Implementing our first web service

Approaches to building web services

◉There are 2 approaches to building web services :

1. Code first (Bottom Up) approach

Allows you to reuse your investment by exposing your existing application.

For example : Credit Card System is an existing Application with Proven business value. Competitive pressure is moving Credit Card System to expose some of this business functionality Like (Credit Card Number Validation) as web service. The implementation class **already exists**, all that is needed is to create a WSDL and expose the class as web service.

2. Contract First (Top Down) Approach

Is the correct way to build new web service from scratch.

This Approach starts with the WSDL (the contract) by defining operation message and so forth. Then you build the endpoint interface and finally the implementation class.

1. Code first (Bottom Up) approach

- In bottom-up development, the interface is generated based on an **existing implementation** thereby exposing its functionality.
- Bottom-up development can be a fast way of creating interfaces but gives you less control over the exact definition of the interface.

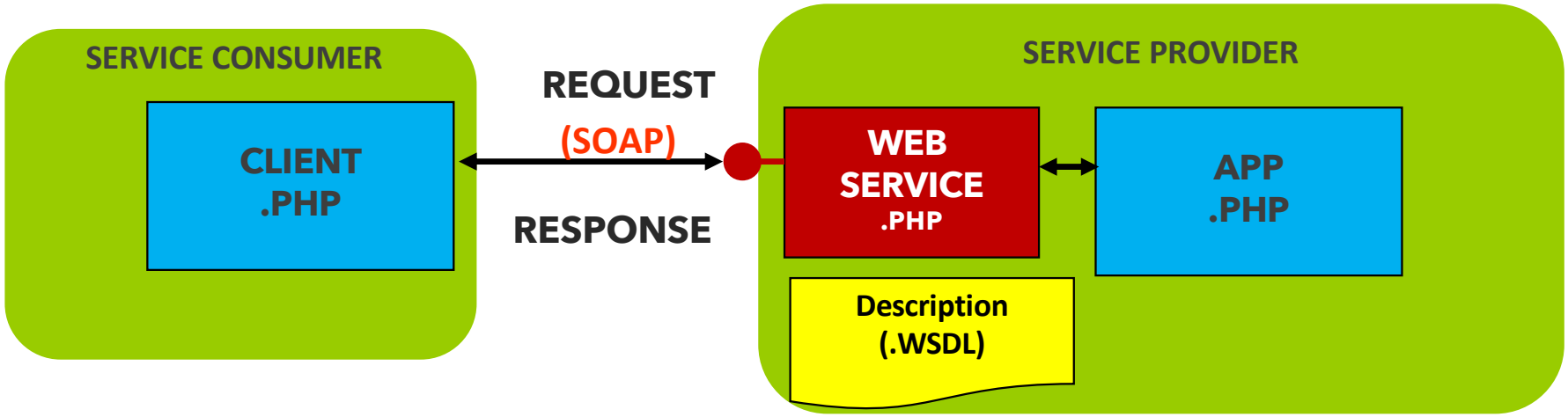
2. Contract First (Top Down) Approach

- In top-down or contract-first development, the service interface is the starting point for developing services.
- A framework is used to generate a code skeleton based on the service WSDLs and XSDs. The generated implementation can then be completed.
- This can be a good approach when there is no service implementation yet in place.

Code first (Bottom Up) approach

Implementing service

- We have initially :
  - App (.php)
- We will implement :
  - Web service: Webservice (.php)
  - Client : Client (.php) or any other programming language.
  - Contract : (.WSDL) will be automatically generated by SOAP API.



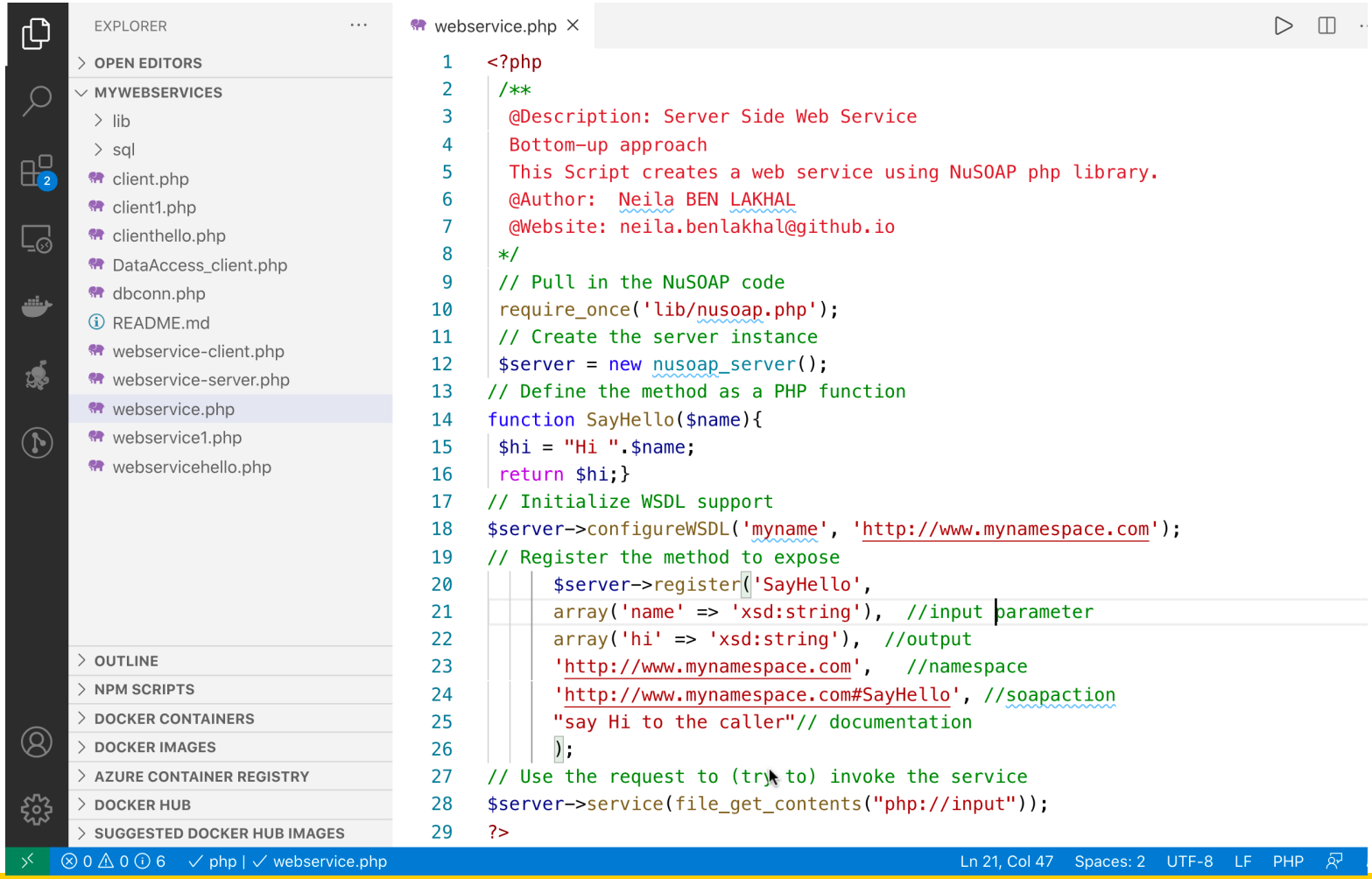
Example 1

- Create a web service for the following php function :

```
function SayHello($name){
$hi = "Hi ".$name;
return $hi;
}
```

Step 1: write down the webservice.php
Step 2: Deploy the WS in an application server (WWW/HTDOCS folder)

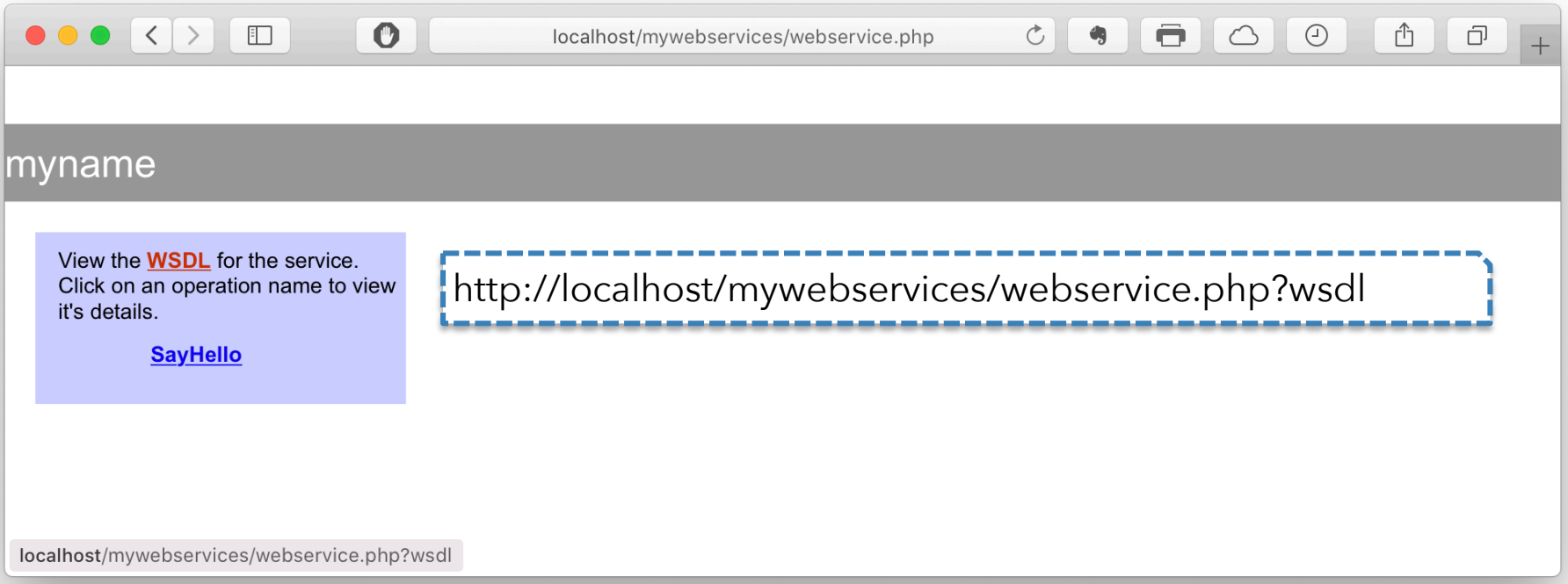
Step 1 : write the code



Step 2: Deploy

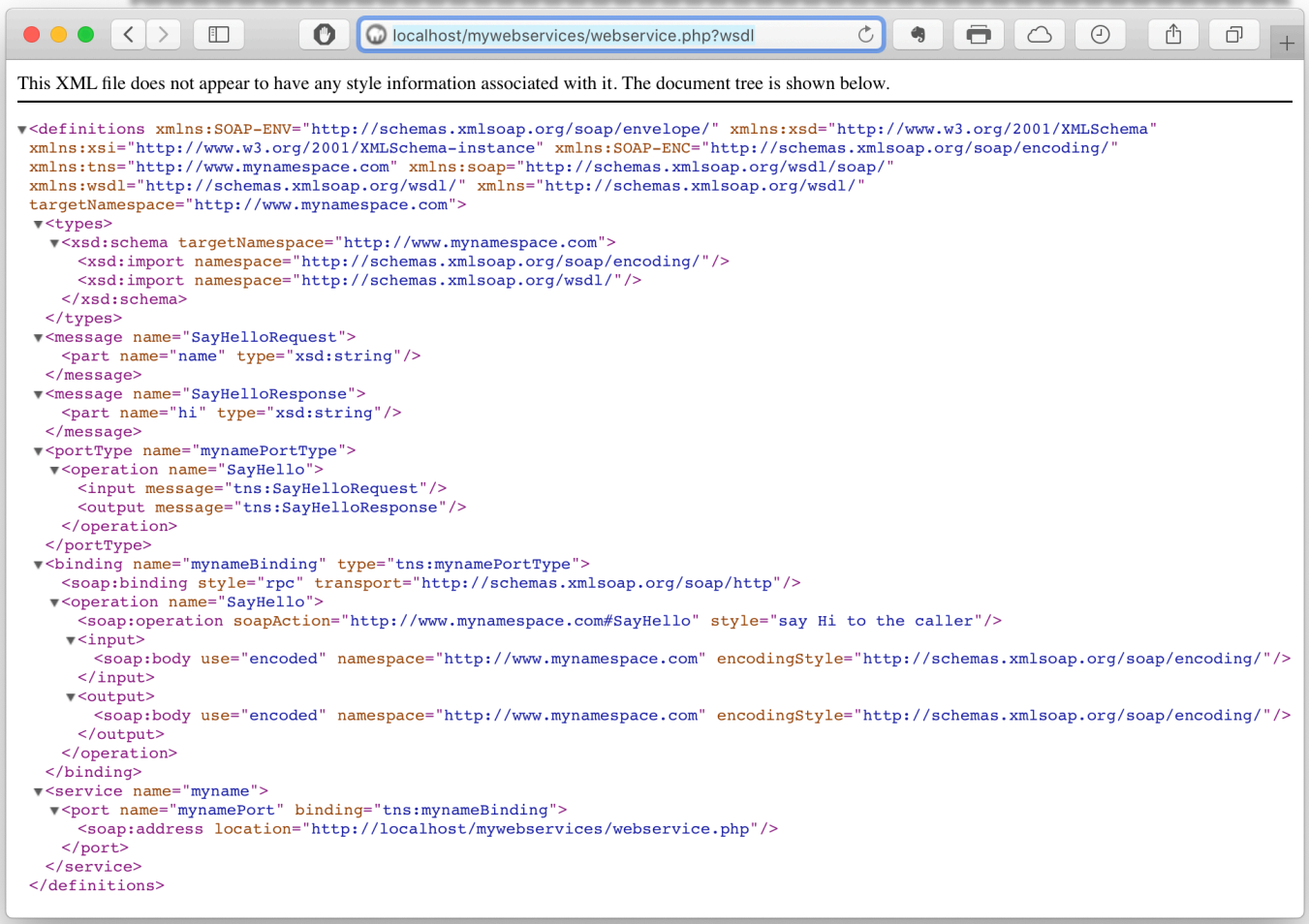
- Add your **webservice** file to **WWW** or **htdocs** folder
- Access the web service :

