Things to Remember:

I'll show you the curl command and then break down the entire flow when you execute it.

Curl Command:

To Check the Wsdl is fine:

curl -X GET http://localhost:8686/services/weather?wsdl

To Initiate the Process:

curl -X GET "<http://localhost:8686/weather/London/UK>"

To Check via SOAP request:

curl -X POST http://localhost:8686/services/weather \

-H "Content-Type: text/xml;charset=UTF-8" \

-H "SOAPAction: http://weather.example.com/getWeather" \

-d '

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"

xmlns:web="http://weather.example.com/">

<soapenv:Header/>

<soapenv:Body>

<web:getWeatherRequest>

<city>London</city>

<country>UK</country>

</web:getWeatherRequest>

</soapenv:Body>

</soapenv:Envelope>'

Here's the step-by-step flow when you execute these commands:

Detailed Flow Explanation:

1. Initial REST Request:

- Your curl command hits the REST endpoint at `http://localhost:8686/weather/London/UK`

- The request is handled by the REST definition in your route: `rest("/weather").get("/{city}/{country}")`

2. Camel Route Processing:

- The request is forwarded to `direct:getWeather`

- The first processor creates a SOAP request:

```java

GetWeatherRequest request = new GetWeatherRequest();

request.setCity("London");

request.setCountry("UK");

```

3. SOAP Service Call:

- The route sends the SOAP request to your mock service via CXF

- The request goes to `http://localhost:8686/services/weather`

- Your mock service (`WeatherSoapEndpoint`) receives the request

4. Mock Service Processing:

- The mock service generates a response:

```java

GetWeatherResponse response = new GetWeatherResponse();

response.setTemperature(BigDecimal.valueOf(20.5));

response.setConditions("Partly Cloudy");

response.setHumidity(BigDecimal.valueOf(65.0));

```

5. Response Transformation:

- The SOAP response is received back in the Camel route

- The second processor converts it to your DTO:

```java

WeatherDto dto = new WeatherDto();

dto.setTemperature(response.getTemperature());

dto.setConditions(response.getConditions());

dto.setHumidity(response.getHumidity());

```

6. Final Response:

- The DTO is automatically converted to JSON

- The JSON response is sent back to the client

- Expected response would look like:

```json

{

"temperature": 20.5,

"conditions": "Partly Cloudy",

"humidity": 65.0

}

```

Alternative curl commands:

```bash

# With verbose output

curl -v -X GET "http://localhost:8686/weather/London/UK"

# Pretty print JSON response

curl -X GET "http://localhost:8686/weather/London/UK" | json\_pp

# Specify accept header

curl -H "Accept: application/json" -X GET "http://localhost:8686/weather/London/UK"

```

If any errors occur during this process, your error handlers will catch them and return an error response like:

```json

{

"error": "Error message here"

}

```

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

DETAILED EXPLANATION:

I'll break down the entire flow of your weather service application:

1. Initial REST Endpoint

```java

rest("/weather")

.get("/{city}/{country}")

.produces("application/json")

.to("direct:getWeather");

```

- This defines a REST endpoint at `/weather/{city}/{country}`

- It accepts HTTP GET requests

- Parameters `city` and `country` are path parameters

- `.produces("application/json")` tells clients that this endpoint returns JSON data

- Routes the request to an internal direct endpoint named "getWeather"

2. Main Route Processing

```java

from("direct:getWeather")

.log("Fetching weather for ${header.city}, ${header.country}")

```

- The route starts processing from the direct:getWeather endpoint

- Logs the request details for monitoring/debugging

3. SOAP Request Preparation

```java

.process(exchange -> {

GetWeatherRequest request = new GetWeatherRequest();

request.setCity(exchange.getIn().getHeader("city", String.class));

request.setCountry(exchange.getIn().getHeader("country", String.class));

exchange.getIn().setBody(request);

})

```

- Creates a SOAP request object (GetWeatherRequest)

- Extracts city and country from the HTTP headers

- Sets these values in the SOAP request

- Places the SOAP request in the exchange body

4. SOAP Service Call

```java

.to("cxf://http://localhost:8686/services/weather"

+ "?serviceClass=com.example.weather.generated.WeatherPortType"

+ "&wsdlURL=/weather-service.wsdl"

+ "&defaultOperationNamespace=http://weather.example.com/"

+ "&loggingFeatureEnabled=false")

```

- Calls the SOAP web service running on localhost:8686

- Uses CXF component for SOAP communication

- Specifies the WSDL and service details

- The service returns a GetWeatherResponse object

5. Response Transformation

```java

.process(exchange -> {

GetWeatherResponse response = exchange.getIn().getBody(GetWeatherResponse.class);

WeatherDto dto = new WeatherDto();

dto.setTemperature(response.getTemperature());

dto.setConditions(response.getConditions());

dto.setHumidity(response.getHumidity());

// Convert WeatherDto to JSON string

String jsonString = JsonbBuilder.create().toJson(dto);

exchange.getIn().setBody(jsonString);

})

```

- Extracts the SOAP response

- Creates a WeatherDto object (your internal data model)

- Copies data from SOAP response to DTO

- Converts DTO to JSON string using JSON-B

- Sets the JSON string as the response body

6. Response Headers

```java

.setHeader(Exchange.CONTENT\_TYPE, constant("application/json"));

```

- Sets the HTTP content type header to application/json

7. Error Handling

```java

onException(Exception.class)

.handled(true)

.log("Error processing request: ${exception.message}")

.setHeader(Exchange.CONTENT\_TYPE, constant("application/json"))

.setBody(simple("{\"error\": \"${exception.message}\"}"));

```

- Catches any exceptions during processing

- Logs the error

- Returns a JSON error response

- Ensures error responses are also JSON formatted

Complete Flow Example:

1. Client makes HTTP GET request to `/weather/London/UK`

2. Camel receives request and extracts "London" and "UK" from path

3. Creates SOAP request with these parameters

4. Calls SOAP service and gets weather data

5. Transforms SOAP response to WeatherDto

6. Converts WeatherDto to JSON

7. Returns JSON response to client

Sample Request:

```bash

GET http://localhost:8080/weather/London/UK

```

Sample Success Response:

```json

{

"temperature": 18.5,

"conditions": "Partly Cloudy",

"humidity": 65

}

```

Sample Error Response:

```json

{

"error": "Service unavailable"

}

```

This architecture demonstrates a common integration pattern: REST-to-SOAP transformation, where a modern REST API facade is placed in front of a legacy SOAP service. The Camel route handles all the necessary transformations transparently to the client.

Process Flow Diagram:

