Instituto Politécnico de Viana do Castelo

Instituto Politécnico de Viana do Castelo

Escola Superior de Tecnologia e Gestão

GRADUATION IN INFORMATICS ENGINEERING

Practical Work nº1

Parsing of XML Files using REST and SOAP via JAVA, PHP and PYTHON

- **Course Unit: Integração de Sistemas**

Eduardo Eiras Nº 21484

- **Coordinator: Prof. Doctor Jorge Ribeiro**

Instituto Politécnico de Viana do Castelo

Escola Superior de Tecnologia e Gestão

www.ipvc.pt

Summary

Summary

Summary

1. Introduction and Objectives

The present document demonstrates the work done, following the proposal of the first practical work which includes, in the first part, the execution of tutorial number 8, using different parsers with different XML files, both in structure and size, implementing concepts already learnt from the execution of tutorial number 7, as well as the implementation of SOAP and REST web services using the client/server approach and different XML files. Additionally, the implementation of web services in tutorial number 8 is done in two different languages which are JAVA and PHP.

The second part of this practical work, aims to approach and use XPATH syntax to filter data in XML files, creating web services in Python and JavaScript, accessing a MySQL database using Python, uploading big XML files to a server and publish a java project to a Tomcat server.

2. Parsers XML in JAVA

There are several different XML parsers, these can be used to de-serialize XML files that serve as their input. This section demonstrates the use of five different parsers those being:

- DOM;
- SAX
- JDOM
- JAXP
- JAXB

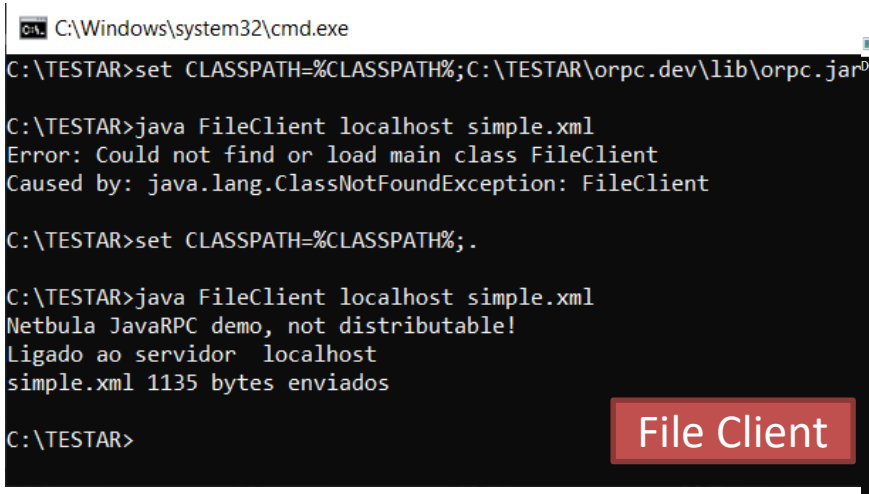
These will be used to de-serialize the input of three different sized files sent from a client to a server. The demonstration of this execution will be done locally, using four CMD windows, one running the server, the second running the JavaRPC, the third the client and the fourth the output of the file received and parsed by the server. The window running the JavaRPC is only going to be presented in the first test, the reason for this is that this window's content is the same for all the tests and it is necessary to be always running to ensure that the connection between the server and the client is successful.

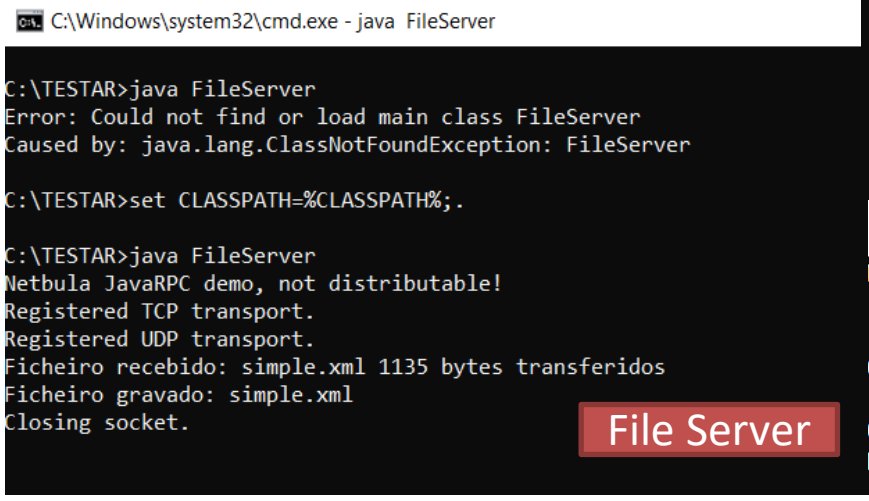
2. Parsers XML in JAVA

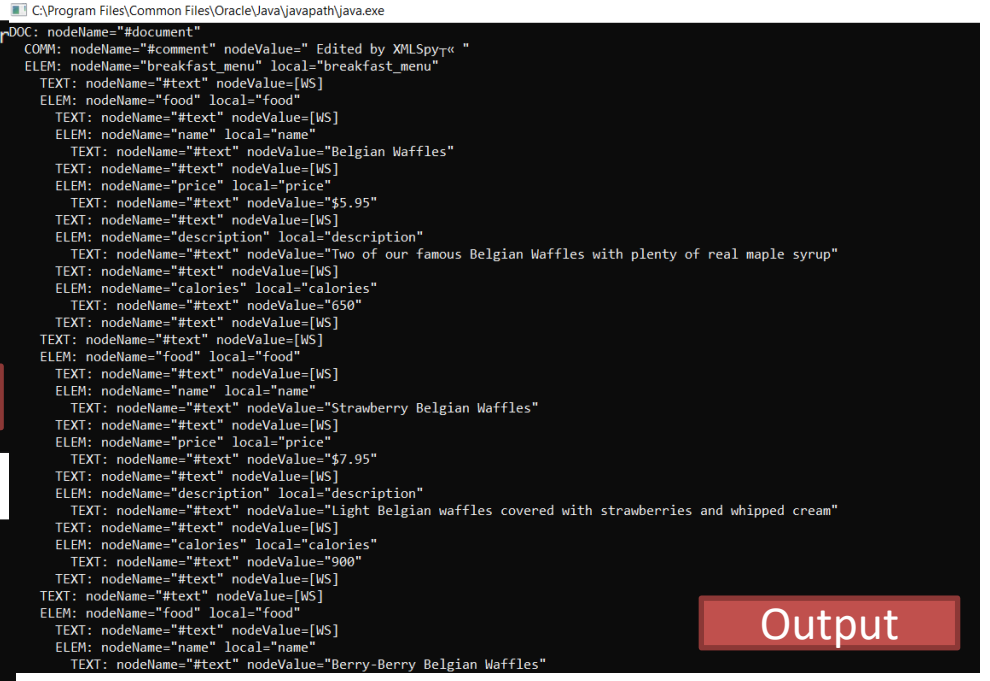
- Additionally for the tests involving the files medsamp2013.xml and medsamp2014.xml, it was necessary to correct the structure of both XML files, in order to be able to parse them correctly, this because there were several tags missing at the end of the files.

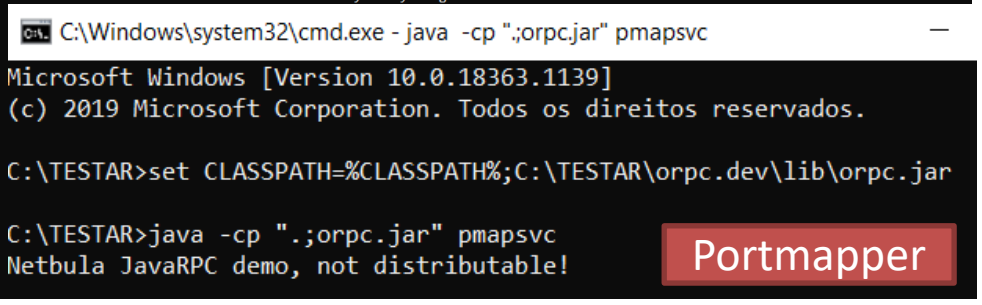
2.1. Parser DOM

- simple.xml









2.1. Parser DOM

- books.xml

C:\Program Files\Common Files\Oracle\Java\javapath\java.exe

```
DOC: nodeName="#document"
  ELEM: nodeName="catalog" local="catalog"
    TEXT: nodeName="#text" nodeValue=[WS]
    ELEM: nodeName="book" local="book"
        ATTR: nodeName="id" local="id" nodeValue="bk101"
          TEXT: nodeName="#text" nodeValue="bk101"
      TEXT: nodeName="#text" nodeValue=[WS]
      ELEM: nodeName="author" local="author"
        TEXT: nodeName="#text" nodeValue="Gambardella, Matthew"
      TEXT: nodeName="#text" nodeValue=[WS]
      ELEM: nodeName="title" local="title"
        TEXT: nodeName="#text" nodeValue="XML Developer's Guide"
      TEXT: nodeName="#text" nodeValue=[WS]
      ELEM: nodeName="genre" local="genre"
        TEXT: nodeName="#text" nodeValue="Computer"
      TEXT: nodeName="#text" nodeValue=[WS]
      ELEM: nodeName="price" local="price"
        TEXT: nodeName="#text" nodeValue="44.95"
      TEXT: nodeName="#text" nodeValue=[WS]
      ELEM: nodeName="publish_date" local="publish_date"
        TEXT: nodeName="#text" nodeValue="2000-10-01"
      TEXT: nodeName="#text" nodeValue=[WS]
      ELEM: nodeName="description" local="description"
        TEXT: nodeName="#text" nodeValue="An in-depth look at creating applications
      with XML."
      TEXT: nodeName="#text" nodeValue=[WS]
    TEXT: nodeName="#text" nodeValue=[WS]
    ELEM: nodeName="book" local="book"
        ATTR: nodeName="id" local="id" nodeValue="bk102"
          TEXT: nodeName="#text" nodeValue="bk102"
      TEXT: nodeName="#text" nodeValue=[WS]
      ELEM: nodeName="author" local="author"
        TEXT: nodeName="#text" nodeValue="Ralls, Kim"
      TEXT: nodeName="#text" nodeValue=[WS]
      ELEM: nodeName="title" local="title"
        TEXT: nodeName="#text" nodeValue="Midnight Rain"
      TEXT: nodeName="#text" nodeValue=[WS]
      ELEM: nodeName="genre" local="genre"
        TEXT: nodeName="#text" nodeValue="Fantasy"
      TEXT: nodeName="#text" nodeValue=[WS]
      ELEM: nodeName="price" local="price"
        TEXT: nodeName="#text" nodeValue="5.95"
      TEXT: nodeName="#text" nodeValue=[WS]
      ELEM: nodeName="publish_date" local="publish_date"
        TEXT: nodeName="#text" nodeValue="2000-12-16"
      TEXT: nodeName="#text" nodeValue=[WS]
      ELEM: nodeName="description" local="description"
        TEXT: nodeName="#text" nodeValue="A former architect battles corporate zombies,
```

Output

C:\Windows\system32\cmd.exe - run1.bat

```
Microsoft Windows [Version 10.0.18363.1139]
(c) 2019 Microsoft Corporation. Todos os direitos reservados.

C:\TESTAR>set CLASSPATH=%CLASSPATH%;.

C:\TESTAR>run1.bat

C:\TESTAR>set CLASSPATH=%CLASSPATH%;.;C:\TESTAR\orpc.dev\lib\orpc.jar

C:\TESTAR>java FileServer
Netbula JavaRPC demo, not distributable!
Registered TCP transport.
Registered UDP transport.
Ficheiro recebido: books.xml 4550 bytes transferidos
Ficheiro gravado: books.xml
Closing socket.
```

File Server

C:\Windows\system32\cmd.exe

```
C:\TESTAR>java FileClient localhost books.xml
Netbula JavaRPC demo, not distributable!
Ligado ao servidor  localhost
books.xml 4550 bytes enviados

C:\TESTAR>
```

File Client

2.1. Parser DOM

- medsamp2014.xml

C:\Program Files\Common Files\Oracle\Java\javapath\java.exe

```
DOC: nodeName="#document"
  ELEM: nodeName="MedlineCitationSet" local="MedlineCitationSet"
    TEXT: nodeName="#text" nodeValue=[WS]
    ELEM: nodeName="MedlineCitation" local="MedlineCitation"
        ATTR: nodeName="Owner" local="Owner" nodeValue="NLM"
          TEXT: nodeName="#text" nodeValue="NLM"
        ATTR: nodeName="Status" local="Status" nodeValue="MEDLINE"
          TEXT: nodeName="#text" nodeValue="MEDLINE"
      TEXT: nodeName="#text" nodeValue=[WS]
      ELEM: nodeName="PMID" local="PMID"
          ATTR: nodeName="Version" local="Version" nodeValue="1"
            TEXT: nodeName="#text" nodeValue="1"
        TEXT: nodeName="#text" nodeValue="10540283"
      TEXT: nodeName="#text" nodeValue=[WS]
      ELEM: nodeName="DateCreated" local="DateCreated"
        TEXT: nodeName="#text" nodeValue=[WS]
        ELEM: nodeName="Year" local="Year"
          TEXT: nodeName="#text" nodeValue="1999"
        TEXT: nodeName="#text" nodeValue=[WS]
        ELEM: nodeName="Month" local="Month"
          TEXT: nodeName="#text" nodeValue="12"
        TEXT: nodeName="#text" nodeValue=[WS]
        ELEM: nodeName="Day" local="Day"
          TEXT: nodeName="#text" nodeValue="17"
        TEXT: nodeName="#text" nodeValue=[WS]
      TEXT: nodeName="#text" nodeValue=[WS]
      ELEM: nodeName="DateCompleted" local="DateCompleted"
        TEXT: nodeName="#text" nodeValue=[WS]
        ELEM: nodeName="Year" local="Year"
          TEXT: nodeName="#text" nodeValue="1999"
        TEXT: nodeName="#text" nodeValue=[WS]
        ELEM: nodeName="Month" local="Month"
```

Output

Selecionar Administrator: Linha de comandos - run1.bat

```
C:\TESTAR>run1.bat

C:\TESTAR>set CLASSPATH=;C:\TESTAR\orpc.dev\lib\orpc.jar

C:\TESTAR>set CLASSPATH=;C:\TESTAR\orpc.dev\lib\orpc.jar;.

C:\TESTAR>java FileServer
Netbula JavaRPC demo, not distributable!
Registered TCP transport.
Registered UDP transport.
Ficheiro recebido: medsamp2014.xml 61450 bytes transferidos
Ficheiro gravado: medsamp2014.xml
Closing socket.
```

File Server

Administrator: Linha de comandos

```
C:\TESTAR>run2.bat

C:\TESTAR>set CLASSPATH=;C:\TESTAR\orpc.dev\lib\orpc.jar

C:\TESTAR>set CLASSPATH=;C:\TESTAR\orpc.dev\lib\orpc.jar;.

C:\TESTAR>java FileClient localhost medsamp2014.xml
Netbula JavaRPC demo, not distributable!
Ligado ao servidor  localhost
medsamp2014.xml 61450 bytes enviados
```

File Client

2.2. Parser SAX

- books.xml

Administrator: Linha de comandos

```
C:\TESTAR_SAX\sax>java SAXLocalNameCount.java "..\books.xml"
Note: SAXLocalNameCount.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
Note: SAXLocalNameCount.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
Local Name "genre" occurs 12 times
Local Name "catalog" occurs 1 times
Local Name "book" occurs 12 times
Local Name "author" occurs 12 times
Local Name "description" occurs 12 times
Local Name "price" occurs 12 times
Local Name "publish_date" occurs 12 times
Local Name "title" occurs 12 times

C:\TESTAR_SAX\sax>
```

Output

C:\Windows\system32\cmd.exe - run1.bat

```
Microsoft Windows [Version 10.0.18363.1139]
(c) 2019 Microsoft Corporation. Todos os direitos reservados.

C:\TESTAR_SAX>run1.bat

C:\TESTAR_SAX>set CLASSPATH=;C:\TESTAR_SAX\orpc.dev\lib\orpc.jar

C:\TESTAR_SAX>java FileServer
Netbula JavaRPC demo, not distributable!
Registered TCP transport.
Registered UDP transport.
Ficheiro recebido: books.xml 4550 bytes transferidos
Ficheiro gravado: books.xml
Closing socket.
```

File Server

C:\Windows\system32\cmd.exe

```
C:\TESTAR_SAX>java FileClient localhost books.xml
Netbula JavaRPC demo, not distributable!
Ligado ao servidor  localhost
books.xml 4550 bytes enviados

C:\TESTAR_SAX>
```

File Client

2.2. Parser SAX

- note.xml

Administrator: Linha de comandos

```
C:\TESTAR_SAX>java FileClient localhost note.xml
Netbula JavaRPC demo, not distributable!
Ligado ao servidor  localhost
note.xml 145 bytes enviados

C:\TESTAR_SAX>cd sax

C:\TESTAR_SAX\sax>java SAXLocalNameCount.java "..\note.xml"
Note: SAXLocalNameCount.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
Note: SAXLocalNameCount.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
Local Name "body" occurs 1 times
Local Name "note" occurs 1 times
Local Name "heading" occurs 1 times
Local Name "from" occurs 1 times
Local Name "to" occurs 1 times

C:\TESTAR_SAX\sax>
```

Output

Administrator: Linha de comandos - run1.bat

```
C:\TESTAR_SAX>run1.bat

C:\TESTAR_SAX>set CLASSPATH=%CLASSPATH%;.;C:\TESTAR_SA
\orpc.jar;C:\TESTAR_SAX\orpc.dev\lib\orpc.jar

C:\TESTAR_SAX>java FileServer
Netbula JavaRPC demo, not distributable!
Registered TCP transport.
Registered UDP transport.
Ficheiro recebido: note.xml 145 bytes transferidos
Ficheiro gravado: note.xml
Closing socket.

C:\TESTAR_SAX>java FileClient localhost note.xml
Netbula JavaRPC demo, not distributable!
Ligado ao servidor  localhost
note.xml 145 bytes enviados
```

File Server

File Client

2.2. Parser SAX

- medsamp2013.xml

```
C:\TESTAR_SAX\sax>java SAXLocalNameCount.java ..\medsamp2013.xml
Note: SAXLocalNameCount.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
Note: SAXLocalNameCount.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
Local Name "RefSource" occurs 9 times
Local Name "Grant" occurs 6 times
Local Name "NameOfSubstance" occurs 47 times
Local Name "AbstractText" occurs 18 times
Local Name "AccessionNumber" occurs 4 times
Local Name "AuthorList" occurs 11 times
Local Name "MeshHeadingList" occurs 10 times
Local Name "ForeName" occurs 328 times
Local Name "ArticleTitle" occurs 11 times
Local Name "MedlineCitationSet" occurs 1 times
Local Name "Volume" occurs 10 times
Local Name "ChemicalList" occurs 7 times
Local Name "Day" occurs 36 times
Local Name "Affiliation" occurs 11 times
Local Name "ISOAbbreviation" occurs 11 times
Local Name "Note" occurs 1 times
Local Name "Acronym" occurs 5 times
Local Name "CopyrightInformation" occurs 2 times
Local Name "DescriptorName" occurs 151 times
Local Name "PublicationType" occurs 25 times
Local Name "MedlineJournalInfo" occurs 11 times
Local Name "MedlineTA" occurs 11 times
Local Name "DateCompleted" occurs 10 times
Local Name "OtherID" occurs 2 times
Local Name "Title" occurs 11 times
Local Name "PMID" occurs 18 times
Local Name "MeshHeading" occurs 151 times
Local Name "CommentsCorrectionsList" occurs 3 times
```

Output

```
Administrator: C:\WINDOWS\system32\cmd.exe - run1.bat

C:\TESTAR>run1.bat

C:\TESTAR>set CLASSPATH=;C:\TESTAR\orpc.dev\lib\orpc.jar;C:\TE
pc.jar

C:\TESTAR>java FileServer
Netbula JavaRPC demo, not distributable!
Registered TCP transport.
Registered UDP transport.
Ficheiro recebido: medsamp2014.xml 61440 bytes transferidos
Ficheiro gravado: medsamp2014.xml
Closing socket.
```

File Server

```
Administrator: C:\WINDOWS\system32\cmd.exe

C:\TESTAR>run2.bat

C:\TESTAR>set CLASSPATH=;C:\TESTAR\orpc.dev\lib\orpc.ja

C:\TESTAR>java FileClient localhost medsamp2014.xml
Netbula JavaRPC demo, not distributable!
Ligado ao servidor  localhost
medsamp2014.xml 61440 bytes enviados

C:\TESTAR>
```

File Client

2.3. Parser JDOM

- books.xml

C:\Program Files\Common Files\Oracle\Java\javapath\java.exe

```
<------------XML DOCUMENT------------>
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
   <book id="bk101">
      <author>Gambardella, Matthew</author>
      <title>XML Developer's Guide</title>
      <genre>Computer</genre>
      <price>44.95</price>
      <publish_date>2000-10-01</publish_date>
      <description>An in-depth look at creating applications
      with XML.</description>
   </book>
   <book id="bk102">
      <author>Ralls, Kim</author>
      <title>Midnight Rain</title>
      <genre>Fantasy</genre>
      <price>5.95</price>
      <publish_date>2000-12-16</publish_date>
      <description>A former architect battles corporate zombies,
      an evil sorceress, and her own childhood to become queen
      of the world.</description>
   </book>
   <book id="bk103">
      <author>Corets, Eva</author>
      <title>Maeve Ascendant</title>
      <genre>Fantasy</genre>
      <price>5.95</price>
      <publish_date>2000-11-17</publish_date>
      <description>After the collapse of a nanotechnology
      society in England, the young survivors lay the
      foundation for a new society.</description>
   </book>
```

Output

Administrator: C:\WINDOWS\system32\cmd.exe - run1.bat

```
C:\TESTAR_JDOM>run1.bat

C:\TESTAR_JDOM>set CLASSPATH=;C:\TESTAR\orpc.dev\lib\orpc.jar

C:\TESTAR_JDOM>java FileServer
Netbula JavaRPC demo, not distributable!
Registered TCP transport.
Registered UDP transport.
Ficheiro recebido: books.xml 4548 bytes transferidos
Ficheiro gravado: books.xml
Closing socket.
```

File Server

Administrator: C:\WINDOWS\system32\cmd.exe

```
C:\TESTAR_JDOM>set CLASSPATH=%CLASSPATH%;C:\TESTAR\orpc.dev\lib\orpc.jar

C:\TESTAR_JDOM>java FileClient localhost books.xml
Error: Could not find or load main class FileClient
Caused by: java.lang.ClassNotFoundException: FileClient

C:\TESTAR_JDOM>set CLASSPATH=%CLASSPATH%;.

C:\TESTAR_JDOM>java FileClient localhost books.xml
Netbula JavaRPC demo, not distributable!
Ligado ao servidor  localhost
books.xml 4548 bytes enviados

C:\TESTAR_JDOM>
```

File Client

2.3. Parser JDOM

- note.xml

Selecionar C:\Windows\system32\cmd.exe - java Pla

```
<------------XML DOCUMENT------------>
<?xml version="1.0" encoding="UTF-8"?>
<!-- Edited by XMLSpy┬« --><note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Output

C:\Windows\system32\cmd.exe - run1.bat

```
C:\TESTAR_JDOM>run1.bat

C:\TESTAR_JDOM>set CLASSPATH=;C:\TESTAR\orpc.dev\lib\
c.jar

C:\TESTAR_JDOM>java FileServer
Netbula JavaRPC demo, not distributable!
Registered TCP transport.
Registered UDP transport.
Ficheiro recebido: note.xml 145 bytes transferidos
Ficheiro gravado: note.xml
Closing socket.
```

File Server

C:\Windows\system32\cmd.exe

```
C:\TESTAR_JDOM>java FileClient localhost note.xml
Netbula JavaRPC demo, not distributable!
Ligado ao servidor  localhost
note.xml 145 bytes enviados

C:\TESTAR_JDOM>
```

File Client

2.3. Parser JDOM

- medsamp2013.xml

C:\Windows\system32\cmd.exe - java PlayWithJDom medsamp2013.xml

```
<------------XML DOCUMENT------------>
<?xml version="1.0" encoding="UTF-8"?>
<MedlineCitationSet>
<MedlineCitation Owner="NLM" Status="MEDLINE">
<PMID Version="1">10540283</PMID>
<DateCreated>
<Year>1999</Year>
<Month>12</Month>
<Day>17</Day>
</DateCreated>
<DateCompleted>
<Year>1999</Year>
<Month>12</Month>
<Day>17</Day>
</DateCompleted>
<DateRevised>
<Year>2006</Year>
<Month>11</Month>
<Day>15</Day>
</DateRevised>
<Article PubModel="Print">
<Journal>
<ISSN IssnType="Print">0950-382X</ISSN>
<JournalIssue CitedMedium="Print">
<Volume>34</Volume>
<Issue>1</Issue>
<PubDate>
<Year>1999</Year>
<Month>Oct</Month>
</PubDate>
</JournalIssue>
<Title>Molecular microbiology</Title>
<ISOAbbreviation>Mol. Microbiol.</ISOAbbreviation>
</Journal>
<ArticleTitle>Transcription regulation of the nir gene cluster encoding nitrite
<Pagination>
<MedlinePgn>24-36</MedlinePgn>
</Pagination>
```

Output

C:\Windows\system32\cmd.exe - run1.bat

```
C:\TESTAR_JDOM>run1.bat

C:\TESTAR_JDOM>set CLASSPATH=;C:\TESTAR\orpc.dev\lib\orpc.jar;C:\TESTA
c.jar;C:\TESTAR\orpc.dev\lib\orpc.jar

C:\TESTAR_JDOM>java FileServer
Netbula JavaRPC demo, not distributable!
Registered TCP transport.
Registered UDP transport.
Ficheiro recebido: medsamp2013.xml 77824 bytes transferidos
Ficheiro gravado: medsamp2013.xml
Closing socket.
```

File Server

C:\Windows\system32\cmd.exe

```
C:\TESTAR_JDOM>java FileClient localhost medsamp2013.xml
Netbula JavaRPC demo, not distributable!
Ligado ao servidor  localhost
medsamp2013.xml 77824 bytes enviados

C:\TESTAR_JDOM>
```

File Client

2.4. Parser JAXP

- books.xml

C:\Program Files\Common Files\Oracle\Java\javapath\java.exe

```
FACTORY: com.sun.xml.internal.stream.XMLInputFactoryImpl@1324409e
filename = books.xml

<?xml version="1.0" encoding="null"?>
<catalog>
   <book id="bk101">
      <author>Gambardella, Matthew</author>
      <title>XML Developer's Guide</title>
      <genre>Computer</genre>
      <price>44.95</price>
      <publish_date>2000-10-01</publish_date>
      <description>An in-depth look at creating applications
      with XML.</description>
   </book>
   <book id="bk102">
      <author>Ralls, Kim</author>
      <title>Midnight Rain</title>
      <genre>Fantasy</genre>
      <price>5.95</price>
      <publish_date>2000-12-16</publish_date>
      <description>A former architect battles corporate zombies,
      an evil sorceress, and her own childhood to become queen
      of the world.</description>
   </book>
   <book id="bk103">
      <author>Corets, Eva</author>
      <title>Maeve Ascendant</title>
      <genre>Fantasy</genre>
      <price>5.95</price>
      <publish_date>2000-11-17</publish_date>
      <description>After the collapse of a nanotechnology
      society in England, the young survivors lay the
      foundation for a new society.</description>
   </book>
```

Output

Administrator: C:\WINDOWS\system32\cmd.exe - run1.bat

```
C:\TESTAR_JAXP>run1.bat

C:\TESTAR_JAXP>set CLASSPATH=;C:\TESTAR\orpc.dev\lib\orpc.jar

C:\TESTAR_JAXP>java FileServer
Netbula JavaRPC demo, not distributable!
Registered TCP transport.
Registered UDP transport.
Ficheiro recebido: books.xml 4550 bytes transferidos
Ficheiro gravado: books.xml
Closing socket.
```

File Server

C:\Windows\system32\cmd.exe

```
C:\TESTAR_JAXP>java FileClient localhost books.xml
Netbula JavaRPC demo, not distributable!
Ligado ao servidor  localhost
books.xml 4550 bytes enviados

C:\TESTAR_JAXP>
```

File Client

2.4. Parser JAXP

- simple.xml

C:\Program Files\Common Files\Oracle\Java\javapath\java.exe

```
FACTORY: com.sun.xml.internal.stream.XMLInputFactoryImpl@1324409e
filename = simple.xml

<?xml version="1.0" encoding="ISO-8859-1"?>
 Edited by XMLSpy? <!-- Edited by XMLSpy? --><breakfast_menu>
        <food>
                <name>Belgian Waffles</name>
                <price>$5.95</price>
                <description>Two of our famous Belgian Waffles with plenty of real maple syrup</description>
                <calories>650</calories>
        </food>
        <food>
                <name>Strawberry Belgian Waffles</name>
                <price>$7.95</price>
                <description>Light Belgian waffles covered with strawberries and whipped cream</description>
                <calories>900</calories>
        </food>
        <food>
                <name>Berry-Berry Belgian Waffles</name>
                <price>$8.95</price>
                <description>Light Belgian waffles covered with an assortment of fresh berries and whipped cream</description>
                <calories>900</calories>
        </food>
        <food>
                <name>French Toast</name>
                <price>$4.50</price>
                <description>Thick slices made from our homemade sourdough bread</description>
                <calories>600</calories>
        </food>
        <food>
                <name>Homestyle Breakfast</name>
                <price>$6.95</price>
                <description>Two eggs, bacon or sausage, toast, and our ever-popular hash browns</description>
                <calories>950</calories>
        </food>
</breakfast_menu> Parsing Time = 29
```

Output

Administrator: C:\WINDOWS\system32\cmd.exe - run1.bat

```
C:\TESTAR_JAXP>run1.bat

C:\TESTAR_JAXP>set CLASSPATH=;C:\TESTAR\orpc.dev\lib\orpc
orpc.jar

C:\TESTAR_JAXP>java FileServer
Netbula JavaRPC demo, not distributable!
Registered TCP transport.
Registered UDP transport.
Ficheiro recebido: simple.xml 1135 bytes transferidos
Ficheiro gravado: simple.xml
Closing socket.
```

File Server

C:\Windows\system32\cmd.exe

```
C:\TESTAR_JAXP>java FileClient localhost simple.xml
Netbula JavaRPC demo, not distributable!
Ligado ao servidor  localhost
simple.xml 1135 bytes enviados

C:\TESTAR_JAXP>
```

File Client

2.4. Parser JAXP

- medsamp2014.xml

C:\Program Files\Common Files\Oracle\Java\javapath\java.exe

```
FACTORY: com.sun.xml.internal.stream.XMLInputFactoryImpl@1324409e
filename = medsamp2014.xml

<?xml version="1.0" encoding="UTF-8"?>
<MedlineCitationSet>
<MedlineCitation Owner="NLM" Status="MEDLINE">
<PMID Version="1">10540283</PMID>
<DateCreated>
<Year>1999</Year>
<Month>12</Month>
<Day>17</Day>
</DateCreated>
<DateCompleted>
<Year>1999</Year>
<Month>12</Month>
<Day>17</Day>
</DateCompleted>
<DateRevised>
<Year>2006</Year>
<Month>11</Month>
<Day>15</Day>
</DateRevised>
<Article PubModel="Print">
<Journal>
<ISSN IssnType="Print">0950-382X</ISSN>
<JournalIssue CitedMedium="Print">
<Volume>34</Volume>
<Issue>1</Issue>
<PubDate>
<Year>1999</Year>
<Month>Oct</Month>
</PubDate>
</JournalIssue>
<Title>Molecular microbiology</Title>
<ISOAbbreviation>Mol. Microbiol.</ISOAbbreviation>
</Journal>
<ArticleTitle>Transcription regulation of the nir gene cluster encoding nitrite reductase
<Pagination>
```

Output

Administrator: C:\WINDOWS\system32\cmd.exe - run1.bat

```
C:\TESTAR_JAXP>run1.bat

C:\TESTAR_JAXP>set CLASSPATH=;C:\TESTAR\orpc.dev\lib\orpc.jar;C
orpc.jar;C:\TESTAR\orpc.dev\lib\orpc.jar

C:\TESTAR_JAXP>java FileServer
Netbula JavaRPC demo, not distributable!
Registered TCP transport.
Registered UDP transport.
Ficheiro recebido: medsamp2014.xml 102321 bytes transferidos
Ficheiro gravado: medsamp2014.xml
Closing socket.
```

File Server

C:\Windows\system32\cmd.exe