  http://localhost/mywebservices/webservice.php

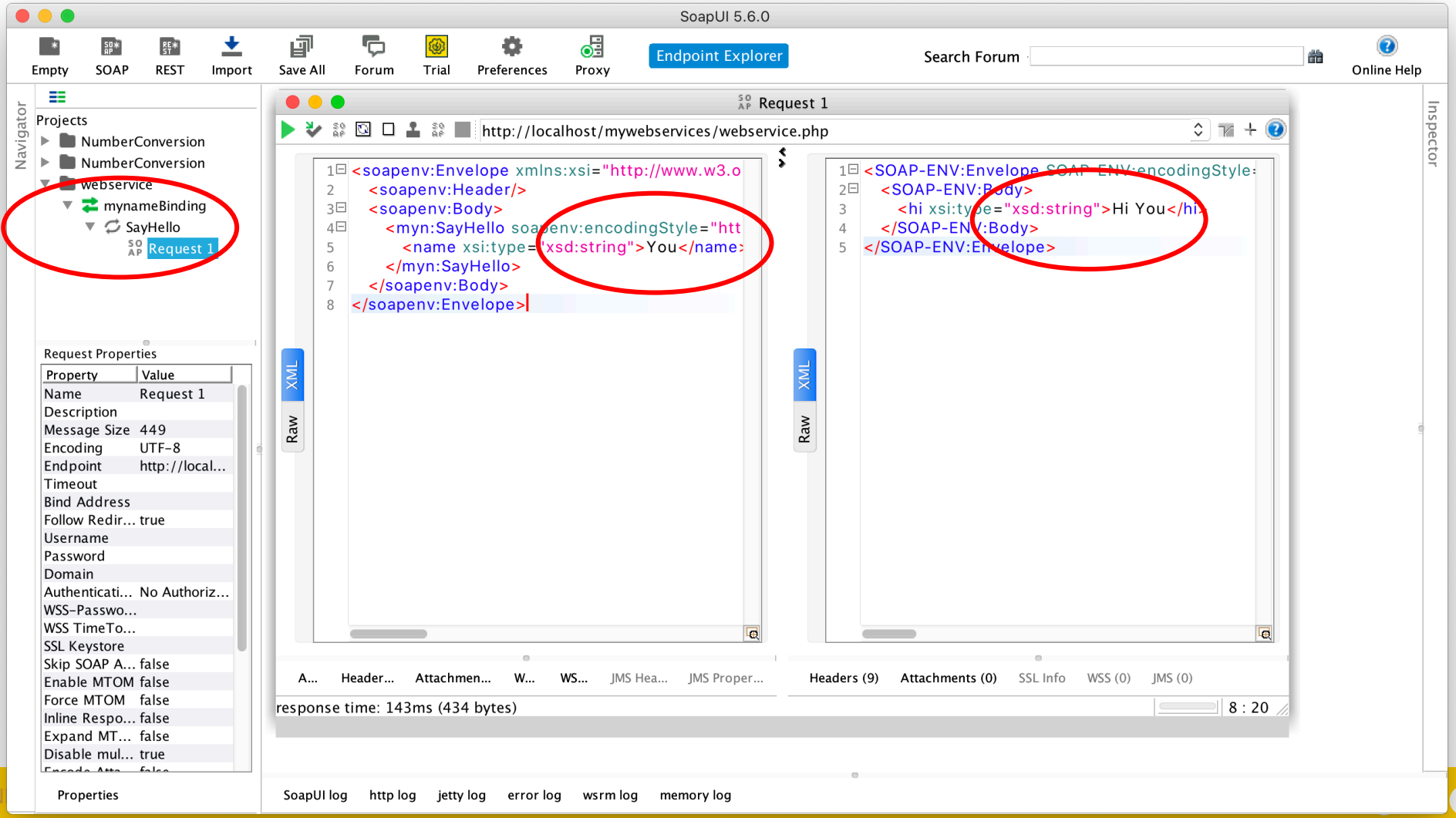


Web service Contract

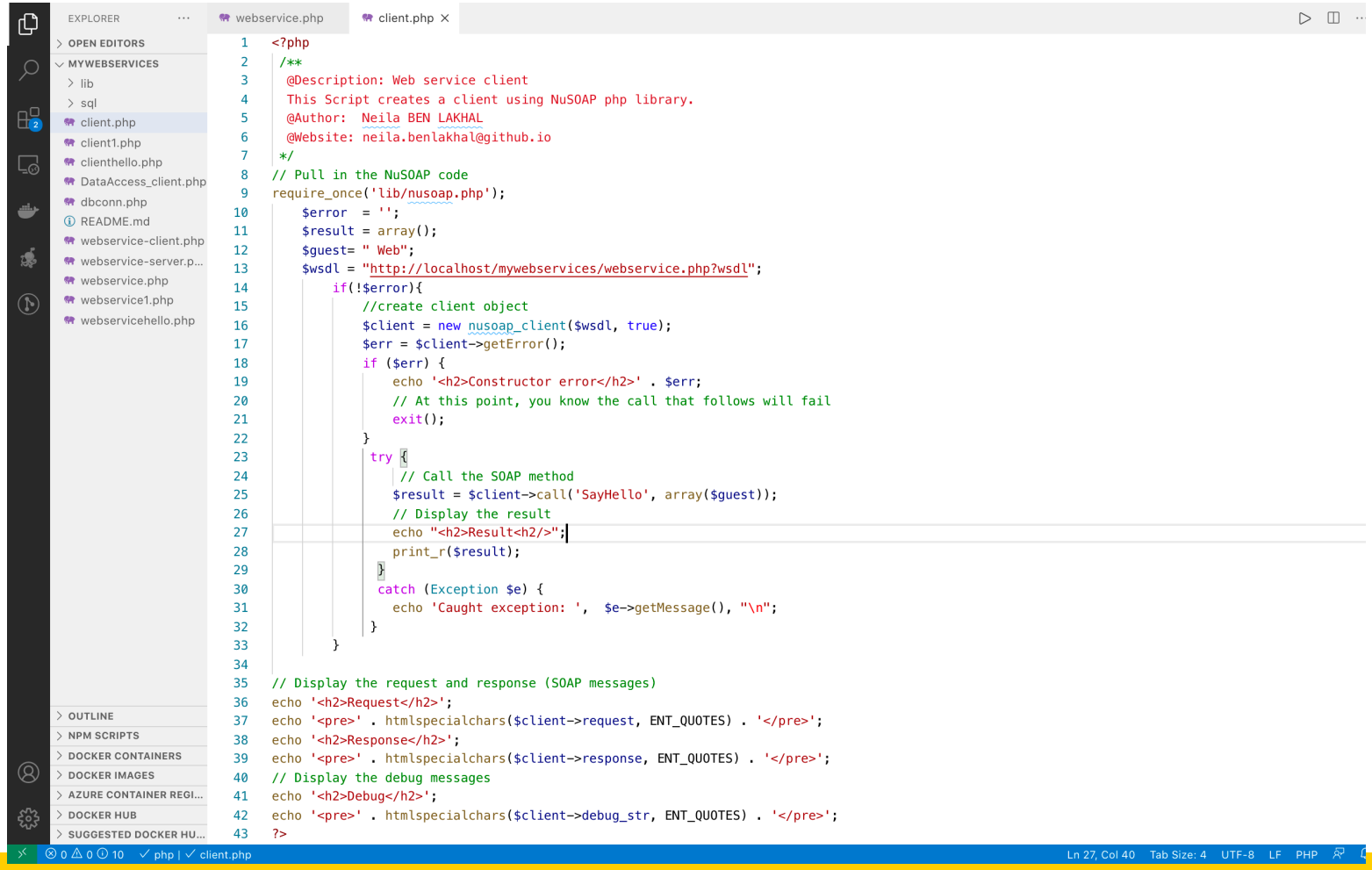
http://localhost/mywebservices/webservice.php?wsdl



Web service Test (SOAPUI)



Web service test (client)



Example 2

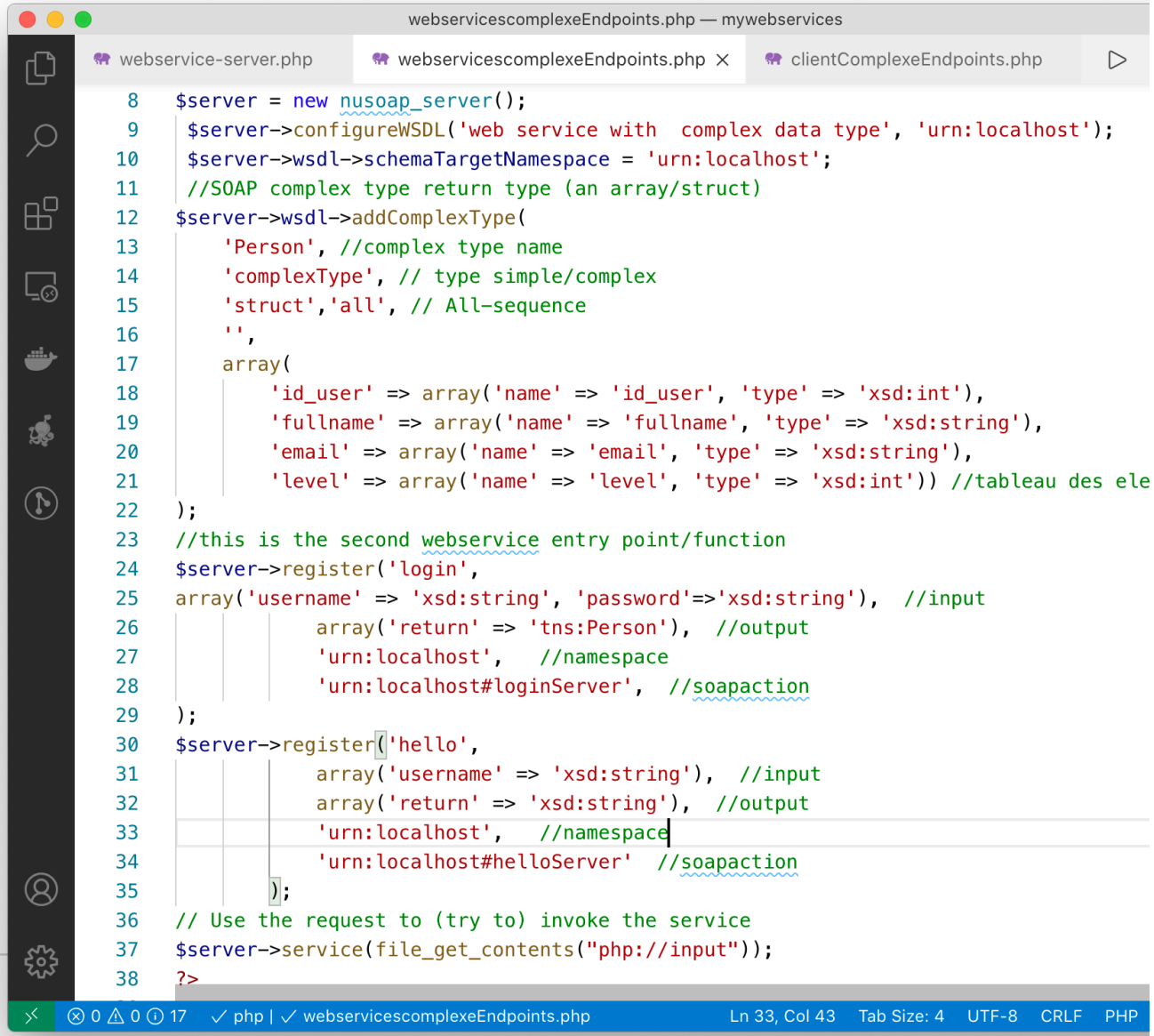
- Web service with complex data type (.xsd)

```php
//(first endpoint)
function hello($username) {
return 'Howdy, '.$username.'!';
}
```

