**Exercise 15b**

*Learn about mediating between services and APIs with Ballerina*

**Prior Knowledge**

Basic understanding HTTP verbs, REST architecture, SOAP and XML

**Objectives**

Understand Ballerina, and simple JSON to XML mapping.

**Software Requirements**

(see separate document for installation of these)

* Java Development Kit 11
* Ballerina Swan Lake alpha 4
* Visual Studio Code
* Ballerina Swan Lake alpha 4 VSIX
* Docker

*Overview*

Ballerina is a programming language that is designed around services, microservices and mediation. It also has a graphical view, but unfortunately that is temporarily not working (at the time of writing)!

In this lab we are going to work with a “legacy” payment service based on XML and SOAP.

In this lab, we are going to take a WSDL/SOAP payment service, which is loosely modeled on a real SOAP API (Barclaycard SmartPay <https://www.barclaycard.co.uk/business/accepting-payments/website-payments/web-developer-resources/smartpay#tabbox1>).

Our aim is to convert this into a simpler HTTP/JSON interface. We probably won’t get as far as any truly RESTful concepts as we won’t have the opportunity to add resources, HATEOAS, etc. But will look at how those could be added with more time.

**PART A - Simple Ballerina Service**Check Ballerina is installed:  
  
bal version

Ballerina Swan Lake Alpha 4

Language specification v2020-12-17

Update Tool 1.3.1

1. Make a new Ballerina project

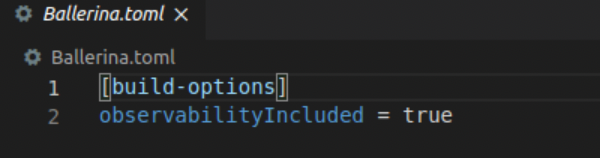
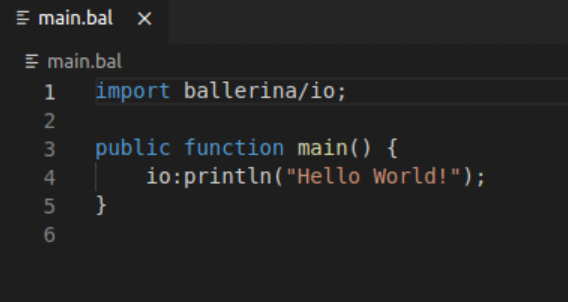
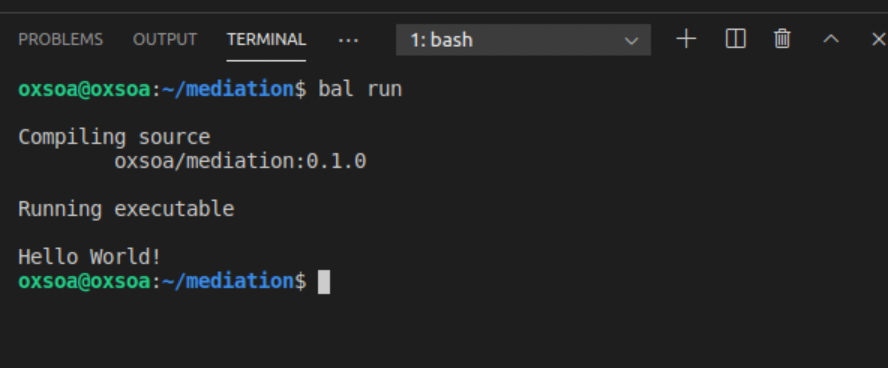
cd ~

bal new mediation  
   
Created new Ballerina package 'mediation' at mediation.

1. This has created a simple package including a config for the project, gitignore and main.bal:

cd ~/mediation

code .

1. Take a look at Ballerina.toml  
     
     
   This is a config file for the project.
2. Take a look at main.bal  
   
3. We can quickly run this:  
   bal run  
     
   
4. Ballerina is based on the JVM, and this is actually compiling the ballerina code into Java bytecode, making a .JAR file and then executing that.

We can also build manually:  
  
bal build

Compiling source

oxsoa/mediation:0.1.0

Generating executable

target/bin/mediation.jar

This JAR file can also be run directly by the JVM:  
  
java -jar target/bin/mediation.jar  
  
Hello World!

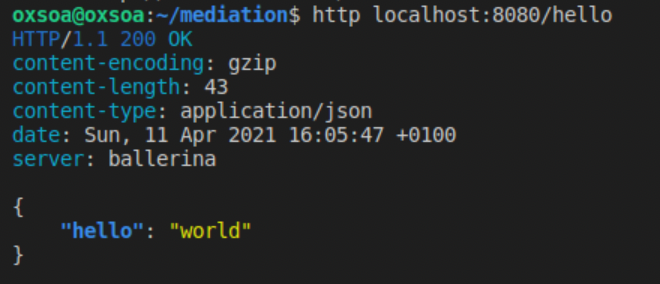
1. However, we do not really want a “main” method. We are going to create a service instead. Let’s delete main.bal and instead create a new file “mediation.bal” which will expose an HTTP service.

rm main.bal

bal clean

code mediation.bal

1. Hit Ctrl-s to save, as vscode intelligent completion only kicks in when the file is saved.
2. Let’s start by creating a simple HTTP server first.
3. Type the following code.  
   
4. Notice that json is a first class type in this language.
5. Save it.
6. Start a command line in vscode (Ctrl+`)  
     
   *A big hint. Under the covers of “yarn dev” was a tool called nodemon that was watching for changes to the files. We can get this to work for Ballerina as well:*  
     
   cd ~/mediation  
   nodemon -e bal --signal SIGINT --exec bal run  
     
   
7. Start another terminal and   
     
   http http://localhost:8080/hello



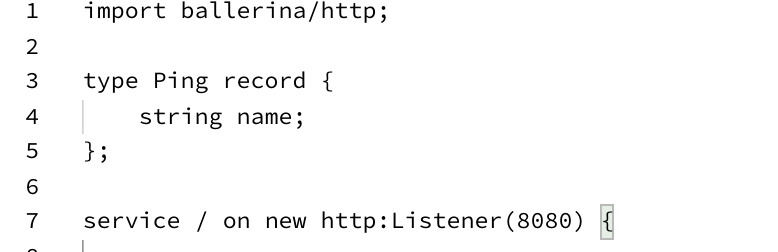
1. Let’s evolve this service a little bit first before we tackle calling the SOAP backend.
2. Let’s start by defining a new type. Ballerina does have classes, but it also has a simpler concept called a **record** that is better for mapping to JSON, etc. You can read more here: <https://ballerina.io/learn/by-example/records.html>

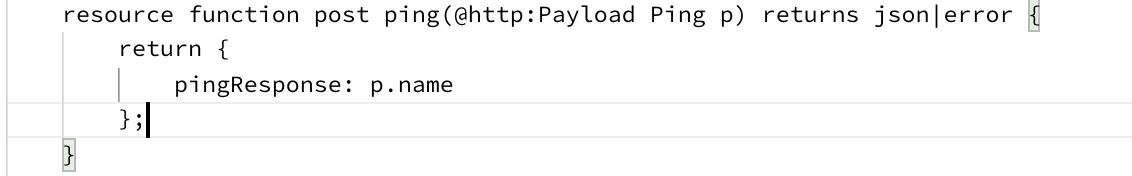
Add this code above your service definition:

type Ping record {

string name;

};

1. Your code should look like:  
     
   
2. Let’s add a ping function that expects a POST message and reads a JSON body into a record of type Ping:

  
  
This is reasonably obvious. The @http:Payload indicates that the parameter is coming from mapping the payload into a record of type Ping.

Note the “union type” json|error   
This means that this method can return either a json or an error.

1. You can test your code like this:  
   
2. Start the “backend” SOAP service:  
    docker run -d -p 8888:8080 pizak/pay
3. We are also going to intercept all the messages between the Ballerina and the backend using mitmdump. In a new terminal window start:  
     
   mitmdump --listen-port 8000 --set flow\_detail=3 --mode reverse:<http://localhost:8888>

1. Check that it is running:  
   Browse: <http://localhost:8000/pay/services/paymentSOAP?wsdl>
2. The SOAP service we are calling has two methods. The first is just a “ping/echo” that will return whatever string we send it. This is a useful test that the service is working. The second is the actual payment service, which takes various credit card details and then returns a response.