```
C:\TESTAR_JAXP>java FileClient localhost medsamp2014.xml
Netbula JavaRPC demo, not distributable!
Ligado ao servidor  localhost
medsamp2014.xml 102321 bytes enviados
```

File Client

2.5. Parser JAXB

C:\Program Files\Common Files\Oracle\Java\javapath\java.exe

```
Ship the following items to:
        Alice Smith
        123 Maple Street
        Cambridge, MA 12345
        US

        5 copies of "Nosferatu - Special Edition (1929)"
        3 copies of "The Mummy (1959)"
        3 copies of "Godzilla and Mothra: Battle for Earth/Godzilla vs. King Ghidora"
```

Output

C:\Windows\system32\cmd.exe - run1.bat

```
C:\TESTAR_JAXB\samples\unmarshal-read>run1.bat

C:\TESTAR_JAXB\samples\unmarshal-read>java FileServer
Netbula JavaRPC demo, not distributable!
Registered TCP transport.
Registered UDP transport.
Ficheiro recebido: po.xml 2835 bytes transferidos
Ficheiro gravado: po.xml
Closing socket.
```

File Server

C:\Windows\system32\cmd.exe

```
C:\TESTAR_JAXB\samples\unmarshal-read>run2.bat

C:\TESTAR_JAXB\samples\unmarshal-read>java FileClient localhost po.xml
Netbula JavaRPC demo, not distributable!
Ligado ao servidor  localhost
po.xml 2835 bytes enviados
```

File Client

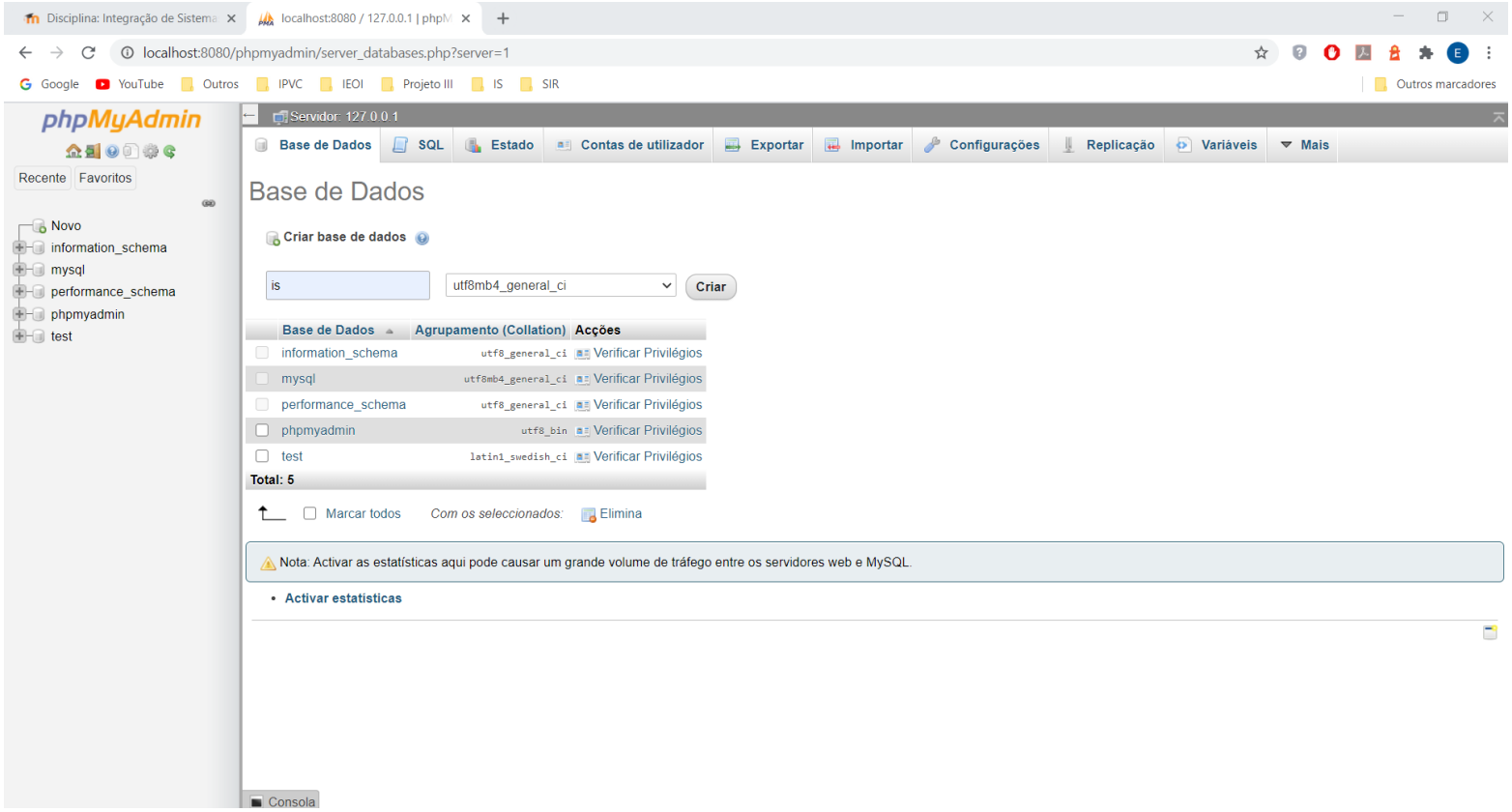
3. SOAP and REST WebServices Implementation integrating XML files in a Data Base

- In this section, it is described the second part of the 8th tutorial, which consists of testing and analyzing the given source code for different webservices created in JAVA and PHP, using some of the parsers tested in the first part of the tutorial (correspondent to the 7th tutorial).

- In order to start this part of the tutorial, it was required to adjust some configurations to the NetBeans IDE to allow webservices to be tested locally in the already installed glassfish server, as well as creating a new MySQL database which runs on the local server provided by XAMPP, with one table called Alunos, and two columns, one for the id (which is the primary key), and another one for the student's name, this to match the xml file alunos.xml which will be used later in the tutorial.

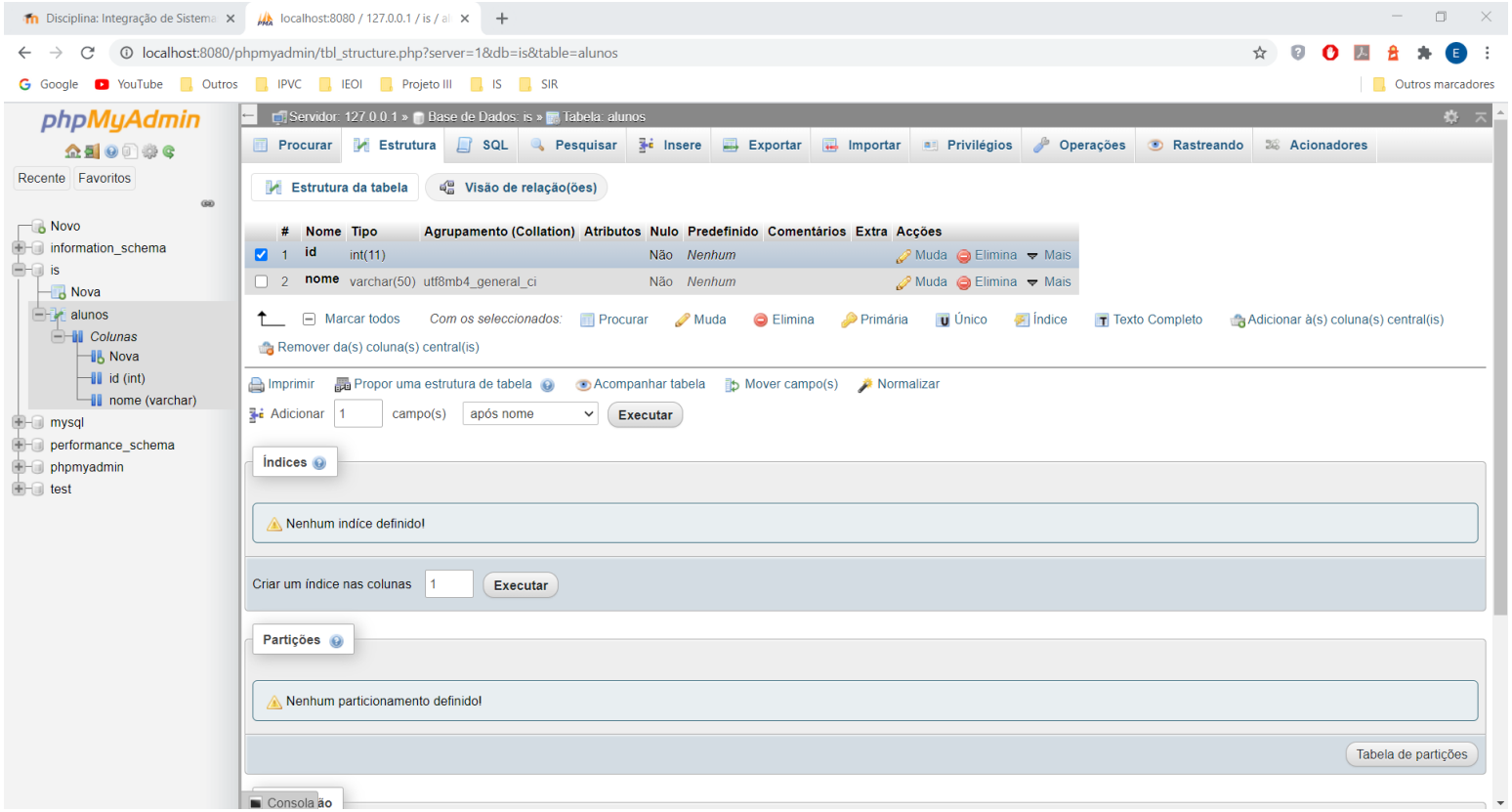
3. SOAP and REST WebServices Implementation integrating XML files in a Data Base

- Database creation using phpMyAdmin



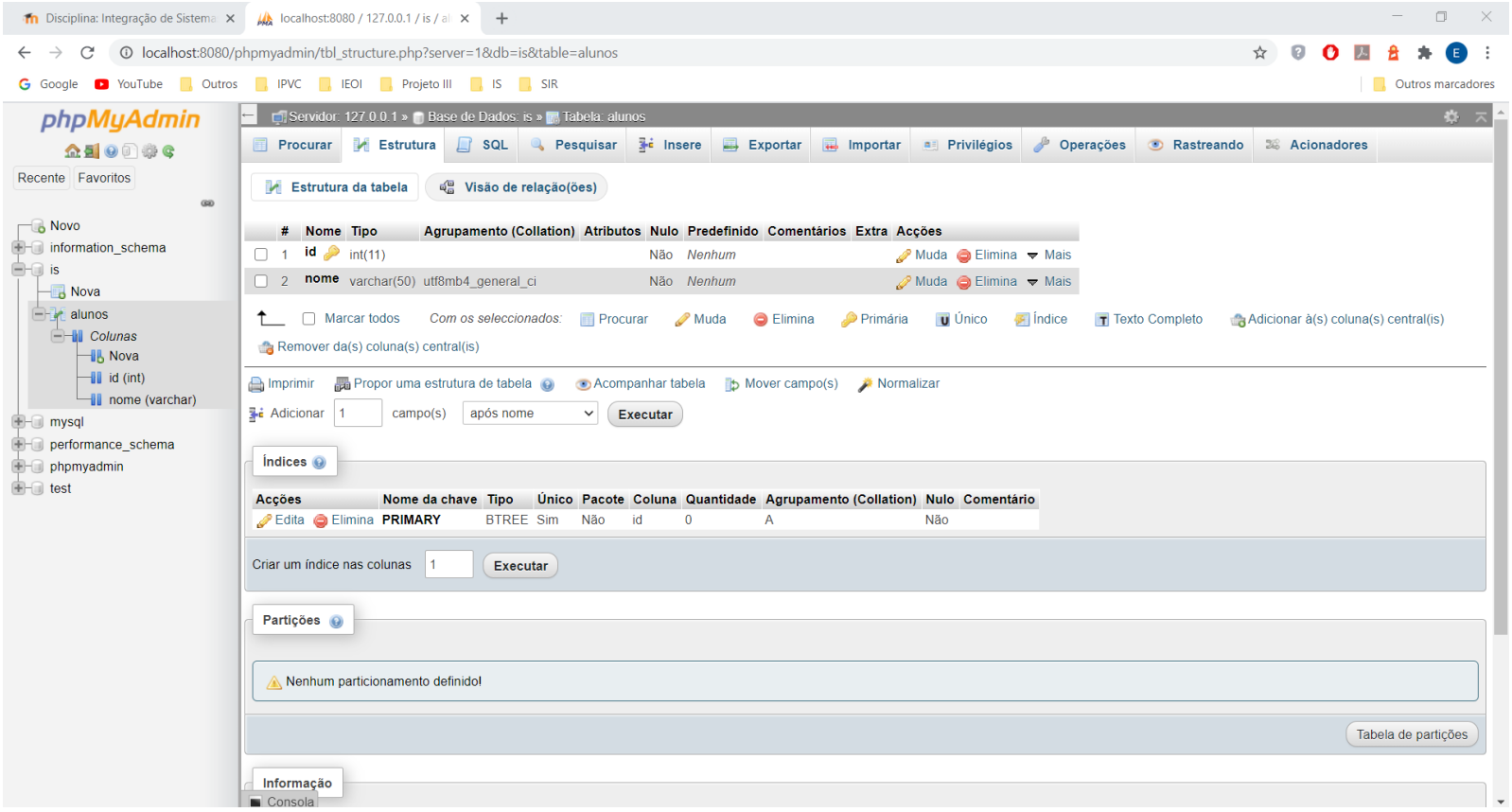
3. SOAP and REST WebServices Implementation integrating XML files in a Data Base

- Table “alunos” and columns “id” and “nome” created in the new database



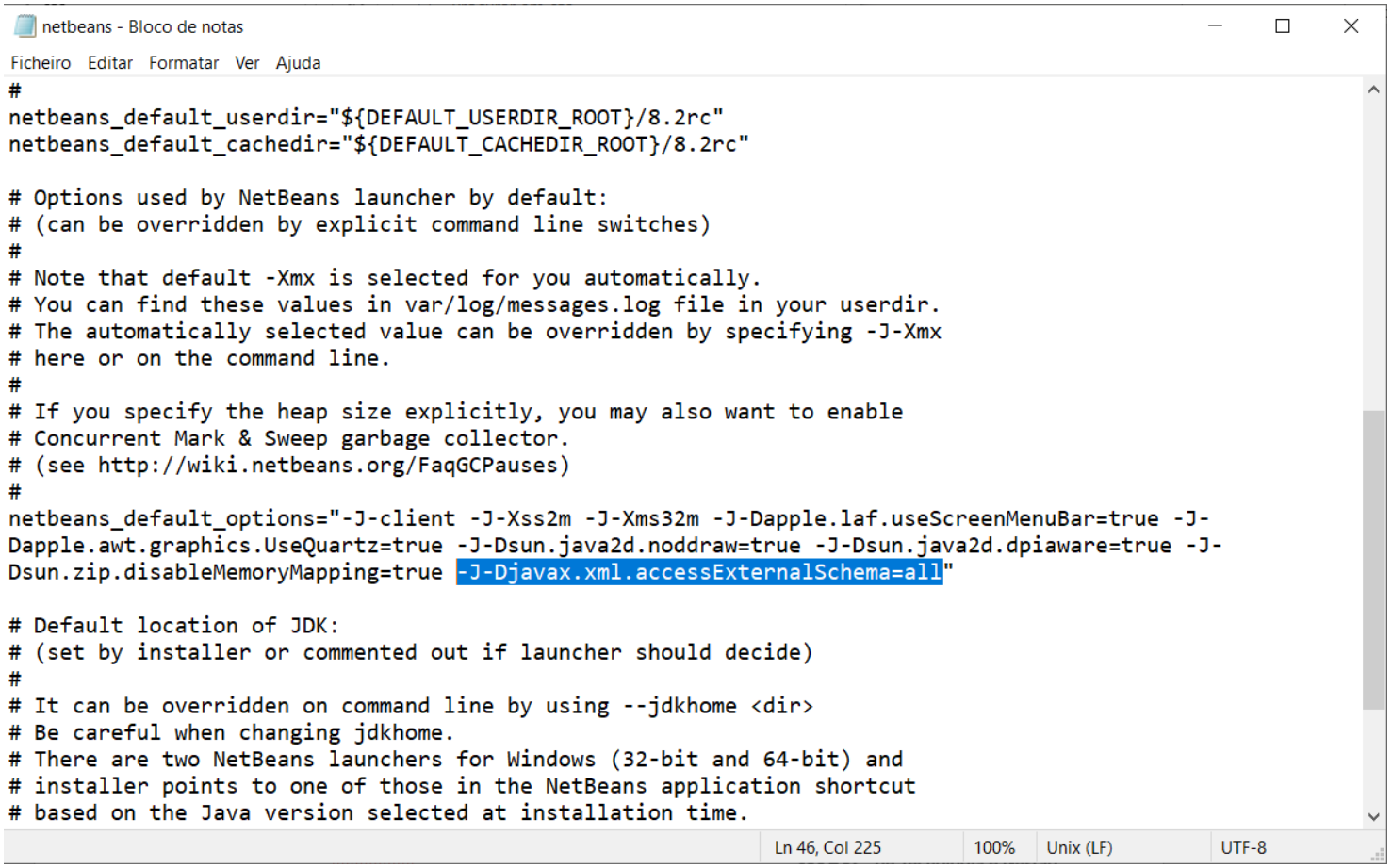
3. SOAP and REST WebServices Implementation integrating XML files in a Data Base

- Definition of the column “id” as the table’s primary key



3. SOAP and REST WebServices Implementation integrating XML files in a Data Base

- Option added to NetBeans default configuration



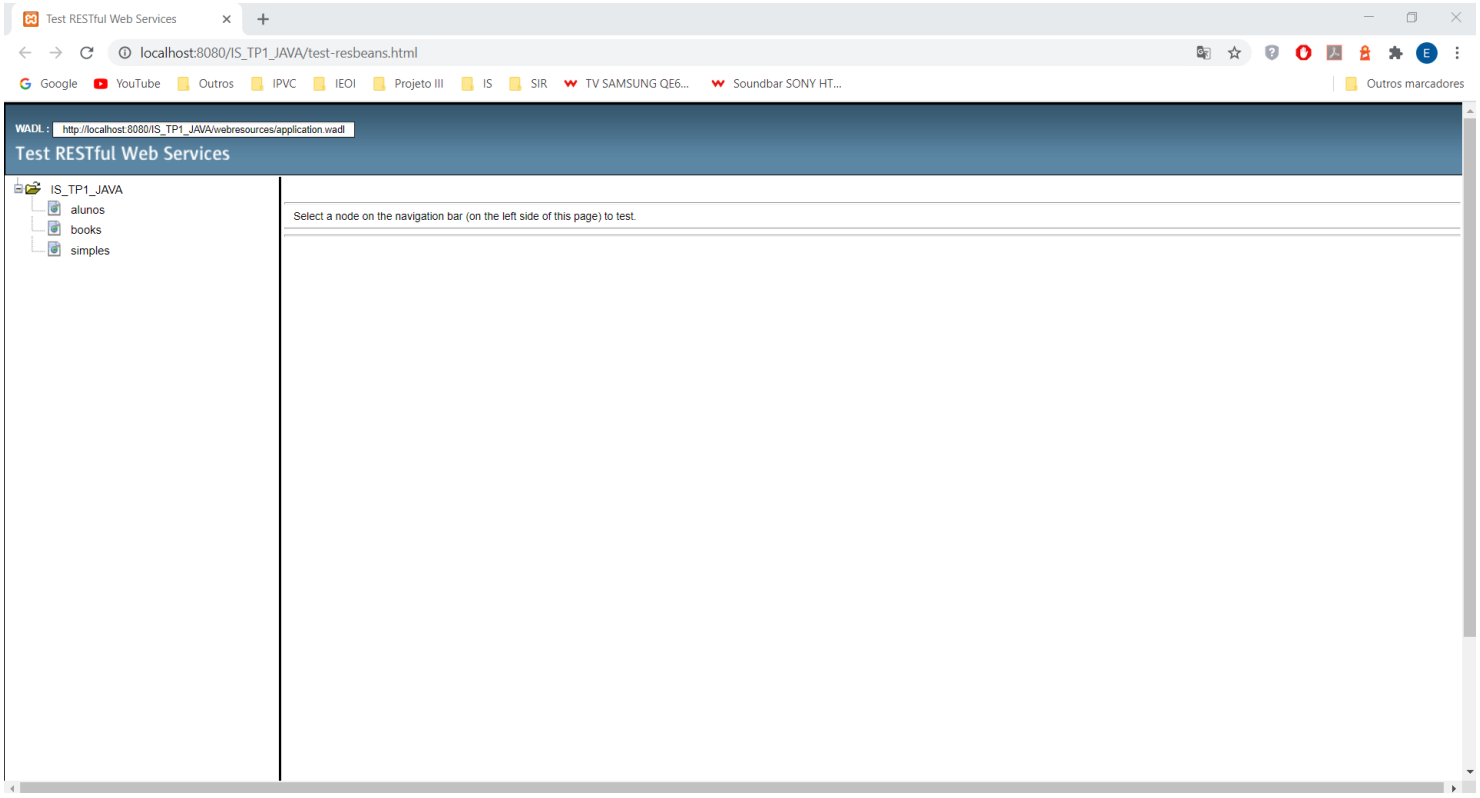
3.1. Java webservices implementation

- This section demonstrates the output of the execution of SOAP and REST webservices using the NetBeans IDE, executing them in the glassfish server along with a MySQL database with the information present in the alunos.xml file. The result of the execution of the webservices is demonstrated by screenshots, as well as the headers of the request and the server's response, verified on the browser's developer tools.

- These webservices are implemented using three different Java classes:
  - One for the declaration of the class that defines an object present in the XML file, with all the XML tags declared as attributes of this same class, as well as all the necessary getters and setters;
  - The second one for the declaration of the array of objects defined in the first class, being the match for the root tag in the XML file;
  - The third corresponds to the definition of the webservice, implementing the method to get all the information from the XML file, using a parser to de-serialize its contents, and create java objects from the different XML tags.

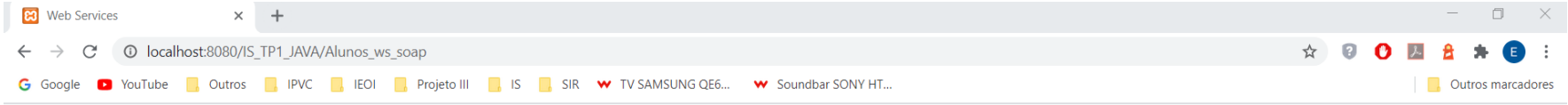
3.1. Java webservices implementation

- Available REST webservices



3.1. Java webservices implementation

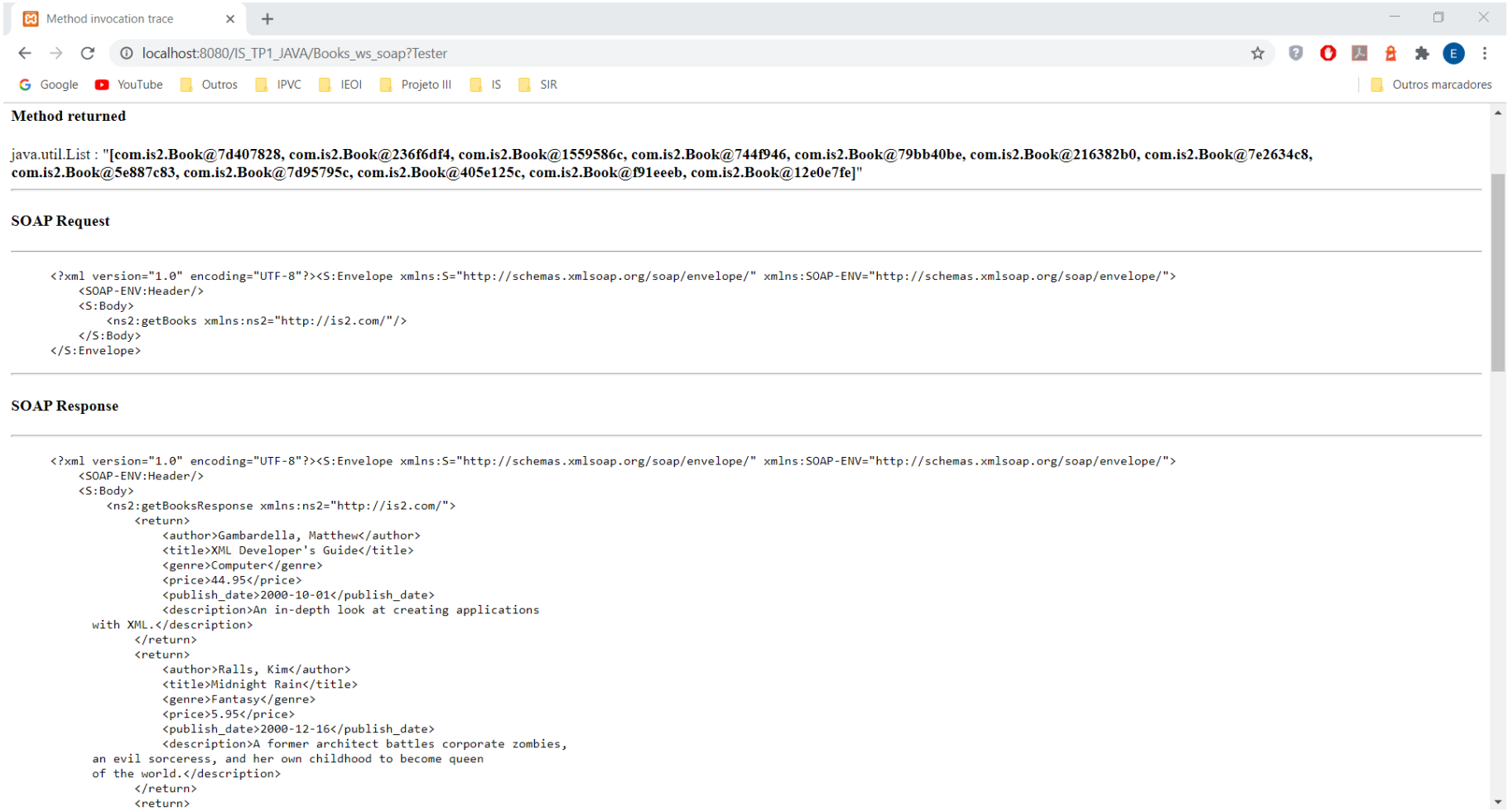
- Available SOAP webservices



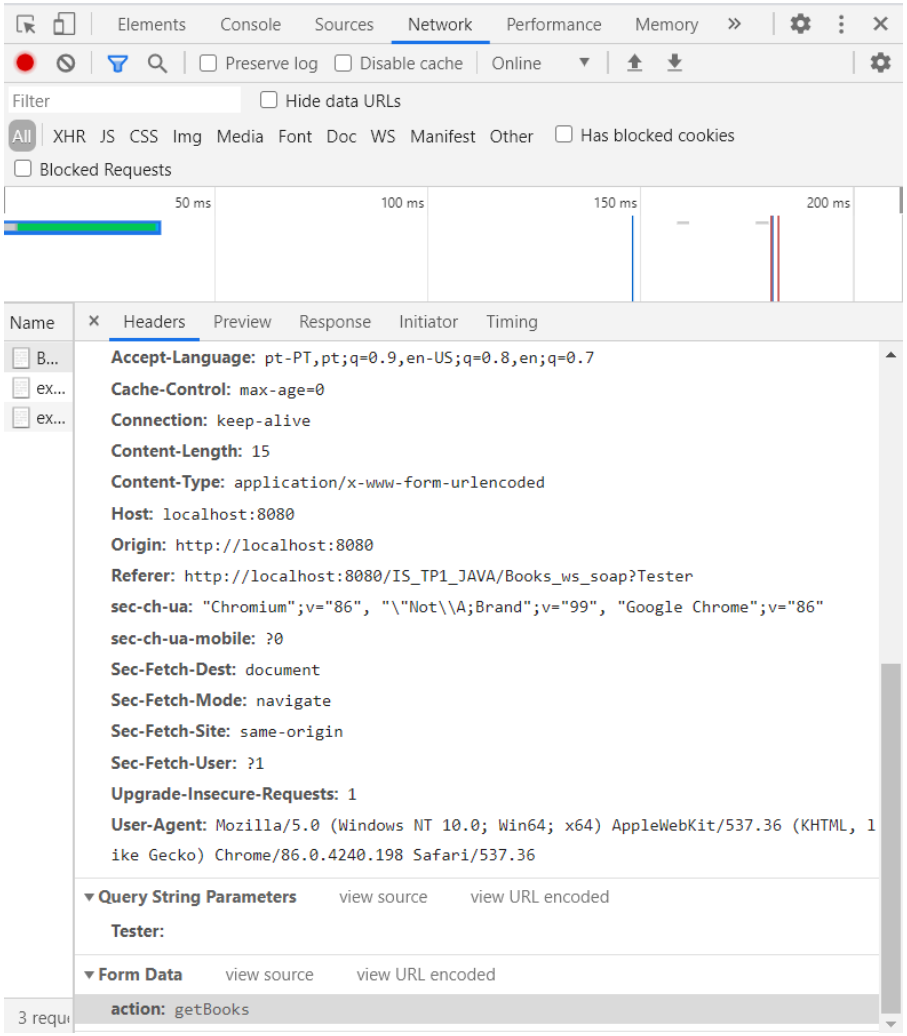
Web Services

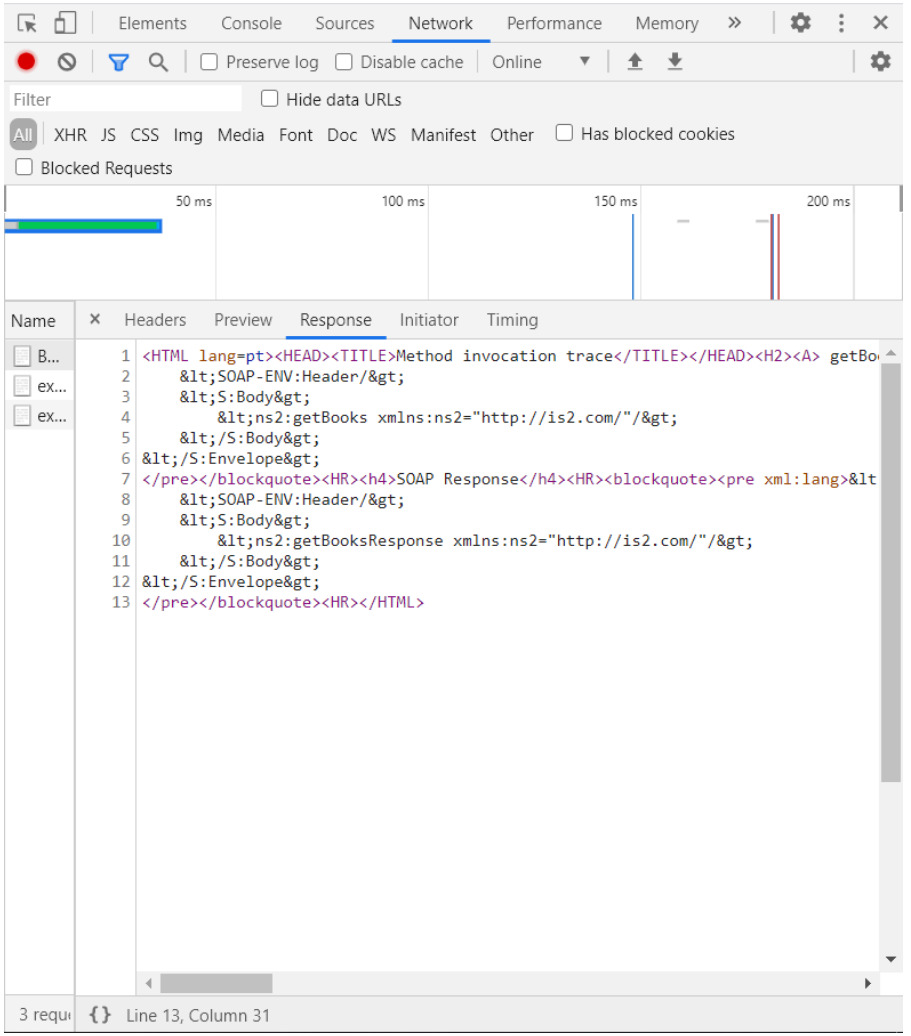
Endpoint	Information
Service Name: {http://is2.com/}Books_ws_soap Port Name: {http://is2.com/}Books_ws_soapPort	Address: http://localhost:8080/IS_TP1_JAVA/Books_ws_soap WSDL: http://localhost:8080/IS_TP1_JAVA/Books_ws_soap?wsdl Implementation class: com.is2.Books_ws_soap
Service Name: {http://is2.com/}Simples_ws_soap Port Name: {http://is2.com/}Simples_ws_soapPort	Address: http://localhost:8080/IS_TP1_JAVA/Simples_ws_soap WSDL: http://localhost:8080/IS_TP1_JAVA/Simples_ws_soap?wsdl Implementation class: com.is2.Simples_ws_soap
Service Name: {http://db.is.com/}Alunos_ws_soap Port Name: {http://db.is.com/}Alunos_ws_soapPort	Address: http://localhost:8080/IS_TP1_JAVA/Alunos_ws_soap WSDL: http://localhost:8080/IS_TP1_JAVA/Alunos_ws_soap?wsdl Implementation class: com.is.db.Alunos_ws_soap

3.1.1. SOAP webservice with file books.xml and JAXB parser

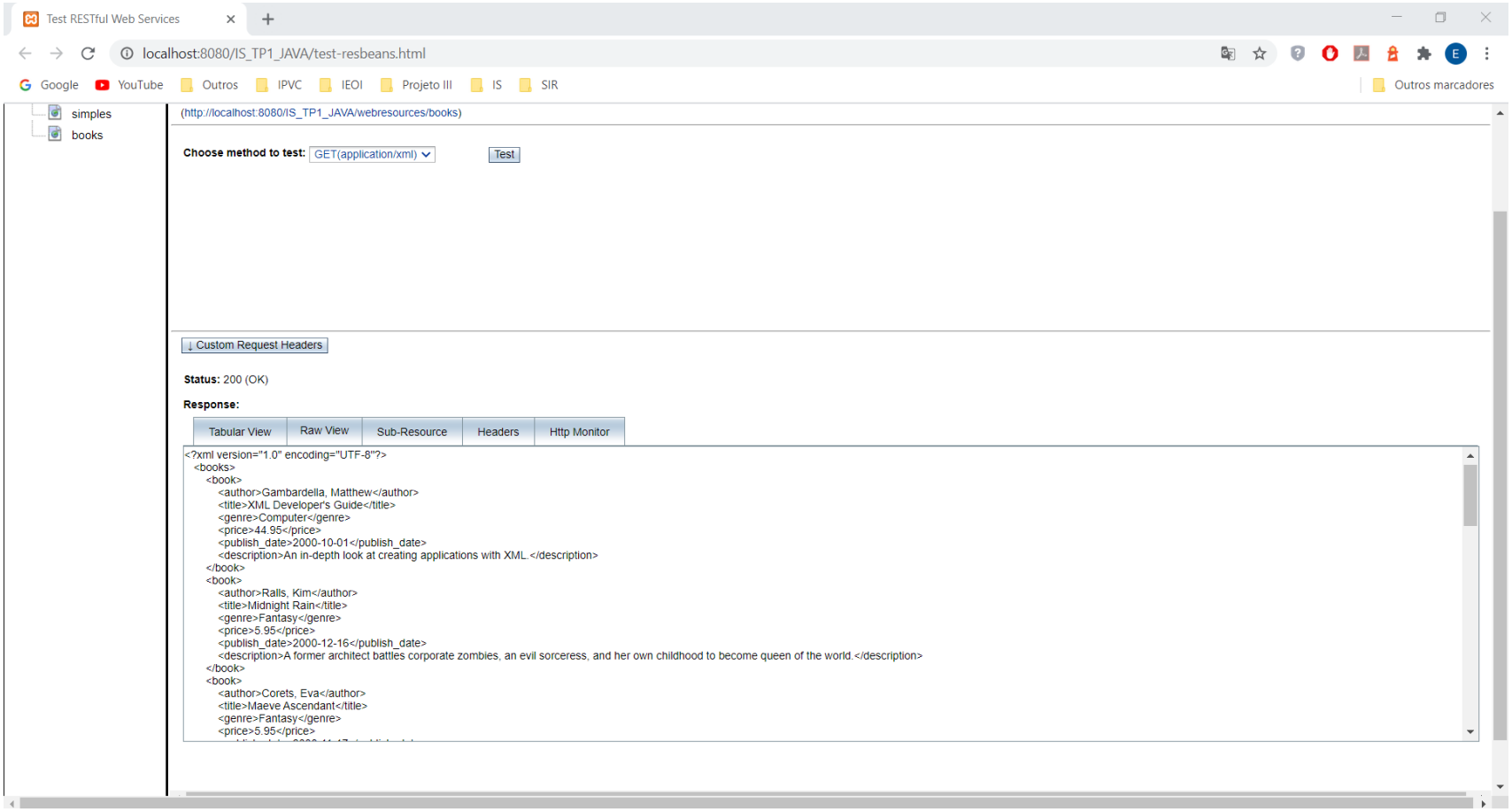


3.1.1. SOAP webservice with file books.xml and JAXB parser

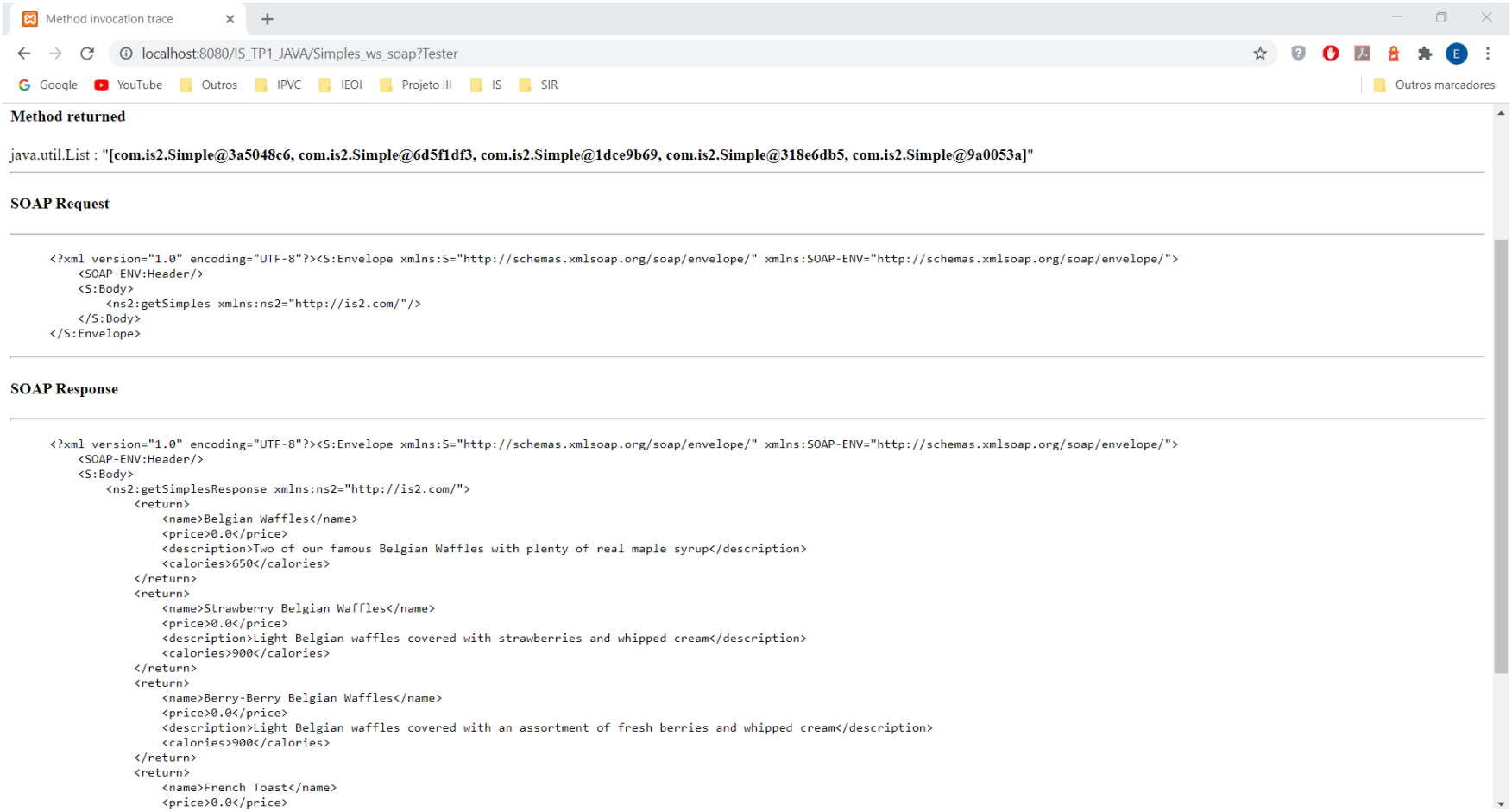




3.1.2. REST webservice with file books.xml and JAXB parser



3.1.3. SOAP webservice with file simple.xml and JAXB parser



Method returned

java.util.List : "**[com.is2.Simple@3a5048c6, com.is2.Simple@6d5f1df3, com.is2.Simple@1dce9b69, com.is2.Simple@318e6db5, com.is2.Simple@9a0053a]**"

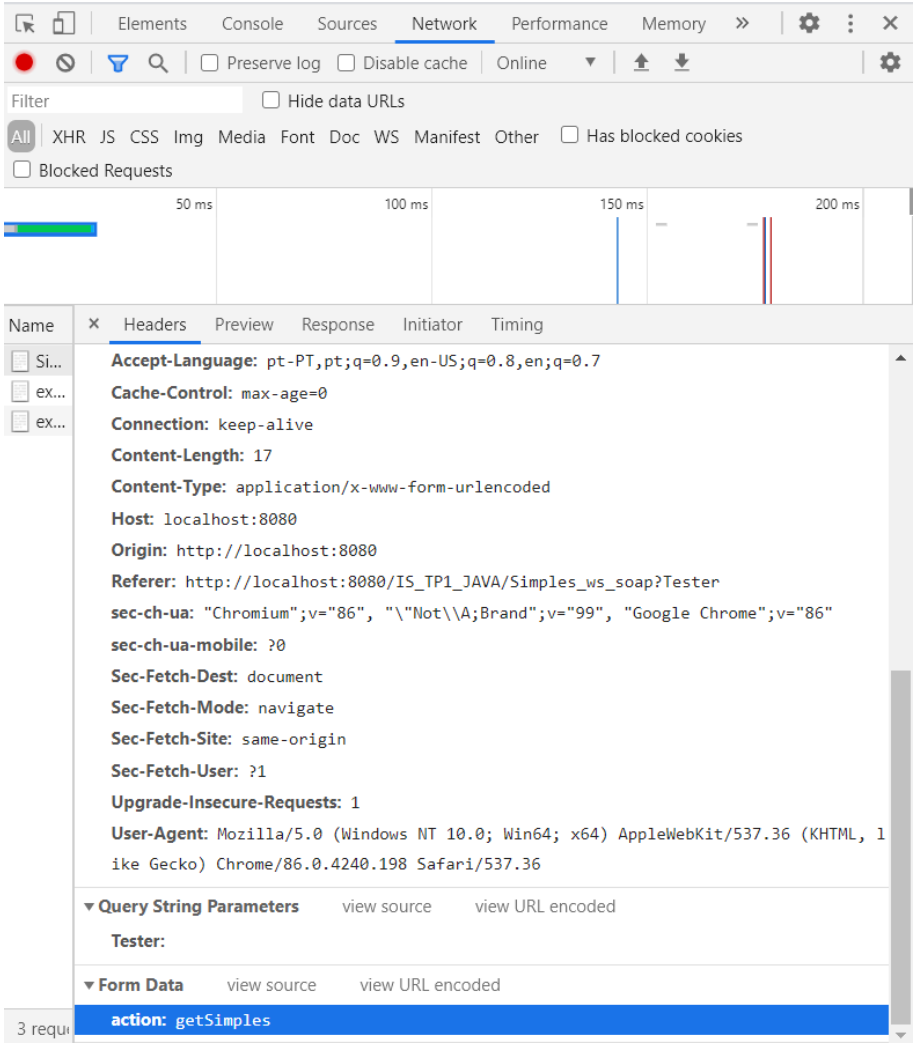
SOAP Request

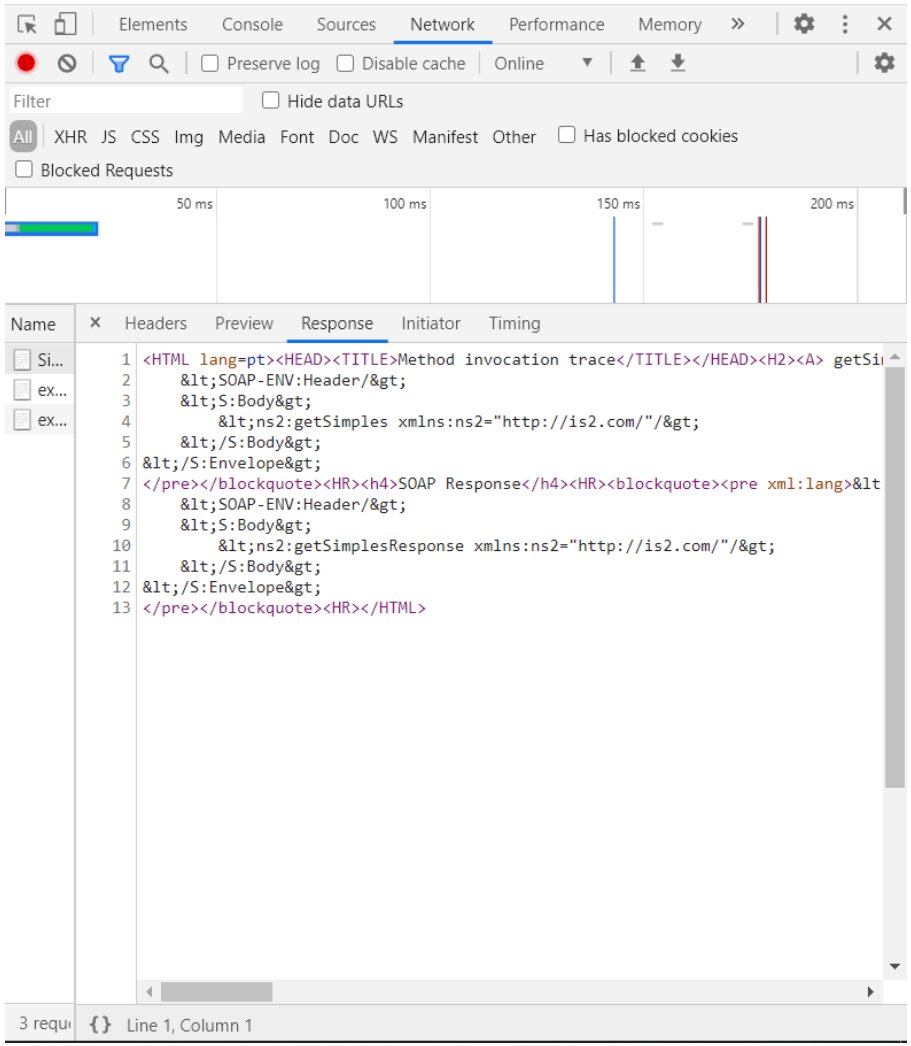
```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <S:Body>
        <ns2:getSimples xmlns:ns2="http://is2.com/"/>
    </S:Body>
</S:Envelope>
```

SOAP Response

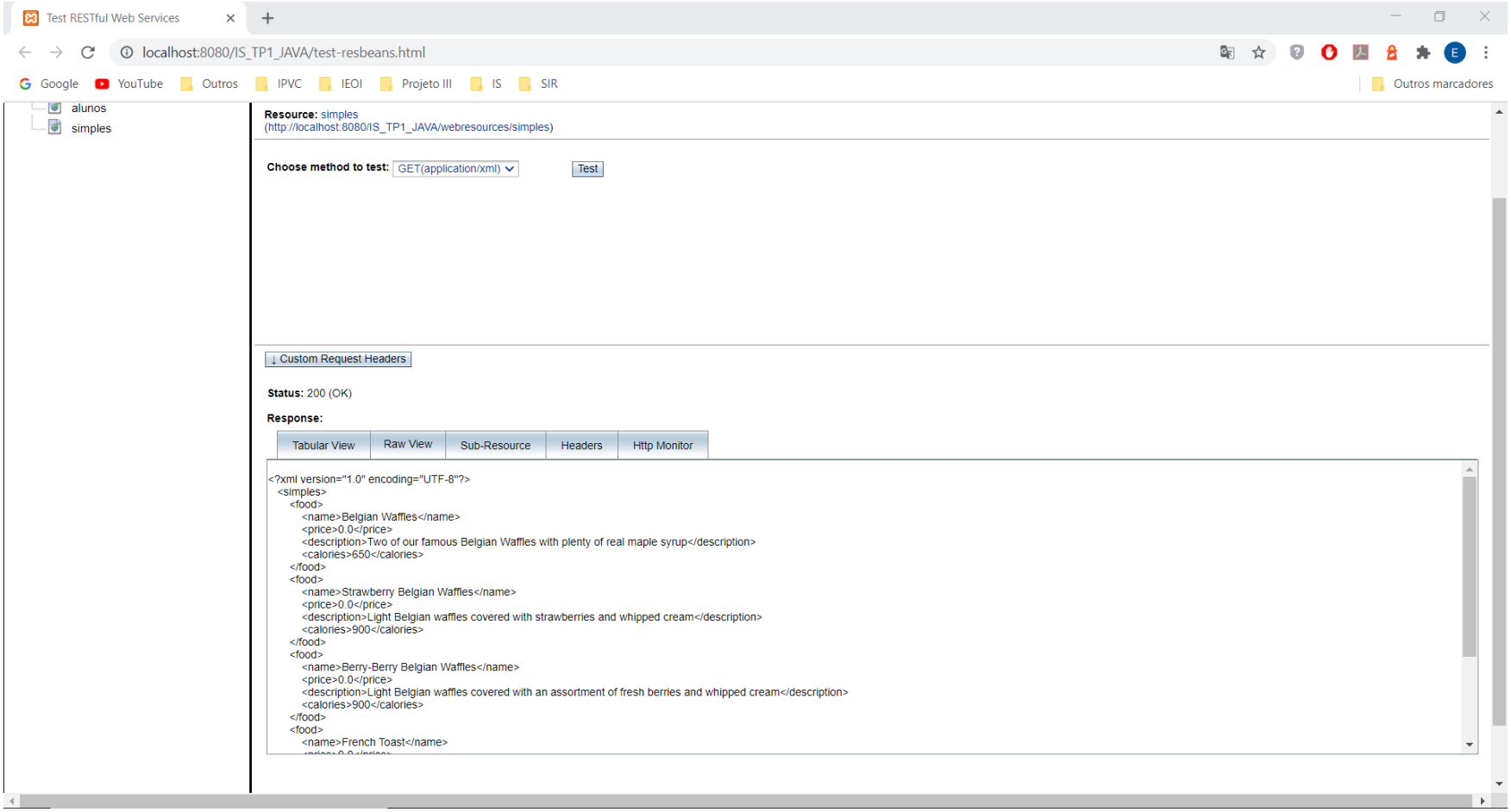
```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <S:Body>
        <ns2:getSimplesResponse xmlns:ns2="http://is2.com/">
            <return>
                <name>Belgian Waffles</name>
                <price>0.0</price>
                <description>Two of our famous Belgian Waffles with plenty of real maple syrup</description>
                <calories>650</calories>
            </return>
            <return>
                <name>Strawberry Belgian Waffles</name>
                <price>0.0</price>
                <description>Light Belgian waffles covered with strawberries and whipped cream</description>
                <calories>900</calories>
            </return>
            <return>
                <name>Berry-Berry Belgian Waffles</name>
                <price>0.0</price>
                <description>Light Belgian waffles covered with an assortment of fresh berries and whipped cream</description>
                <calories>900</calories>
            </return>
            <return>
                <name>French Toast</name>
                <price>0.0</price>
```

■ 3.1.3. SOAP webservice with file simple.xml and JAXB parser

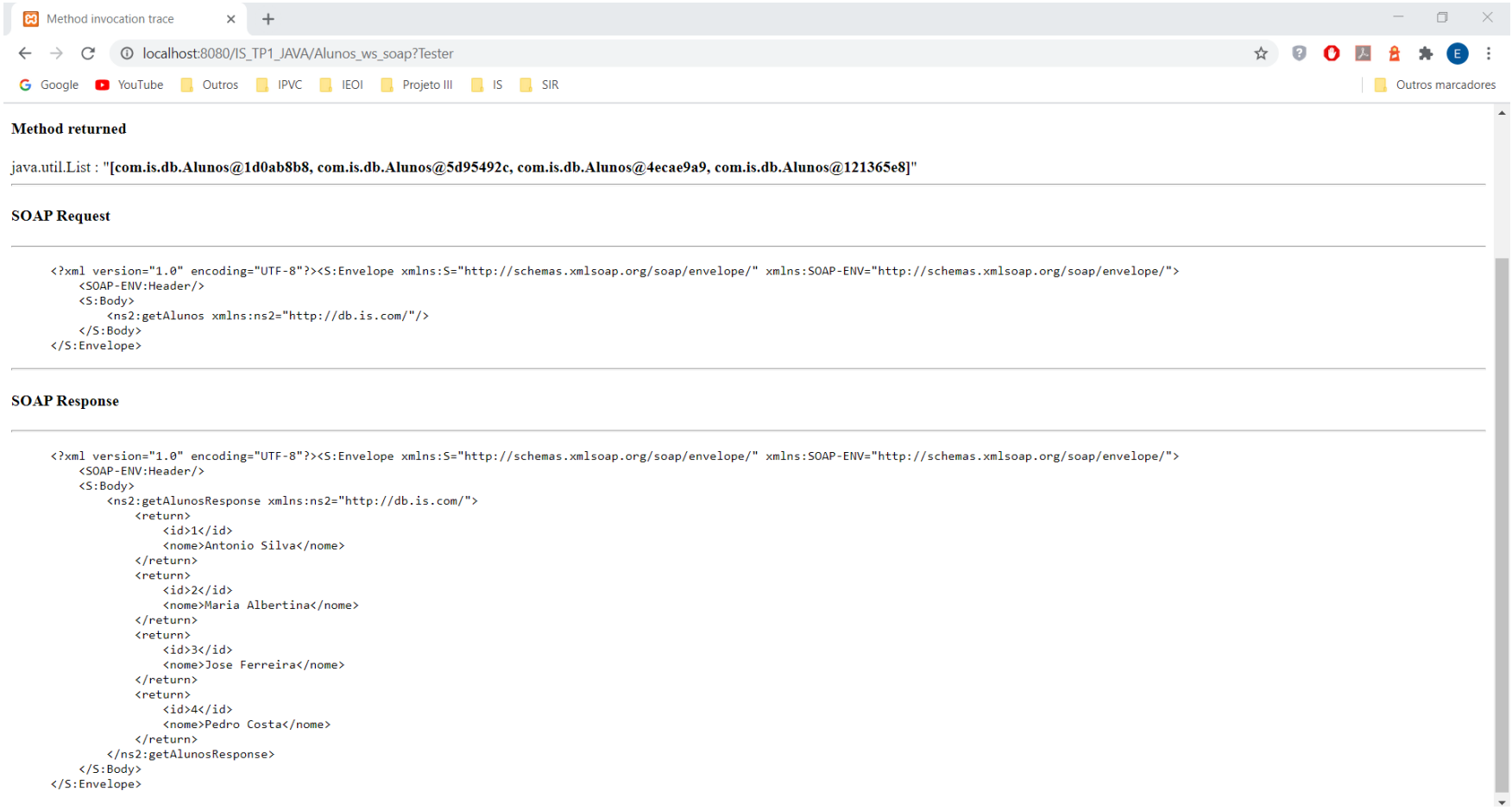




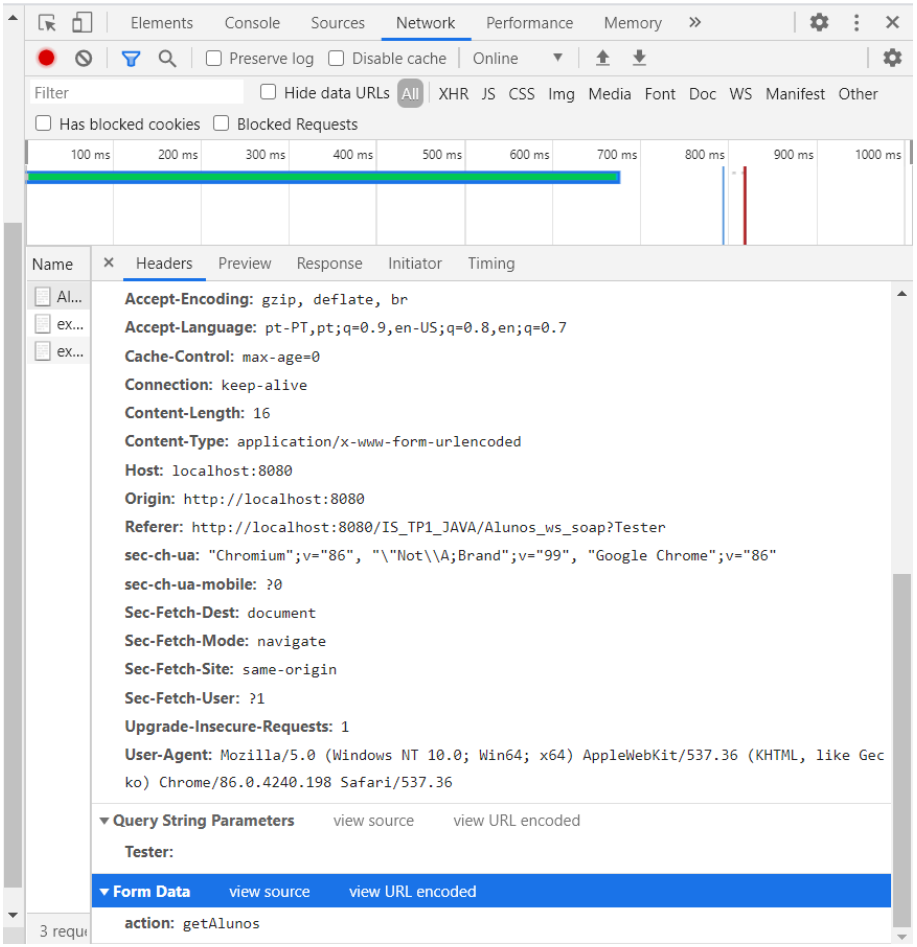
3.1.4. REST webservice with file simple.xml and JAXB parser

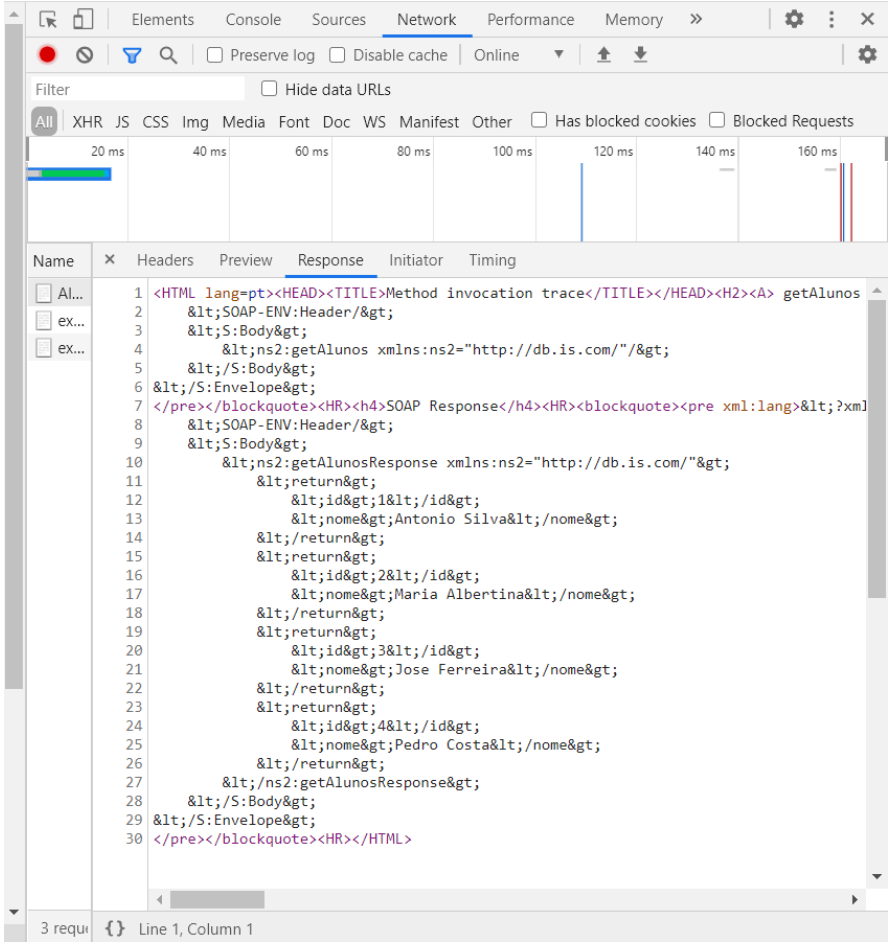


3.1.5. SOAP webservice with file alunos.xml from database

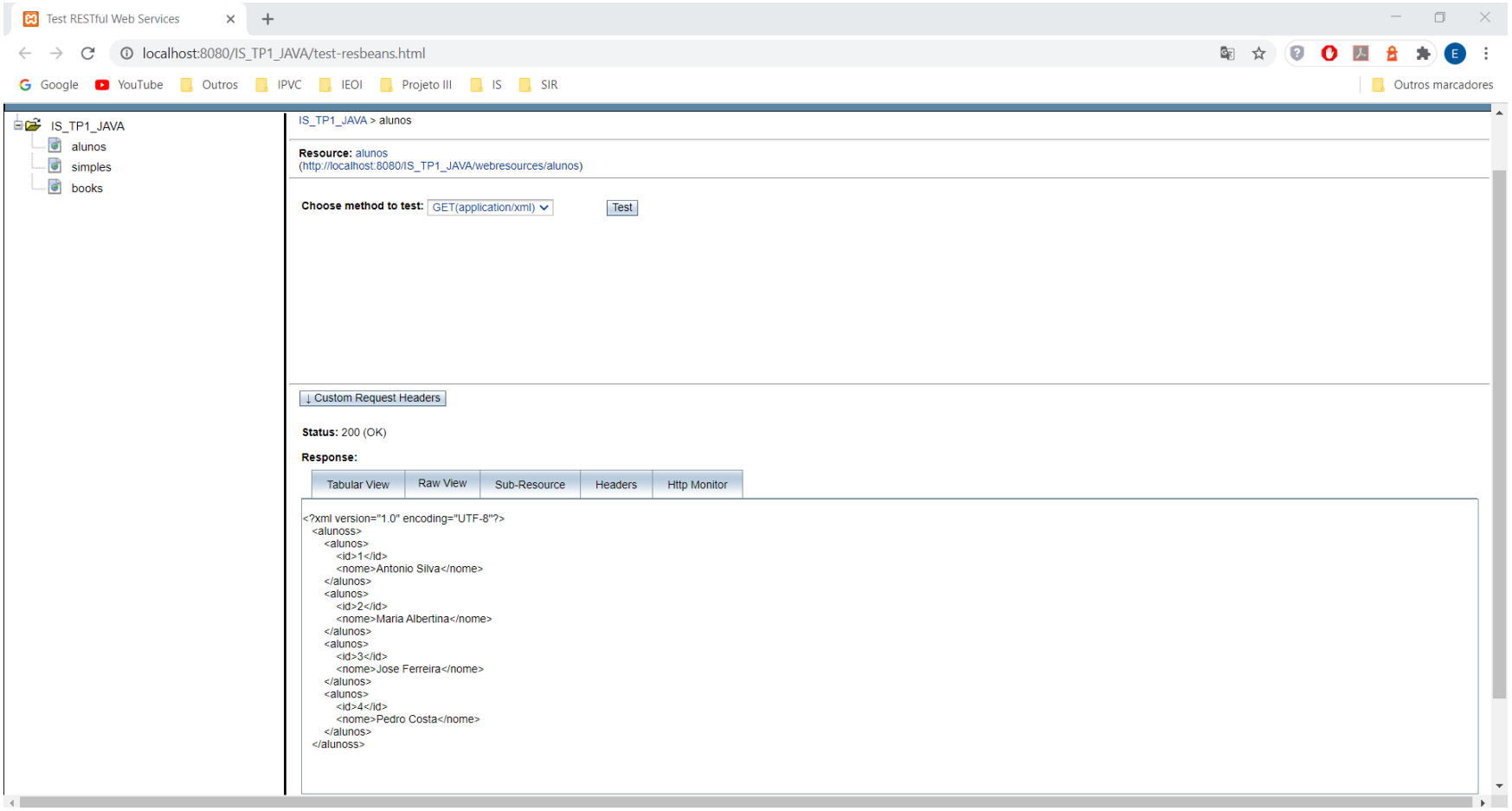


■ 3.1.5. SOAP webservice with file alunos.xml from database





3.1.6. REST webservice with file alunos.xml from database

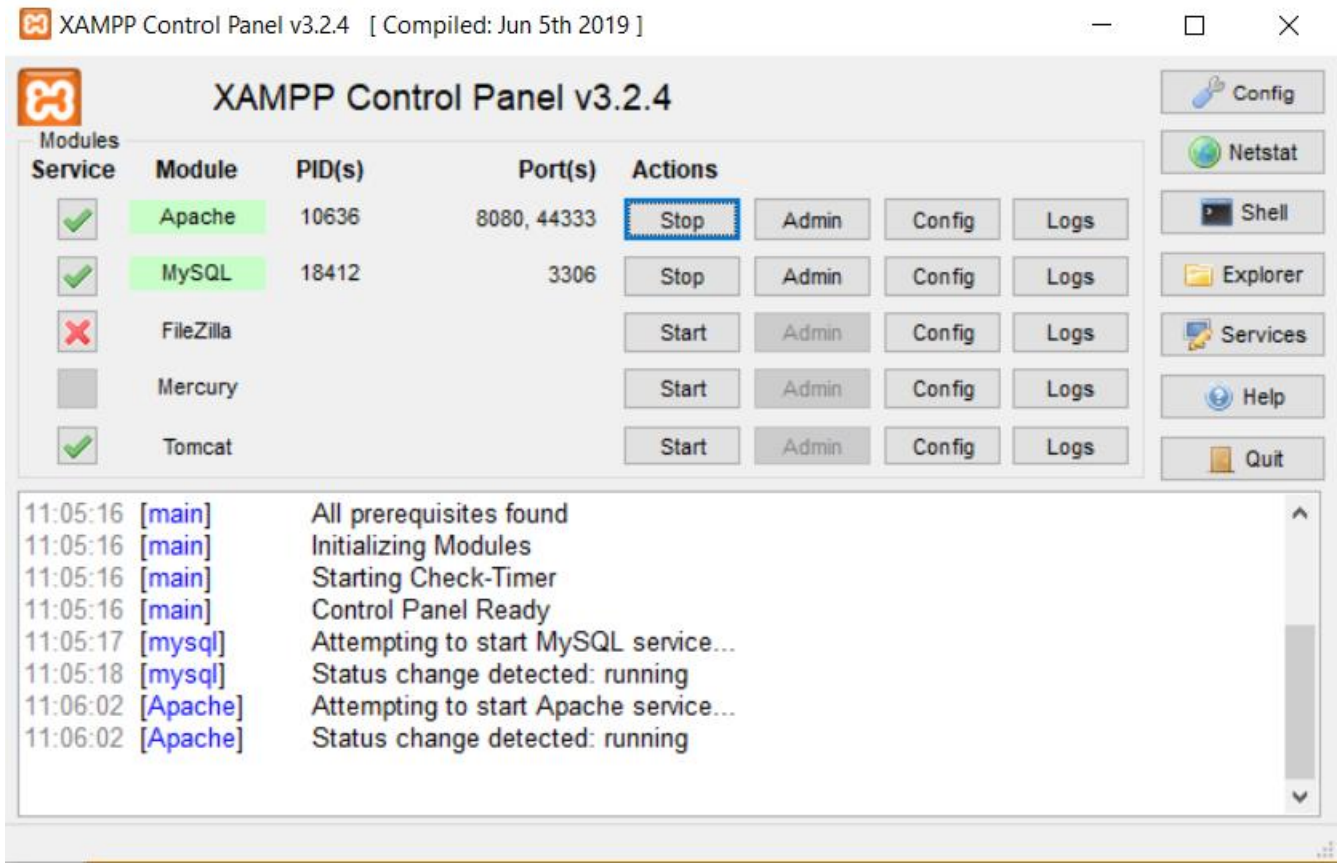


3.2. PHP webservices implementation

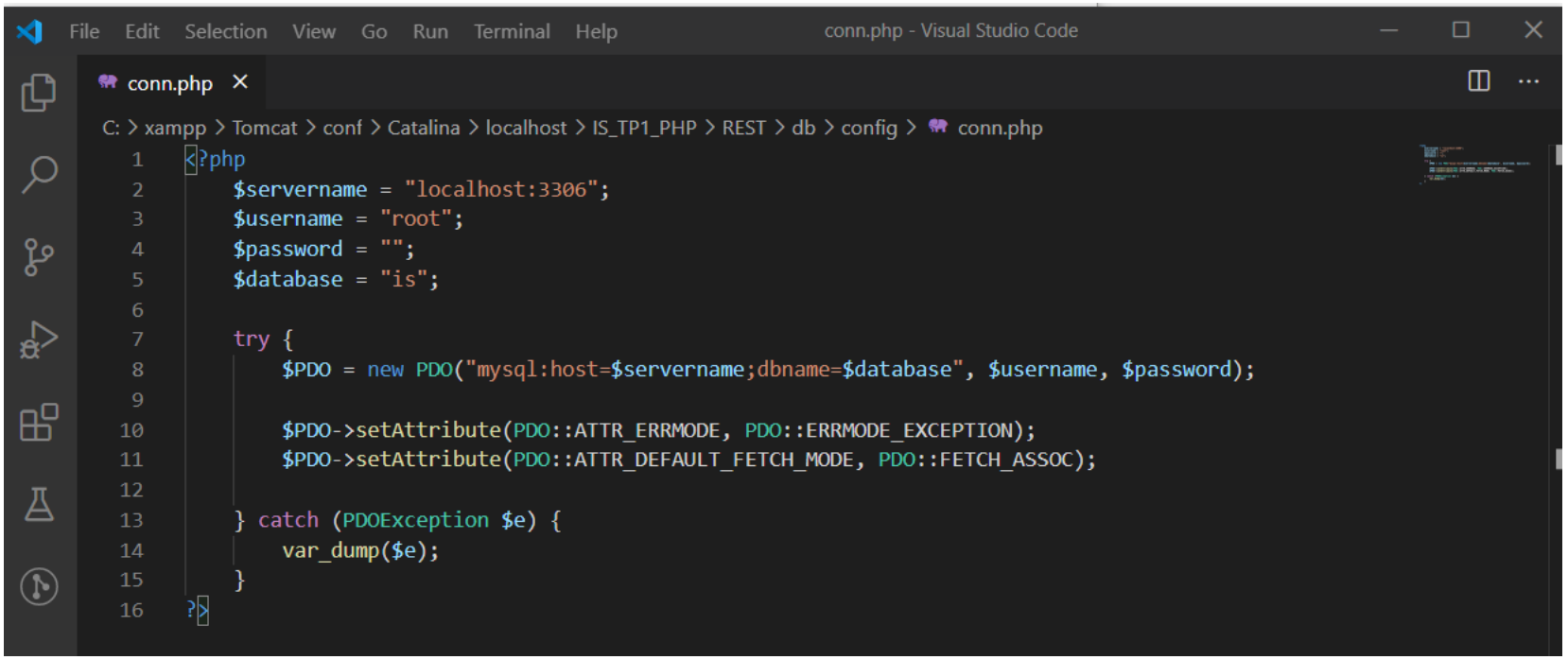
- For the test of the implemented REST and SOAP webservices, it was started a local Apache server using XAMPP, and the php files accessed via web browser.

- For some of the tests it was necessary to change the values of the variables *username* and *password* in the file *conn.php* which configures the connection to the MySQL database running on the port 3306 in the server localhost, as it is possible to confirm on the next slide.

3.2. PHP webservices implementation



3.2. PHP webservices implementation




```php
<?php
    $servername = "localhost:3306";
    $username = "root";
    $password = "";
    $database = "is";

    try {
        $PDO = new PDO("mysql:host=$servername;dbname=$database", $username, $password);

        $PDO->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        $PDO->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);

    } catch (PDOException $e) {
        var_dump($e);
    }
?>
```

3.2. PHP webservices implementation

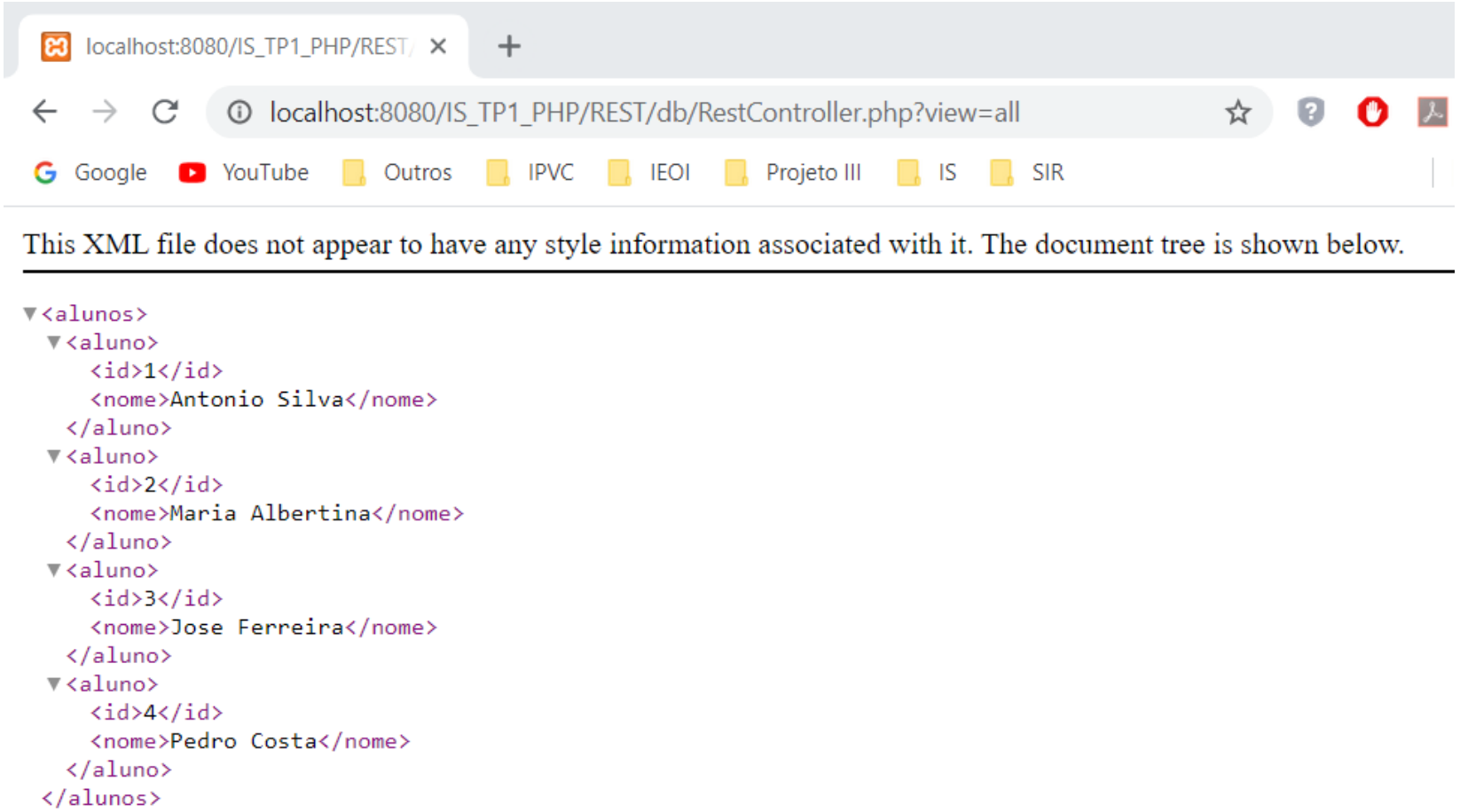
The implementation of a REST webservice using PHP, is done by using four different files:

- One file for the declaration of the class SimpleRest, which implements the REST protocol;
- The second file defines a specific class to get and handle the data form the database or XML file, depending on the example;
- The third, also depending on the example, is optional because it defines and creates the connection to the database using the configuration present in the file *conn.php;*
- The fourth usually contains the class RestController, which uses the specific class (second file) defined to handle the request.

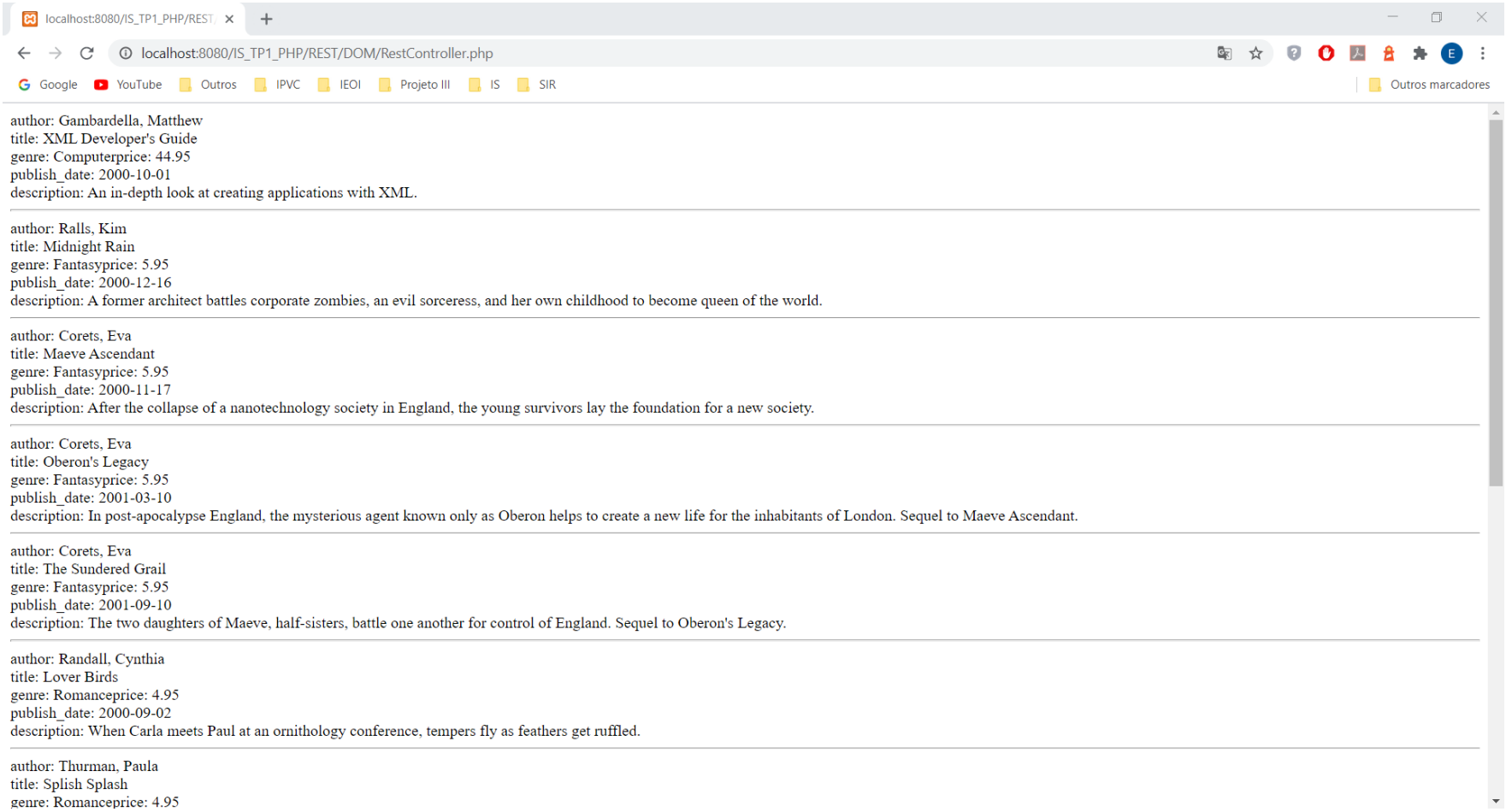
3.2. PHP webservices implementation

- As far as the implementation of webservices using SOAP, it is done with:
  - One file for the client, which uses *nusoap* to make a request, in order to get the data and print the results to the user;
  - The second file is optional, because it defines and executes the connection to the database, and returns the results;
  - The third file defines the server which has the necessary methods to de-serialize the data contained in the XML file or treat the data obtained from the database, depending on the example.

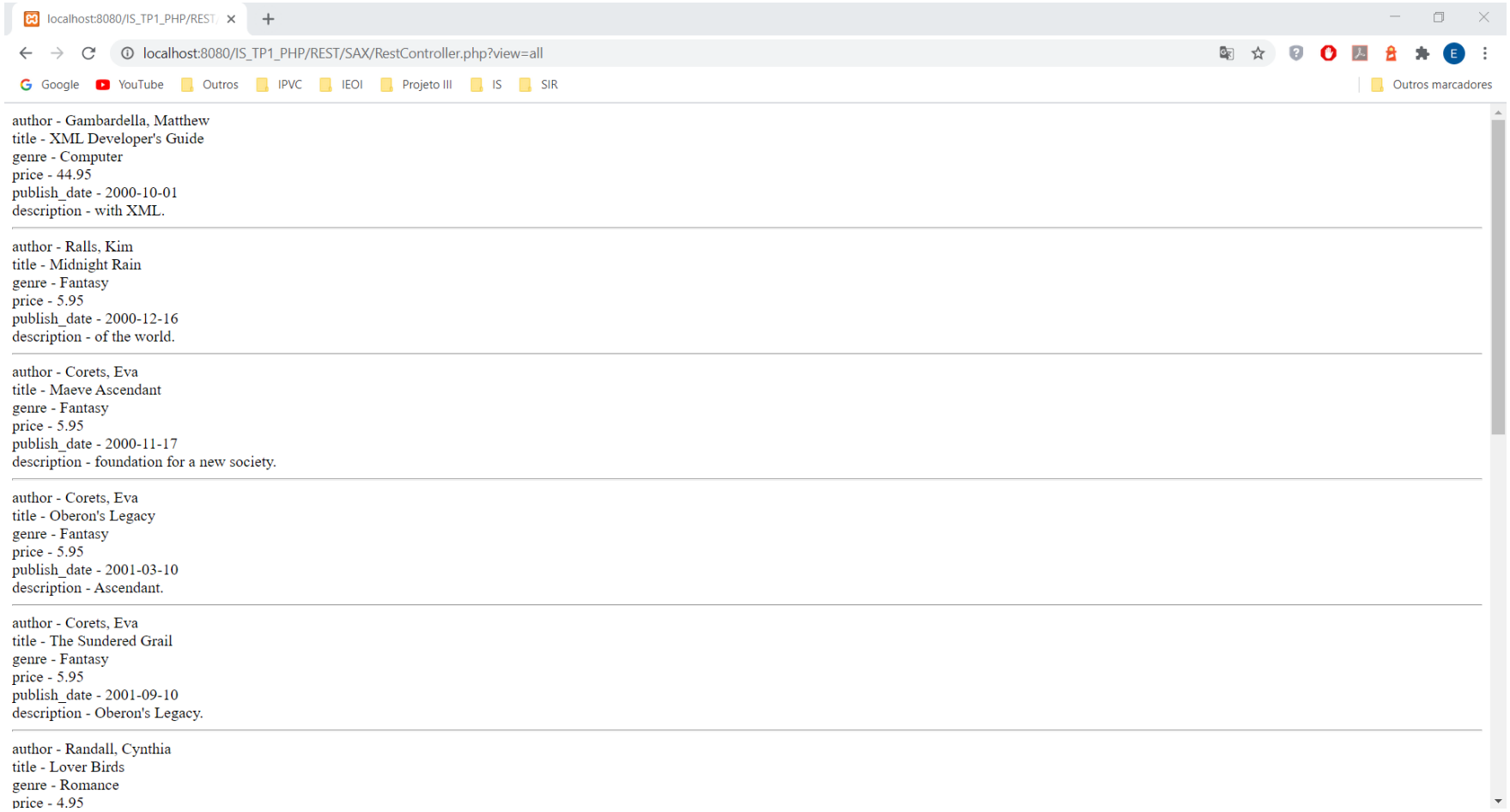
3.2.1. REST webservice to read data form the database



3.2.2. REST webservice using parser DOM/SAX and the books.xml file

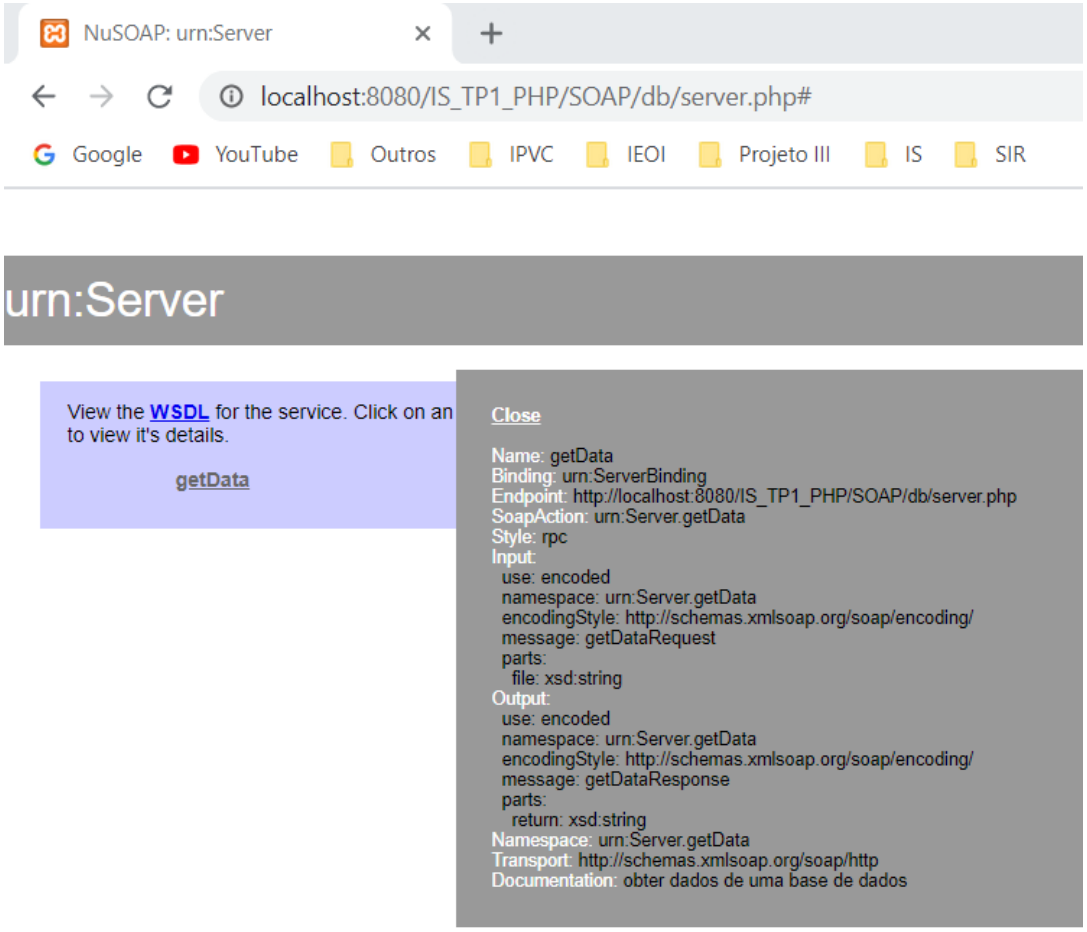


3.2.2. REST webservice using parser DOM/SAX and the books.xml file



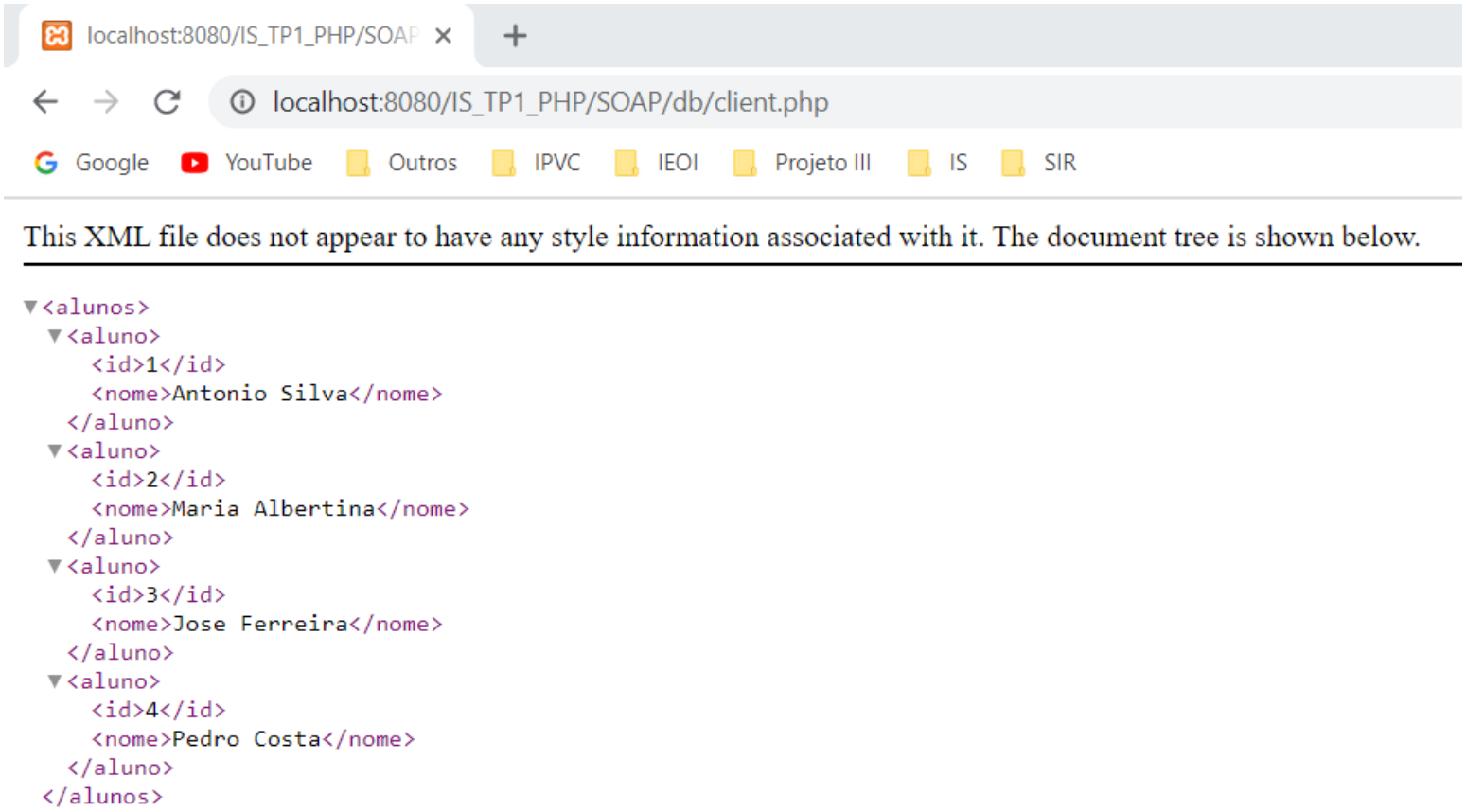
3.2.3. SOAP webservice to read data from the database

- *Server.php* execution

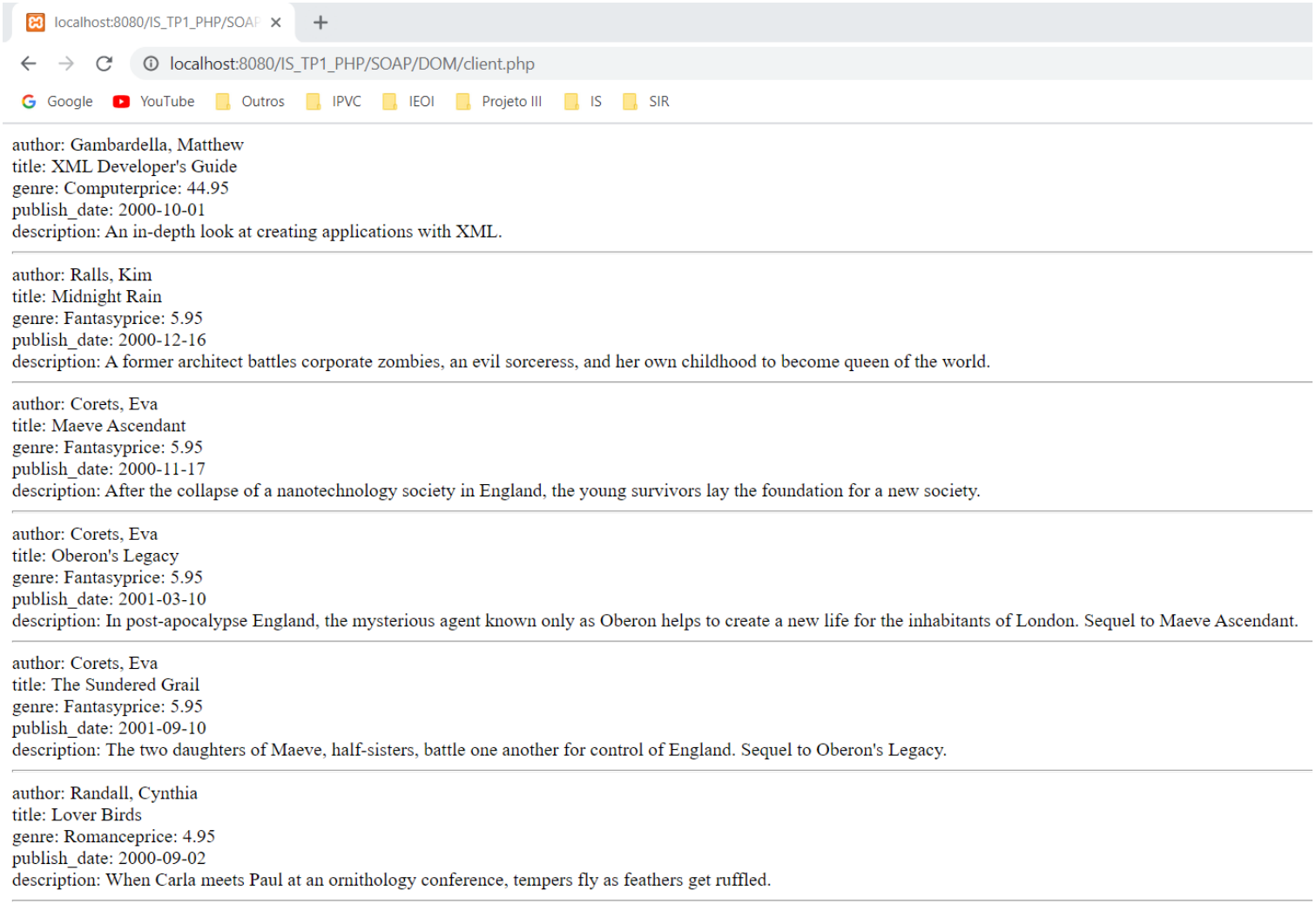


3.2.3. SOAP webservice to read data from the database

- *Client.php* execution



■ 3.2.4. SOAP webservice to read data from the file books.xml using DOM parser



author: Gambardella, Matthew
title: XML Developer's Guide
genre: Computerprice: 44.95
publish_date: 2000-10-01
description: An in-depth look at creating applications with XML.

---

author: Ralls, Kim
title: Midnight Rain
genre: Fantasyprice: 5.95
publish_date: 2000-12-16
description: A former architect battles corporate zombies, an evil sorceress, and her own childhood to become queen of the world.

---

author: Corets, Eva
title: Maeve Ascendant
genre: Fantasyprice: 5.95
publish_date: 2000-11-17
description: After the collapse of a nanotechnology society in England, the young survivors lay the foundation for a new society.

---

author: Corets, Eva
title: Oberon's Legacy
genre: Fantasyprice: 5.95
publish_date: 2001-03-10
description: In post-apocalypse England, the mysterious agent known only as Oberon helps to create a new life for the inhabitants of London. Sequel to Maeve Ascendant.

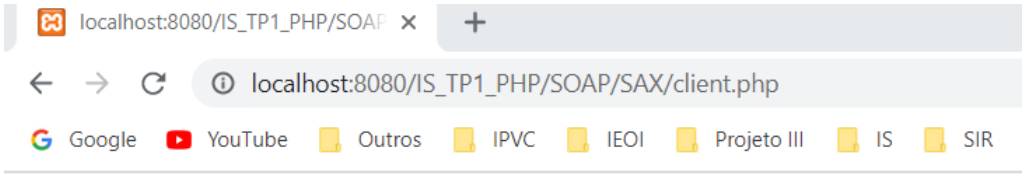
---

author: Corets, Eva
title: The Sundered Grail
genre: Fantasyprice: 5.95
publish_date: 2001-09-10
description: The two daughters of Maeve, half-sisters, battle one another for control of England. Sequel to Oberon's Legacy.

---

author: Randall, Cynthia
title: Lover Birds
genre: Romanceprice: 4.95
publish_date: 2000-09-02
description: When Carla meets Paul at an ornithology conference, tempers fly as feathers get ruffled.

■ 3.2.5. SOAP webservice to read data from the file books.xml using SAX parser



author - Gambardella, Matthew
title - XML Developer's Guide
genre - Computer
price - 44.95
publish_date - 2000-10-01
description - with XML.

---

author - Ralls, Kim
title - Midnight Rain
genre - Fantasy
price - 5.95
publish_date - 2000-12-16
description - of the world.

---

author - Corets, Eva
title - Maeve Ascendant
genre - Fantasy
price - 5.95
publish_date - 2000-11-17
description - foundation for a new society.

---

author - Corets, Eva
title - Oberon's Legacy
genre - Fantasy
price - 5.95
publish_date - 2001-03-10
description - Ascendant.

---

4. Output correction in client side

- In this section, the code in the cliente side for both the SOAP and REST web services implemented in PHP, was altered in order to present the result of the parse of the books.xml file, not in html format but in XML. The code and result can be examined in the next two points.

4.1 REST webservice with the file books.xml and parser DOM

- In this file the only changes made were:

- Commented all the code beneath line 7;
- Added the code on line two, indicating that the php file will return XML;
- Instead of getting the content by its elements and building a string to send to the user, the file is saved as XML and this action printed back to the user.

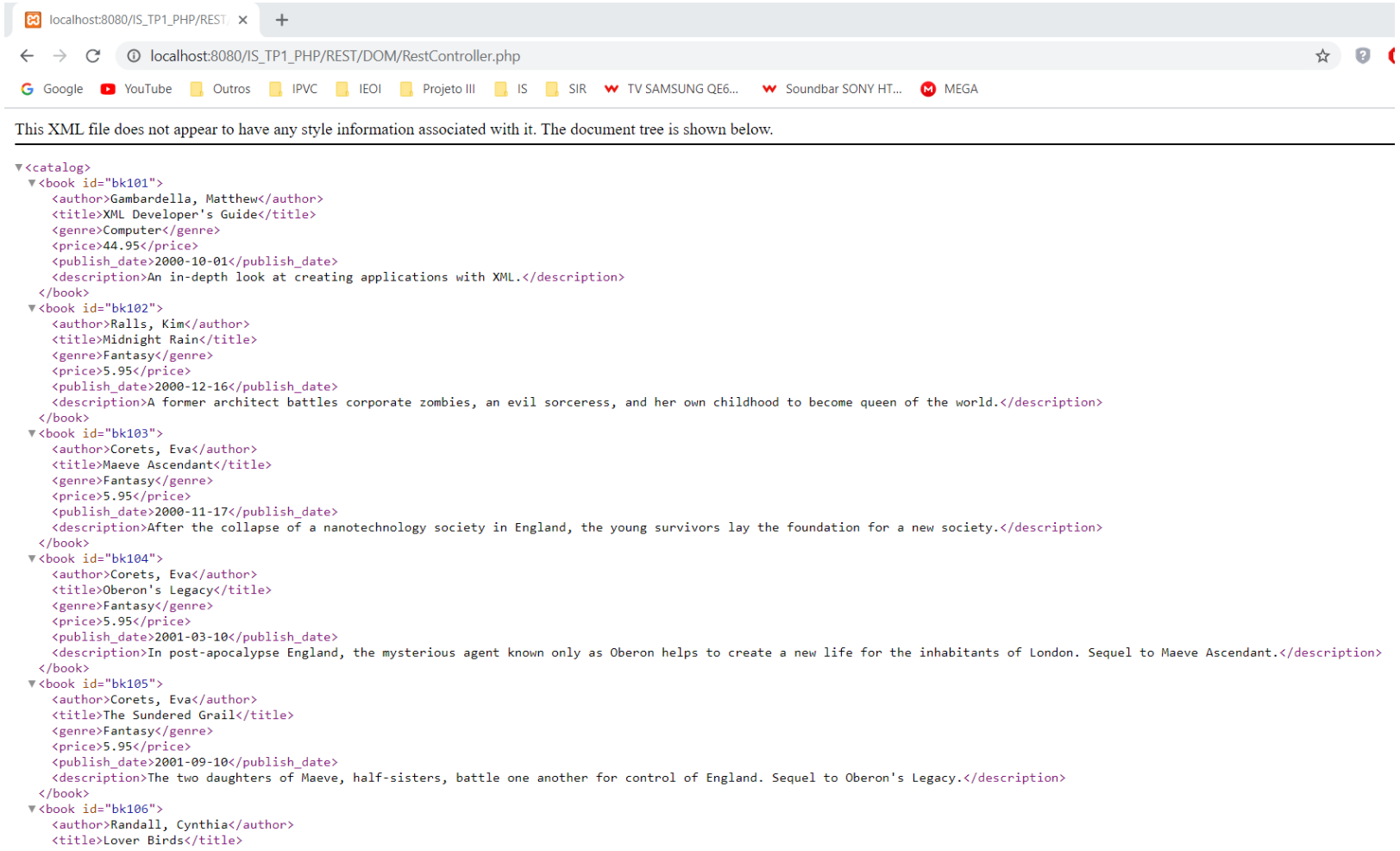
RestController.php | BooksRestHandler.php | readFile.php

C: > xampp > htdocs > IS_TP1_PHP > REST > DOM > readFile.php

```php
<?php
header('content-type: application/xml');
$dom = new domDocument;

    $dom->load("files/books.xml");

    echo $dom->saveXML();
    //$dom->preserveWhiteSpace = false;

    //$tables = $dom->getElementsByTagName('catalog');

    //$rows = $tables->item(0)->getElementsByTagName('book');

    /*
    foreach ($rows as $row) {

        $author = $row->getElementsByTagName('author');
        $title = $row->getElementsByTagName('title');
        $genre = $row->getElementsByTagName('genre');
        $price = $row->getElementsByTagName('price');
        $publish_date = $row->getElementsByTagName('publish_date');
        $description = $row->getElementsByTagName('description');


        echo 'author: '.$author->item(0)->nodeValue.'<br />';
        echo 'title: '.$title->item(0)->nodeValue.'<br />';
        echo 'genre: '.$genre->item(0)->nodeValue;
        echo 'price: '.$price->item(0)->nodeValue.'<br />';
        echo 'publish_date: '.$publish_date->item(0)->nodeValue.'<br />';
        echo 'description: '.$description->item(0)->nodeValue;
        echo '<hr />';
    }
    */
?>
```

4.1 REST webservice with the file books.xml and parser DOM

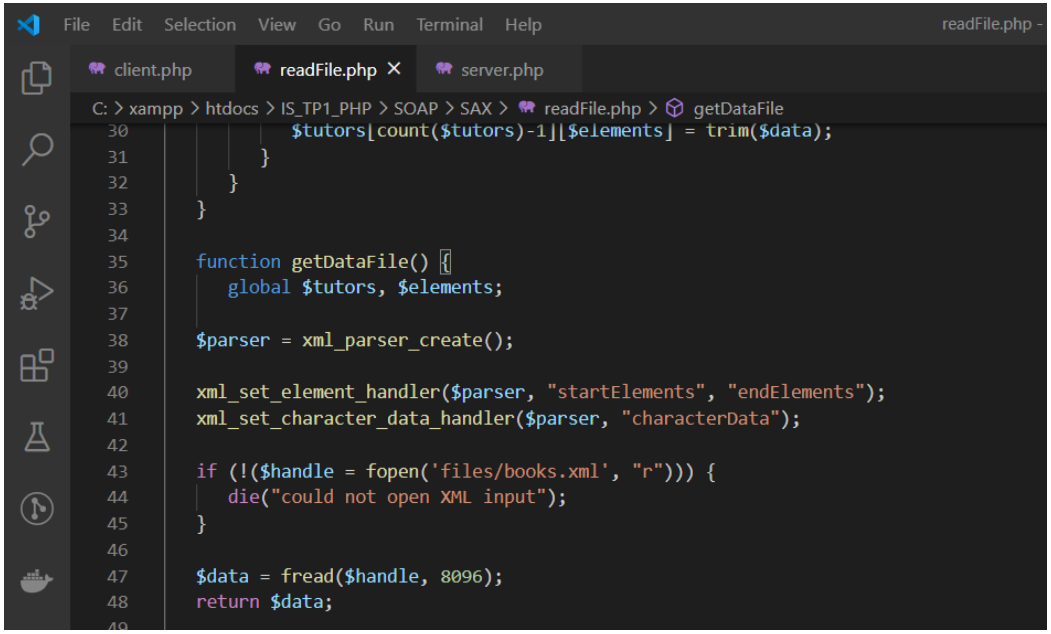
- Obtained result:



4.2 SOAP webservice with the file books.xml and parser SAX

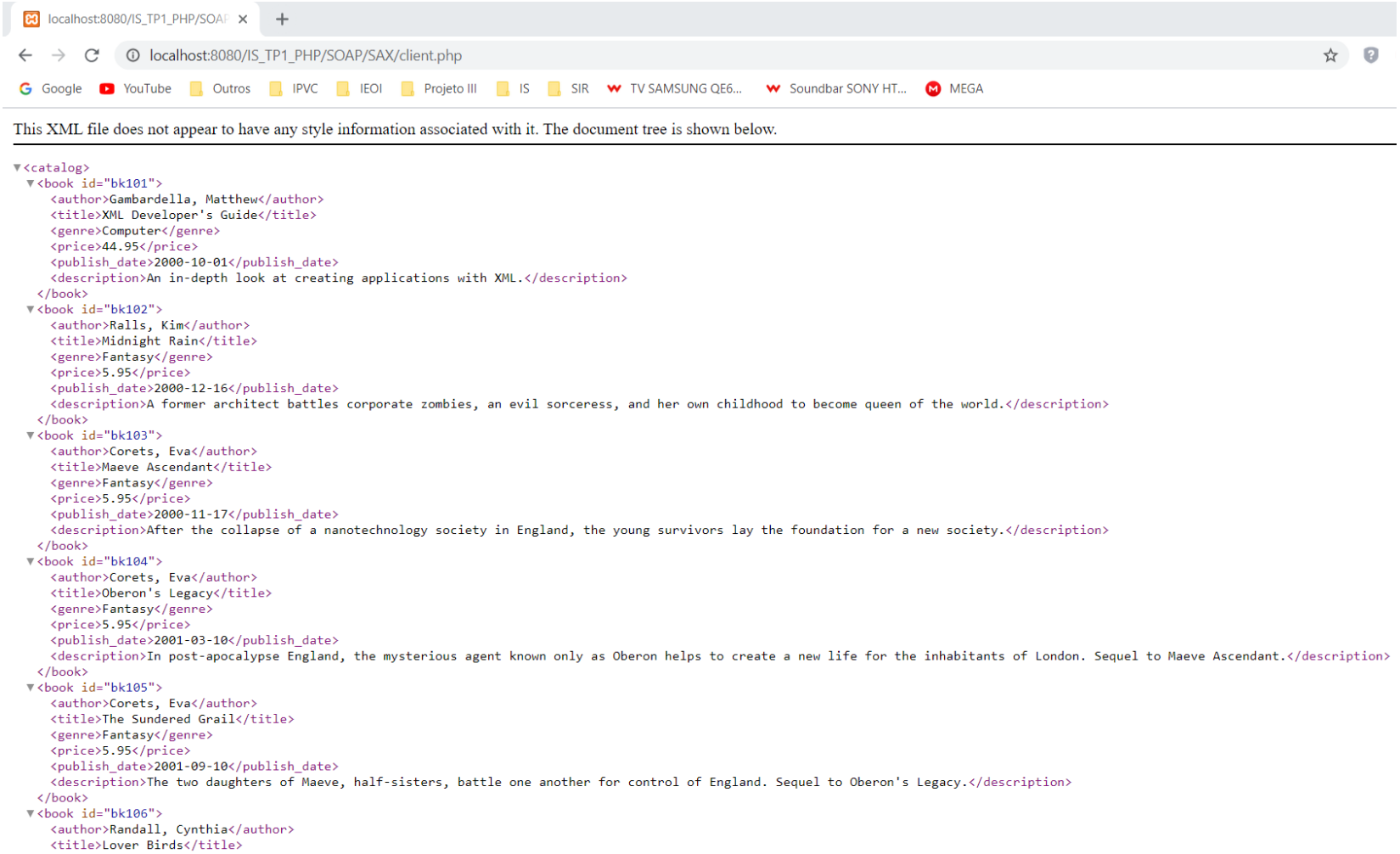
- In this file the changes include:

- Commented all the code beneath line 48;
- The file opened in line 43, has his data fully extracted and directly printed to the client;
- Added line 10 on the client.php file to indicate that the data received is XML;



4.2 SOAP webservice with the file books.xml and parser SAX

- Obtained result:

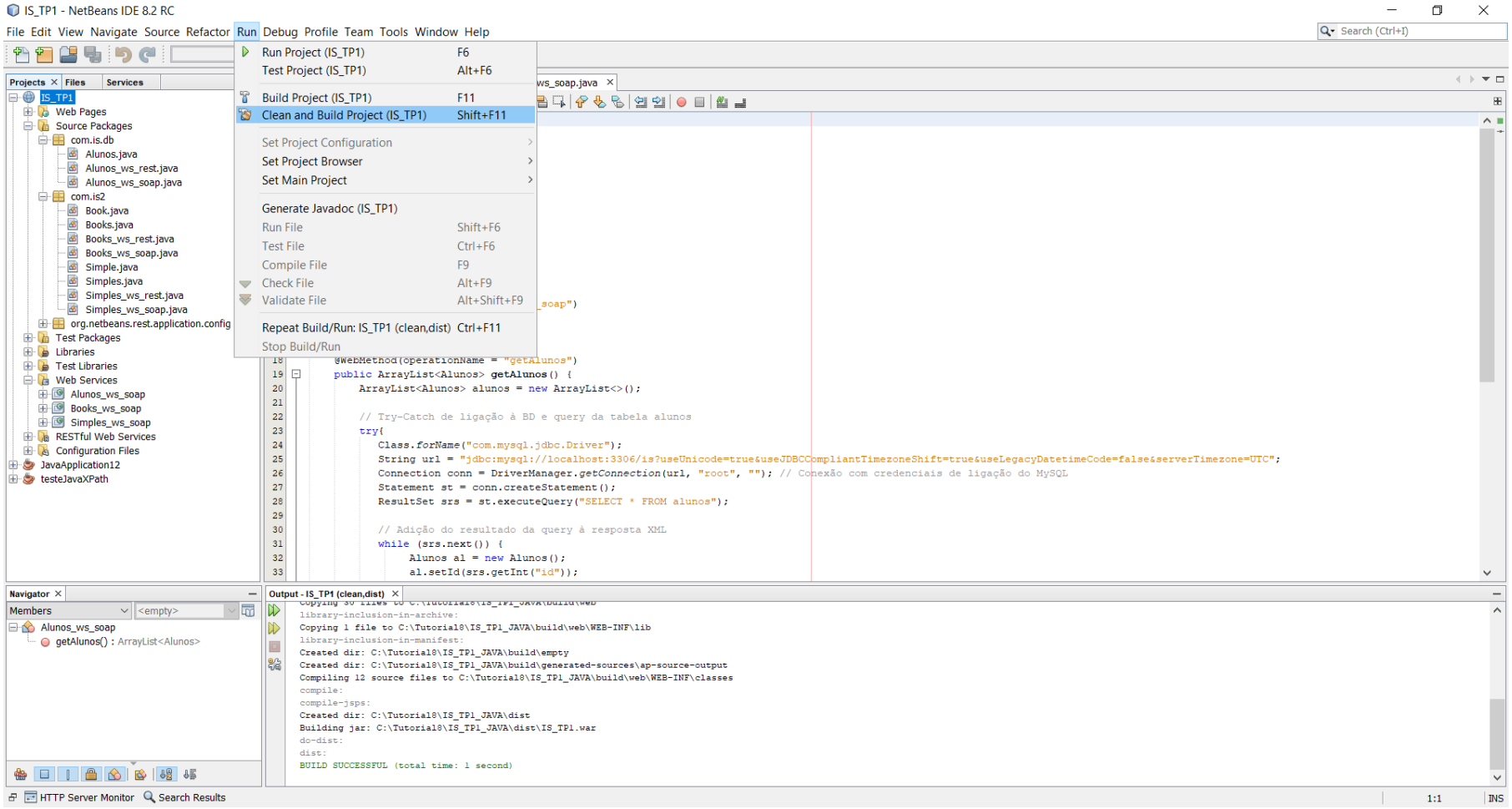


5 Publish a Java project in a Tomcat server

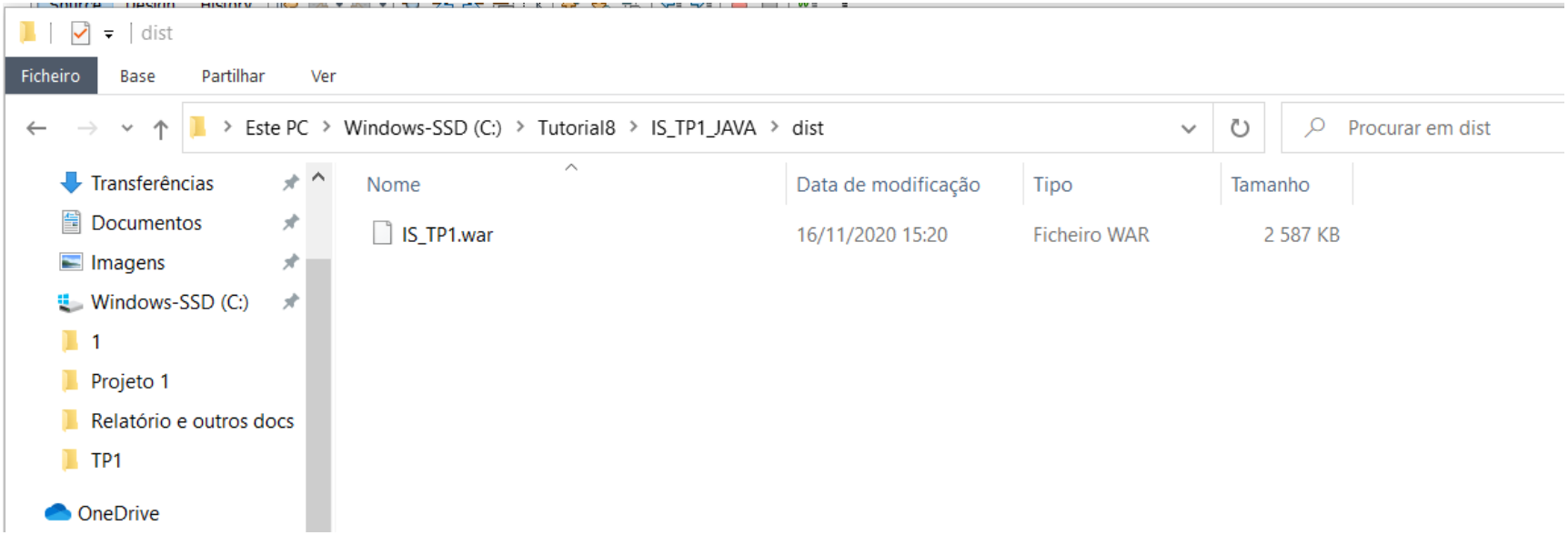
- In this part of the practical work, the main objective is the generation of a *war* file from an existing NetBeans web project. This project includes the code for some SOAP and REST web services implemented in Java, and the purpose of publishing it is to place these web services in the server without allowing access to the source code, functioning outside of the development environment.

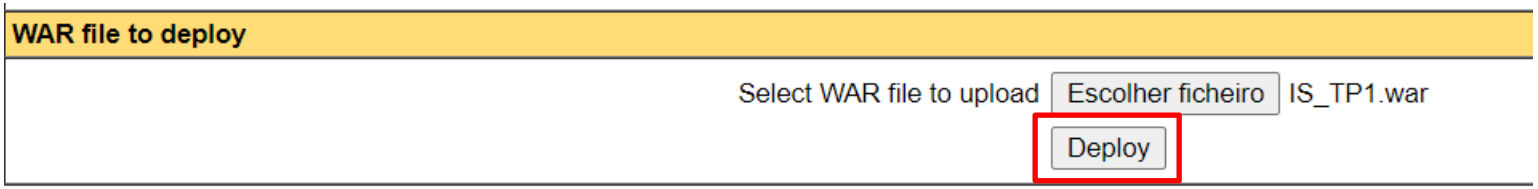
- The first step of this process is to clean and build the project, this will create a war file that can be uploaded in the Tomcat server web application manager, the second step is to select the war file and hit the deploy button and the final step is to check the web application manager and see if the project is deployed in the server.

5 Publish a Java project in a Tomcat server

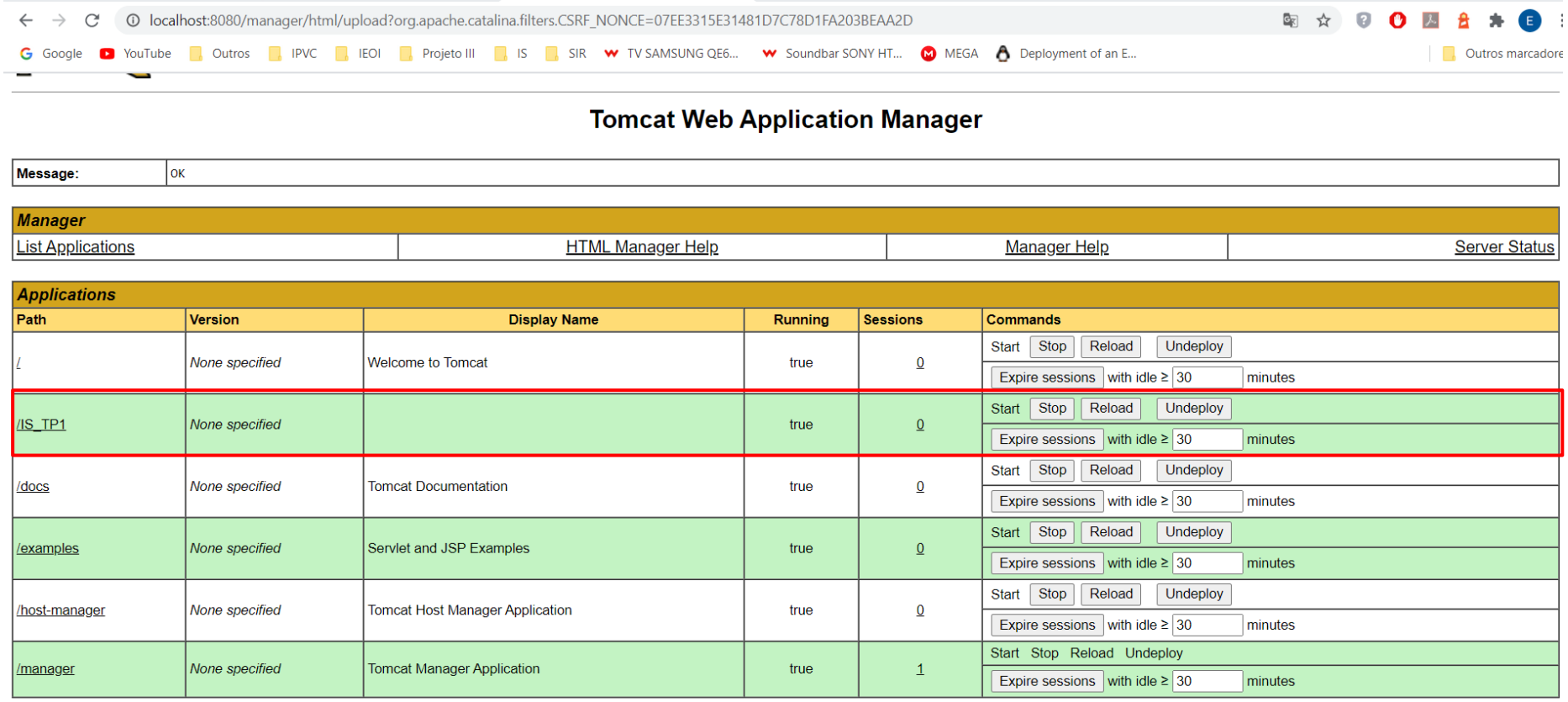


5 Publish a Java project in a Tomcat server





5 Publish a Java project in a Tomcat server



6 XPATH exploration in XML files

- The XPATH exploration was done by changing the code on two files, the first one was the REST web service with parser JAXB and the file books.xml in Java, and the second one the SOAP web service using the parser DOM and the file books.xml in PHP.

- The changes made were the implementation of XPATH, using expressions to filter through the existing books in the referenced file, only obtaining the books where the author value was equal to “Corets, Eva” and “Thurman, Paula”.

6.1 REST webservice with the file books.xml and parser JAXB - Java

- Java code with necessary changes made

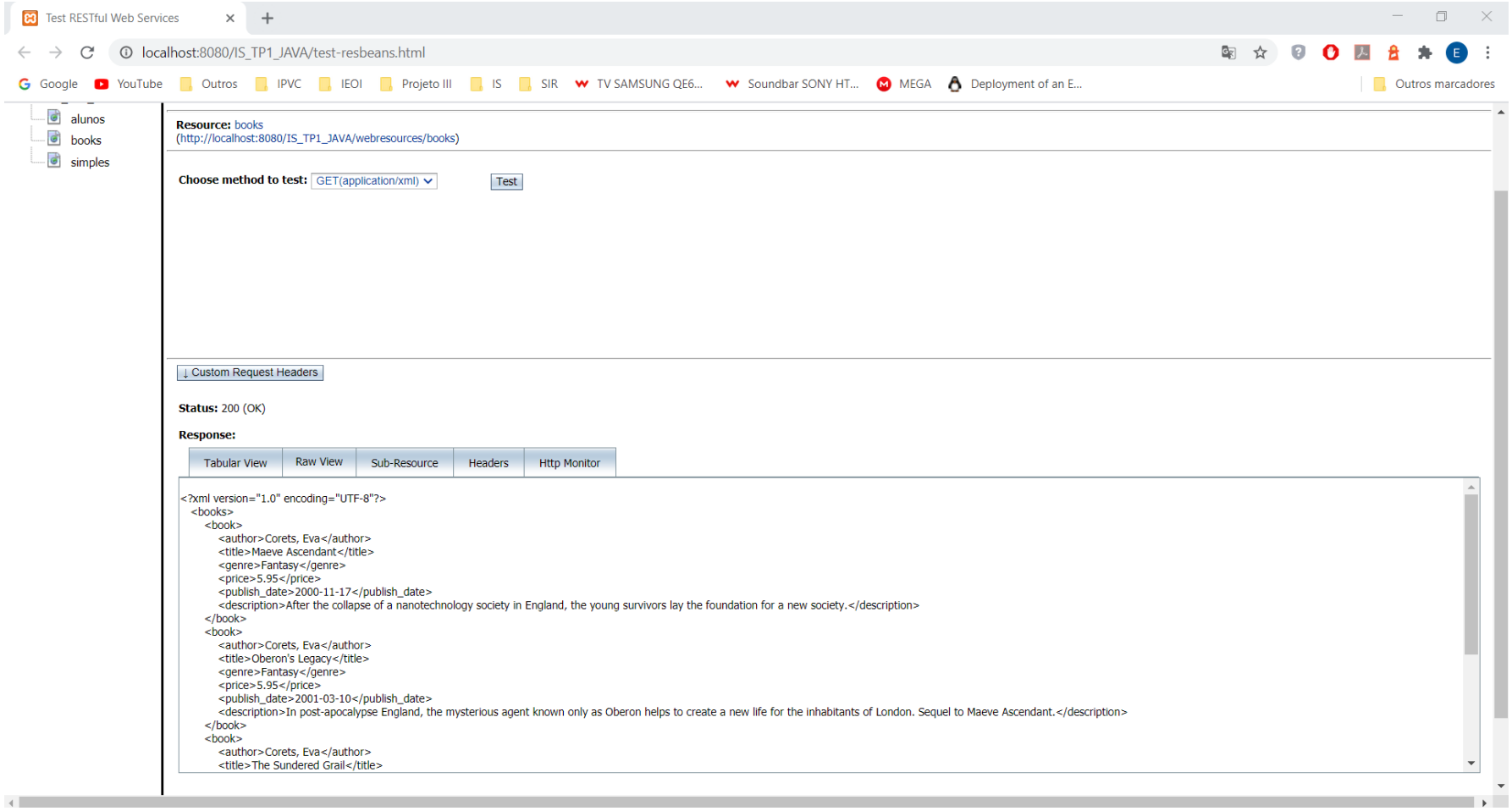
Books_ws_rest.java | Book.java | Alunos_ws_soap.java

Source | History

```
@Produces(MediaType.APPLICATION_XML) // Anotação que permite que a ArrayList seja devolvida em formato XML
public ArrayList<Book> getBooks() throws JAXBException, InvocationTargetException, XPathExpressionException, ParserConfigurationException, SAXException, IOException {

    // Try-Catch do parsing via JAXB do ficheiro
    try {
        JAXBContext jaxbContext; // Parser usado (JAXB)
        File xmlFile = new File("C:\\Tutorial8\\FicheirosXML\\books.xml"); // Cria variável de ficheiro local indicando o path do ficheiro "books.xml"
        jaxbContext = JAXBContext.newInstance(Books.class); // Cria instance do parse do JAXB na classe da lista de livros
        Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller(); // Cria interface responsável para "des-serializar" o ficheiro XML para objeto Java
        Books books = (Books) jaxbUnmarshaller.unmarshal(xmlFile); // Des-serializa o ficheiro XML para objeto Java


        DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder;

        dBuilder = dbFactory.newDocumentBuilder();

        Document doc = dBuilder.parse(xmlFile);
        doc.getDocumentElement().normalize();

        XPath xPath =  XPathFactory.newInstance().newXPath();

        String expression = "/catalog/book[author='Corets, Eva' or author='Thurman, Paula']";
        NodeList nodeList = (NodeList) xPath.compile(expression).evaluate(
           doc, XPathConstants.NODESET);

        ArrayList<String> booksToKeep = new ArrayList<>();
        for (int i = 0; i < nodeList.getLength(); i++) {
           Node nNode = nodeList.item(i);

           if (nNode.getNodeType() == Node.ELEMENT_NODE) {
              Element eElement = (Element) nNode;
              booksToKeep.add(eElement.getElementsByTagName("title").item(0).getTextContent());
           }
        }