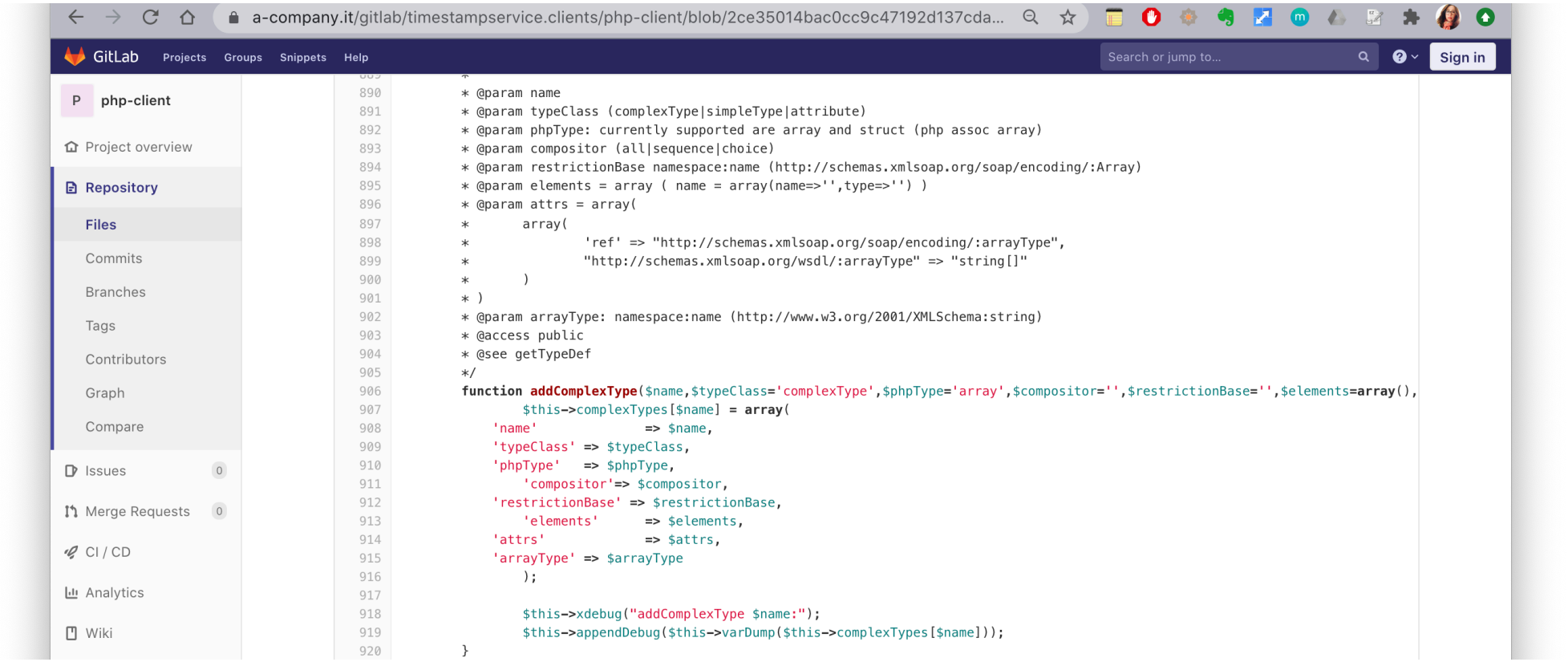
```php
//(second endpoint)
// function with complex XML (data type)
function login($username, $password){
return array('id_user'=>1,
'fullname'=>'WEB',
'email'=>'web@soa.com', 'level'=>100);
}
```

```php
//(first endpoint)
function hello($username) {
return 'Howdy, '.$username.'!';
}
```

```php
//(second endpoint)
// function with complex XML (data type)
function login($username, $password){
return array('id_user'=>1,
'fullname'=>'WEB',
'email'=>'web@soa.com', 'level'=>100);
}
```



AddComplexType method



https://www.a-company.it/gitlab/timestampservice.clients/php-client/blob/2ce35014bac0cc9c47192d137cda61fd9ddac31c/nusoap/class.xmlschema.php

