UNIVERSIDAD DE GRANADA

Práctica 2: Memoria BPEL

Desarrollo de Sistemas de Software basados en Componentes y Servicios
curso:2024/2025

Adriano García - Giralda Milena
DNI:77944452-M adrianoggm@correo.ugr.es

Universidad de Granada
España

15 de noviembre de 2024

Índice

1. Ejercicio 1

1.1. Estructura del compuesto SOA

Para la realización de este ejercicio se ha planteado una lógica de negocio enfocada en el diagrama de actividades que encabeza esta práctica.

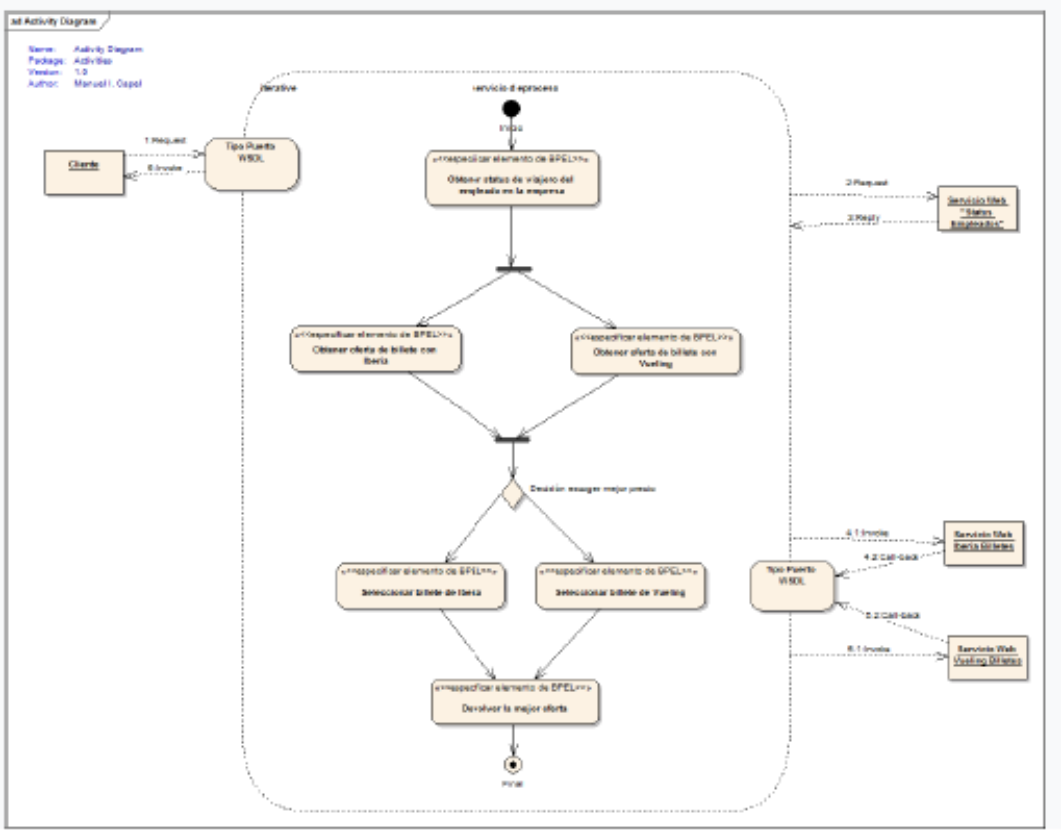

Figura 1: Enter Caption

En el sistema podemos destacar los siguientes componentes:

- **Empleado:** Recibe como entrada el **nombre del empleado**. Proporciona como salida el **tipo de billete asociado**, que puede ser Primera Clase, Jet Privado, o Turista, dependiendo del nombre ingresado (Pedro, María, u otro nombre, respectivamente).
- **Iberia:** Recibe como entrada el **tipo de billete**, el **Aeropuerto de Origen**, el **Aeropuerto de Destino**, la **Fecha y hora de salida**, y la **Fecha y hora de llegada**. Proporciona como salida el **precio final** basado en estos parámetros.
- **Vueling:** Recibe como entrada el **tipo de billete**, el **Aeropuerto de Origen**, el **Aeropuerto de Destino**, la **Fecha y hora de salida**, y la **Fecha y hora de llegada**. Proporciona como salida el **precio final** basado en estos parámetros.
- **Gestor:** Es el componente principal encargado de la orquestación. Como entrada, recibe el **nombre del empleado**, el **Aeropuerto de Origen**, el **Aeropuerto de Destino**, la **Fecha**

y hora de salida, y la **Fecha y hora de llegada**. Como salida, proporciona el **precio final** más bajo y la **aerolínea elegida**.

La orquestación se realiza a través del componente **Gestor**, que mediante los invoke coordina las interacciones con las distintas interfaces para implementar la lógica de negocio.