        ArrayList<Book> listaBooksFiltrada = new ArrayList<>();
        for(String title : booksToKeep) {
            for(Book book : books.getBooks()) {
                if(book.getTitle().equals(title)) {
                    listaBooksFiltrada.add(book);
                }
            }
        }


        return listaBooksFiltrada;
```

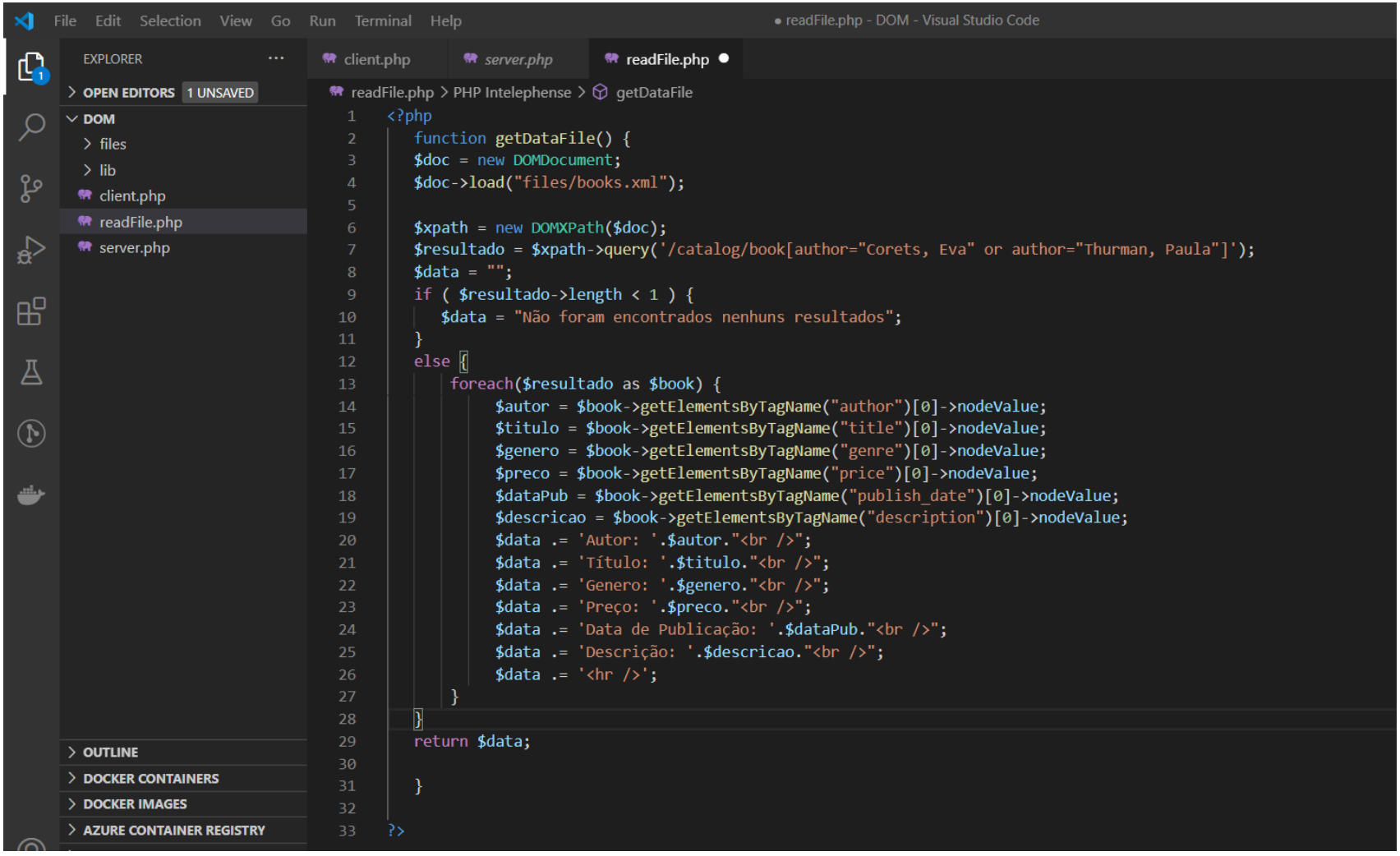
6.1 REST webservice with the file books.xml and parser JAXB - Java

- Web service test and results



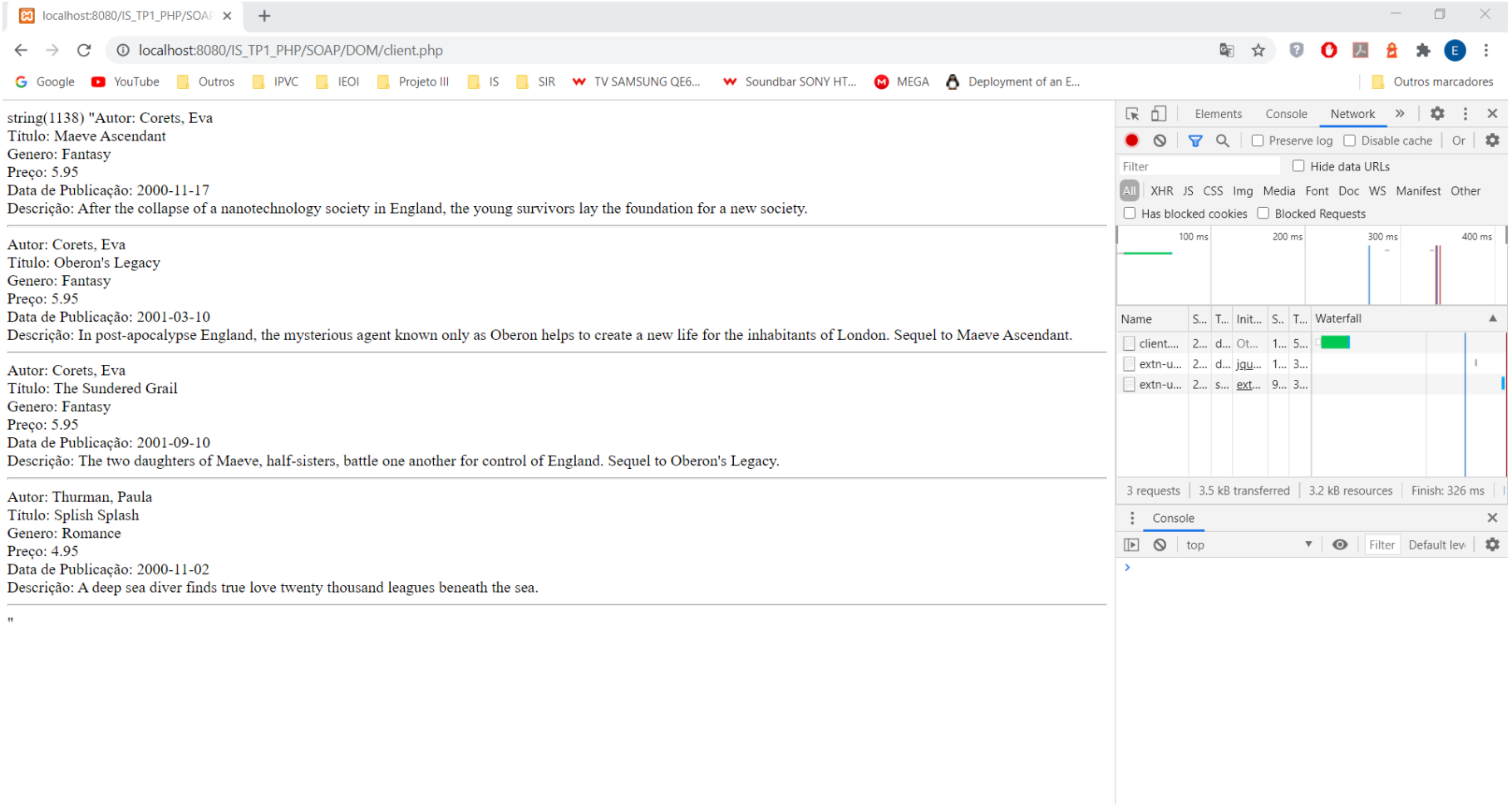
6.2 SOAP webservice with the file books.xml and parser DOM - PHP

- PHP with changes applied



6.2 SOAP webservice with the file books.xml and parser DOM - PHP

- PHP obtained result

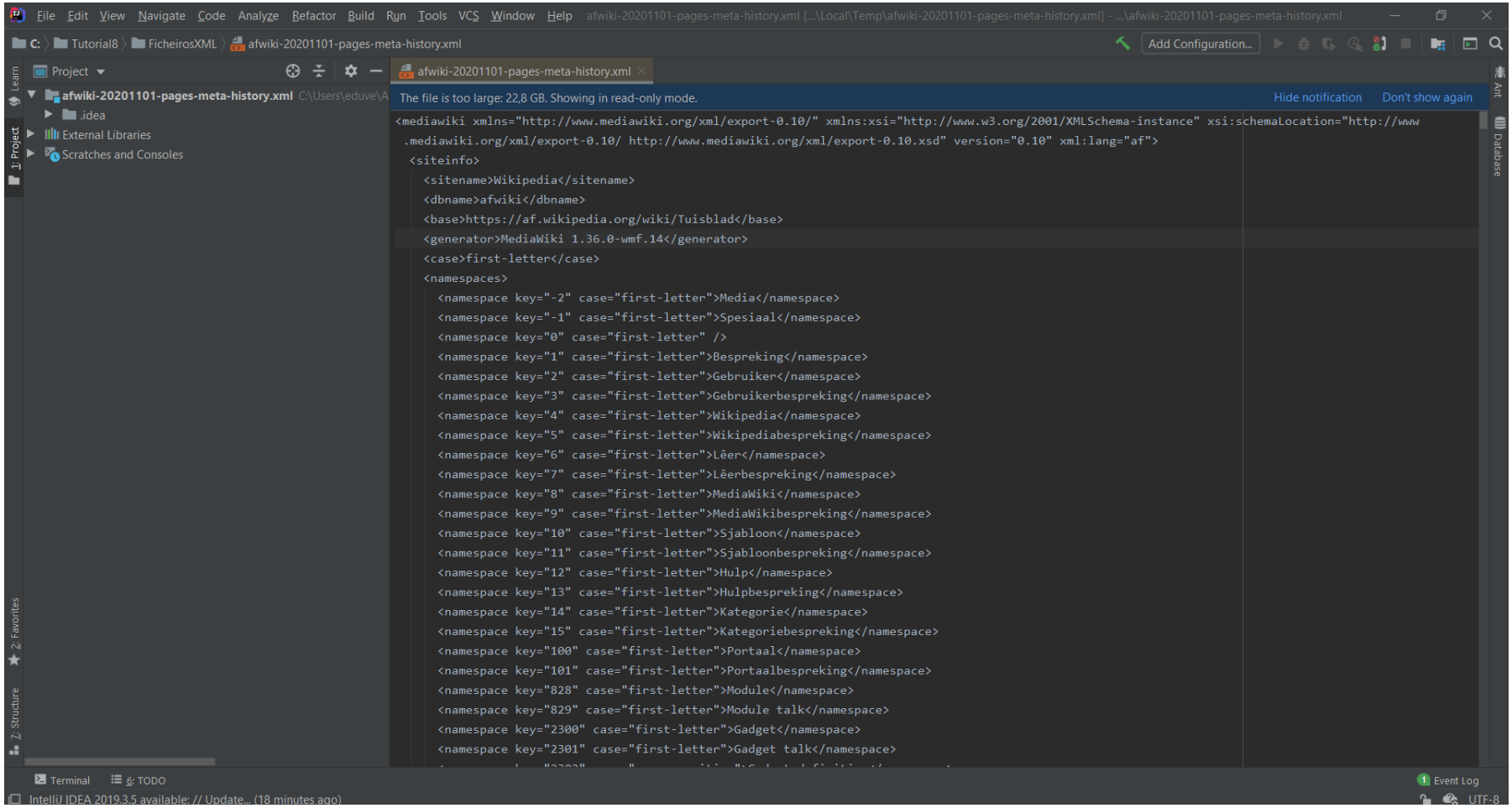


7 Submitting large XML files

- This section demonstrates the attempt of submitting and parsing a large XML file, in this case with 485 MB, to different web services, in Java and PHP. This file was downloaded from https://dumps.wikimedia.org/mirrors.html, and contains several types of data from multiple Wikipedia pages.
- This will be accomplished by changing the code in the REST web service and parser JAXB, in Java to import the new file and create the necessary classes that will be the model of the file, allowing to convert the file into a Java object. In the SOAP web service and parser DOM in PHP, it was only changed the code to import the new file, parsing and printing its contents to the client.

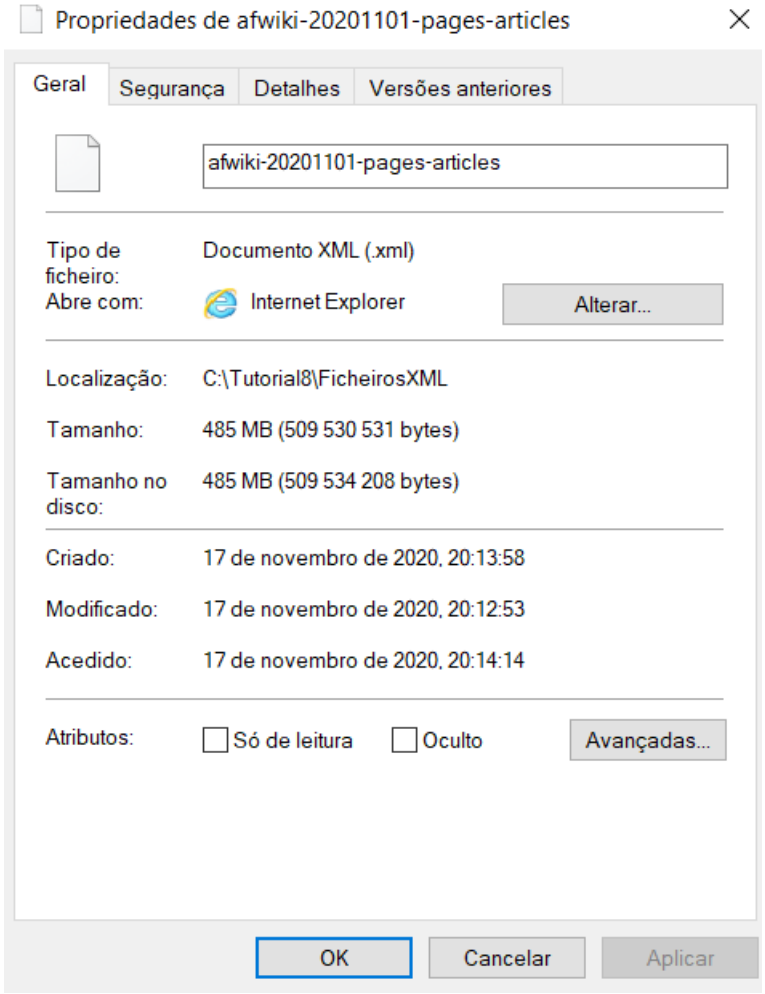
7 Submitting large XML files

- afwiki-20201101-pages-articles.xml file structure



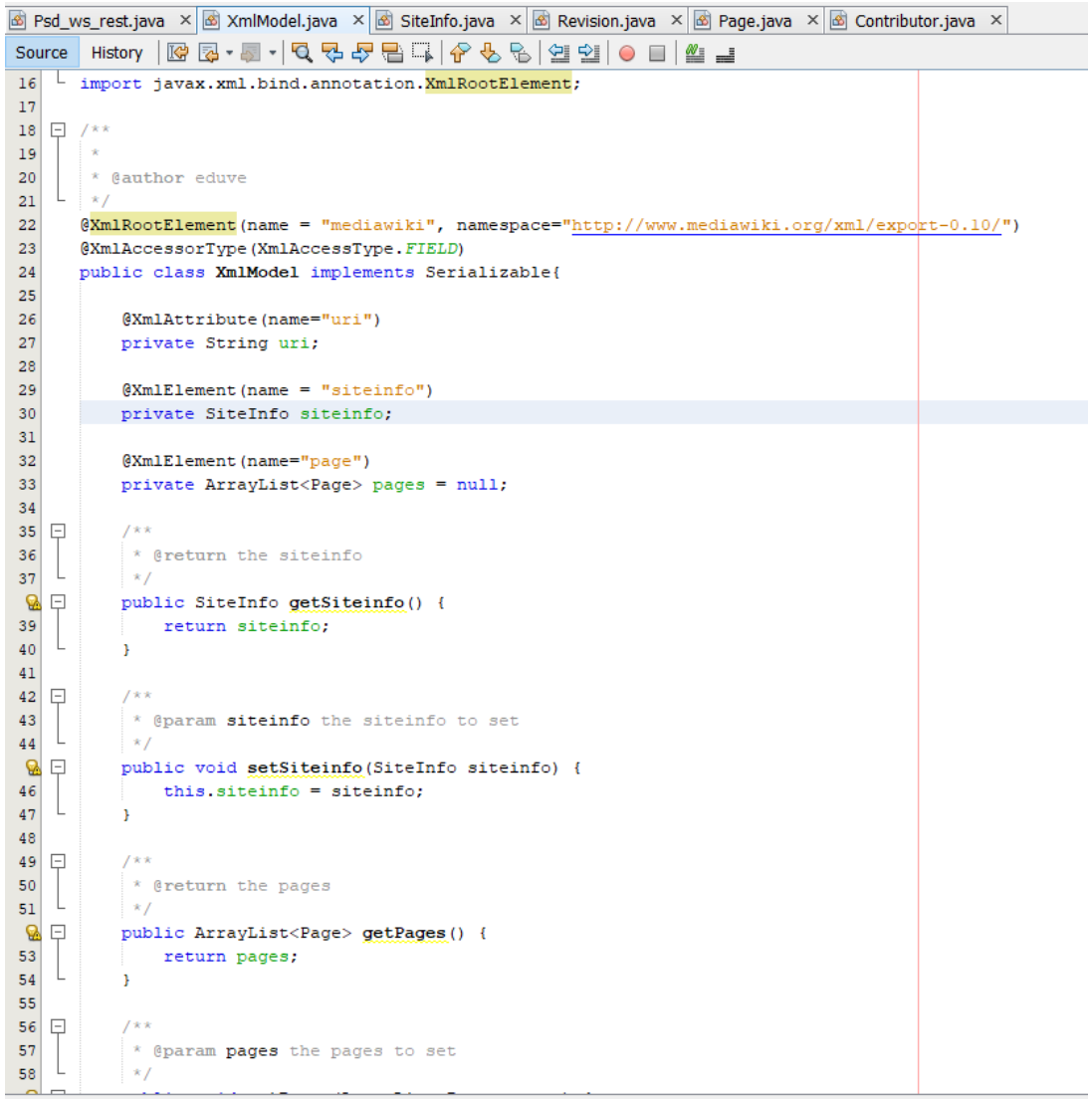
7 Submitting large XML files

- afwiki-20201101-pages-articles.xml file details



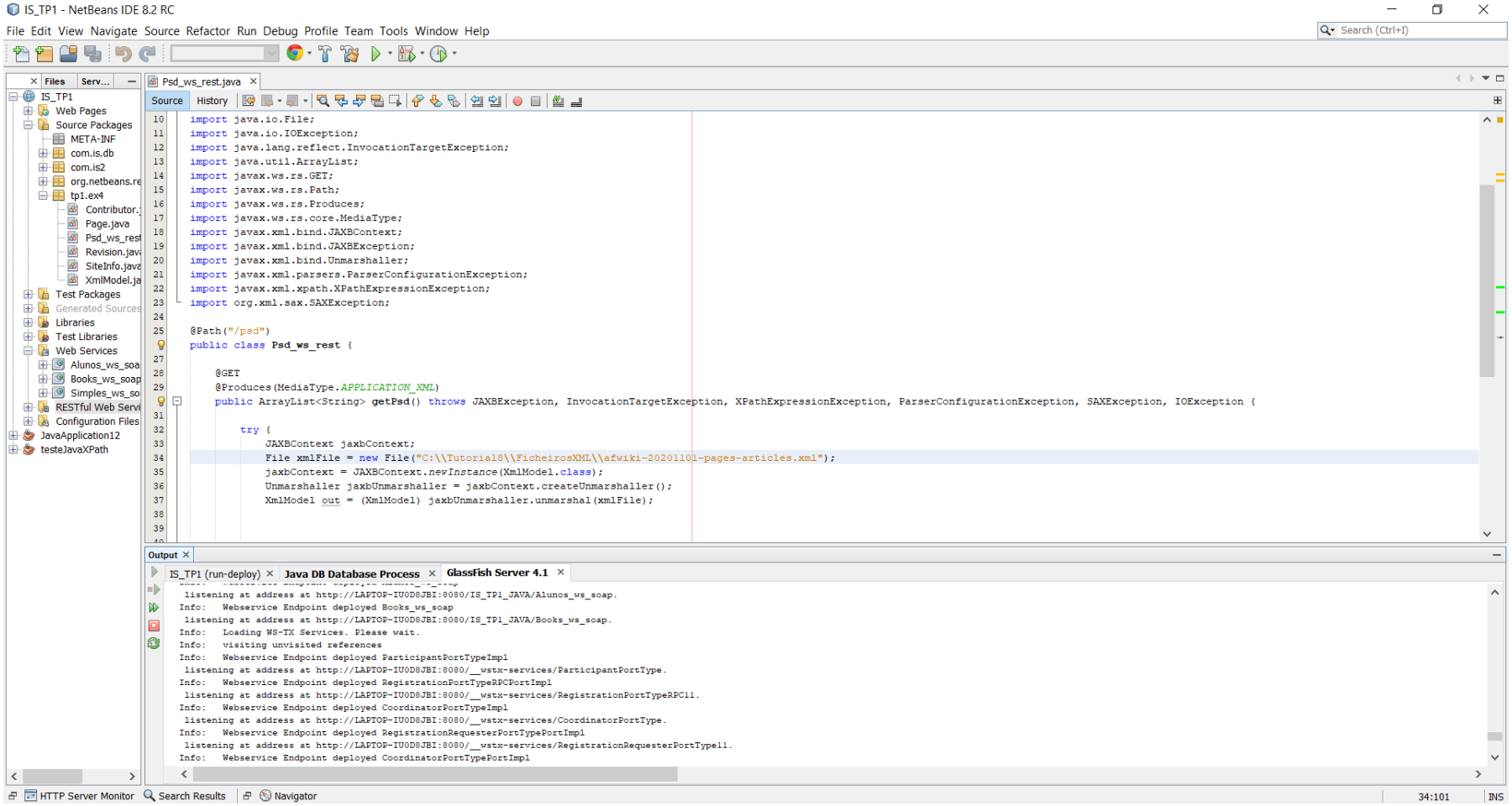
7.1 REST webservice and parser JAXB - Java

- Java class XmlModel, to represent the structure of the xml file used.
- The rest of the classes can be examined by opening the project and analyzing the code.



7.1 REST webservice and parser JAXB - Java

- Attempt to load and parse the xml file with out of memory error.



7.2 SOAP webservice and parser DOM – PHP

- afwiki-20201101-pages-articles.xml imported in the readFile.php file

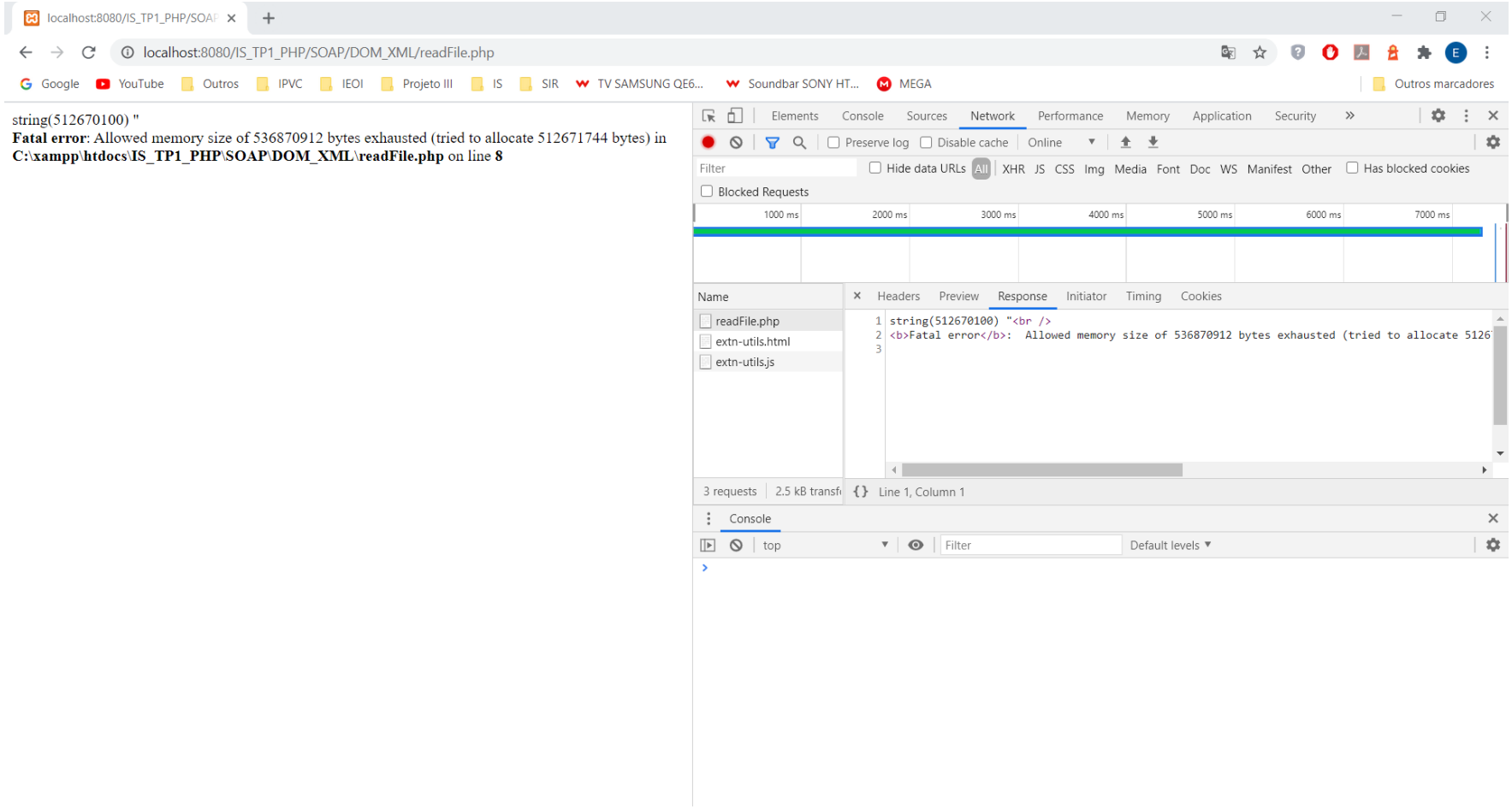
readFile.php

SOAP > DOM_XML > readFile.php > ...

```php
<?php
    getDataFile();
    function getDataFile() {
    try {
        $dom = new domDocument;

        $dom->load("files/afwiki-20201101-pages-articles.xml");

        echo $dom->saveXML();
    } catch(Exception $e) {
        echo $e->getMessage();
    }
```

7.2 SOAP webservice and parser DOM – PHP

- Attempt to parse and load the xml file with out of memory error.



7.3 Problems and approaches to process large files

- Main problems in processing and parsing a large files [8]:

- The first and more obvious problem is the lack of memory on the server side, which results in a fatal error and the immediate termination of the process during the processing stage of the referenced file. This error was the one verified in the attempts registered in the points 7.2 and 7.3, where both the glassfish server and apache server could not parse and process the file due to the allowed memory size for these servers.
- Being the file large, the transfer could be interrupted because of the server timeout, not receiving the file consequently.
- The server includes a limit to the files uploaded, if the file's size is over the imposed limit, the connection is severed, and the file is not uploaded.

7.3 Problems and approaches to process large files

- Ways to handle the upload and transfer of large files [9]:

- The most common way to be able to upload and transfer large files, is breaking them into several pieces and processing them one at a time. This process has one big advantage, and it is that if eventually one of the chunks arrive with errors the process can restart from where it was without losing all the progress done until that point.
- A second way could involve asynchronous uploads, where the file is uploaded in the background allowing the software to continue functioning, reducing the risks of a timeout or memory caps.
- Uploading the file directly to a cloud service and accessing it from there could be an option but it has drawbacks, like the long upload time if it needs to be processed in several locations before it reaches the cloud.

7.3 Problems and approaches to process large files

- What would need to be changed in order to receive a large file in the Java or PHP examples:

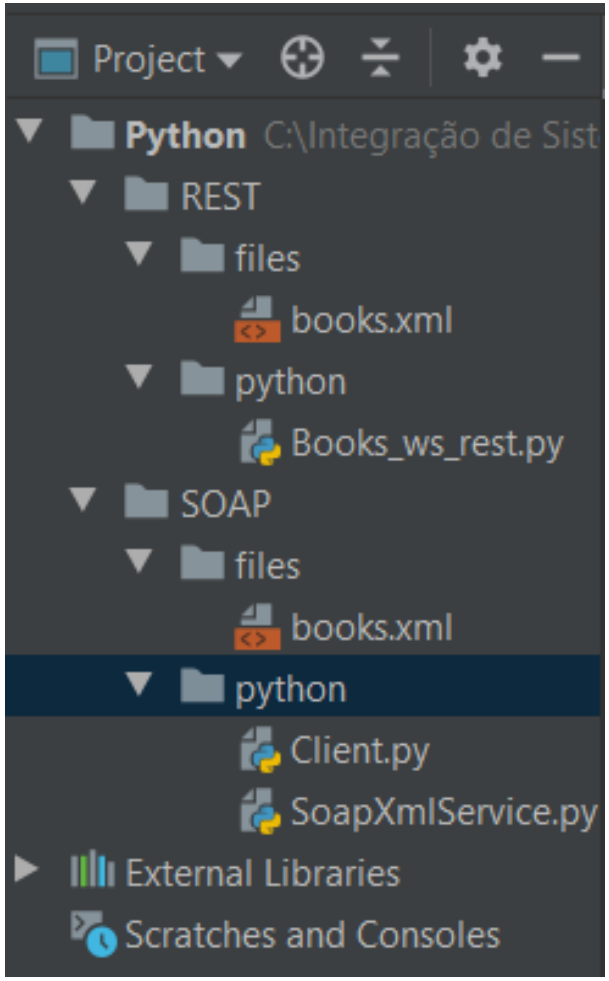
- In Java, a solution could be the use of the *AsyncXMLInputFactory*, in order to parse the incoming XML file in an asynchronous way, parsing it in the background while the service continues to run. This also allows to pull parts of the file, parsing it piece by piece, reducing the risk of hitting the server's memory and timeout caps [10].
- In PHP, instead of using an XML parser like DOM, which loads all the contents of the XML file into memory, using the XMLReader and the functions it provides, it would be possible to pull one or more elements at a time from the file, loading them into memory, not overloading the server and using a very reduced memory size and with almost no risk of timeouts [11].

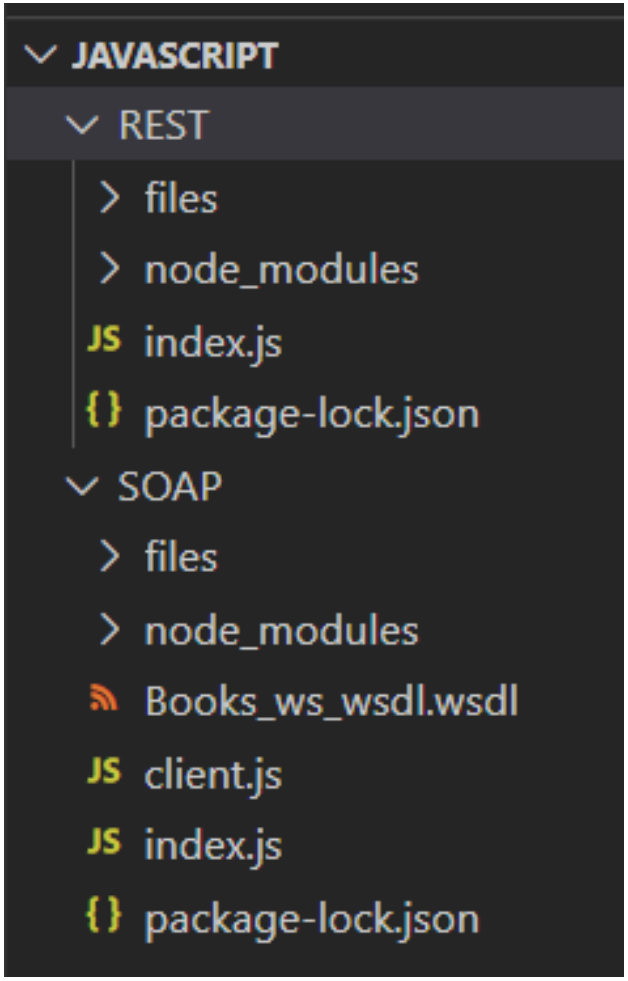
8 SOAP and REST web services implementation in Python and JavaScript

- This section shows the implementation done of REST and SOAP web services in Python and in JavaScript, using the file books.xml and XML parsers to de-serialize the contents of this file and return the data obtained to the client. Additionally these web services will also use XPATH to filter through the contents of the XML file and return only the results matching a certain expression.

- To achieve this goal, it was created a project for the web services in Python using the PyCharm IDE and for the JavaScript a project with Visual Studio Code. The final project's structure is the one presented in the following slide.

8 SOAP and REST web services implementation in Python and JavaScript





8.1 Python – REST web service

- All the REST web services in Python were created using the library *Flask*, which allows the creation of a server where it is possible to register several endpoints to be by the client, which in this case is the browser. Additionally it was also used the *lxml* and the *xml.dom* libraries more specifically in the parsing and reading of the xml file.

- In this first implementation there are two endpoints available, in which one of them execute the parse of the file books.xml, whereas the second one executes the parts of the file and uses XPath expressions to filter through it's contents, returning only the ones who match the referred expression. Accessing the root endpoint, two web links are shown and if clicked they call correspondent method on the web service.

- In the following slides the pictures are accompanied by a description of what they are presenting

8.1 Python – REST web service

- Imports, root endpoint (at '/'), *getBooks* function and the correspondent endpoint to parse the books.xml file, for argument type html.

Books_ws_rest.py