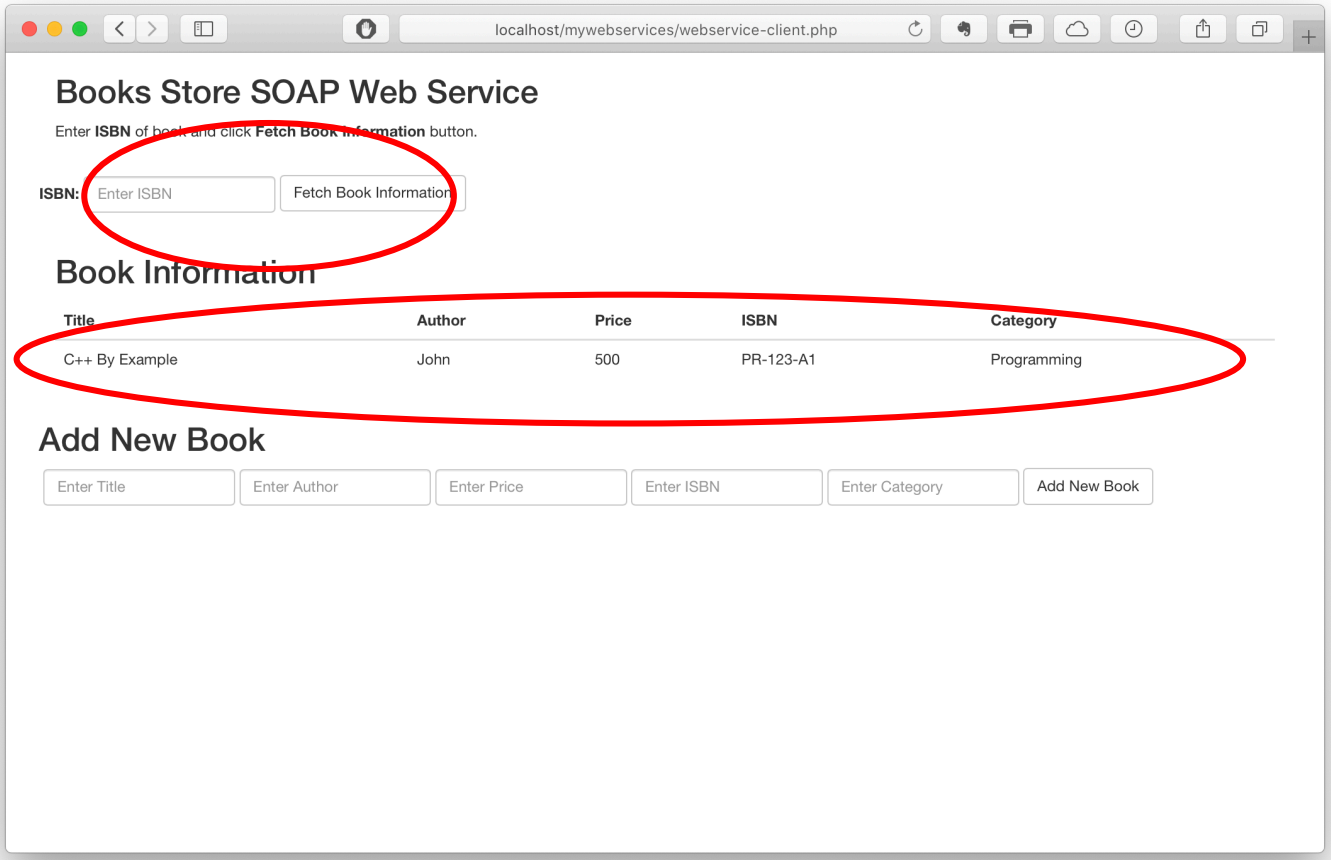
Added complex type in WSDL contract

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<definitions xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tns="urn:localhost"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="urn:localhost">
  ▼<types>
    ▼<xsd:schema targetNamespace="urn:localhost">
       <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
       <xsd:import namespace="http://schemas.xmlsoap.org/wsdl/"/>
      ▼<xsd:complexType name="Person">
        ▼<xsd:all>
           <xsd:element name="id_user" type="xsd:int"/>
           <xsd:element name="fullname" type="xsd:string"/>
           <xsd:element name="email" type="xsd:string"/>
           <xsd:element name="level" type="xsd:int"/>
         </xsd:all>
       </xsd:complexType>
     </xsd:schema>
   </types>
```

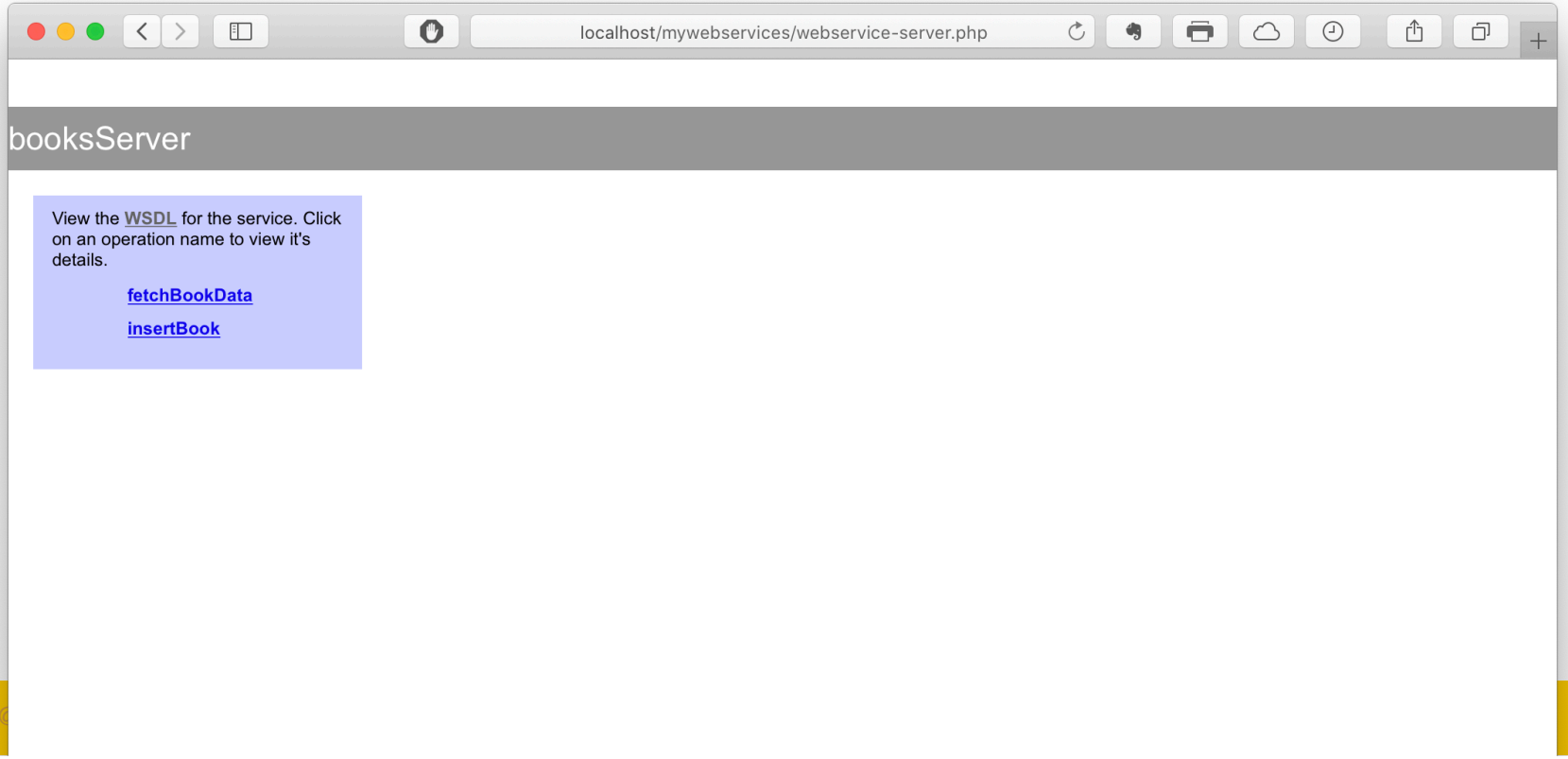
Exemple 3

- Web service call (Web App)



Exemple 3

- Web service :
- http://localhost/mywebservices/webservice-server.php



More examples

- Currently, platforms like Paypal, ebay.. are offering access to their API and support multiple SOAP-based web services:

- NVP/SOAP API (PAYPAL)

  https://developer.paypal.com/docs/nvp-soap-api/PayPalSOAPAPIArchitecture/

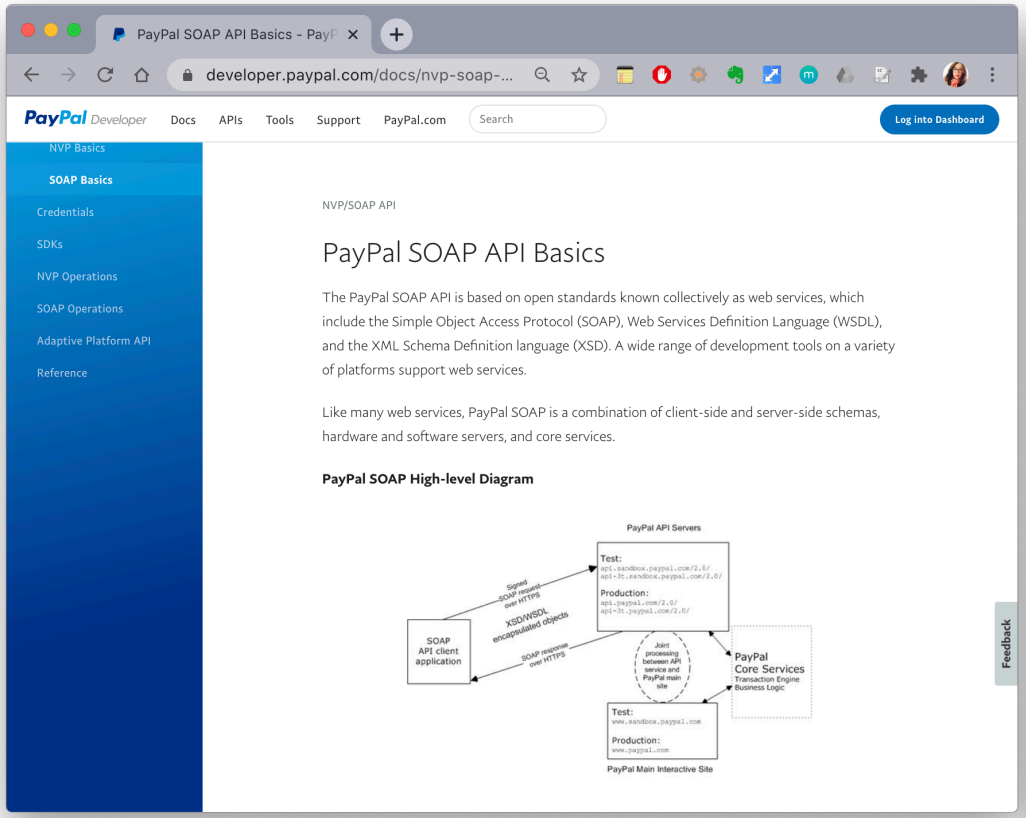
- EBAY

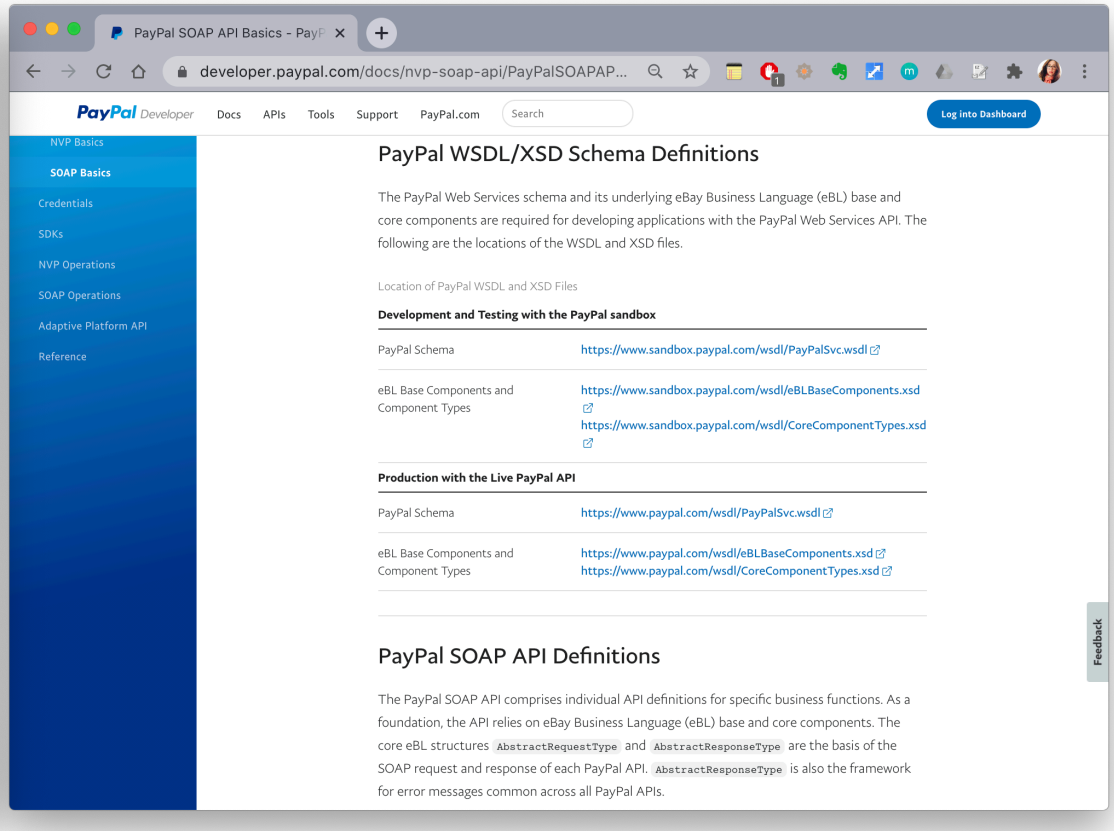
  https://developer.ebay.com/devzone/shopping/docs/CallRef/GetItemStatus.html#Samples

- FLIGHT STAT

  https://developer.flightstats.com/api-docs

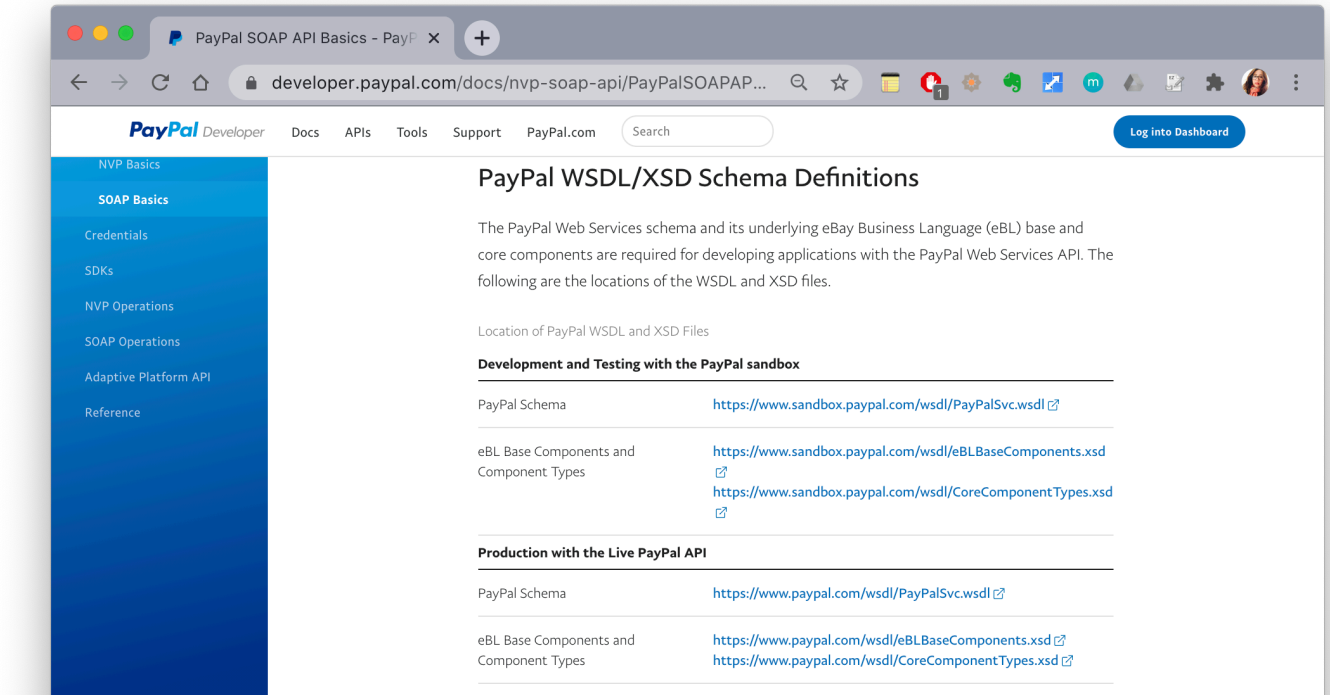
https://developer.paypal.com/docs/nvp-soap-api/PayPalSOAPAPIArchitecture/



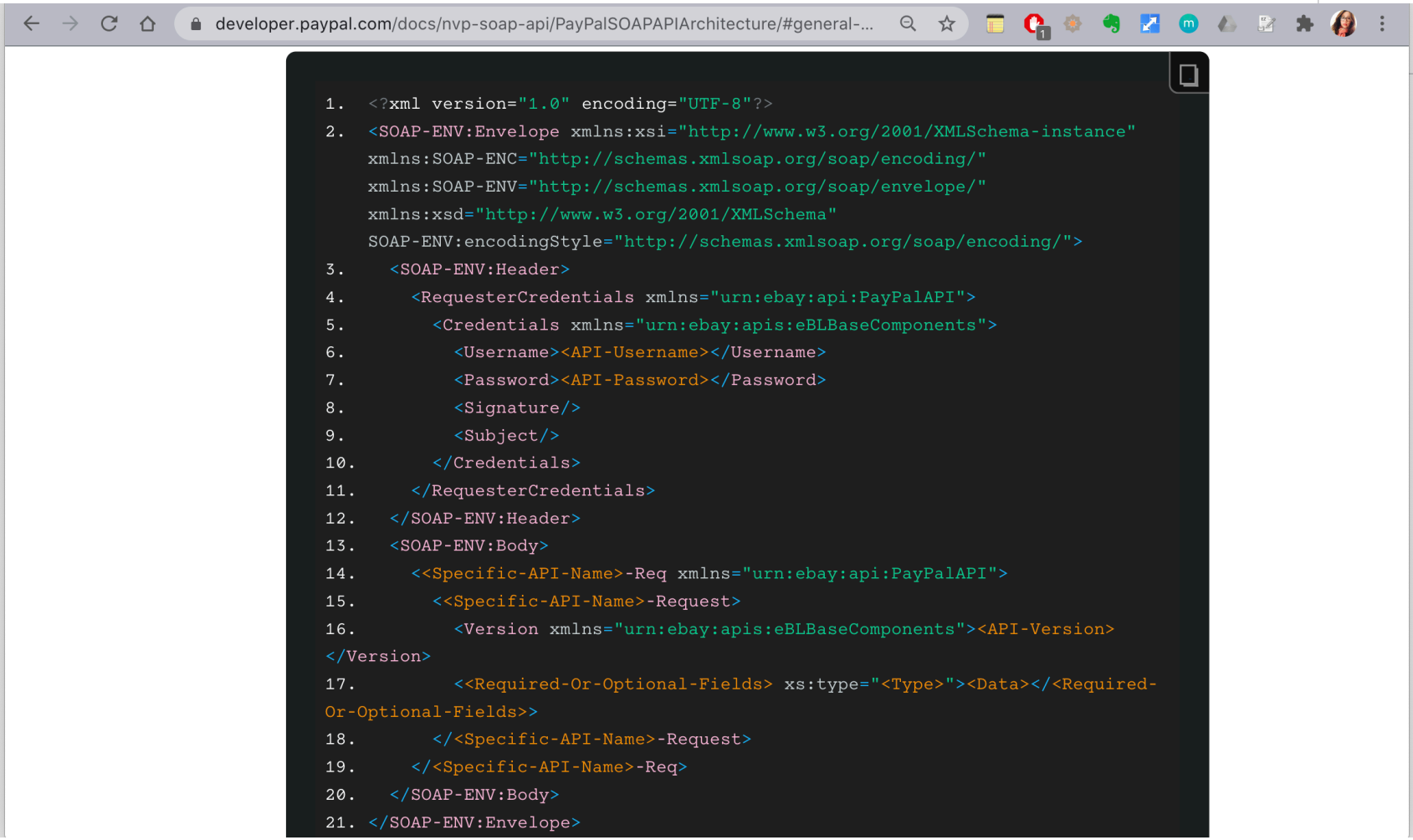


PAYPAL Service Access

https://www.paypalobjects.com/wsdl/paypalsvc.**wsdl**



PayPal API SOAP Request




```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Header>
    <RequesterCredentials xmlns="urn:ebay:api:PayPalAPI">
      <Credentials xmlns="urn:ebay:apis:eBLBaseComponents">
        <Username><API-Username></Username>
        <Password><API-Password></Password>
        <Signature/>
        <Subject/>
      </Credentials>
    </RequesterCredentials>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <<Specific-API-Name>-Req xmlns="urn:ebay:api:PayPalAPI">
      <<Specific-API-Name>-Request>
        <Version xmlns="urn:ebay:apis:eBLBaseComponents"><API-Version>
</Version>
        <<Required-Or-Optional-Fields> xs:type="<Type>"><Data></<Required-
Or-Optional-Fields>>
      </<Specific-API-Name>-Request>
    </<Specific-API-Name>-Req>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

FLIGHT STAT API

- https://developer.flightstats.com/api-docs

Flex API Quick Reference

Aircraft

The *Aircraft* API provides extensive information on specific aircraft.

Base URI `https://api.flightstats.com/flex/aircraft/rest/`
Version `v1`
WADL `https://api.flightstats.com/flex/aircraft/rest/v1/schema/wadl`
JSON `https://api.flightstats.com/flex/aircraft/rest/v1/schema/json`

Entry Points	Name	Description	Response Elements
aircraft/	Aircraft details	Retrieve details about an aircraft	aircraft
aircraft/search/	Aircraft details search	Retrieve details about a matching aircraft	aircraft
availableFields/	User fields	List the fields the user is allowed to receive	field list
allFields/	All fields	List all possible fields.	field list
company/search/	Company search	List the companies matching the search term	companies
fleet/	Company aircraft	List *Aircraft* belonging to a company.	company aircraft

Airlines

The *Airlines* API provides basic reference information about one or more airlines.