Como resumen de la lógica de negocio:

- El proceso inicia con un **invoke** al componente **Empleado**. Según el **nombre** ingresado, se determina el **tipo de billete** asociado: Pedro, María o cualquier otro nombre corresponden a Primera Clase, Jet Privado, o Turista, respectivamente.
- A continuación, se realizan dos flujos de ejecución paralelos hacia los componentes **Vueling** e **Iberia**. Ambos componentes comparten la misma interfaz de entrada y salida. Reciben como entrada el **tipo de billete**, el **Aeropuerto de Origen**, el **Aeropuerto de Destino**, la **Fecha y hora de salida**, y la **Fecha y hora de llegada**. Devuelven como salida el **precio final**.
- Una vez obtenidos los precios ofertados por ambos componentes, el **Gestor** compara los precios y selecciona el más bajo. Finalmente, se ofrece al cliente el precio más bajo y la aerolínea correspondiente.

1.2. Casos de prueba y testeo

Para validar el correcto funcionamiento del sistema, se pueden probar los componentes de manera individual o evaluar directamente el proceso completo mediante el **Gestor**, verificando su comportamiento bajo diferentes condiciones.

A continuación, se describen los casos de prueba realizados, incluyendo los resultados esperados y observados:

1.2.1. Primer caso: Caso base con Pedro

En este caso, se utiliza como entrada el **nombre: Pedro**, lo que corresponde a un **billete de Primera Clase**. Se verifica que los multiplicadores se ejecuten correctamente según la lógica de negocio.

- **IBERIA:** $5 \times 25 \times 25 = 3125$
- **VUELING:** $4 \times 25 \times 20 = 2000$

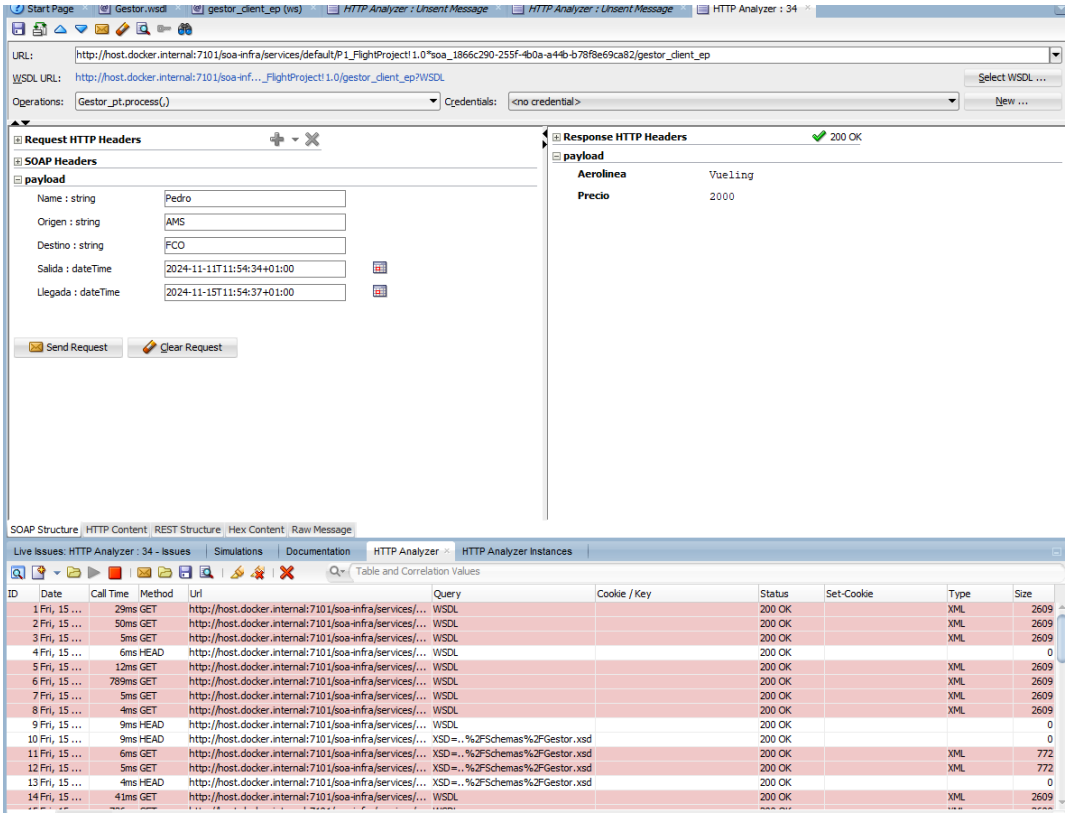

Figura 2: Primer caso

Observamos que el sistema selecciona correctamente la aerolínea con el precio más bajo: **VUELING, con un precio de 2000**.

1.2.2. Segundo caso: Caso con María

En este caso, se utiliza como entrada el **nombre: María**, lo que corresponde a un **billete de Jet Privado**. Debido a las características del billete, se espera un precio considerablemente alto.