```
from xml.dom import minidom
from flask import Flask, request
from lxml import etree
from werkzeug.wrappers import Response
app = Flask(__name__)


@app.route('/')
def index():
    html = "<h3>Esta aplicação contém os seguintes métodos: <h3>"
    html = html + "<a href='/getBooks/books'>XML Parse - Ficheiro books.xml</a><p>Resposta em XML e HTML</p><br>"
    html = html + "<a href='/getBooks/booksXpath'>XML Parse e XPATH - Ficheiro books.xml</a>"
    return html

@app.route('/getBooks/books', methods=['GET'])
def getBooks():
    typeRequest = request.args.get('type')
    if typeRequest.upper() == "HTML":
        xmldoc = minidom.parse('../files/books.xml')
        itemlist = xmldoc.getElementsByTagName('book')
        data = ""
        i = 0
        for i in range(0, len(itemlist)):
            node = itemlist.item(i)
            autor = node.getElementsByTagName('author')[0].childNodes[0].nodeValue
            titulo = node.getElementsByTagName('title')[0].childNodes[0].nodeValue
            genero = node.getElementsByTagName('genre')[0].childNodes[0].nodeValue
            preco = node.getElementsByTagName('price')[0].childNodes[0].nodeValue
            data_publicacao = node.getElementsByTagName('publish_date')[0].childNodes[0].nodeValue
            descricao = node.getElementsByTagName('description')[0].childNodes[0].nodeValue
            data = data + "<h5>Autor: " + autor + "<h5>"
            data = data + "<h5>Título: " + titulo + "<h5>"
```

8.1 Python – REST web service

Books_ws_rest.py

```
33                data = data + "<h5>Genero: " + genero + "<h5>"
34                data = data + "<h5>Preco: " + preco + "<h5>"
35                data = data + "<h5>Data de Publicação: " + data_publicacao + "<h5>"
36                data = data + "<h5>Descrição: " + descricao + "<h5>"
37                data = data + "<br>"
38            return data
39        if typeRequest.upper() == "XML":
40            root = etree.parse('../files/books.xml').getroot()
41            xml = etree.tostring(root, pretty_print=True)
42            return Response(xml, mimetype='text/xml')
43
44    @app.route('/getBooks/booksXpath', methods=['GET'])
45    def getBooksXpath():
46        root = etree.parse('../files/books.xml')
47        books = root.xpath('/catalog//book[price<7.00 and price>5.00]')
48        data = "<h1>Resultados da pesquisa XPATH por: </h1><h2>price < 7.00 and price > 5.00</h2><br>"
49        i = 0
50        for i in range(0, len(books)):
51            autor = books[i][0].text
52            titulo = books[i][1].text
53            genero = books[i][2].text
54            preco = books[i][3].text
55            data_publicacao = books[i][4].text
56            descricao = books[i][5].text
57            data = data + "<h5>Autor: " + autor + "<h5>"
58            data = data + "<h5>Título: " + titulo + "<h5>"
59            data = data + "<h5>Genero: " + genero + "<h5>"
60            data = data + "<h5>Preco: " + preco + "<h5>"
61            data = data + "<h5>Data de Publicação: " + data_publicacao + "<h5>"
62            data = data + "<h5>Descrição: " + descricao + "<h5>"
63            data = data + "<br>"
64        return data
```

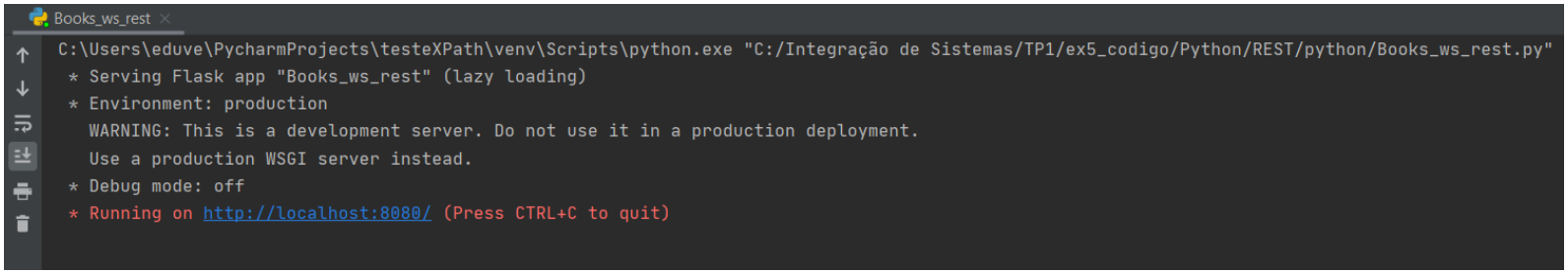
- Return of the constructed string with html, xml type request and the function *getBooksXpath* to filter the results using an Xpath expression.

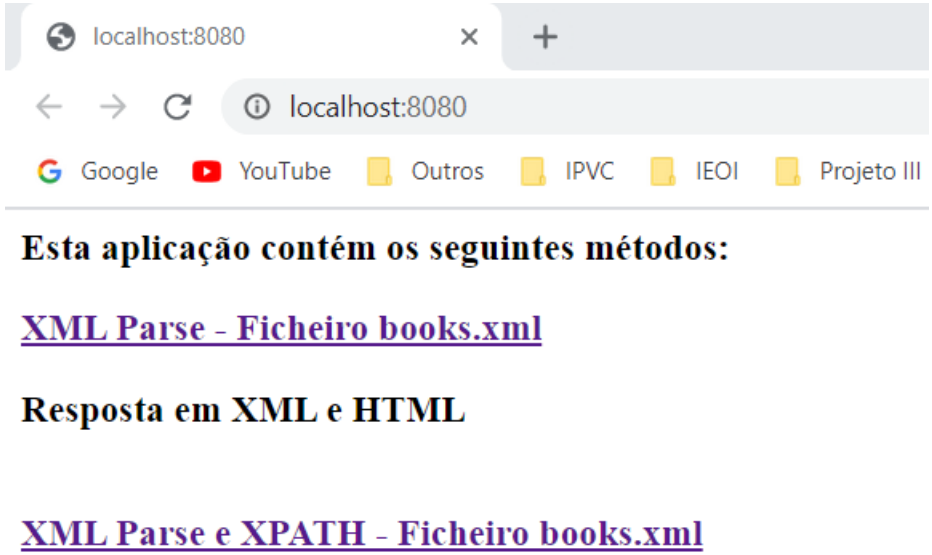
- Main function to start the service

```
65
66  if __name__ == "__main__":
67      app.run(host='localhost', port=8080)
68
```

8.1 Python – REST web service

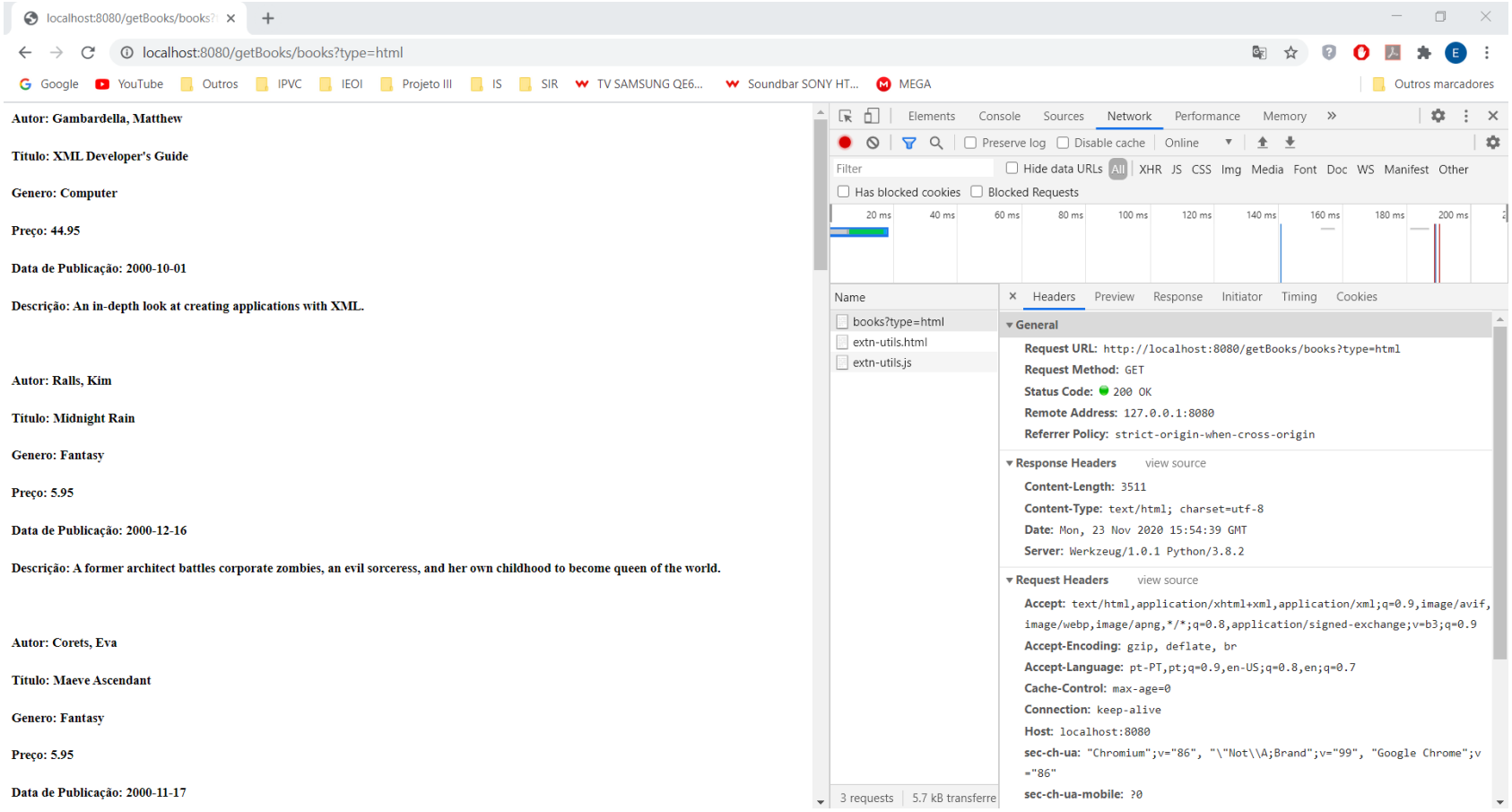
- Web service running on localhost, port 8080, and accessing the root endpoint through the browser





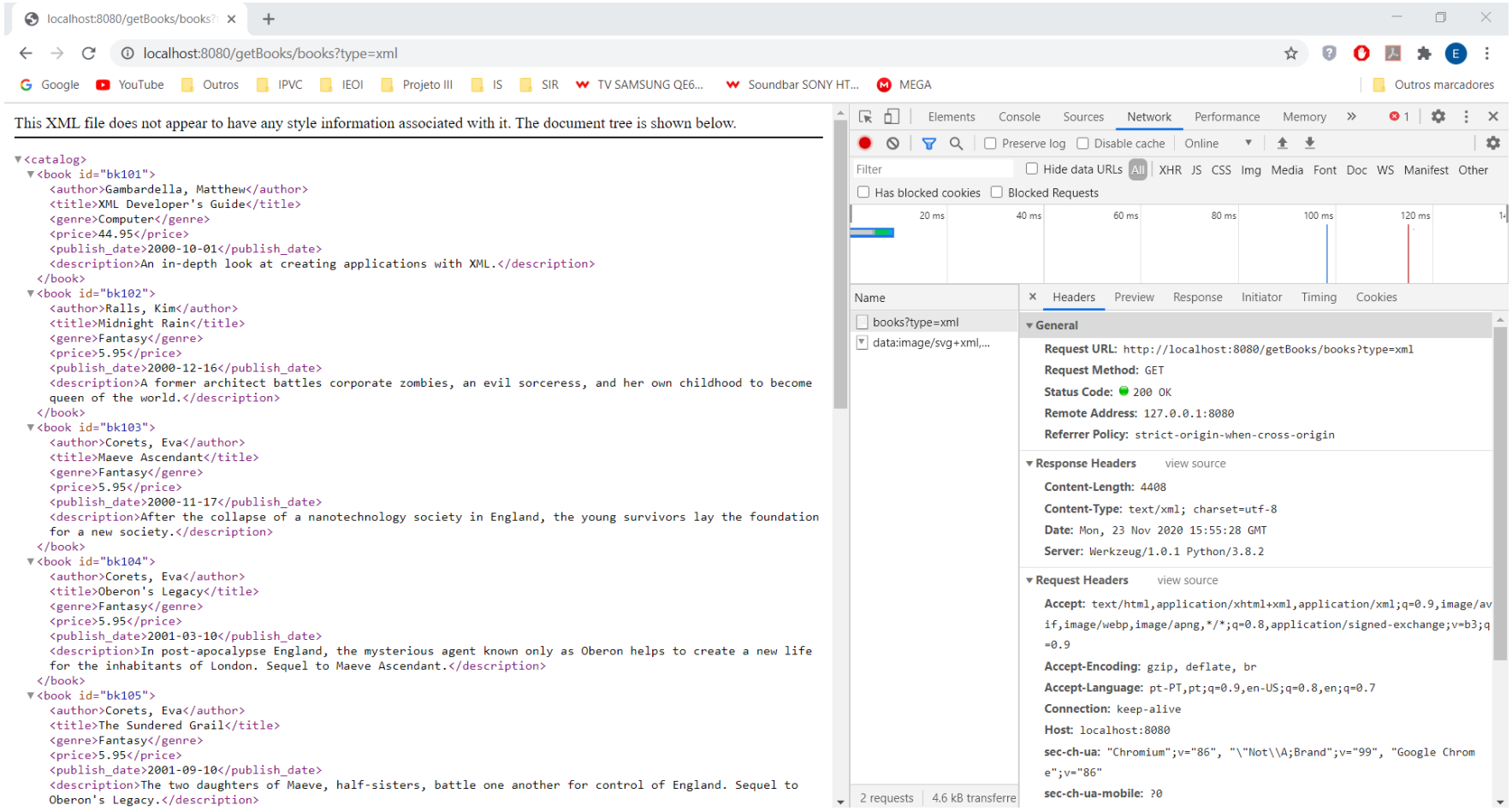
8.1 Python – REST web service

- Calling the *getBooks/books* endpoint specifying the type of output as html



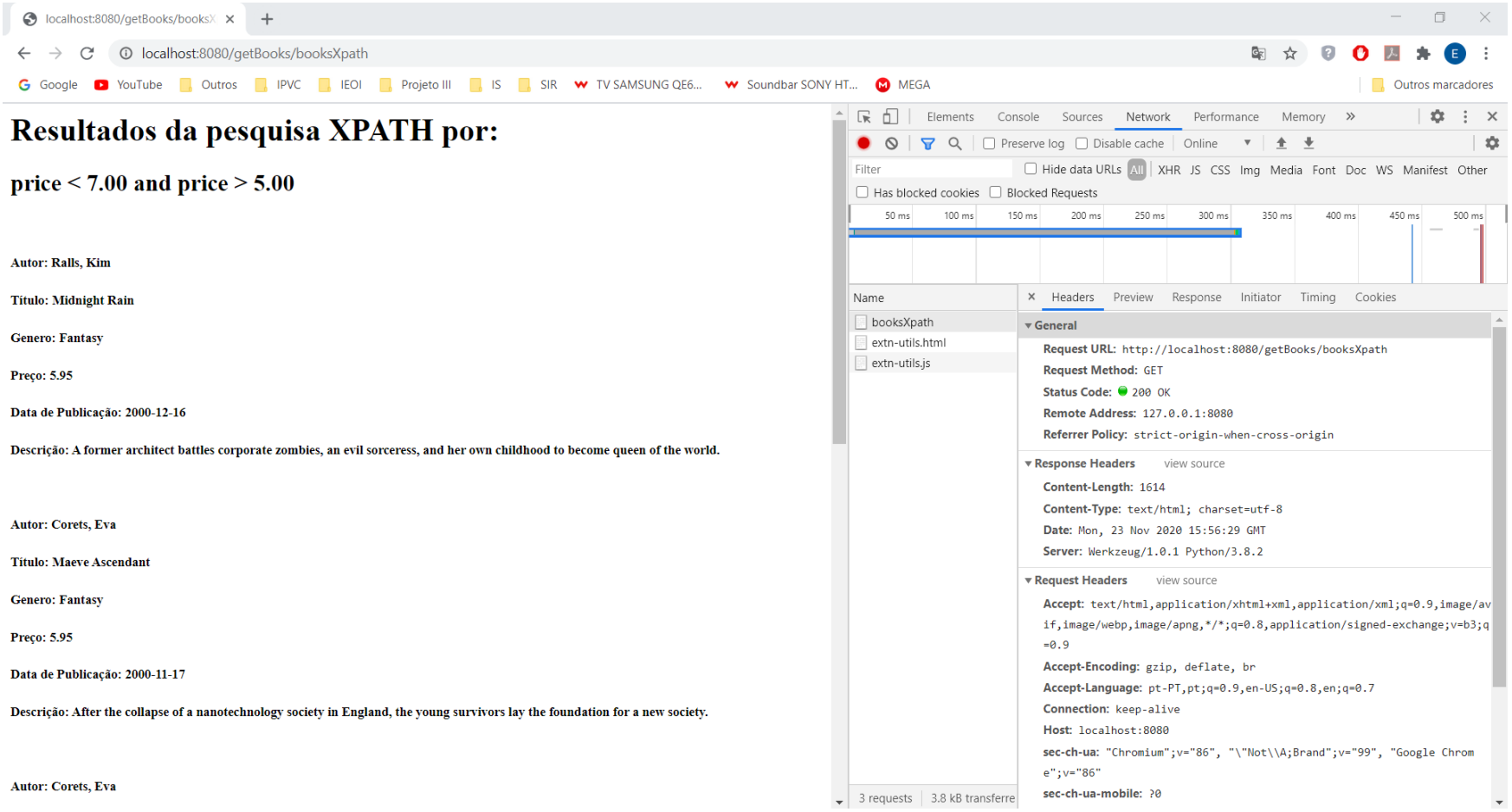
8.1 Python – REST web service

- Calling the *getBooks/books* endpoint specifying the type of output as xml



8.1 Python – REST web service

- Calling the *getBooks/booksXpath* endpoint and the result obtained



8.2 Python – SOAP web service

- As far as soap web services in Python go, in this practical work, they were implemented using the Spyne framework, which allows to create a server, registering created methods and mapping the web service output and input types, which in this case are both SOAP11.

- As it was done for the rest web service this web service has two endpoints available, where one of them executes the parse of the XML file, and the other one uses an XPath expression obtaining only the books which price is between 7.00 and 5.00.

8.2 Python – SOAP web service

- Necessary imports, *books_parse* function, the necessary code using minidom to parse the file and obtain an element list and the construction of the html string to send back to the client.

SoapXmlService.py

```python
from spyne import Application, rpc, ServiceBase, Unicode
from spyne.protocol.soap import Soap11
from spyne.server.wsgi import WsgiApplication
from xml.dom import minidom
from lxml import etree


class SoapXmlService(ServiceBase):
    @rpc(_returns=Unicode)
    def books_parse(ctx):
        xmldoc = minidom.parse('../files/books.xml')
        itemlist = xmldoc.getElementsByTagName('book')
        data = ""
        i = 0
        for i in range(0, len(itemlist)):
            node = itemlist.item(i)
            autor = node.getElementsByTagName('author')[0].childNodes[0].nodeValue
            titulo = node.getElementsByTagName('title')[0].childNodes[0].nodeValue
            genero = node.getElementsByTagName('genre')[0].childNodes[0].nodeValue
            preco = node.getElementsByTagName('price')[0].childNodes[0].nodeValue
            data_publicacao = node.getElementsByTagName('publish_date')[0].childNodes[0].nodeValue
            descricao = node.getElementsByTagName('description')[0].childNodes[0].nodeValue
            data = data + "<h5>Autor: " + autor + "<h5>"
            data = data + "<h5>Título: " + titulo + "<h5>"
            data = data + "<h5>Genero: " + genero + "<h5>"
            data = data + "<h5>Preço: " + preco + "<h5>"
            data = data + "<h5>Data de Publicação: " + data_publicacao + "<h5>"
            data = data + "<h5>Descrição: " + descricao + "<h5>"
            data = data + "<br>"
        return data

```

8.2 Python – SOAP web service

- Second method, *books_Xpath*, using etree from the lxml library to filter through all the books in the books.xml file and return only the list of elements who verify the expression, building the html to send to the client.

SoapXmlService.py

```
31
32      @rpc(_returns=Unicode)
33      def books_Xpath(ctx):
34          root = etree.parse('../files/books.xml')
35          books = root.xpath('/catalog//book[price<7.00 and price>5.00]')
36          data = "<h1>Resultados da pesquisa XPATH por: </h1><h2>price < 7.00 and price > 5.00</h2><br>"
37          i = 0
38          for i in range(0, len(books)):
39              autor = books[i][0].text
40              titulo = books[i][1].text
41              genero = books[i][2].text
42              preco = books[i][3].text
43              data_publicacao = books[i][4].text
44              descricao = books[i][5].text
45              data = data + "<h5>Autor: " + autor + "<h5>"
46              data = data + "<h5>Título: " + titulo + "<h5>"
47              data = data + "<h5>Genero: " + genero + "<h5>"
48              data = data + "<h5>Preço: " + preco + "<h5>"
49              data = data + "<h5>Data de Publicação: " + data_publicacao + "<h5>"
50              data = data + "<h5>Descrição: " + descricao + "<h5>"
51              data = data + "<br>"
52          return data
53
54
55  application = Application([SoapXmlService],
56                            tns='soap.books.parse',
57                            in_protocol=Soap11(validator='lxml'),
58                            out_protocol=Soap11()
59                            )
```

- Additionally the web service is registered in the application, referring a *tns*, the *in_protocol* and the *out_protocol*.

8.2 Python – SOAP web service

- Main function were the server is created and started in localhost in the port 8000, in this case.

```
60  if __name__ == '__main__':
61      from wsgiref.simple_server import make_server
62
63      wsgi_app = WsgiApplication(application)
64      server = make_server('localhost', 8000, wsgi_app)
65      server.serve_forever()
66
```

8.2 Python – SOAP web service

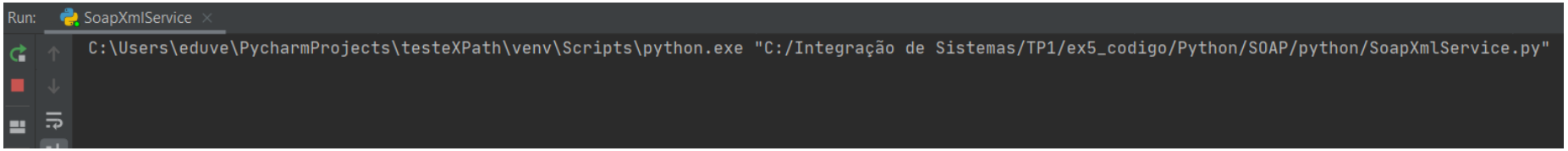
- For the client, it was created a flask server with methods that can be called from the browser and create using the zeep library, a client, which calls the created webservice and the respective method.

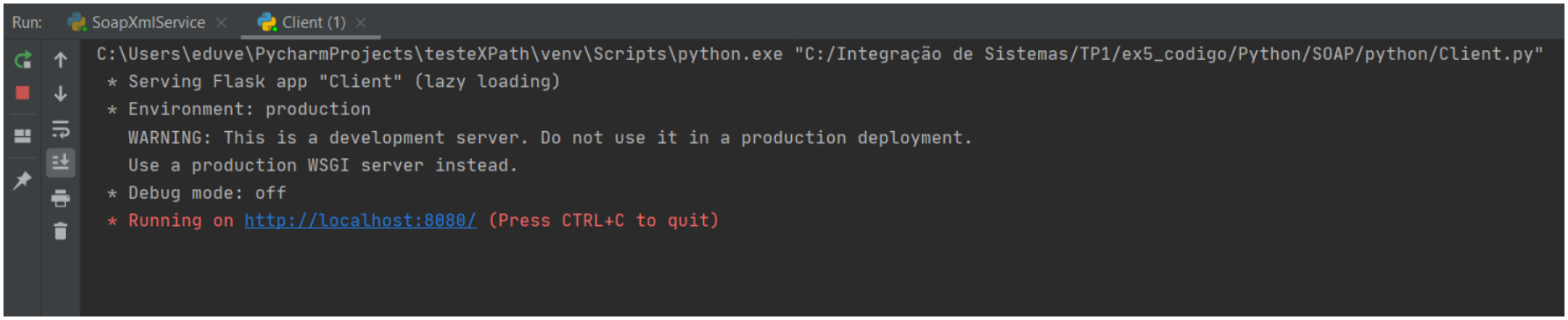
SoapXmlService.py | Client.py

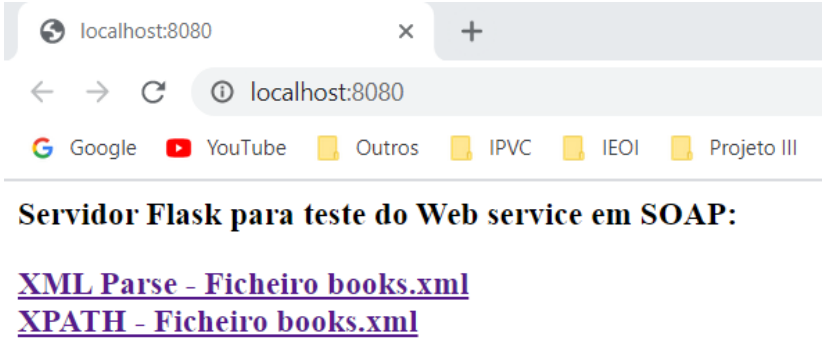
```python
from zeep import Client
from flask import Flask
app = Flask(__name__)


@app.route('/')
def index():
    html = "<h3>Servidor Flask para teste do Web service em SOAP: <h3>"
    html = html + "<a href='/soap/booksParse'>XML Parse - Ficheiro books.xml</a><br>"
    html = html + "<a href='/soap/Xpath'>XPATH - Ficheiro books.xml</a>"
    return html

@app.route('/soap/booksParse', methods=['GET'])
def booksParseSOAP():
    client = Client(wsdl='http://localhost:8000/soap.books.parse?wsdl')
    data = client.service.books_parse()
    return data

@app.route('/soap/Xpath', methods=['GET'])
def booksXpath():
    client = Client(wsdl='http://localhost:8000/soap.books.parse?wsdl')
    data = client.service.books_Xpath()
    return data

if __name__ == "__main__":
    app.run(host='localhost', port=8080)

```

8.2 Python – SOAP web service

- From top to bottom, the first image is the web service running, the second one demonstrates the client running, and the third one presents the root endpoint in the client server, used to present the data.

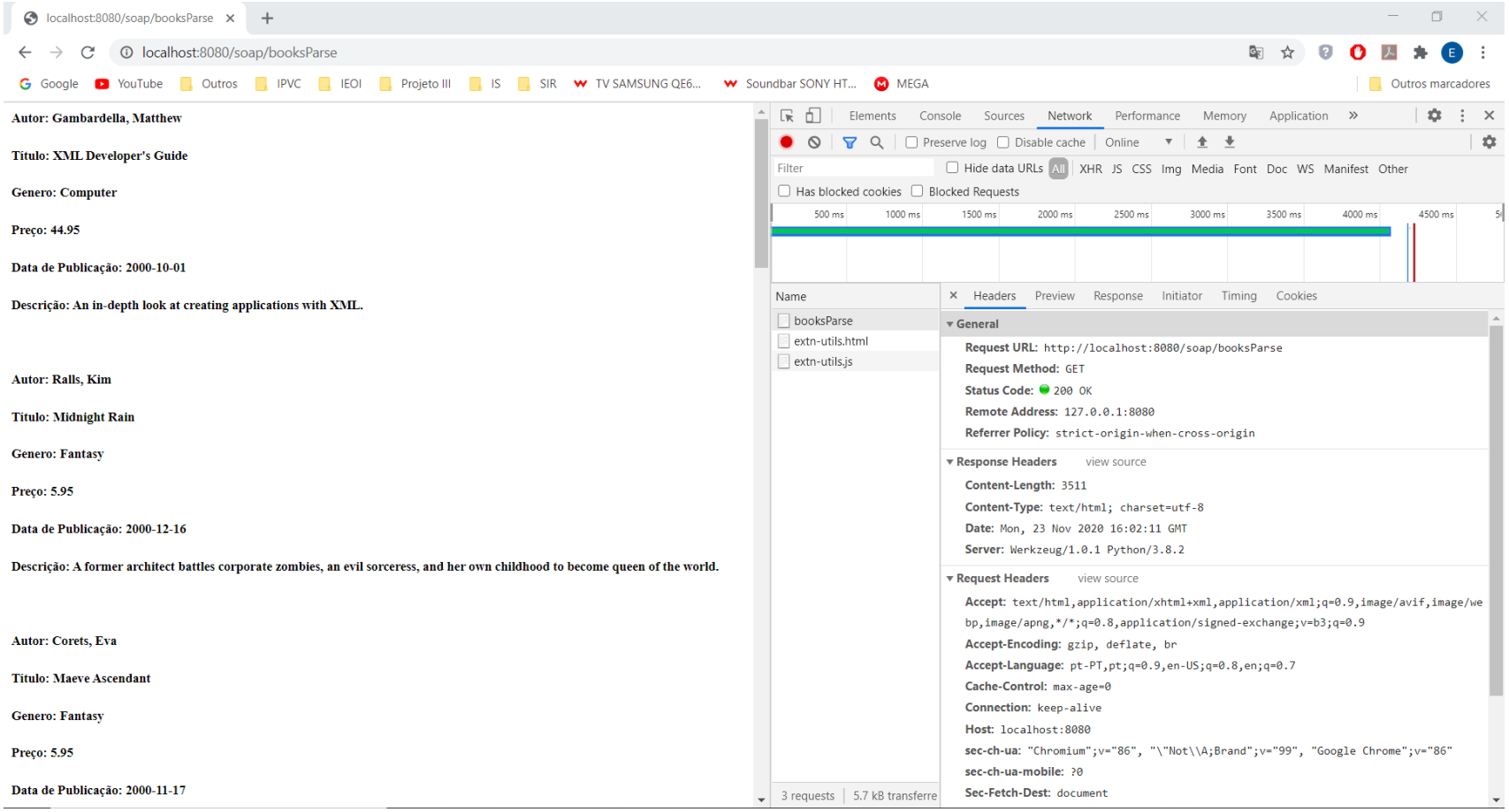






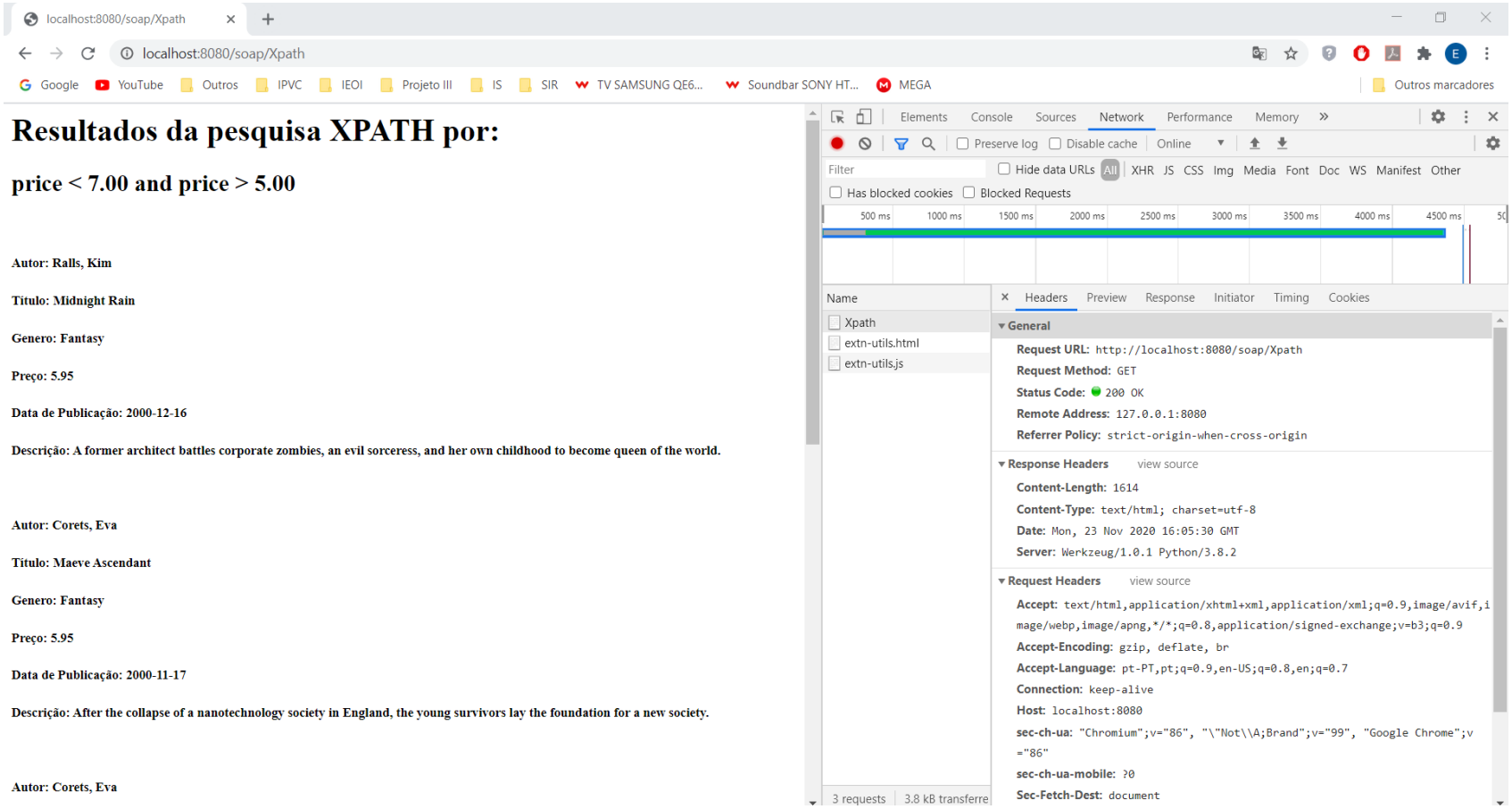
8.2 Python – SOAP web service

- Result of calling the *books_parse* method on the web service, with the data presented on the client side.



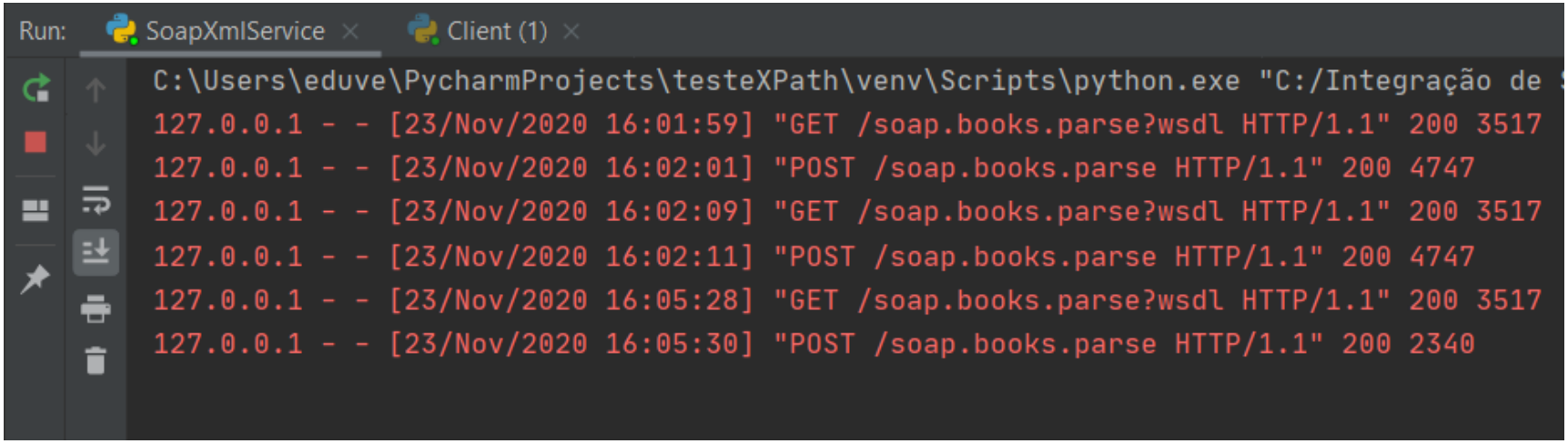
8.2 Python – SOAP web service

- Result of calling the *books_Xpath* method and the result presented in the browser.



8.2 Python – SOAP web service

- Registered calls to the web service.



8.3 JavaScript – REST web service

- For the implementation of a REST web service in JavaScript, it was used Node Js, in order to create a server where all the created methods to access, parse and filer the contents of the books.xml file will be registered.

- The first endpoint, obtains the arguments from the URL and checks if the argument *file* is equal to *books*, and if it is, then the file to execute the parsing is books.xml. In this case the file is parsed, and the contents returned are in a form of a string.

- The second endpoint verifies the arguments in the same way as the first one, but it only extracts the nodes from the XML file that match the XPath expression, and returning html to the client with the results.

8.3 JavaScript – REST web service

- Necessary impots, first endpoint registered to execute the parse of the books.xml file, constructing the string to send the data back to the client.

index.js

REST > index.js > app.get('/booksXpath') callback