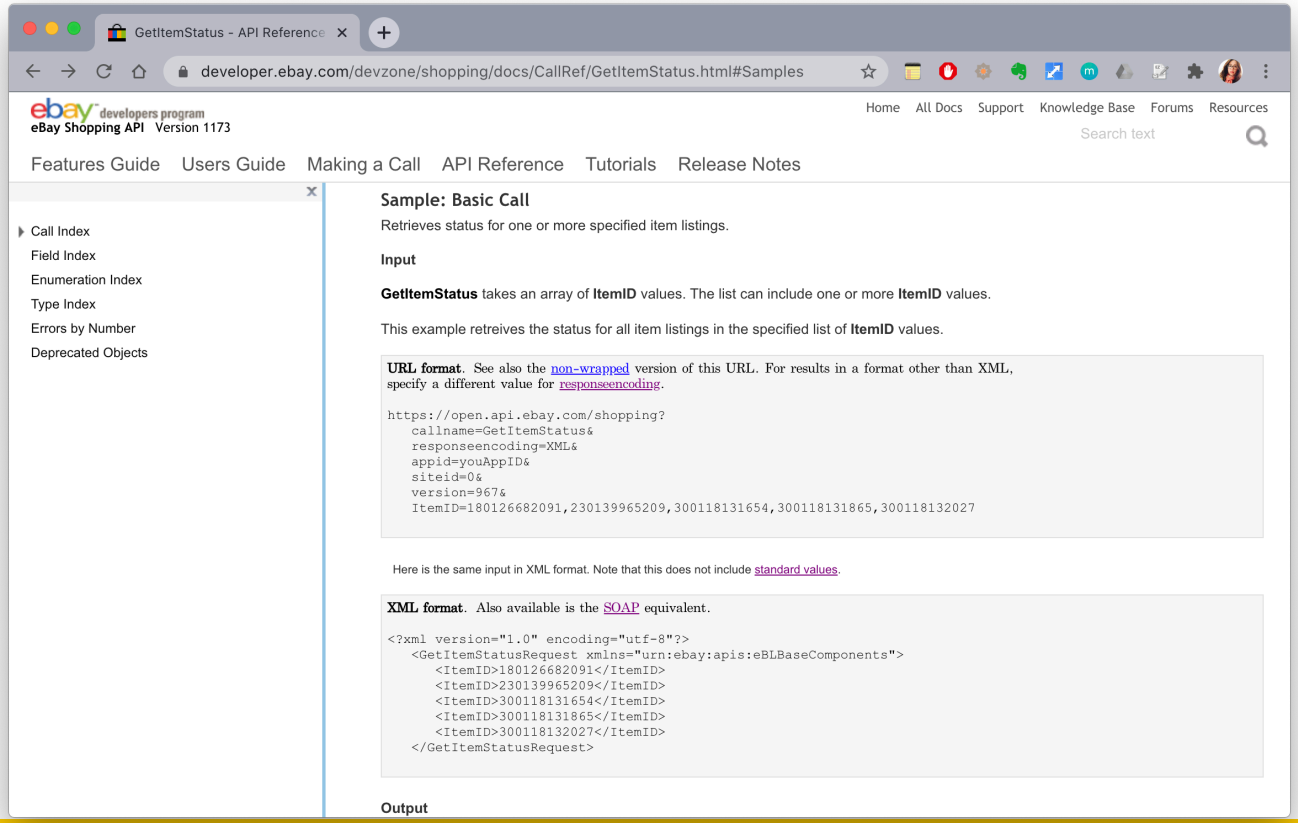
Base URI `https://api.flightstats.com/flex/airlines/rest/`
Version `v1`
WADL `https://api.flightstats.com/flex/airlines/rest/v1/schema/wadl`
WSDL `https://api.flightstats.com/flex/airlines/soap/v1/airlinesService?wsdl`
XSD `https://api.flightstats.com/flex/airlines/rest/v1/schema/xsd`
JSON `https://api.flightstats.com/flex/airlines/rest/v1/schema/json`

Entry Points	Name	Description	Response Elements
active/	Active airlines	*Airline* information for active airlines	airline
all/	All airlines	*Airline* information for all airlines, active and inactive.	airlines
fs/	Airlines by Cirium code	*Airline* information for airlines selected by Cirium code and optionally date.	airline
iata/	Airlines by IATA code	*Airline* information for airlines selected by IATA code and optionally date.	airline
icao/	Airlines by ICAO code	*Airline* information for airlines selected by ICAO code and optionally date.	airline

Airports

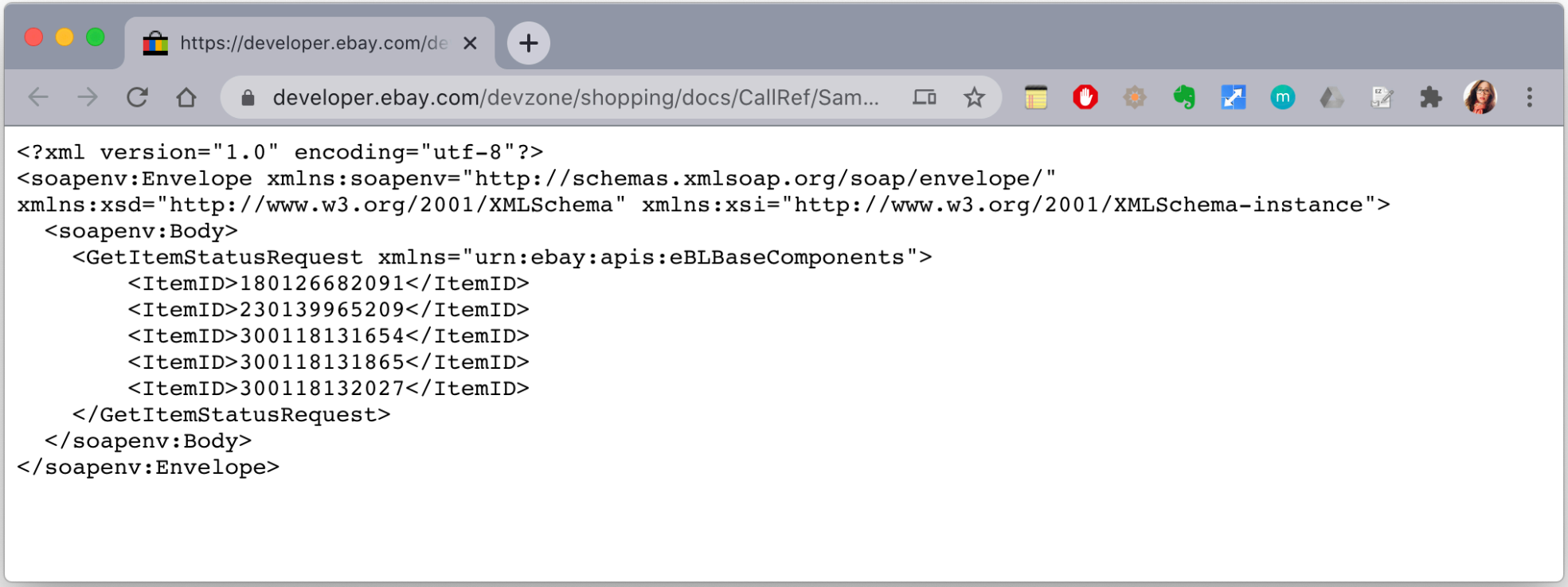
EBAY API

- https://developer.ebay.com/devzone/shopping/docs/CallRef/GetItemStatus.html#Samples