- **IBERIA:** $5 \times 100 \times 25 = 12500$
- **VUELING:** $4 \times 200 \times 20 = 16000$

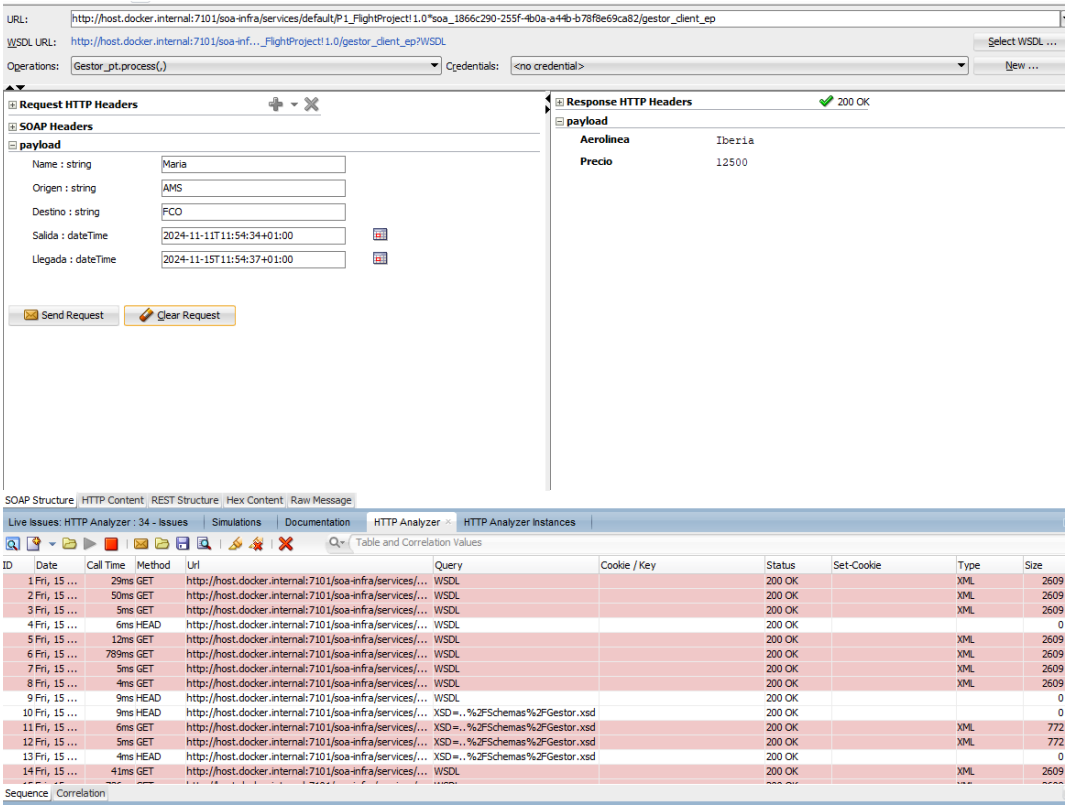

Figura 3: Segundo caso

El sistema selecciona correctamente la aerolínea con el precio más bajo: **IBERIA, con un precio de 12500**.

1.2.3. Tercer caso: Caso con otro nombre

En este caso, se utiliza un nombre diferente a Pedro o María, lo que corresponde a un **billete de Turista**. Este tipo de billete tiene un precio más accesible en comparación con los anteriores.

- **IBERIA:** $5 \times 5 \times 15 = 375$
- **VUELING:** $4 \times 8 \times 30 = 960$

