde a la siguiente tabla:

Campo	Tipo	Campo Java	Relación	Clave foránea	CampoJavaClaveForánea
CITY_CODE	String	code	1-N SHIPMENT	CITY_ORIGIN_FK, CITY_DESTINATION_FK	origin, destination
NAME	String	name	-	-	-
HANDLING_COST	BigDecimal	handlingCost	-	-	-

Renombre el fichero *_data.sql* a *data.sql* para asegurar que los datos por defecto son cargados una vez habilita el adaptador de bbdd.

- Añada un test de integración que verifique que consultar ciudades en base de datos.
- Compruebe que al realizar consultas de envíos en la aplicación, la información ahora proviene de base de datos.

Procesar el cálculo de rutas (Opcional) [#6]

Para la implementación del algoritmo de cálculo, se hará uso de una versión modificada del algoritmo de caminos mínimos (dijkstra). El algoritmo en su ideal original consiste en explorar el camino más corto entre un nodo (ciudad origen) y el resto de nodos del grafo, a efecto de poder determinar el coste del camino óptimo.

Dado que la vida real es un poco más complicada, en nuestro problema existirán distintos grafos de transporte (vías de comunicación entre ciudades) en función del tipo de transporte elegido y, además, el camino más corto podrá componerse por la combinación de tramos de los distintos grafos:

- Existirá un grafo de nodos según el tipo de transporte, donde el coste de conexión en tiempo entre dos nodos puede variar de un transporte a otro.
- Un grafo no tiene porque tener todas las rutas. Por ejemplo, las rutas maritimas no unen ciudades de interior.

> Puede comprobar distintas configuraciones modificando la propiedad 'graph.folder' en el fichero application.yml. En cada carpeta encontrará desde configuraciones sencillas de un sólo tipo de transporte a configuraciones más complejas.

Partiendo del modelo original **(YA FUNCIONAL)** sobre un único tipo de transporte **(TRUCK)**, se solicita la implementación de la lógica necesaria para poder calcular la ruta óptima entre dos ciudades en función de los costes de cada tipo de transporte. Para la implementación del algoritmo, el desarrollador podrá decidir la estrategia que considere apropiada. No existe una única solución y se valorará cualquier iniciativa que ayude a una optimización de cualquier ámbito en la resolución del problema: paralelización, simplificación de grafos, aplicación de IA, etc. **SE PERMITE** el uso de herramientas de ayuda a la generación de código. En dicho caso, se valorará la aportación de los PROMPTS empleados como soporte.

- Tenga en cuenta que el test **ProcessPendingShipmentsUseCaseImplTest** ya implementado asegurará que el nuevo algoritmo sigue siendo efectivo cuando se aplica en las condiciones iniciales: Un único tipo de transporte.
- Añada nuevos test unitarios que validen la variación del algoritmo.
- Compruebe que puede ejecutar la aplicación y que el nuevo procesamiento procesa distintos tipos de transporte en función de la configuración de grafos cargados en el *application.yml*

Reto Algoritmia (Opcional) [#7]

Modifique ligeramente el algoritmo implementado previamente para contemplar un coste adicional si en el trayecto se realiza un cambio de tipo de transporte.

El coste del cambio de transporte es una propiedad de cada Ciudad (handlingCost) y atiende a la logistica necesaria para trasladar la mercancía entre terminales (Puerto, aeropuerto, estación de tren). El cálculo del coste se mide en horas y se suma al coste del trayecto.

Tenga en cuenta que si el tipo de transporte no varía, el coste es 0. ¿Varía el resultado de los cálculos? ¿Cómo lo ha comprobado?

Reto Marshalling (Opcional) [#8]

La API del servicio expone que la consulta de datos puede hacerse en formato JSON y la implementación actual solo permite la consulta en dicho formato. ¿Puede adaptar el servicio para soportar el uso de XML? Las peticiones deben usar el header Accept: application/xml Puede añadir dependencias si es necesario.

Para añadir también soporte para CSV se provee un conversor custom para CSV (MarshallingCSVConverter). ¿Sabría configurar el sistema para incluir también dicha opción en las consultas?

Reto Event sourcing (Opcional) [#9]

¿Sabría aplicar el uso de Spring Events al proyecto en cuestión? ¿Qué eventos serían interesantes de capturar / enviar? ¿Qué principios SOLID se aplicarían en la implementación de eventos y porqué? Si sólo tuviese que implementar eventos para el escenario de procesamiento de rutas, ¿Cómo lo haría? (teórico)

Respuesta téoricas

> TBC