EBAY API

- https://developer.ebay.com/devzone/shopping/docs/callref/samples/getitemstatus_basic_in_soap_soap.txt




```
<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <GetItemStatusRequest xmlns="urn:ebay:apis:eBLBaseComponents">
        <ItemID>180126682091</ItemID>
        <ItemID>230139965209</ItemID>
        <ItemID>300118131654</ItemID>
        <ItemID>300118131865</ItemID>
        <ItemID>300118132027</ItemID>
    </GetItemStatusRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

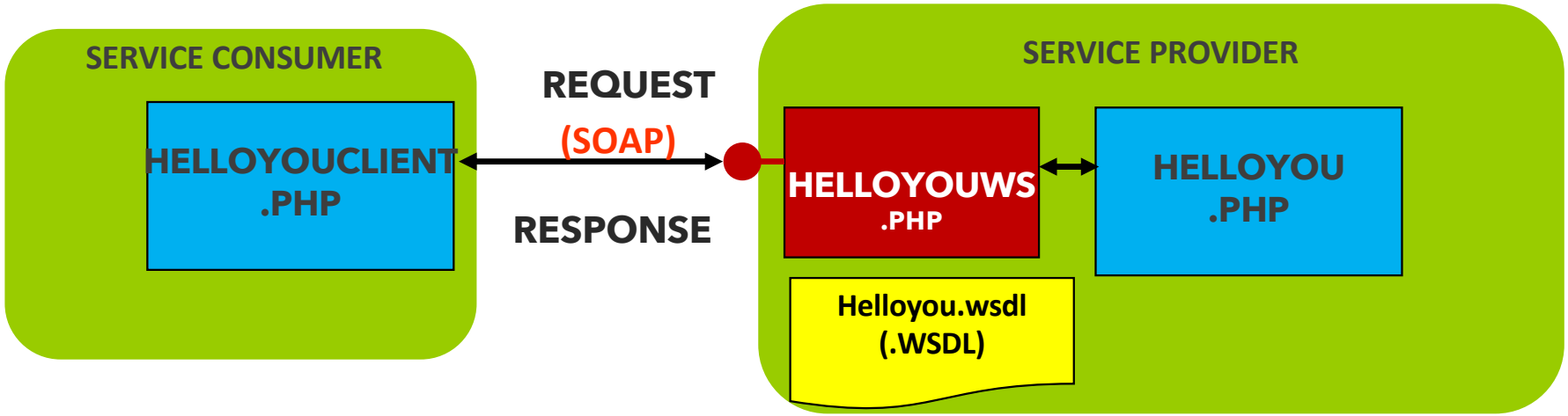
Contract first (Top-down) approach

Top-down approach

- Unlike the bottom-up approach used to migrate to SOA architecture (already implemented IS), where a service composition approach is followed, the top-down approach is used to implement a business process (BP) through a decomposition of the BP and the identification of the different services that compose it.

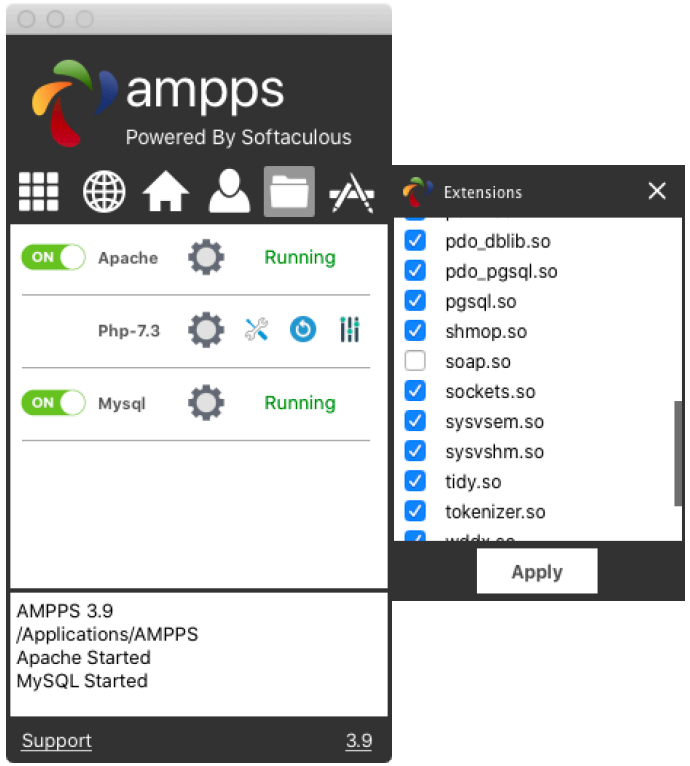
Implementing service

- Steps :
  - Step 1: write App code Helloyou (.php: our App)
  - Step 2: write contract Helloyou.wsdl (contract)
  - Step 3: write Web service **`HelloYouWS.php`** (.php)
  - Step 4 : write Client **`HelloyouClient.php`** (.php) or any other programming language.



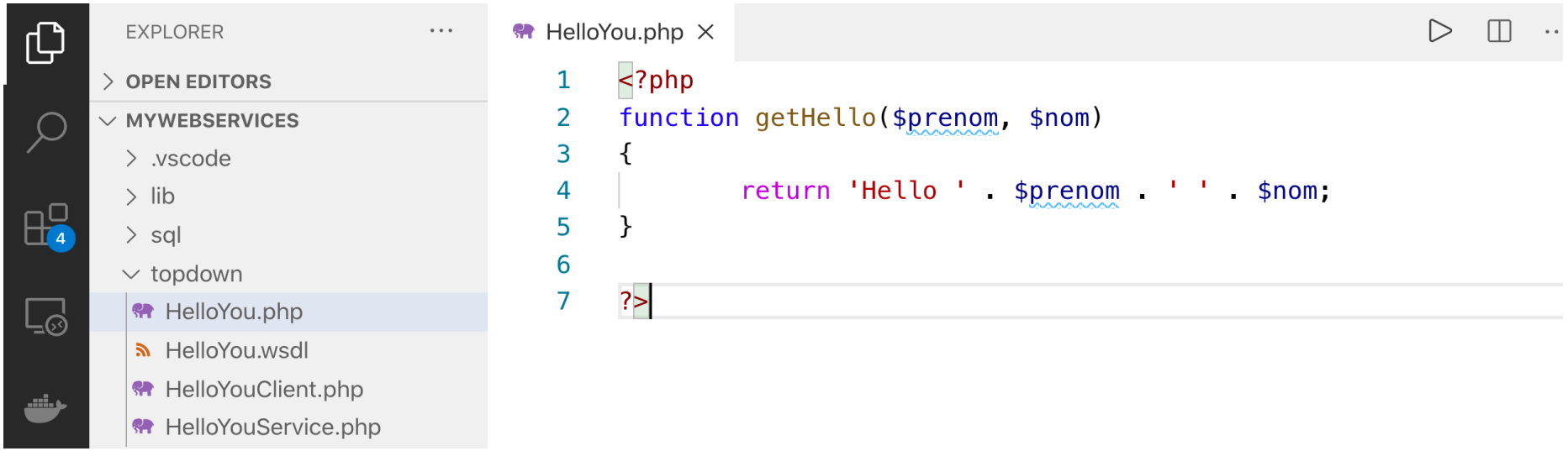
Implementing top-down service

- All programming langages have APIs to implement web services in Top-down approaches
- For php, we will be using : **soap.dll**, an extension to php
- In Ampps, you need to activate it.



Implementing top-down service

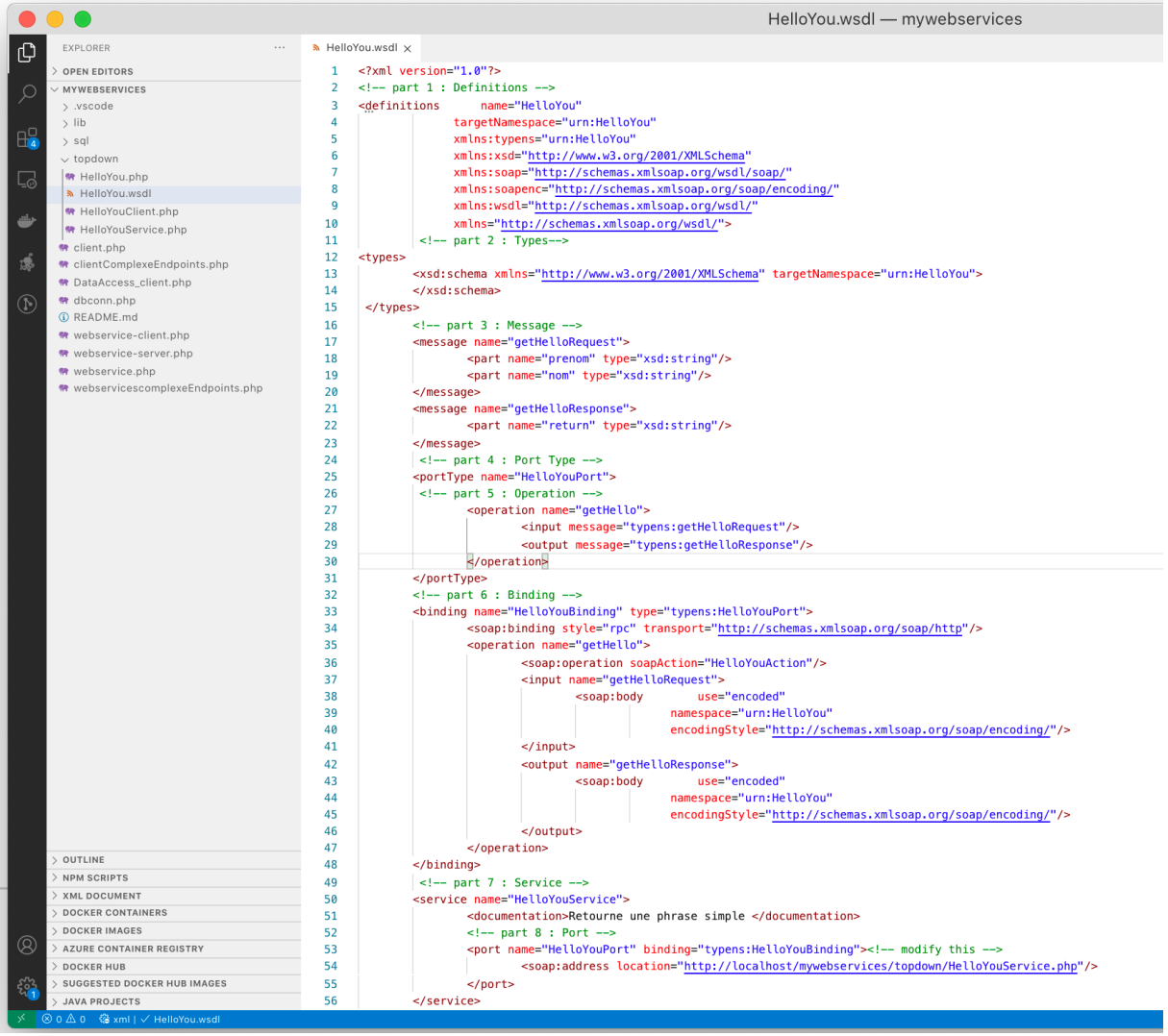
- Step 1: write App code



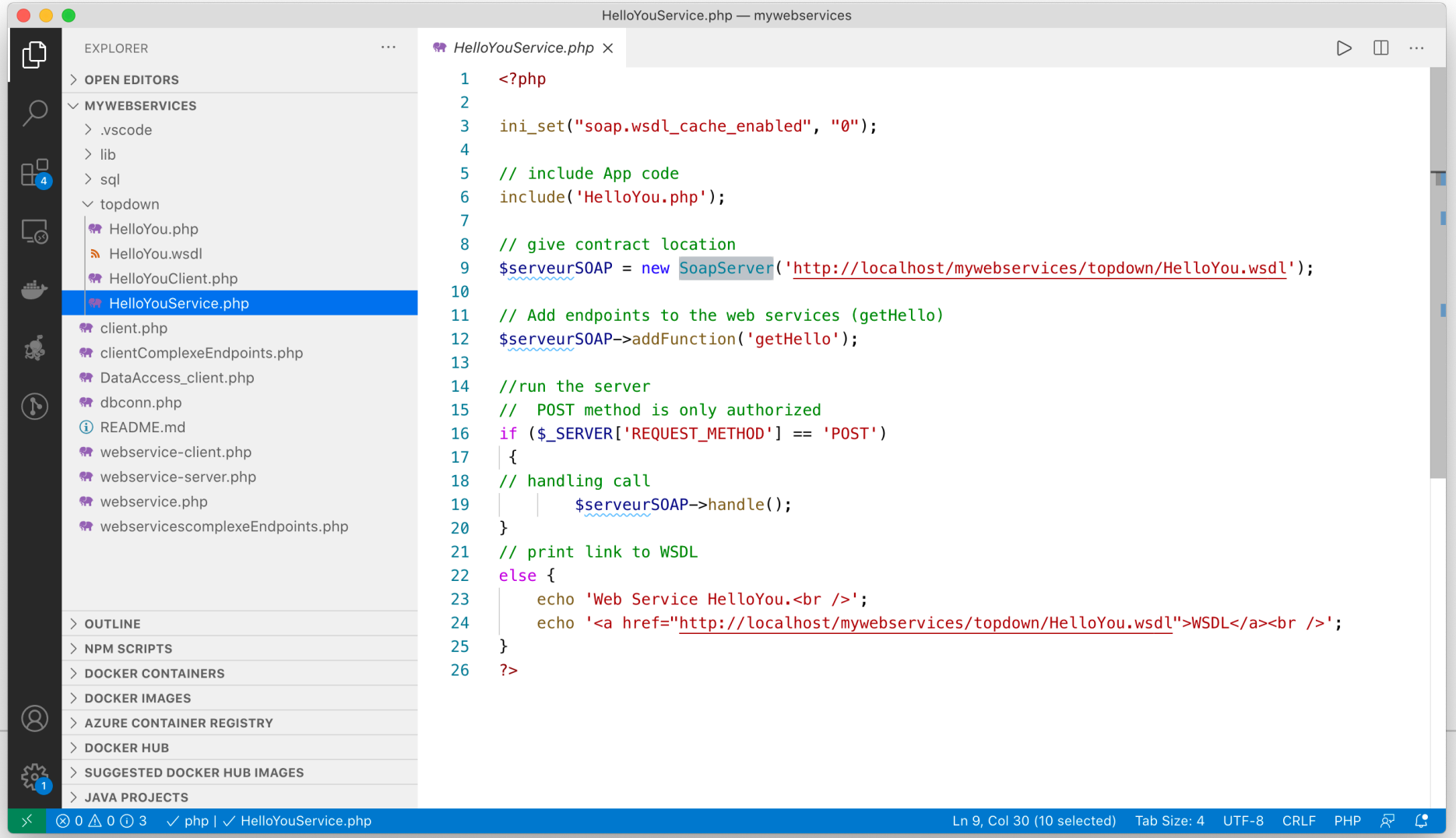
Step 2 : write (generate contract)

http://localhost/mywebservices/topdown/HelloYou.wsdl

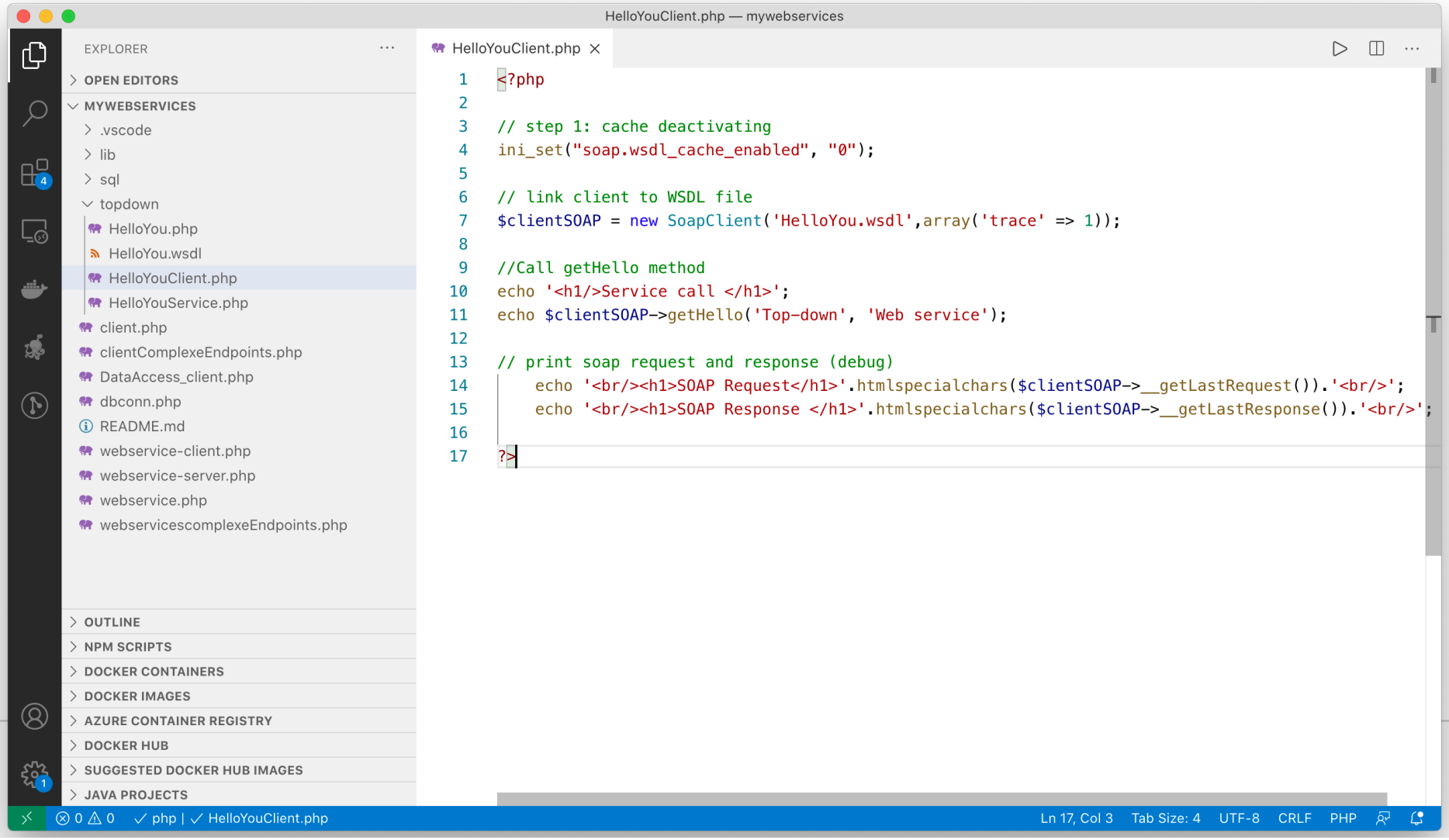
- This approach requires .wsdl (contract file)
- As we have App code in .php, we can use NUSOAP (bottom-up approach) to generate .wsdl file.



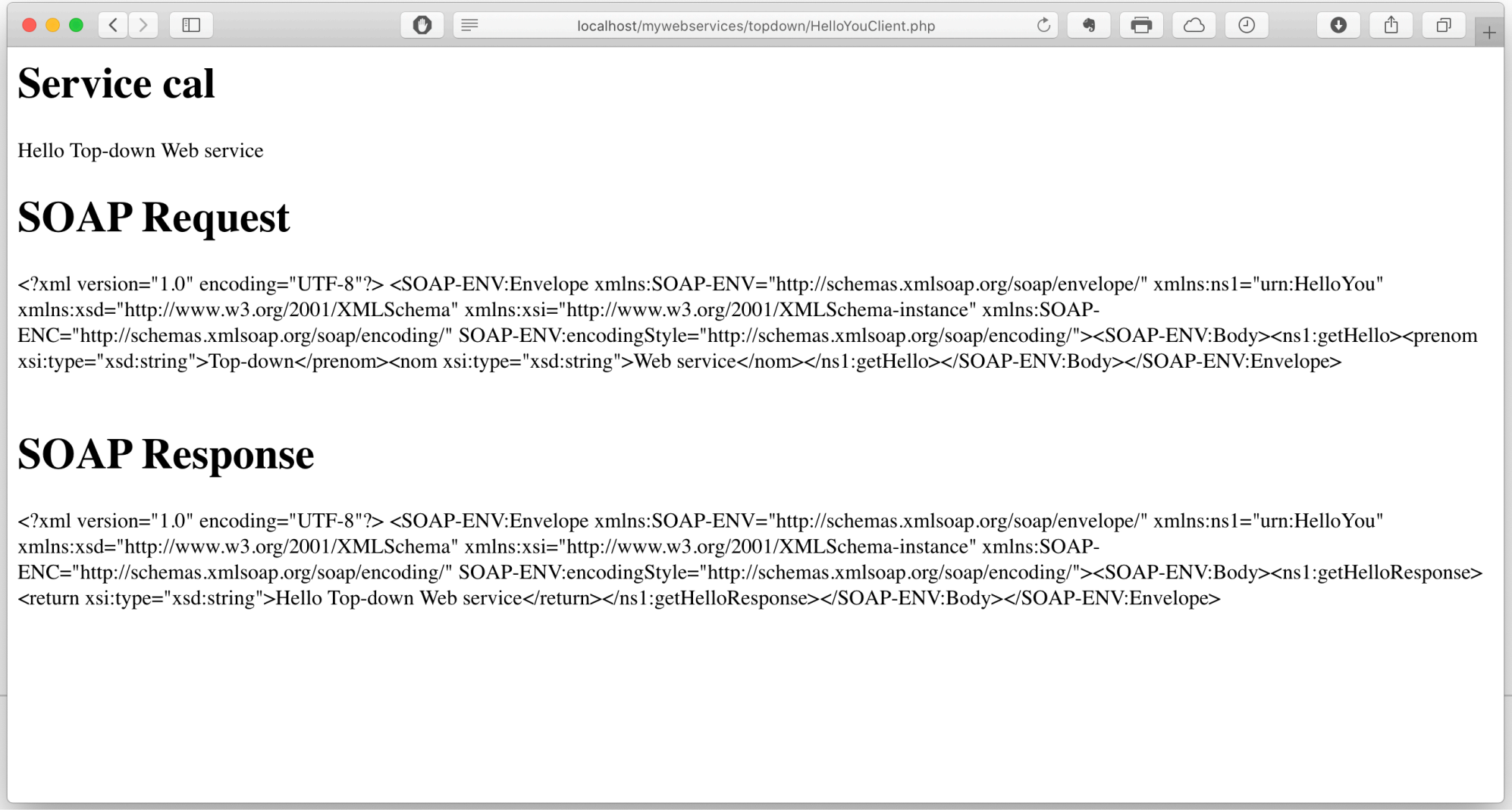
Step 3: write web service



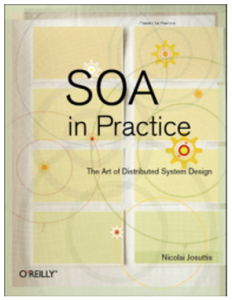
Step 4: Test service (write client)



Running Service client



Nicolai M. Josuttis: SOA in Practice



Nicolai M. Josuttis

SOA in Practice
The Art of Distributed System Design

O'Reilly

ISBN-10: 0-596-52955-4
ISBN-13: 978-0-596-52955-0

Bottom-up vs. Top-down

- In the *top-down* approach, you decompose a problem, system, or process into smaller chunks until you reach the level of (basic) services.
- In the *bottom-up* approach, you build business processes by composing services into more general chunks.
- All experts agree that in practice, a pure application of either of these approaches does not work. Of course, you can design your processes from the top down, which will help you to understand what is needed and what might be a “natural” separation of activities. How- ever, ignoring existing realities might result in high costs compared with an approach that also takes the existing bottom layer into account. On the other hand, designing business processes from the bottom up introduces the danger of proposing technical details and constraints to very high levels, making the processes inflexible and unintuitive.
- [BloombergSchmelzer06] puts it as follows:

  Your approach to SOA should be both top-down (through process decomposition) and bottom-up (exposing existing functionality as Services and composing them into pro- cesses). If you take only a top-down approach, you’re likely to recommend building Services that are technically difficult or complex to implement. Taking solely a bottom-up approach can yield Services you don’t need or, even worse, Services that don’t fulfil the requirements of the business.
- So, in practice, a mixture of both approaches is appropriate. (There are many different names for such a mixed approach, such as “middle-out,” “middle-ground,” “meet in the middle,” or just “agile.”)
- Usually, your aim should be to fulfil a real business need. That is, a new or modified business process or a new functionality should come into play. However, you should also respect what you already have (adapting IT reality).

WSDL

Web Service Description Language

Web Service Description Language

- Initiative of Ariba, IBM and Microsoft
- Standard by W3C

  WSDL 1.1, (2001) : http://www.w3.org/TR/wsdl

  WSDL 2.0 (2007) : http://www.w3.org/TR/wsdl20/

WSDL ?

- Is an **XML-based** specification **schema** for describing the **public** interface of a Web service.

- It is through this service description that the **service provider** communicates all the specifications for invoking a particular WS to the **service requestor.**

- WSDL is used to describe precisely:
  - ***What*** a service does, i.e., the **operations** the service provides,
  - ***Where*** it resides, i.e., details of the **protocol-specific address**, e.g., a URL, and
  - ***How*** to invoke it, i.e., details of the **data formats** and **protocols** necessary to access the service's operations.

WSDL content 1/2

◉The WSDL specification can be conveniently divided into 2 parts. This enables each part to be defined separately and independently and reused by other parts :

1. **Service interface definition** (**abstract** interface)
   - This contains all the **operations** supported by the service, the operation **parameters**, and abstract **data types**.
2. **Service implementation** (**concrete** implementation)
   - This part binds the abstract interface to a concrete **network address**, to a **specific protocol**, and to concrete **data structures.**

WSDL content 2/2

◉This public interface includes **operational** description of a Web service that encompasses:

- All publicly available **operations**,
- The XML **message** protocols supported by the Web service,
- **Data type** information for messages,
- **Binding** information about the specific **transport protocol** to be used, and
- **Address** information for **locating** the Web service.

WSDL structure

- WSDL is layered on top of the **XML schema**. It is stored in **.wsdl** file.
- <definitions> is the root element.
- Abstract interface describes ,messages, operations, and port types in a platform and language independent manner using <types>, <message>, <part>, <portType>, and <operation> elements.
- Concrete implementation part of WSDL contains the elements <binding> <port>, and <service> and describes how a particular service interface is implemented by a given service provider.