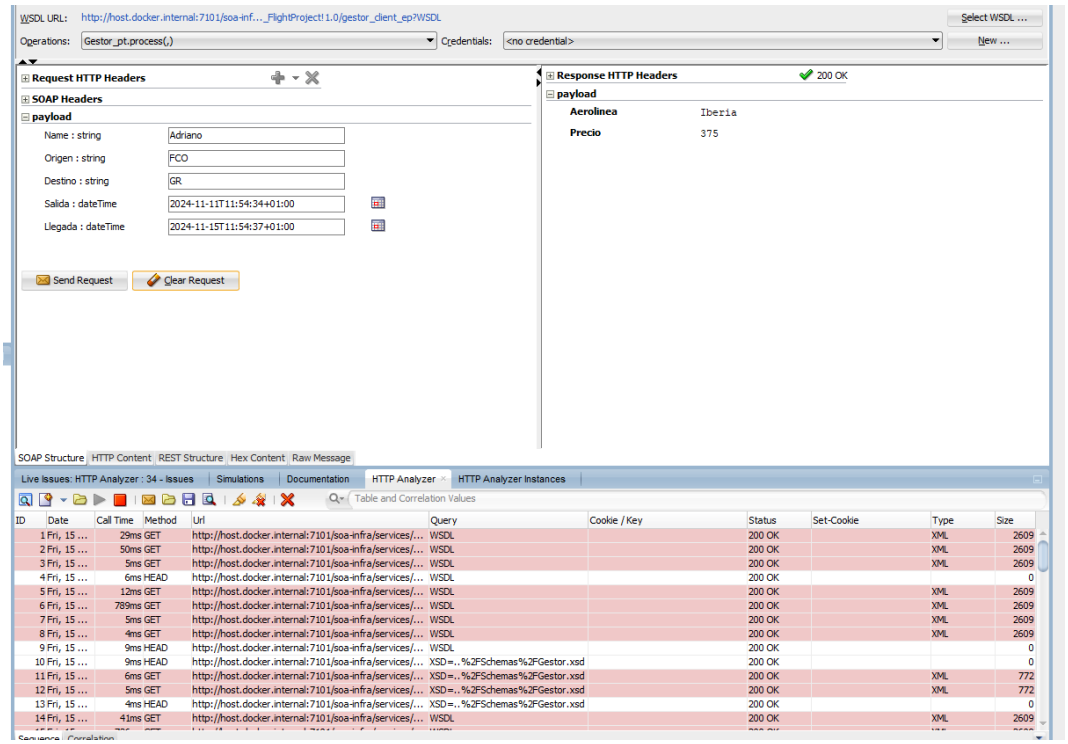

Figura 4: Tercer caso

El sistema selecciona correctamente la aerolínea con el precio más bajo: **IBERIA, con un precio de 375**.

1.2.4. Cuarto caso: Fecha de salida posterior a la fecha de llegada

En este caso, se prueba un escenario donde la **fecha de salida** es posterior a la **fecha de llegada**, lo que constituye un error lógico. Se ha diseñado un **throw fault** en los componentes Iberia y Vueling para manejar este caso.

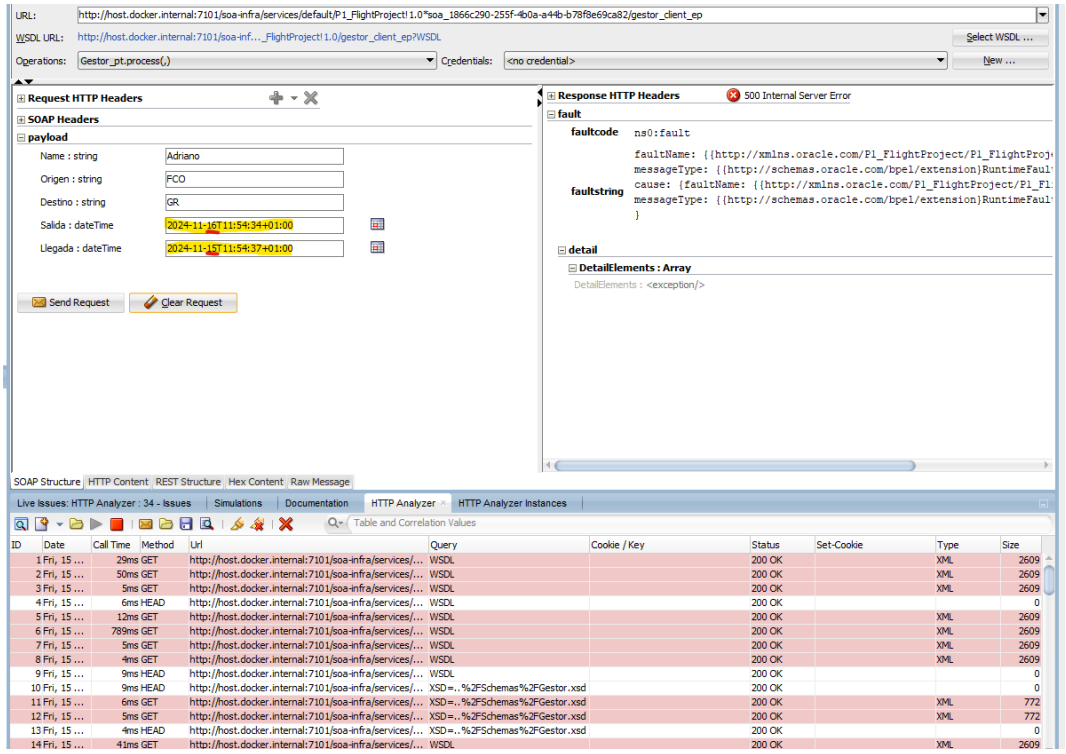

Figura 5: Cuarto caso

Observamos que el sistema responde correctamente con un **fault**, indicando el error de fechas inconsistentes.