Here is an HTTP level log of the ping service.  
  

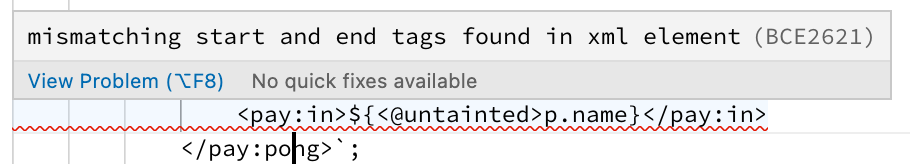

1. We will next create a mediation for ping. Following, we will create the payment method using a freeform approach.
2. There is a SOAP library for Ballerina, but we don’t actually need it.   
   Ballerina has built in support for XML types. We are going to use this to craft the XML message to send to the SOAP backend.  
   You can find out more here:  
   <https://ballerina.io/learn/by-example/xml-literal.html>

<https://ballerina.io/learn/by-example/xml-namespaces.html>

We can really simply create an XML body to send to the payment service. Add this code to the body of your “ping” method. I basically copied this from the SOAPUI method. In addition, I split out the SOAP envelope XML from the body to be clearer and more reusable.



Notice the simple templating ${p.name}

Another thing worth noticing is that the vscode plugin knows when your xml is valid or not. For example if I change the end tag to </pay:pong> instead you will see:  
  
  
It is hard to explain how clever this is unless you’ve struggled with coding XML in other languages!

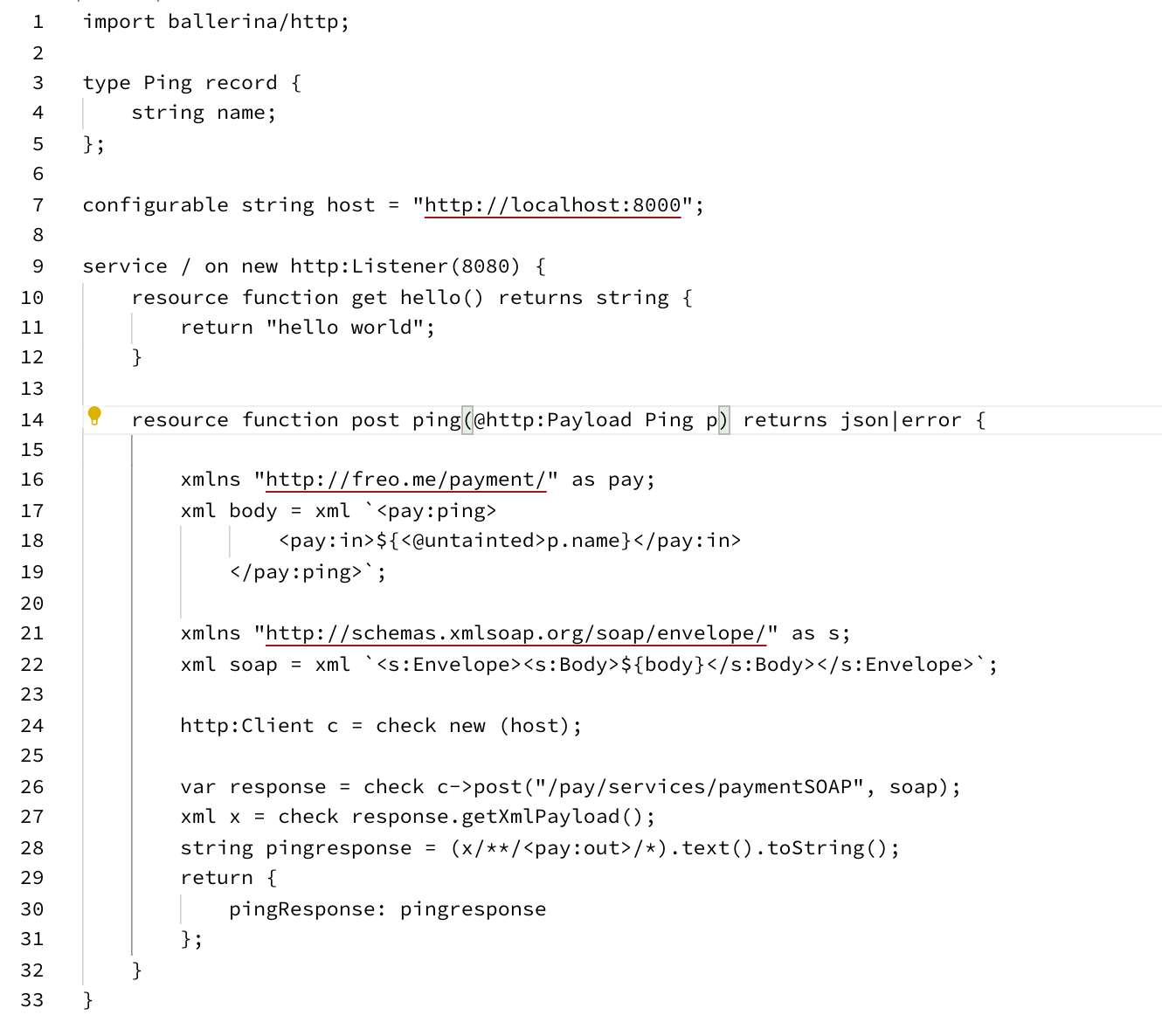
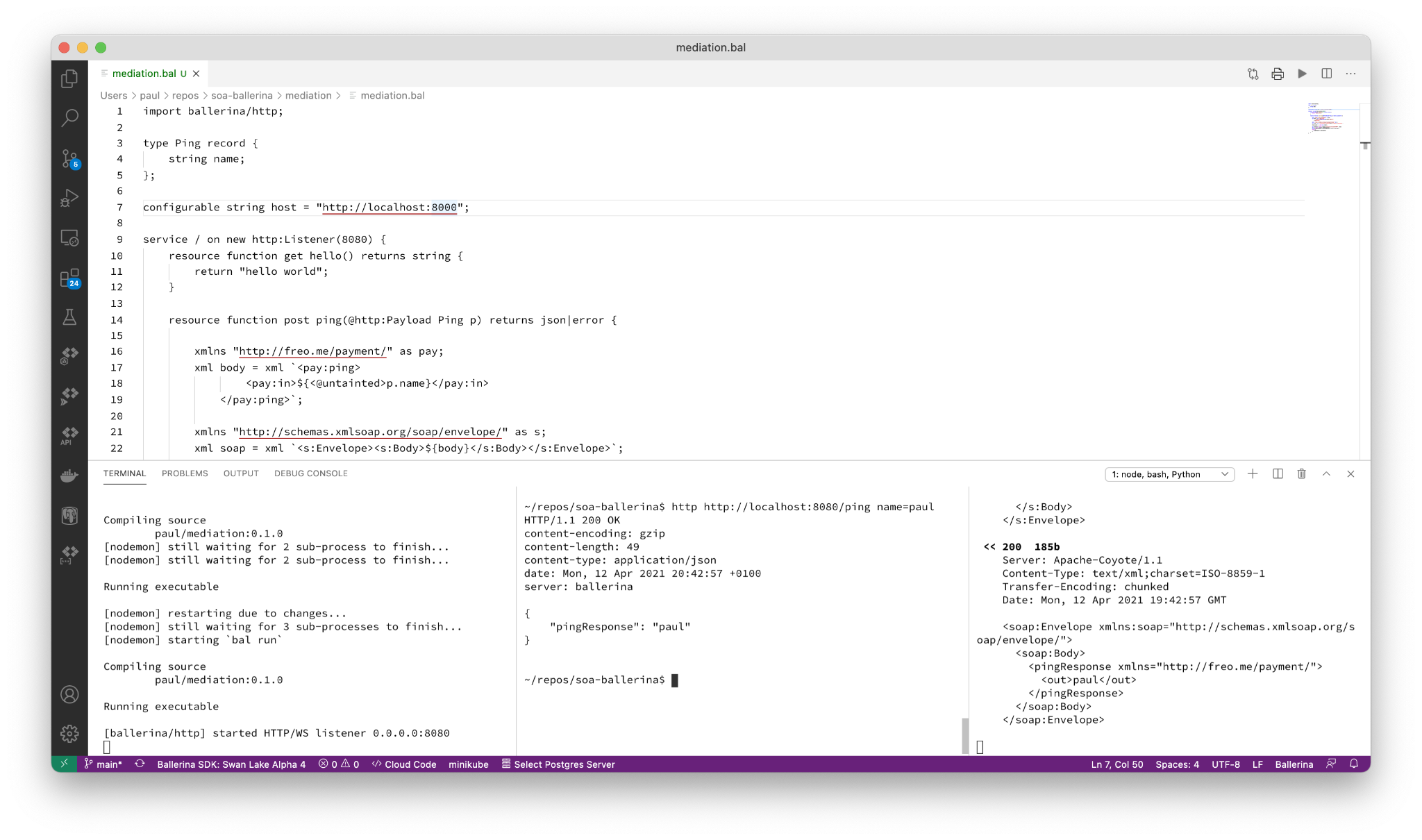
1. Now we need to send the XML to the HTTP backend. At the top of the file, after the imports, add:  
     