```
<definitions>
        <types>… </types>
        <message>… </message>
        <portType>…</portType>
        <binding>…</binding>
        <service>…>/service>
</definitions>
```

Several WSDL documents

- Google Ads Platform
  - https://developers.google.com/adwords/api/docs/guides/call-structure
  - https://adwords.google.com/api/adwords/cm/v201809/CampaignService?wsdl

- EbaY
  - http://developer.ebay.com
  - http://developer.ebay.com/webservices/latest/ebaySvc.wsdl
- PayPAL
  - https://www.paypalobjects.com/wsdl/paypalsvc.wsdl

- National Digital Forecast Database (NDFD)
  - http://www.nws.noaa.gov/xml/
  - http://www.weather.gov/forecasts/xml/DWMLgen/wsdl/ndfdXML.wsdl

- UPS Developer Kit APIs
  - https://www.ups.com/cd/en/help-center/technology-support/developer-resource-center/ups-developer-kit/about.page?

- Flightaware
  - http://flightxml.flightaware.com/soap/FlightXML3/wsdl
  - https://flightaware.com/commercial/flightxml/v3/documentation.rvt

webservice.wsdl

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- part 1 : Definitions -->
<definitions xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
             xmlns:tns="http://www.mynamespace.com"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
             xmlns="http://schemas.xmlsoap.org/wsdl/"
             targetNamespace="http://www.mynamespace.com">
  <!-- part 2 : Types-->
  <types>
    <xsd:schema targetNamespace="http://www.mynamespace.com">
      <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
      <xsd:import namespace="http://schemas.xmlsoap.org/wsdl/" />
    </xsd:schema>
  </types>
  <!-- part 3 : Message -->
  <message name="SayHelloRequest">
    <part name="name" type="xsd:string" />
  </message>
  <message name="SayHelloResponse">
    <part name="hi" type="xsd:string" />
  </message>
  <!-- part 4 : Port Type -->
  <portType name="mynamePortType">
    <operation name="SayHello">
      <input message="tns:SayHelloRequest" />
      <output message="tns:SayHelloResponse" />
    </operation>
  </portType>
```

Abstract part

webservice.wsdl

```xml
  <!-- part 5 : Binding -->
  <binding name="mynameBinding" type="tns:mynamePortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="SayHello">
      <soap:operation soapAction="http://www.mynamespace.com#SayHello" style="rpc" />
      <input>
        <soap:body use="encoded"
                   namespace="http://www.mynamespace.com"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
      <output>
        <soap:body use="encoded"
                   namespace="http://www.mynamespace.com"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
  </binding>
  <!-- part 6 : Service -->
  <service name="myname">
    <!-- part 7 : Port -->
    <port name="mynamePort" binding="tns:mynameBinding">
      <soap:address location="http://localhost/mywebservices/webservice.php" />
    </port>
  </service>
</definitions>
```

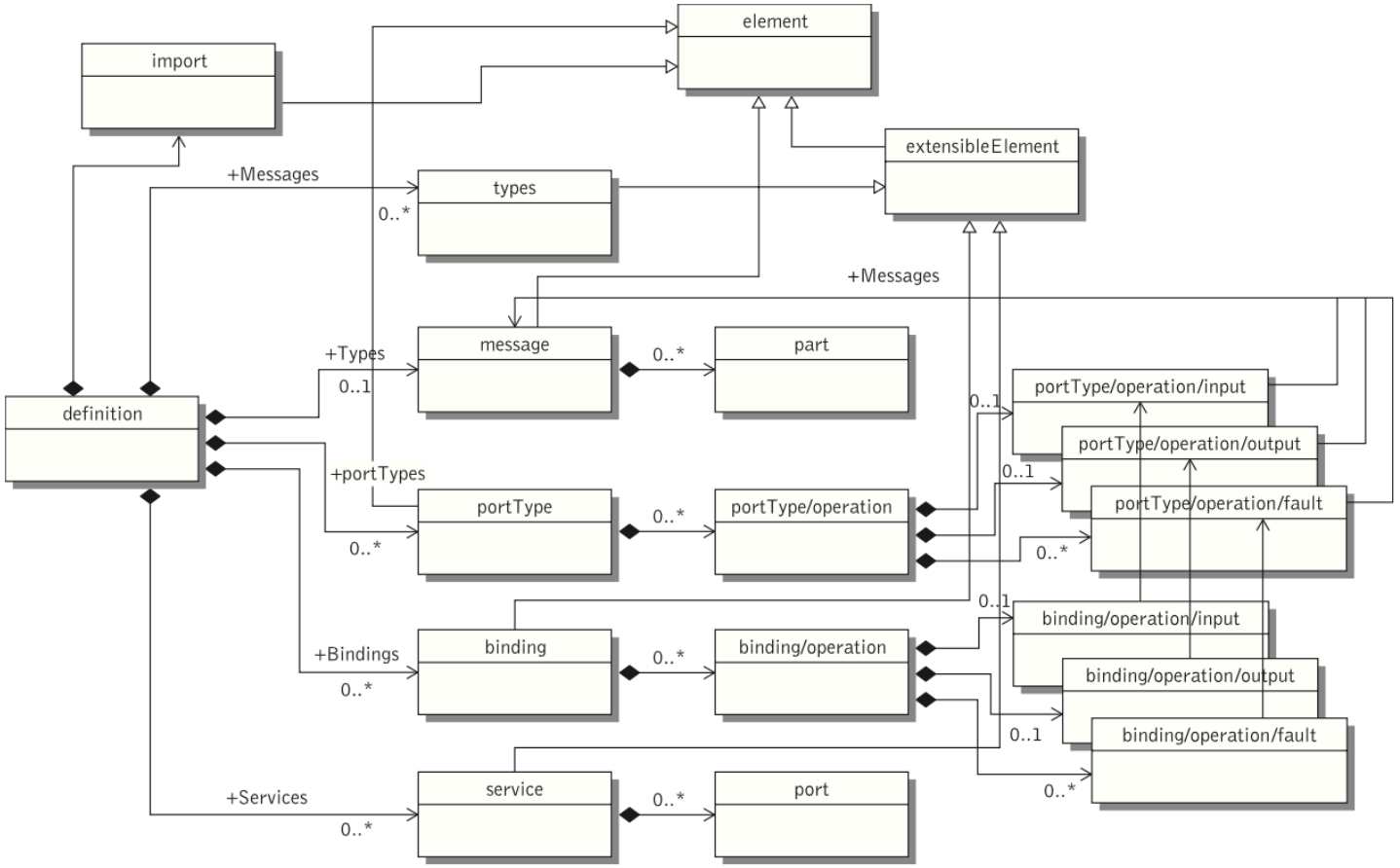
Concrete part

WSDL sample :

http://www.dneonline.com/calculator.asmx?wsdl

```
▼<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:tns="http://tempuri.org/"
 xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://tempuri.org/">
  ▼<wsdl:types>
    ▶<s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">...</s:schema>
   </wsdl:types>
  ▶<wsdl:message name="AddSoapIn">...</wsdl:message>
  ▶<wsdl:message name="AddSoapOut">...</wsdl:message>
  ▶<wsdl:message name="SubtractSoapIn">...</wsdl:message>
  ▶<wsdl:message name="SubtractSoapOut">...</wsdl:message>
  ▶<wsdl:message name="MultiplySoapIn">...</wsdl:message>
  ▶<wsdl:message name="MultiplySoapOut">...</wsdl:message>
  ▶<wsdl:message name="DivideSoapIn">...</wsdl:message>
  ▶<wsdl:message name="DivideSoapOut">...</wsdl:message>
  ▶<wsdl:portType name="CalculatorSoap">...</wsdl:portType>
  ▶<wsdl:binding name="CalculatorSoap" type="tns:CalculatorSoap">...</wsdl:binding>
  ▶<wsdl:binding name="CalculatorSoap12" type="tns:CalculatorSoap">...</wsdl:binding>
  ▼<wsdl:service name="Calculator">
    ▶<wsdl:port name="CalculatorSoap" binding="tns:CalculatorSoap">...</wsdl:port>
    ▶<wsdl:port name="CalculatorSoap12" binding="tns:CalculatorSoap12">...</wsdl:port>
   </wsdl:service>
 </wsdl:definitions>
```

WSDL : Overall structure meta-model expressed in UML



<definitions> element

```
<!-- part 1 : Definitions -->
<definitions xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
             xmlns:tns="http://www.mynamespace.com"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
             xmlns="http://schemas.xmlsoap.org/wsdl/"
             targetNamespace="http://www.mynamespace.com">
```

- The **<definitions>** element signals the start of a WSDL document (i.e., the parent for all other WSDL elements) and it contains all relevant, globally declared namespaces.
- The first attribute in the <definitions> element is **name**, which is used to name the entire WSDL document. The **<definitions>** element also declares an attribute called **targetNamespace**, which identifies the logical namespace for elements defined within the WSDL document and characterizes the service.
- Required element (1 instance)

<types> element

```
<!-- part 2 : Types-->
<types>
  <xsd:schema targetNamespace="http://www.mynamespace.com">
    <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
    <xsd:import namespace="http://schemas.xmlsoap.org/wsdl/" />
  </xsd:schema>
</types>
```

- **<types>…</types>** (0..1, only one is authorized)
- Contains all abstract data types that define a WS interface.
- Contains XML schemas or external references to XML schemas that describe the data type definitions.
- Uses the built-in primitive data types such as int, float, long, short, string, boolean, and so on.

<types> element sample

HTTP://WWW.DNEONLINE.COM/CALCULATOR.ASMX?WSDL

```
<wsdl:types>
 <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
  <s:element name="Add">
   <s:complexType>
    <s:sequence>
     <s:element minOccurs="1" maxOccurs="1" name="intA" type="s:int"/>
     <s:element minOccurs="1" maxOccurs="1" name="intB" type="s:int"/>
    </s:sequence>
   </s:complexType>
  </s:element>
  <s:element name="AddResponse">...</s:element>
  <s:element name="Subtract">...</s:element>
  <s:element name="SubtractResponse">...</s:element>
  <s:element name="Multiply">...</s:element>
  <s:element name="MultiplyResponse">...</s:element>
  <s:element name="Divide">...</s:element>
  <s:element name="DivideResponse">...</s:element>
 </s:schema>
</wsdl:types>
```

<message> element

```
<!-- part 3 : Message -->
<message name="SayHelloRequest">
  <part name="name" type="xsd:string" />
</message>
<message name="SayHelloResponse">
  <part name="hi" type="xsd:string" />
</message>
```

- **<message> ...</message>** (1..N)
- The message element declares the form of a message that the WS sends and receives.
- It describes the payload of outgoing and incoming messages.
- It corresponds to a single piece of information moving between the invoker and a WS.
- The message element defines what kind of message is expected as **input**, **output** and **fault** by this WS. Each message is constructed from a number of XML Schema-typed part elements. A message may contain multiple parts **<part>.**

<portType> element

```
<!-- part 4 : PortType -->
<portType name="mynamePortType">
  <operation name="SayHello">
    <input message="tns:SayHelloRequest" />
    <output message="tns:SayHelloResponse" />
  </operation>
</portType>
```

- A WSDL definition can contain 0..N <portType>.
- <portType> contains a named set of operations. It defines the functionality of the Web service. A portType is a way of grouping operations.
- A WSDL <portType> element may have one or more <operation> elements, each of which defines an RPC-style or document-style Web service method.
- Each <operation> element is composed of :

  0..1 <input> element

  0..1 <output> element and

  0..N (any number) <fault> elements.

Operations in WSDL are the equivalent of method signatures in programming languages. They represent the various methods being exposed by the service. An operation defines a method on a Web service, including the name of the method and the input and output parameters. A typical operation defines the input and output parameters or exceptions (faults) of an operation.

<operation> exchange pattern

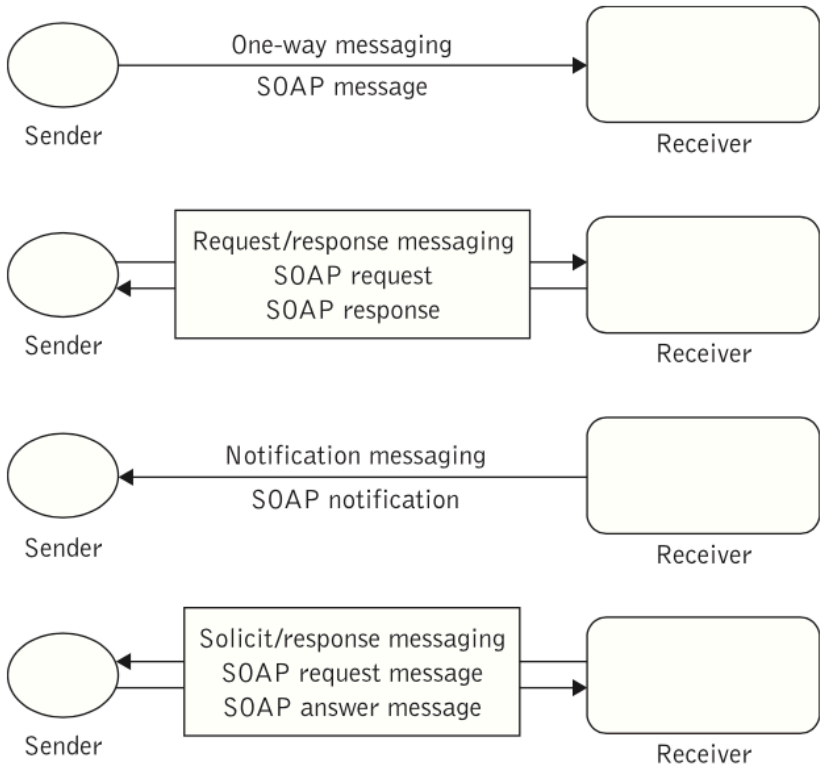
◉The operation element indicates a message exchange pattern:

Synchronous interaction

1. **Request-response** (i.e., Input–Output): The WS receives a message and sends a correlated message (or fault).
2. **Solicit-response** (i.e., Output–Input): The WS generates a message and receives a correlated message (or fault) in return.

Asynchronous interaction

3. **One-way** (i.e., Input only):
4. **Notification** (i.e., Output only): The WS sends a message. It does not expect anything in return.



WSDL messaging patterns

Request-response (i.e., Input–Output)

- The WS receives a message and sends a correlated message (or fault).
- **Example**: a **SendPurchase** operation receives as input a message containing a purchase order (order number and date) and customer details and responds with a message containing an invoice.

```
<!-- portType element describes the abstract interface of a Web
 service -->
<wsdl:portType name="PurchaseOrder_PortType">
     <wsdl:operation name="SendPurchase">
          <wsdl:input message="tns:POMessage"/>
          <wsdl:output message="tns:InvMessage"/>
     </wsdl:operation>
</wsdl:portType>
```

One-way
(i.e., Input only)

- The WS receives a message. The WS consumes the message and does not produce any output or fault message.
- **Example** : an operation representing the submission of an order to a purchasing system. Once the order is sent, no immediate response is expected.

```
<!-- portType element describes the abstract interface of a Web
 service -->
<wsdl:portType name="SubmitPurchaseOrder_PortType">
     <wsdl:operation name="SubmitPurchaseOrder">
          <wsdl:input name="order"
           message="tns:SubmitPurchaseOrder_Message"/>
     </wsdl:operation>
</wsdl:portType>
```

Notification (i.e., Output only)

- The WS sends a message. It does not expect anything in return.
- This type of messaging is used by services that need to **notify** clients of events (*push model).:* The assumption is that the client (subscriber) has registered with the WS to receive messages (notifications) about an event.
- **Example** : a service model in which events are reported to the service and where the endpoint periodically reports its status, for instance subscribing to a flight status.

```
<wsdl:definitions .... >
    <wsdl:portType .... > *
        <wsdl:operation name="nmtoken">
            <wsdl:output name="nmtoken"? message="qname"/>
        </wsdl:operation>
    </wsdl:portType >
</wsdl:definitions>
```

Solicit-response (i.e., Output–Input)

- The WS generates a message and receives a correlated message (or fault) in return.
- the service endpoint is initiating the operation (soliciting the client), rather than responding to a request.
- It is like notification messaging, except that the client is expected to respond to the WS. As with notification messaging, clients of the solicit/response WS must subscribe to the service in order to receive messages.
- An example of this operation might be a service that sends out order status to a client and receives back a receipt.

```
<wsdl:portType .... >
    <wsdl:operation name="nmtoken" parameterOrder="nmtokens">
       <wsdl:output name="nmtoken"? message="qname"/>
       <wsdl:input name="nmtoken"? message="qname"/>
       <wsdl:fault name="nmtoken"message="qname"/>
    </wsdl:operation>
</wsdl:portType >
```

<binding> element

```
<!-- part 5 : Binding -->
<binding name="mynameBinding" type="tns:mynamePortType">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="SayHello">
    <soap:operation soapAction="http://www.mynamespace.com#SayHello" style="rpc" />
    <input>
      <soap:body use="encoded"
                 namespace="http://www.mynamespace.com"
                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </input>
    <output>
      <soap:body use="encoded"
                 namespace="http://www.mynamespace.com"
                 encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </output>
  </operation>
</binding>
```

- **<binding>** element contains information of how the elements in an **abstract service interface** (<portType> and <operation> elements) are mapped to a **concrete representation:**
  - Protocols (e.g., SOAP or HTTP),
  - **Messaging styles (e.g., RPC or documents styles),**
  - Formatting (encoding)
  - Styles (e.g., literal or SOAP encoding).

<binding> element

- A WSDL definition contains 1..N <binding>.
- Several bindings may represent various implementations of the same <portType> element: If a service supports more than one protocol, then the WSDL <portType> element should include a <binding> for each protocol it supports.

```
▶ <wsdl:portType name="CalculatorSoap">...</wsdl:portType>
▶ <wsdl:binding name="CalculatorSoap" type="tns:CalculatorSoap">...</wsdl:binding>
▶ <wsdl:binding name="CalculatorSoap12" type="tns:CalculatorSoap">...</wsdl:binding>
▼ <wsdl:service name="Calculator">
  ▶ <wsdl:port name="CalculatorSoap" binding="tns:CalculatorSoap">...</wsdl:port>
  ▶ <wsdl:port name="CalculatorSoap12" binding="tns:CalculatorSoap12">...</wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Messaging style

- 2 messaging style are used :
- RPC-based style
  - <soap:binding style='**rpc**' ...>

```
<env:Envelope ...>
      <env:Header> some header </env:Header>
      <env:Body>
         <m:GetProductPrice>
             <product-id> 450R60P </product-id>
         </m:GetProductPrice>
      </env:Body>
</env:Envelope>

<env:Envelope ...>
      <env:Header> some header </env:Header>
      <env:Body>
         <m:GetProductPriceResponse>
             <product-price> 134.32 </product-price
         </m:GetProductPriceResponse>
      </env:Body>
</env:Envelope>
```

- Document-based style

  <soap:binding style='**document**' ...>

```
<env:Envelope ...>
  <env:Header> some header </env:Header>
  <env:Body>
    <po:PurchaseOrder orderDate="2004-12-02">
       <po:from>
        <po:accountName> RightPlastics </po:accountName>
        <po:accountNumber> PSC-0343-02 </po:accountNumber>
       </po:from>
       <po:to>
        <po:supplierName>Plastic Supplies Inc.</po:supplierName>
        <po:supplierAddress>Melbourne</po:supplierAddress>
       </po:to>
       <po:product>
        <po:product-name> injection molder </po:product-name>
        <po:product-model> G-100T </po:product-model>
        <po:quantity> 2 </po:quantity>
       <po:product>
    </po:PurchaseOrder>
  </env:Body>
</env:Envelope>
```

Messaging style : RPC

- <soap:binding style='rpc' ...>

- Stems from the conventional remote procedure call (RPC) style.
  - This style supports **synchronous** request-response interaction.

  How to recognize SOAP RPC-style ?
  - The SOAP <Body> must conform to a structure that indicates the **method** name and contains a set of **parameters**.
  - The response always has '**Response**' appended after the request method.

Messaging style : Document

<soap:binding style='**document**' ...>

- Flexible communication style that provides the best interoperability, used by Modern SOAP engines.

- Described as a message-driven, **asynchronous** communication model.

How to recognize a SOAP document-style ?

- In this style, WSDL definitions must contain XML schema definitions of request and response messages.
- The body of each message is an **XML fragment** as defined in the WSDL document.
- The client sends an entire XML document (e.g., purchase order) as a request to an endpoint, rather than a discrete set of parameters and a service operation name.
- The endpoint on the server side is responsible for forwarding and dispatching the request to the right service operation that can process the message.

Why do we need several <binding> and <port> for a same service ?

Scenario :

- A same WS, could, for instance, contain 3 ports, all of which use the same <PortType> but are bound to :
  - SOAP over HTTP,
  - SOAP over SMTP, and
  - HTTP GET/POST, respectively.
- When many protocols are supported:
  - A client that wishes to interact with the service can choose which protocol it wants to use to communicate with the service
  - The service is more readily accessible on a wider range of platforms:
  - A PC desktop application may use SOAP over HTTP,
  - A mobile application may use HTTP GET/POST

<service> Element

- The <service> element appears at the end of the WSDL document and it points to the specific network endpoint for a binding (i.e., access point of the service)

```
  <!-- part 6 : Service -->
  <service name="myname">
    <!-- part 7 : Port -->
    <port name="mynamePort" binding="tns:mynameBinding">
      <soap:address location="http://localhost/mywebservices/webservice.php" />
    </port>
  </service>
</definitions>
```

<service> Element

- It binds the Web service to a specific network addressable location. It takes the bindings that were declared previously and ties them to one or more <port> elements, each of which represents a Web service.

```
▶ <wsdl:binding name="CalculatorSoap" type="tns:CalculatorSoap">...</wsdl:binding>
▶ <wsdl:binding name="CalculatorSoap12" type="tns:CalculatorSoap">...</wsdl:binding>
▼ <wsdl:service name="Calculator">
  ▼ <wsdl:port name="CalculatorSoap" binding="tns:CalculatorSoap">
      <soap:address location="http://www.dneonline.com/calculator.asmx"/>
    </wsdl:port>
  ▼ <wsdl:port name="CalculatorSoap12" binding="tns:CalculatorSoap12">
      <soap12:address location="http://www.dneonline.com/calculator.asmx"/>
    </wsdl:port>
  </wsdl:service>
```

<service> Element with several ports