2. Ejercicio 2

2.1. Estructura del compuesto SOA

El sistema está compuesto por los siguientes componentes principales:

- **VerCantidad:** Este componente recibe como entrada el **nombre del producto**. Como salida, proporciona:
  - El **nombre del producto**.
  - Un **mensaje** informativo.
  - La **cantidad en stock** disponible.
- **Comprador:** Recibe como entrada el **nombre del producto**, el **precio original**, y el **precio ofertado**. Proporciona como salida si **acepta** o no la oferta, basado en la relación entre el precio original y el ofertado.
- **Vendedor:** Recibe como entrada el **precio**. Como salida, genera un **precio ofertado**, aplicando un descuento del 10 % sobre el precio original.
- **Gestor:** Es el componente principal encargado de la orquestación. Como entrada, recibe el **nombre del producto** y el **precio original**. Como salida, proporciona un **mensaje final** y el **precio acordado** para el trato.

La orquestación se realiza a través del componente **Gestor**, que mediante invocaciones ('invoke') coordina las interacciones con las diferentes interfaces para implementar la lógica de negocio.

Resumen de la lógica de negocio

A continuación, se detalla el flujo de la lógica de negocio del sistema:

- El proceso comienza con una invocación al componente **VerCantidad**. Según el **nombre del producto** ingresado, se consulta el stock disponible. Este componente proporciona como salida:
  - Si hay stock disponible o no.
  - La cantidad de stock existente.
  - Un mensaje informativo.
- Si no hay stock disponible, el sistema informa al usuario mediante un mensaje y establece el precio de la operación en **0**. En caso de que haya stock, el flujo continúa con una llamada al componente **Vendedor**, quien realiza una oferta inicial.
- El componente **Comprador** evalúa la oferta inicial. Si la oferta es menor que 0.7 el valor original es aceptada, el sistema marca la variable `aceptable = true` y el proceso termina con éxito. Si no es aceptada, se activa un bucle iterativo para renegociar el precio.
- En el bucle:
  - El precio rechazado por el comprador se pasa nuevamente al **Vendedor**, quien genera una nueva oferta.
  - Esta nueva oferta se envía al **Comprador** para su evaluación.
  - El proceso se repite hasta que el comprador acepte la oferta (`aceptable = true`).
- Una vez finalizado el proceso de negociación, el sistema informa al usuario del **precio final acordado** y del **producto comprado**, acompañados de un mensaje final informativo.

2.2. Casos de prueba y testeo

2.2.1. Primer caso: Caso base con Pantalones

En este caso, utilizamos como entrada el **nombre: Pantalones o PANTALONES** *no es caps sensitivo* y un precio original de 10 euros. La lógica seguida por el vendedor es aplicar un descuento constante del 10 % en cada iteración hasta que el precio del producto sea aceptado por el comprador. Sin embargo, el comprador no aceptará el producto hasta que el precio sea inferior al 70 % del precio original, es decir, hasta que el precio haya bajado a menos de 7 euros.

La secuencia de cálculo para el precio de los pantalones, aplicando un descuento del 10 % de forma sucesiva, es la siguiente:

$$10 \times 0{,}9 = 9 \quad \text{(primer descuento)}$$

$$9 \times 0{,}9 = 8{,}1 \quad \text{(segundo descuento)}$$

$$8{,}1 \times 0{,}9 = 7{,}29 \quad \text{(tercer descuento)}$$

$$7{,}29 \times 0{,}9 = 6{,}561 \quad \text{(cuarto descuento)}$$

Como podemos ver, después de aplicar cuatro descuentos sucesivos del 10 %, el precio del producto llega a 6.561 euros, lo cual es aceptable para el comprador ya que está por debajo del 70 % del valor original.