   configurable string host = "<http://localhost:8000>";
2. Now, back under the xml code, add:  
     
     
   You can delete the old “return” lines.

What is going on here?

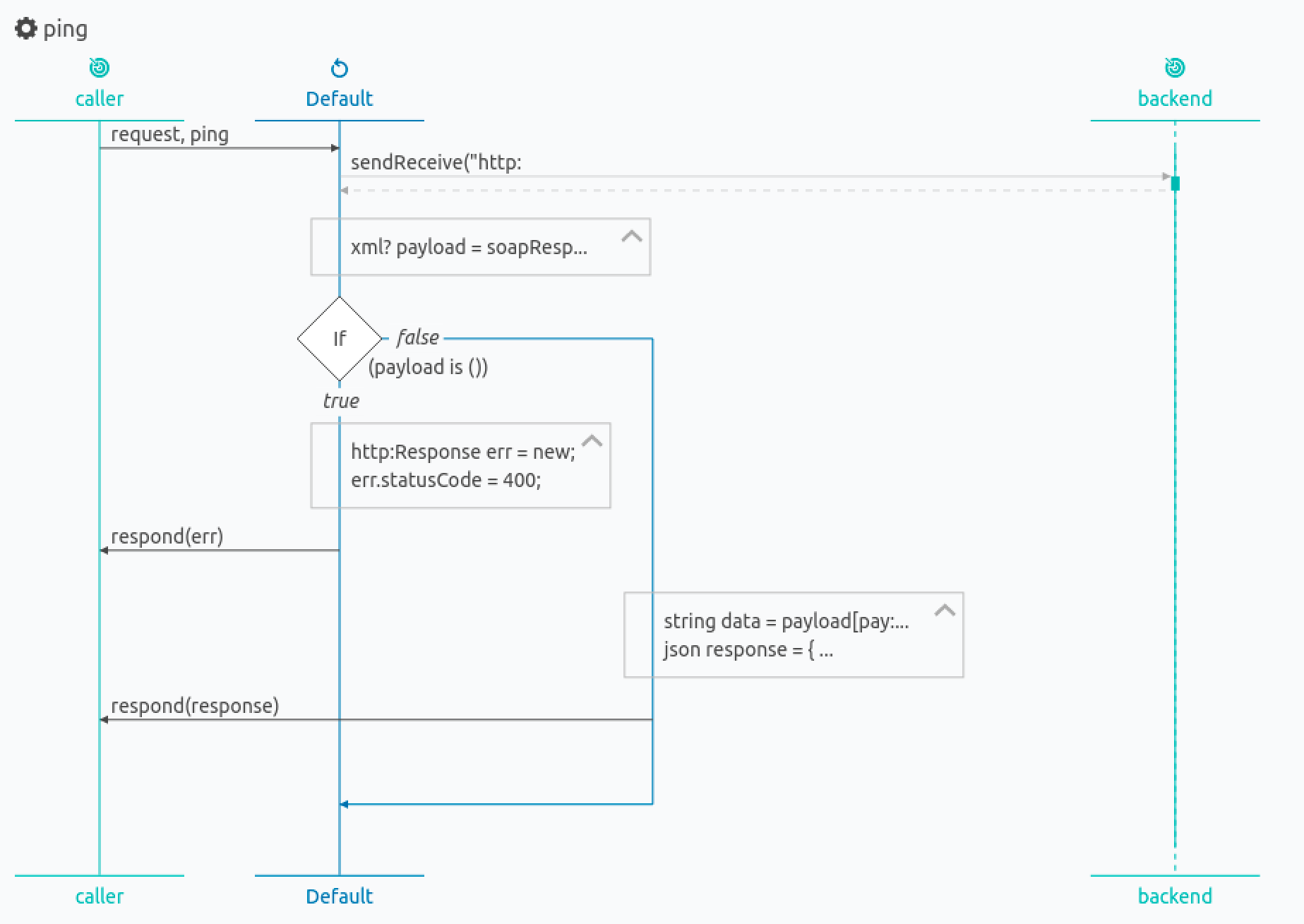
Firstly, let’s understand “check”. This looks for an error and throws it if it finds it (<https://ballerina.io/learn/by-example/check.html>)   
Secondly this is an interesting line:

string pingresponse = (x/\*\*/<pay:out>/\*).text().toString();

Basically this looks inside the xml of variable “x” at any depth for an element <pay:out>, then for its child (.\*) and then gets the text and converts to a string.

1. Your code should now look like:  
     
   Try it out with the same request as before.
2. This time you should see a successful call in the MITMDUMP terminal. Here is a screenshot:  
   

Ballerina is a language specifically designed for orchestration of services, modelled on sequence diagrams. Unfortunately the vscode support for the graphical view is temporarily disabled in this alpha preview. Here is a screenshot from a previous release of some similar but slightly different code.



The cool thing is that these sequence diagrams are automatically generated and updated from the code. According to this link, they should be enabled again soon:  
<https://stackoverflow.com/questions/66401970/ballerina-swan-lake-an-ides-how-to-get-sequence-diagrams> **PART C - ACTUAL CARD PAYMENT**

1. Now we need to take an incoming JSON in a POST, parse it, and then convert it into a more complex XML.
2. Here is the XML interaction we need to talk to the SOAP service  
   
3. I want you to parse an incoming JSON, e.g.:   
   
4. There is a postman collection you can download and install here:  
   <https://raw.githubusercontent.com/pzfreo/soa-ballerina/main/payment-collection.json>
5. You should know almost enough to complete this but there are a couple of things you will need to know extra.
6. Firstly, obviously you need to copy and paste the resource function for **ping** and rename the new version to be **authorise**. This will make the request path be /authorise
7. Here is a record definition for the incoming JSON that you can use:

Also available here <https://freo.me/pay-struct>   
  
You can also read more about Records here: <https://ballerina.io/learn/by-example/records.html>  


1. Copy and paste this and put it at the same level as the service in the .bal file (so it is globally defined for this ballerina program).
2. You need to construct an XML that looks like this. You can cut and paste this from SOAPUI.

You will need to use the same templating as above to insert the data.  


1. Finally, I want you to return a JSON that looks like:  
   {

"authcode": "FAILED",

"reference": "b23f8aad-766d-46c9-98c2-f328ce8ed594",

"refusalreason": "INSUFFICIENT FUNDS"

}

or

{

"authcode": "AUTH0234",

"reference": "07b82cad-cb55-428d-b02c-c5619bbbed4d",

"refusalreason": "OK"

}

I have created a Postman collection json to test this:

1. If you get stuck, my version is available here:   
   <https://github.com/pzfreo/soa-ballerina/blob/main/mediation/mediation.bal>

**Extensions**

1. Try the OpenAPI ballerina to swagger interface:  
     
   bal openapi -i mediator.bal
2. Try generating Ballerina server-side skeletons from the Purchase swagger:  
     
   <https://ballerina.io/learn/tooling-guide/cli-tools/openapi/>
3. Take a look at Ballerina’s support for gRPC:  
   <https://ballerina.io/learn/by-example/proto-to-ballerina.html>
4. And AWS Lambda:

<https://ballerina.io/learn/by-example/aws-lambda-deployment.html>