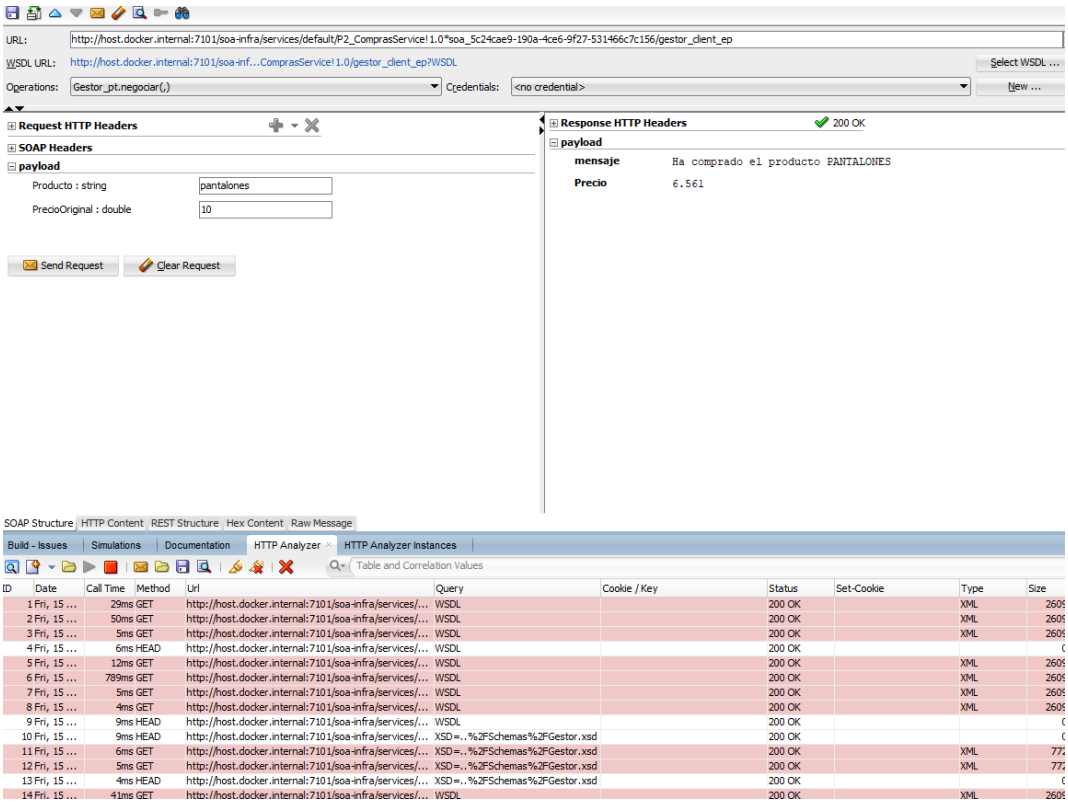

Figura 6: Primer caso: Precio de los pantalones con descuento sucesivo

2.2.2. Segundo caso: Caso con Camiseta

En este caso, se utiliza como entrada el **nombre: Camiseta**y un precio original de 15 euros. La lógica seguida por el vendedor es aplicar un descuento constante del 10 % en cada iteración hasta que el precio del producto sea aceptado por el comprador. Sin embargo, el comprador no aceptará el producto hasta que el precio sea inferior al 70 % del precio original, es decir, hasta que el precio haya bajado a menos de 10.5 euros.

$$15 \times 0{,}9 = 13{,}5 \quad \text{(primer descuento)}$$
$$13{,}5 \times 0{,}9 = 12{,}5 \quad \text{(segundo descuento)}$$
$$12{,}5 \times 0{,}9 = 10{,}935 \quad \text{(tercer descuento)}$$
$$10{,}935 \times 0{,}9 = 9{,}8415 \quad \text{(cuarto descuento)}$$

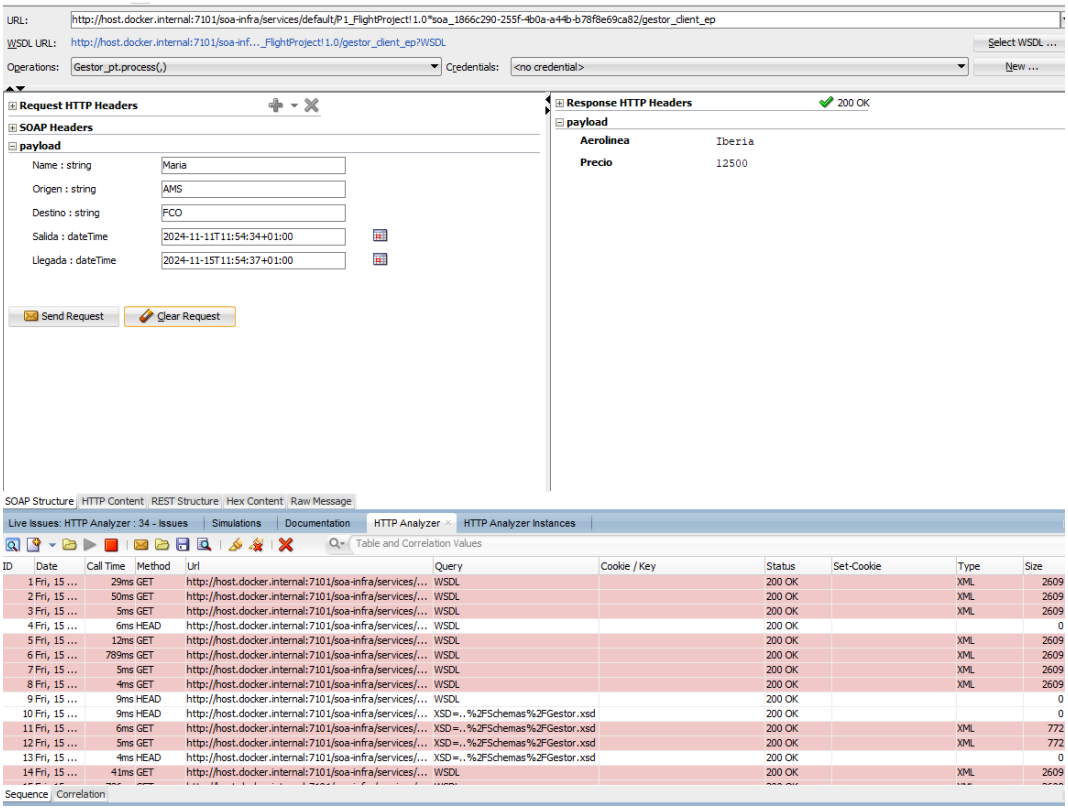

Figura 7: Segundo caso: Precio de la camiseta

2.2.3. Tercer caso: Caso con otro nombre

En este caso, se utiliza un nombre diferente y por lo tanto deberá informar que no queda stock en la tienda.

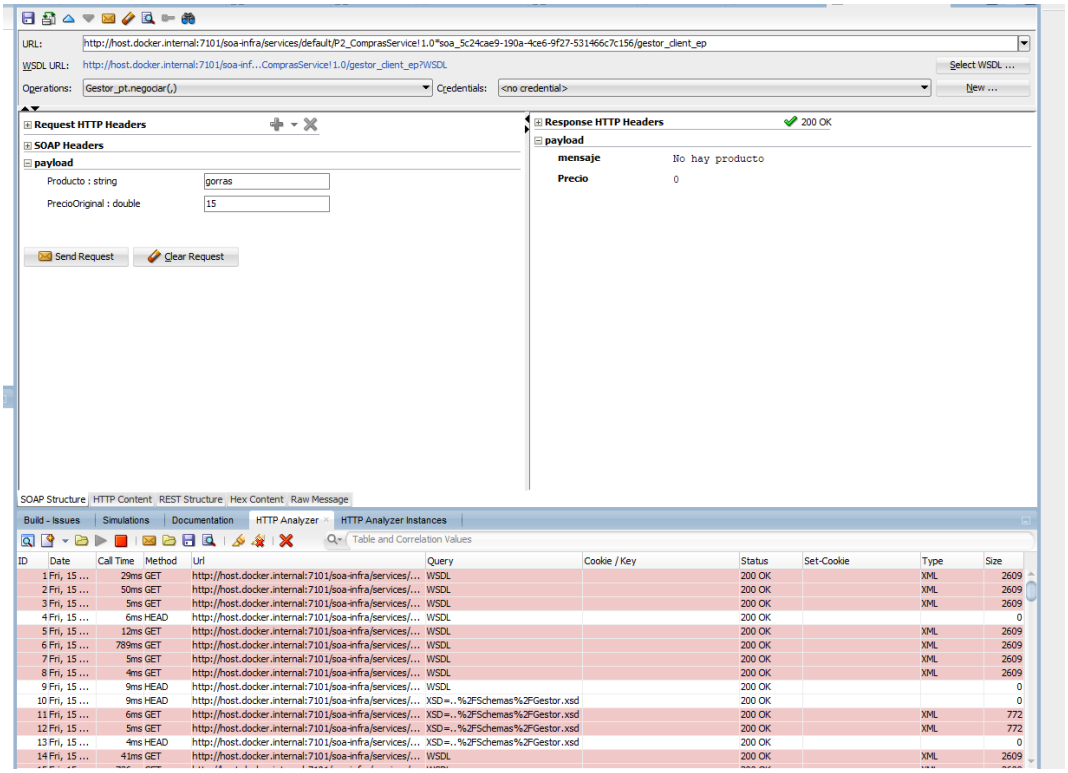

Figura 8: Tercer caso,No hay stock

3. Instrucciones de despliegue

El despliegue de la aplicación puede generar errores la primera vez, específicamente en el proceso **Gestor**. Esto suele deberse a que los enlaces de los servicios relacionados no están correctamente configurados en el proyecto. A continuación, se explica cómo verificar y corregir esta configuración utilizando un ejemplo basado en el Ejercicio 2.

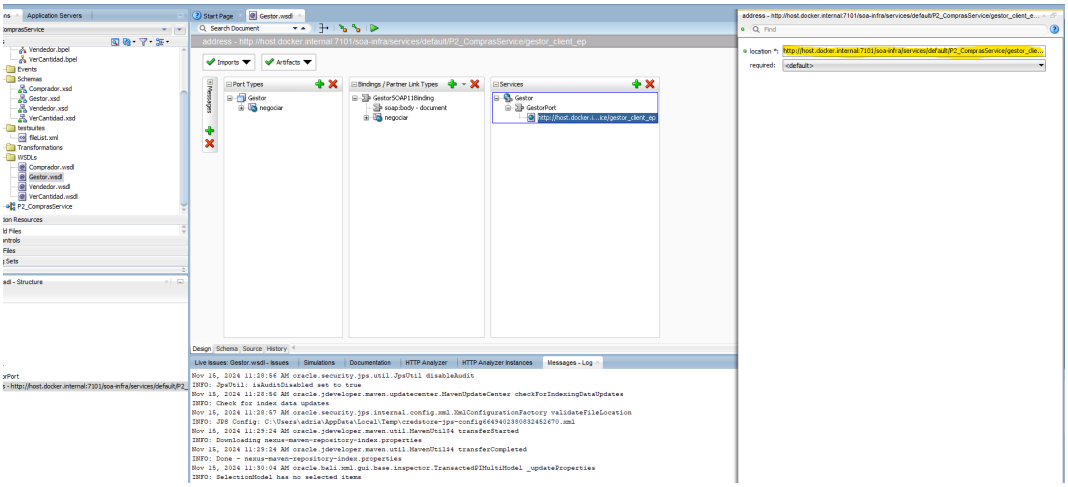

Figura 9: Ubicación de los endpoints

Por ejemplo, para el servicio **Gestor**, el enlace generado podría ser: `http://host.docker.internal:7101/soa-infra/services/default/P2_ComprasService/gestor_client_ep`.

Los servicios relacionados, como **VerCantidad**, **Vendedor** y **Comprador**, deben configurarse con enlaces similares, cambiando únicamente la parte final del endpoint. Por ejemplo:

- Para **VerCantidad**: `http://host.docker.internal:7101/soa-infra/services/default/P2_ComprasService/vercantidad_client_ep`
- Para **Vendedor**: `http://host.docker.internal:7101/soa-infra/services/default/P2_ComprasService/vendedor_client_ep`
- Para **Comprador**: `http://host.docker.internal:7101/soa-infra/services/default/P2_ComprasService/comprador_client_ep`

Sin embargo, la dirección base del servidor puede variar según el entorno de despliegue. Por ejemplo, en lugar de `host.docker.internal`, podría utilizarse `localhost`. En este caso, la configuración sería: `http://localhost:7101/soa-infra/services/default/P2_ComprasService/gestor_client_ep`.

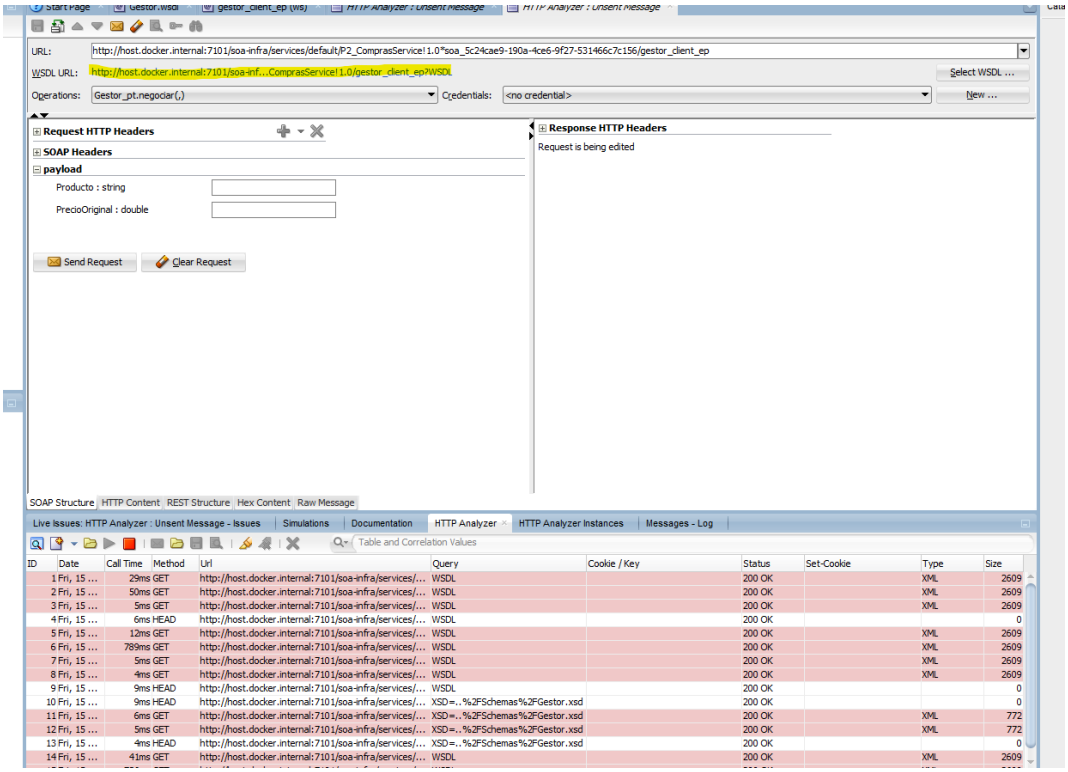

Figura 10: Configuración de endpoints relativos

Solución en caso de error:
Si al probar el servicio aparece un **fault** relacionado con la dirección del endpoint, es probable que esta no esté configurada correctamente. Para solucionarlo:

1. Despliega la aplicación en el servidor.
2. Copia la dirección del endpoint directamente desde la consola o el entorno de despliegue.
3. Asegúrate de que cada servicio (Gestor, VerCantidad, Vendedor, Comprador) tenga configurada la dirección correcta en función del entorno (por ejemplo, `localhost` o `host.docker.internal`).

Esta verificación asegura que los servicios puedan comunicarse correctamente y evita errores durante el testeo de la aplicación.