```
1  var express = require('express');
2  var app = express();
3  var fs = require("fs");
4  var parser = require('xml2json-light');
5  var xpath = require('xpath')
6  var dom = require('xmldom').DOMParser
7  const http = require('http');
8  const url = require('url');
9
10 app.get('/booksParse', function (req, res) {
11    const queryObject = url.parse(req.url, true).query;
12    if (queryObject["file"] == "books") {
13       fs.readFile("files/books.xml", 'utf8', function (err, data) {
14          var json = parser.xml2json(data);
15          var i = 0;
16          var dados = "";
17          for (i = 0; i < json.catalog.book.length; i++) {
18             var book = json.catalog.book[i]
19             dados = dados + `Id: ${book.id}\n`
20             dados = dados + `Autor: ${book.author}\n`
21             dados = dados + `Título: ${book.title}\n`
22             dados = dados + `Genero: ${book.genre}\n`
23             dados = dados + `Preço: ${book.price}\n`
24             dados = dados + `Data de Publicação: ${book.publish_date}\n`
25             dados = dados + `Descrição: ${book.description}\n\n`
26          }
27
28          res.setHeader("Content-Type", "text/plain; charset=utf-8");
29          res.end(dados);
30       });
31    }
32    else {
33       res.setHeader("Content-Type", "text/plain; charset=utf-8");
34       res.end("O URL não contém o nome do ficheiro XML para a realização do parse!\n\n Pedido Inválido!");
35    }
36
37 })
```

8.3 JavaScript – REST web service

- Second endpoint, that uses XPath expression to get all the books, where price is higher than 30.50 and the book's genre is Computer.
- It also builds a string containing the data in html format and returns it to the client.

```
38
39  app.get('/booksXpath', function (req, res) {
40     const queryObject = url.parse(req.url, true).query;
41     if (queryObject["file"] == "books") {
42        fs.readFile("files/books.xml", 'utf8', function (err, data) {
43           var doc = new dom().parseFromString(data)
44           var nodes = xpath.select("/catalog/book[price>30.50 and genre='Computer']", doc)
45           var ids = xpath.select("/catalog/book/@id", doc)
46           var i = 0;
47           var dados = "<h2>RESULTADOS FILTRAGEM XPATH POR:</h2><br><p>price > 30.50 e genre = Computer</p><br>";
48
49           for (i = 0; i < nodes.length; i++) {
50              var bookId = ids[i].value
51              var book = nodes[0];
52              var autor = book.getElementsByTagName('author')[0].childNodes[0].nodeValue;
53              var titulo = book.getElementsByTagName('title')[0].childNodes[0].nodeValue;
54              var genero = book.getElementsByTagName('genre')[0].childNodes[0].nodeValue;
55              var preco = book.getElementsByTagName('price')[0].childNodes[0].nodeValue;
56              var dataPublicacao = book.getElementsByTagName('publish_date')[0].childNodes[0].nodeValue;
57              var descricao = book.getElementsByTagName('description')[0].childNodes[0].nodeValue;
58              dados = dados + `<p>Id: ${bookId}</p>`
59              dados = dados + `<p>Autor: ${autor}</p>`
60              dados = dados + `<p>Título: ${titulo}</p>`
61              dados = dados + `<p>Genero: ${genero}</p>`
62              dados = dados + `<p>Preço: ${preco}</p>`
63              dados = dados + `<p>Data de Publicação: ${dataPublicacao}</p>`
64              dados = dados + `<p>Descrição: ${descricao}</p><br>`
65           }
66           res.setHeader("Content-Type", "text/html; charset=utf-8");
67           res.end(dados);
68        });
69     }
70     else {
71        res.setHeader("Content-Type", "text/plain; charset=utf-8");
72        res.end("O URL não contém o nome do ficheiro XML para a realização do parse!\n\n Pedido Inválido!");
73     }
```

8.3 JavaScript – REST web service

- The first image, presents the code to create and start the server, configured to listen in the port 3000 in the localhost. The second one is the execution of the created file, through the command line, and the web service running.

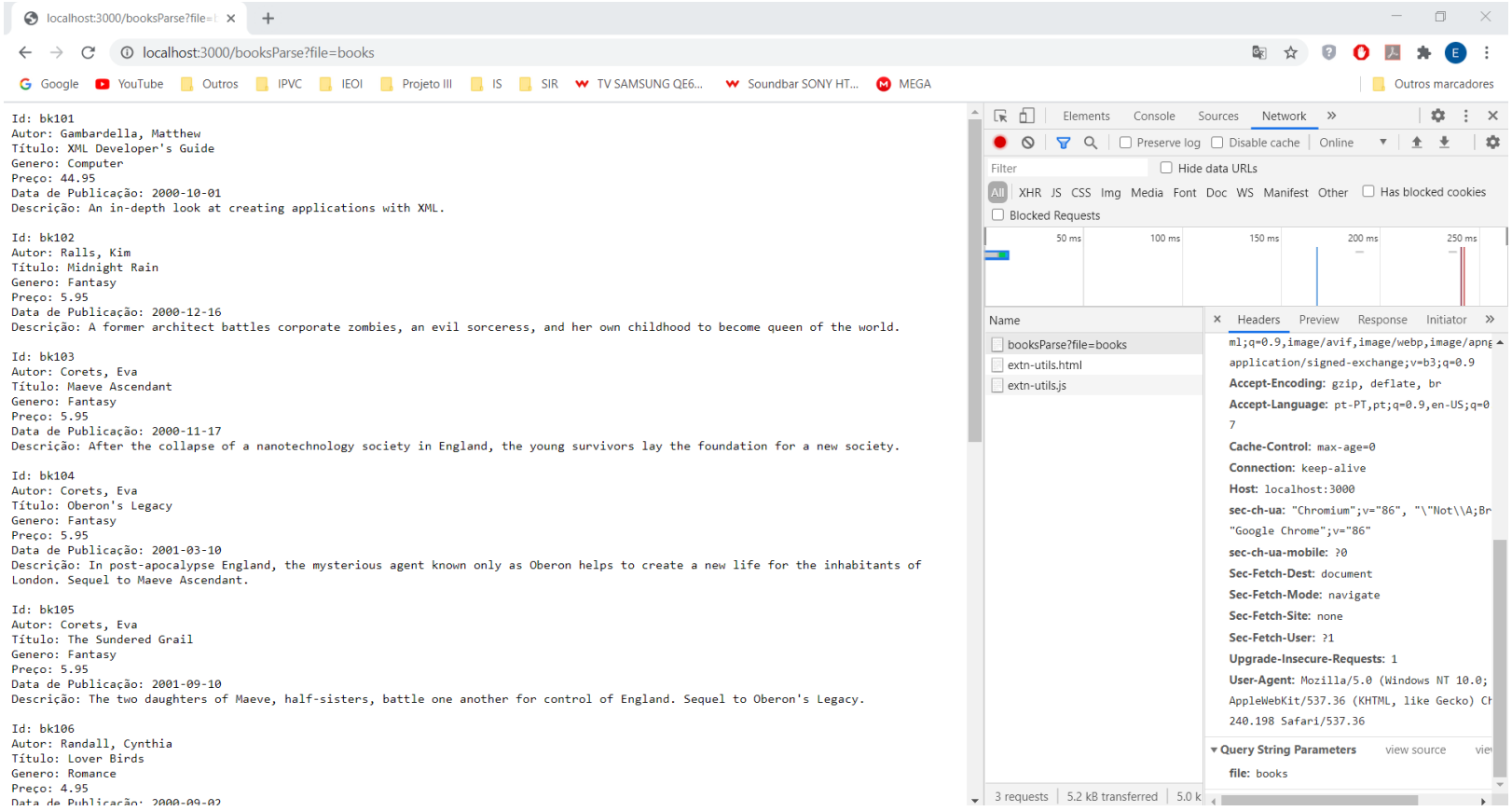
```
77  var server = app.listen(3000, function () {
78      var host = server.address().address
79      var port = server.address().port
80      console.log("Books REST Web Service listening at http://localhost:3000")
81  })
```

Linha de comandos - node index.js

```
C:\Integração de Sistemas\TP1\ex5_codigo\JavaScript\REST>node index.js
Books REST Web Service listening at http://localhost:3000
```

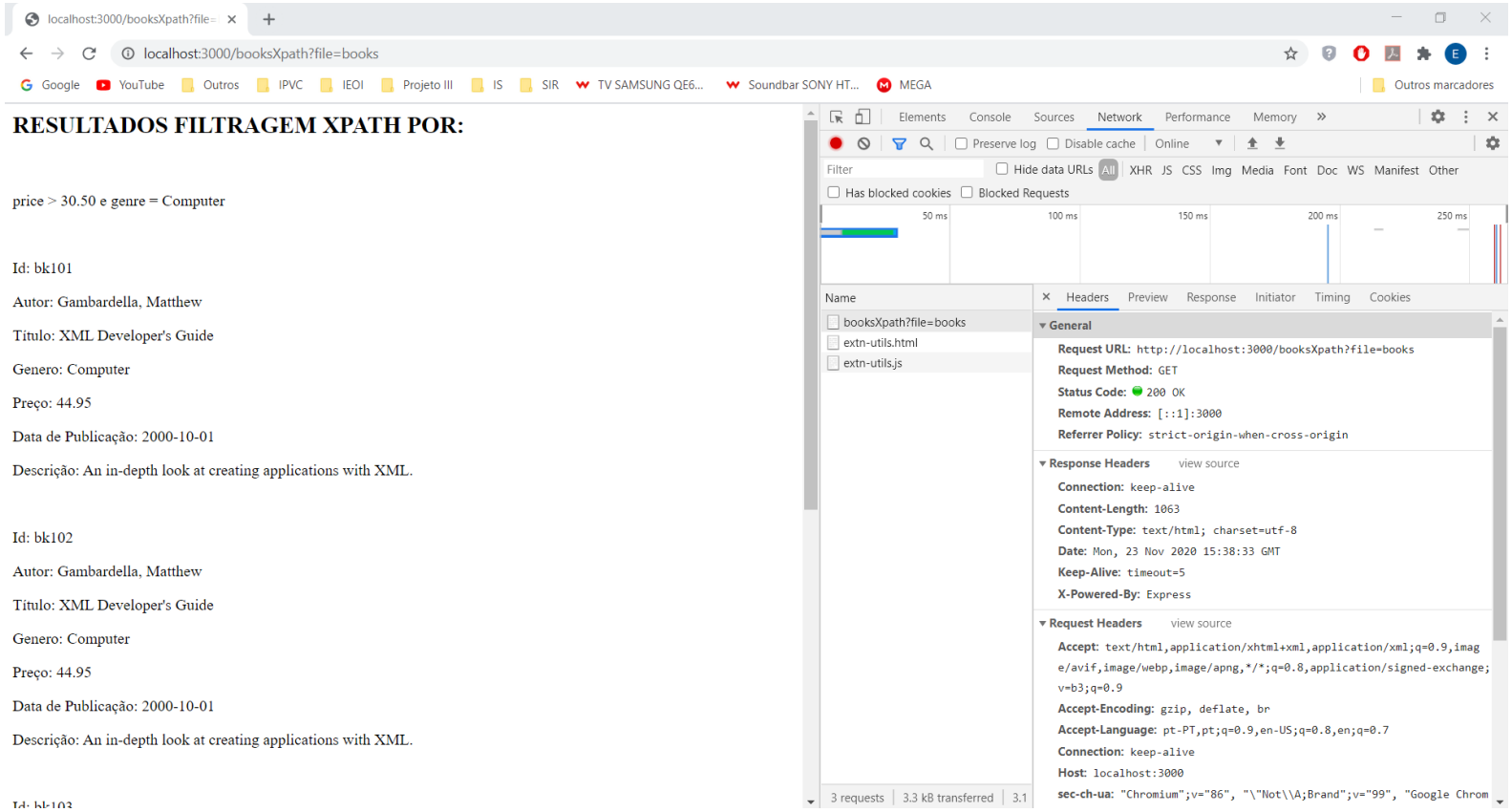
8.3 JavaScript – REST web service

- Result obtained calling the first endpoint to parse the books.xml file.



8.3 JavaScript – REST web service

- Result obtained calling the second endpoint to parse and filter the books.xml file with XPath.



8.4 JavaScript – SOAP web service

- In order to create a SOAP web service, it was necessary to create a wsdl file which specifies the web service, defining the operations that it includes, the arguments and the type of call, which in this case is soap, and the names and port where the service can be called.

- The code for the web service was created using the express server and soap packages for Node Js, and a SOAP client to call the created web service, as demonstrated in the next slides.

8.4 JavaScript – SOAP web service

- First part of the Books_ws_wsdl file, which defines the requests and the elements they include.

Books_ws_wsdl.wsdl

SOAP > Books_ws_wsdl.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="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:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  xmlns:tns="http://tempuri.org/"
  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:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <!-- Definição dos pedidos e respostas para o método parse_books  -->
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
      <s:element name="BooksParseRequest">
        <s:complexType>
          <s:sequence>
          <s:element minOccurs="1" maxOccurs="unbounded" name="type" type="s:string"/>
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="BooksParseResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="unbounded" name="result" type="s:string"/>
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:schema>
  </wsdl:types>
```

8.4 JavaScript – SOAP web service

- This second part defines the type of communications between the server and client, the operation on the web service , and the way the messages will be transmitted.

Books_ws_wsdl.wsdl

SOAP > Books_ws_wsdl.wsdl

```
31  <!-- Definição dos dados e do seu tipo entre o cliente e o servidor -->
32  <wsdl:message name="BooksParseSoapIn">
33    <wsdl:part name="parameters" element="tns:BooksParseRequest"/>
34  </wsdl:message>
35  <wsdl:message name="BooksParseSoapOut">
36    <wsdl:part name="parameters" element="tns:BooksParseResponse"/>
37  </wsdl:message>
38  <wsdl:portType name="BooksParseSoapPort">
39    <!-- Definição da operação books_parse, definida no index.js -->
40    <wsdl:operation name="BooksParse">
41      <wsdl:input message="tns:BooksParseSoapIn"/>
42      <wsdl:output message="tns:BooksParseSoapOut"/>
43    </wsdl:operation>
44  </wsdl:portType>
45  <!-- Definição de como as mensagens serão transmitidas -->
46  <wsdl:binding name="BooksParseServiceSoapBinding" type="tns:BooksParseSoapPort">
47    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
48    <wsdl:operation name="BooksParse">
49      <soap:operation soapAction="BooksParse" style="document"/>
50      <wsdl:input>
51        <soap:body use="literal"/>
52      </wsdl:input>
53      <wsdl:output>
54        <soap:body use="literal"/>
55      </wsdl:output>
56    </wsdl:operation>
57  </wsdl:binding>
```

8.4 JavaScript – SOAP web service

- The final part creates the definition of the service where the method is going to be binded to.

Books_ws_wsdl.wsdl

SOAP > Books_ws_wsdl.wsdl

```
58    <wsdl:binding name="BooksParseServiceSoap12Binding" type="tns:BooksParseSoapPort">
59      <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"/>
60      <wsdl:operation name="BooksParse">
61        <soap12:operation soapAction="BooksParse" style="document"/>
62        <wsdl:input>
63          <soap12:body use="literal"/>
64        </wsdl:input>
65        <wsdl:output>
66          <soap12:body use="literal"/>
67        </wsdl:output>
68      </wsdl:operation>
69    </wsdl:binding>
70    <!-- Definição do serviço  -->
71    <wsdl:service name="BooksParseService">
72      <wsdl:port name="BooksParseServiceSoapPort" binding="tns:BooksParseServiceSoapBinding">
73        <soap:address location="http://localhost:8000/wsdl"/>
74      </wsdl:port>
75      <wsdl:port name="BooksParseServiceSoap12Port" binding="tns:BooksParseServiceSoap12Binding">
76        <soap12:address location="http://localhost:8000/wsdl"/>
77      </wsdl:port>
78    </wsdl:service>
79  </wsdl:definitions>
```

8.4 JavaScript – SOAP web service

- This first part of the file presents the initial imports necessary for the web service, and the method *books_parse*, which depending on the argument given in the call, will execute the parse or filter and parse the books.xml file.

index.js

SOAP > index.js > ...

```
1  var soap = require('soap');
2  var express = require('express');
3  var fs = require('fs');
4  var parser = require('xml2json-light');
5  var xpath = require('xpath')
6  var dom = require('xmldom').DOMParser
7
8  function books_parse(args) {
9    if(args.type == "parse") {
10     var data = fs.readFileSync( "files/books.xml", 'utf8');
11     var json = parser.xml2json(data);
12     var dados = "<h2>PARSE DO FICHEIRO - books.xml</h2><br>";
13     var i = 0;
14     for(i = 0; i < json.catalog.book.length; i++) {
15       var book = json.catalog.book[i]
16       dados = dados + `<p>Id: ${book.id}</p>`
17       dados = dados + `<p>Autor: ${book.author}</p>`
18       dados = dados + `<p>Título: ${book.title}</p>`
19       dados = dados + `<p>Genero: ${book.genre}</p>`
20       dados = dados + `<p>Preço: ${book.price}</p>`
21       dados = dados + `<p>Data de Publicação: ${book.publish_date}</p>`
22       dados = dados + `<p>Descrição: ${book.description}</p><br>`
23     }
24     return {
25         result: dados
26     }
27   }
```

8.4 JavaScript – SOAP web service

```
29    if(args.type == "xpath") {
30      var data = fs.readFileSync( "files/books.xml", 'utf8');
31      var doc = new dom().parseFromString(data)
32      var nodes = xpath.select("/catalog/book[genre='Fantasy' and contains(author, 'Eva')]", doc)
33      var ids = xpath.select("/catalog/book/@id", doc)
34      var i = 0;
35      var dados = "<h2>XPATH - books.xml</h2><p>Filtragem por livros do género <i>Fantasy</i>,\
36       e onde o nome do autor contém a string, <i>Eva</i></p><br>";
37      for(i = 0; i < nodes.length; i++) {
38        var bookId = ids[i].value
39        var book = nodes[0];
40        var autor = book.getElementsByTagName('author')[0].childNodes[0].nodeValue;
41        var titulo = book.getElementsByTagName('title')[0].childNodes[0].nodeValue;
42        var genero = book.getElementsByTagName('genre')[0].childNodes[0].nodeValue;
43        var preco = book.getElementsByTagName('price')[0].childNodes[0].nodeValue;
44        var dataPublicacao = book.getElementsByTagName('publish_date')[0].childNodes[0].nodeValue;
45        var descricao = book.getElementsByTagName('description')[0].childNodes[0].nodeValue;
46        dados = dados + `<p>Id: ${bookId}</p>`
47        dados = dados + `<p>Autor: ${autor}</p>`
48        dados = dados + `<p>Título: ${titulo}</p>`
49        dados = dados + `<p>Genero: ${genero}</p>`
50        dados = dados + `<p>Preço: ${preco}</p>`
51        dados = dados + `<p>Data de Publicação: ${dataPublicacao}</p>`
52        dados = dados + `<p>Descrição: ${descricao}</p><br>`
53      }
54      return {
55        result: dados
56      }
57    }
58  }
59
```

8.4 JavaScript – SOAP web service

- Service object created and the wsdl file readed, configuring the *book_parse* method to be executed when the server is accessed in the port 8000 on the localhost.

index.js

SOAP > index.js > ...

```
59
60  var serviceObject = {
61    BooksParseService: {
62          BooksParseServiceSoapPort: {
63            BooksParse: books_parse
64          },
65          BooksParseServiceSoap12Port: {
66            BooksParse: books_parse
67          }
68      }
69  };
70
71  var xml = fs.readFileSync('Books_ws_wsdl.wsdl', 'utf8');
72  var app = express();
73
74  app.get('/', function (req, res) {
75    res.send('SOAP webservice - Parse de ficheiros XML e XPATH<br />');
76  })
77
78  var port = 8000;
79  app.listen(port, function () {
80    console.log('Listening on port ' + port);
81    var wsdl_path = "/wsdl";
82    soap.listen(app, wsdl_path, serviceObject, xml);
83    console.log("Check http://localhost:" + port + wsdl_path +"?wsdl to see if the service is working");
84  });
```

8.4 JavaScript – SOAP web service

- Client code, using the soap and express libraries to call the web service and present the results, the first endpoint for the parsing of the books.xml file.

client.js

SOAP > client.js > app.get('/callBooksParse') callback > soap.createClient() callback

```
1  var express = require('express');
2  var app = express();
3  var soap = require('soap');
4  var url = 'http://localhost:8000/wsdl?wsdl';
5
6  app.get('/callBooksParse', function (req, res) {
7      soap.createClient(url, function (err, client) {
8        if (err){
9          throw err;
10       }
11       var args = {
12         type: 'parse'
13       }
14       client.BooksParse(args, function (err, result) {
15         if (err) {
16           throw err;
17         }
18         else {
19           res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
20           res.end(result["result"][0]);
21         }
22       });
23     });
24 })
25
```

8.4 JavaScript – SOAP web service

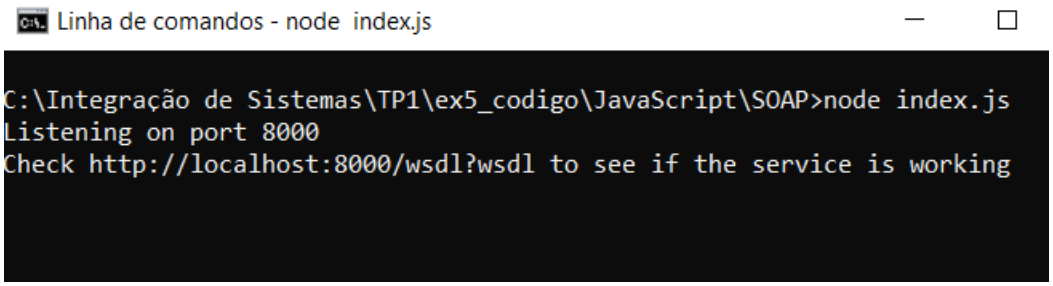
- Second endpoint where the web service is called specifying the type of action as XPath, presenting the results as html.

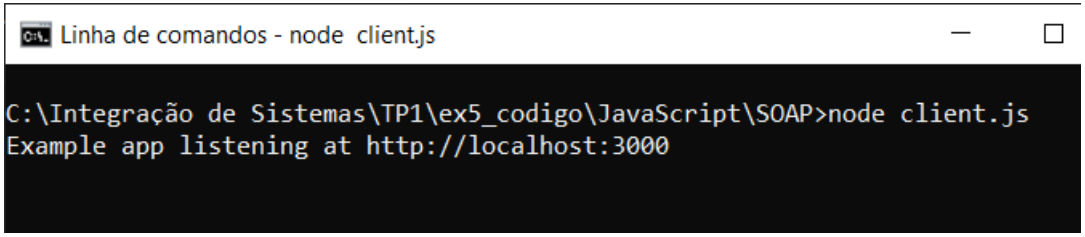
client.js

SOAP > client.js > app.get('/callBooksParse') callback > soap.createClient() callback

```
25
26  app.get('/callBooksXpath', function (req, res) {
27   soap.createClient(url, function (err, client) {
28     if (err){
29       throw err;
30     }
31     var args = {
32       type: 'xpath'
33     }
34     client.BooksParse(args, function (err, result) {
35       if (err) {
36         throw err;
37       }
38       else {
39         res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
40         res.end(result["result"][0]);
41       }
42     });
43   });
44 })
45
46  var server = app.listen(3000, function () {
47     var host = server.address().address
48     var port = server.address().port
49     console.log("Example app listening at http://localhost:3000")
50  })
```

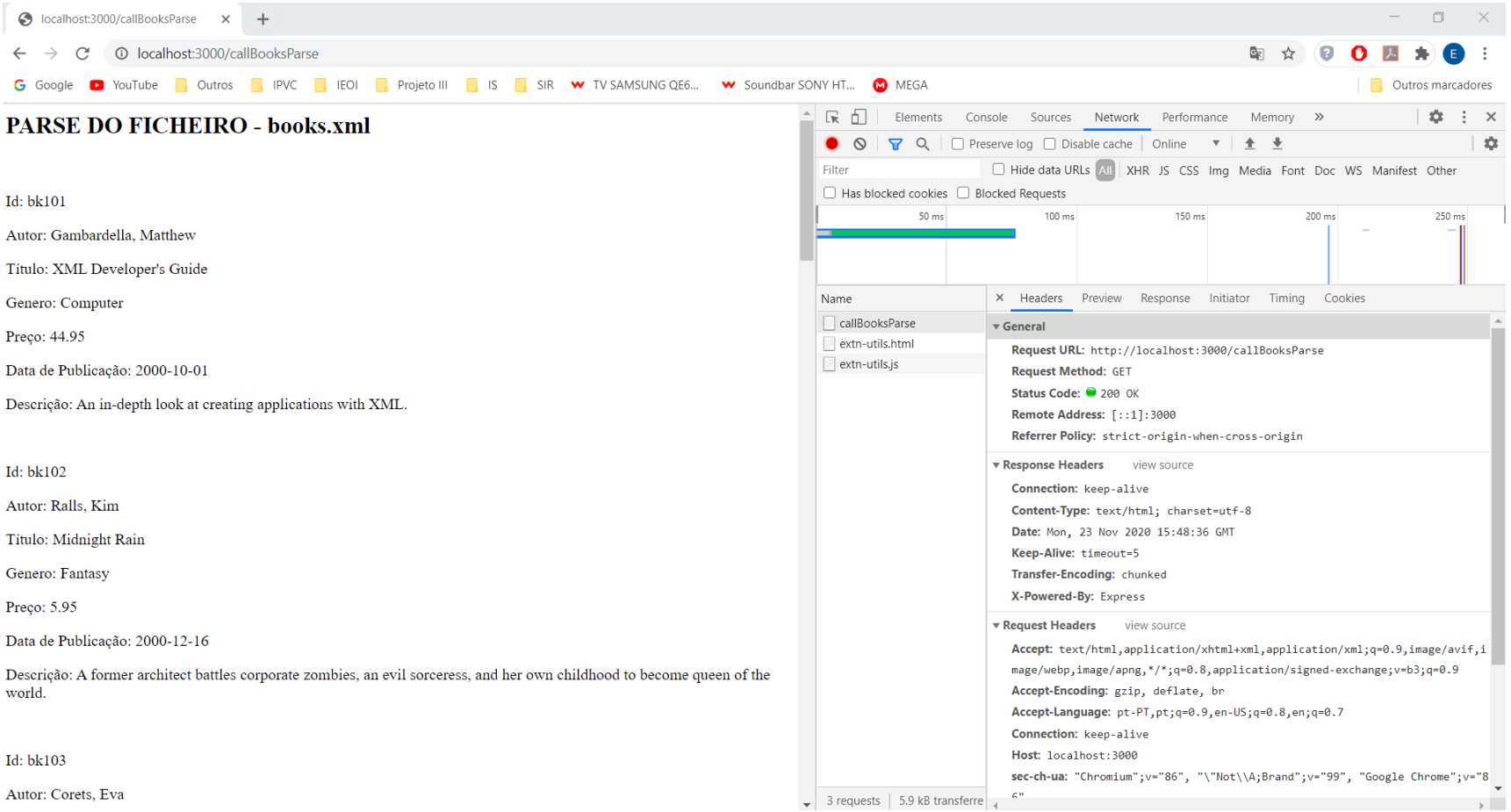
- Running the server (image on top), running the client (image on the bottom).





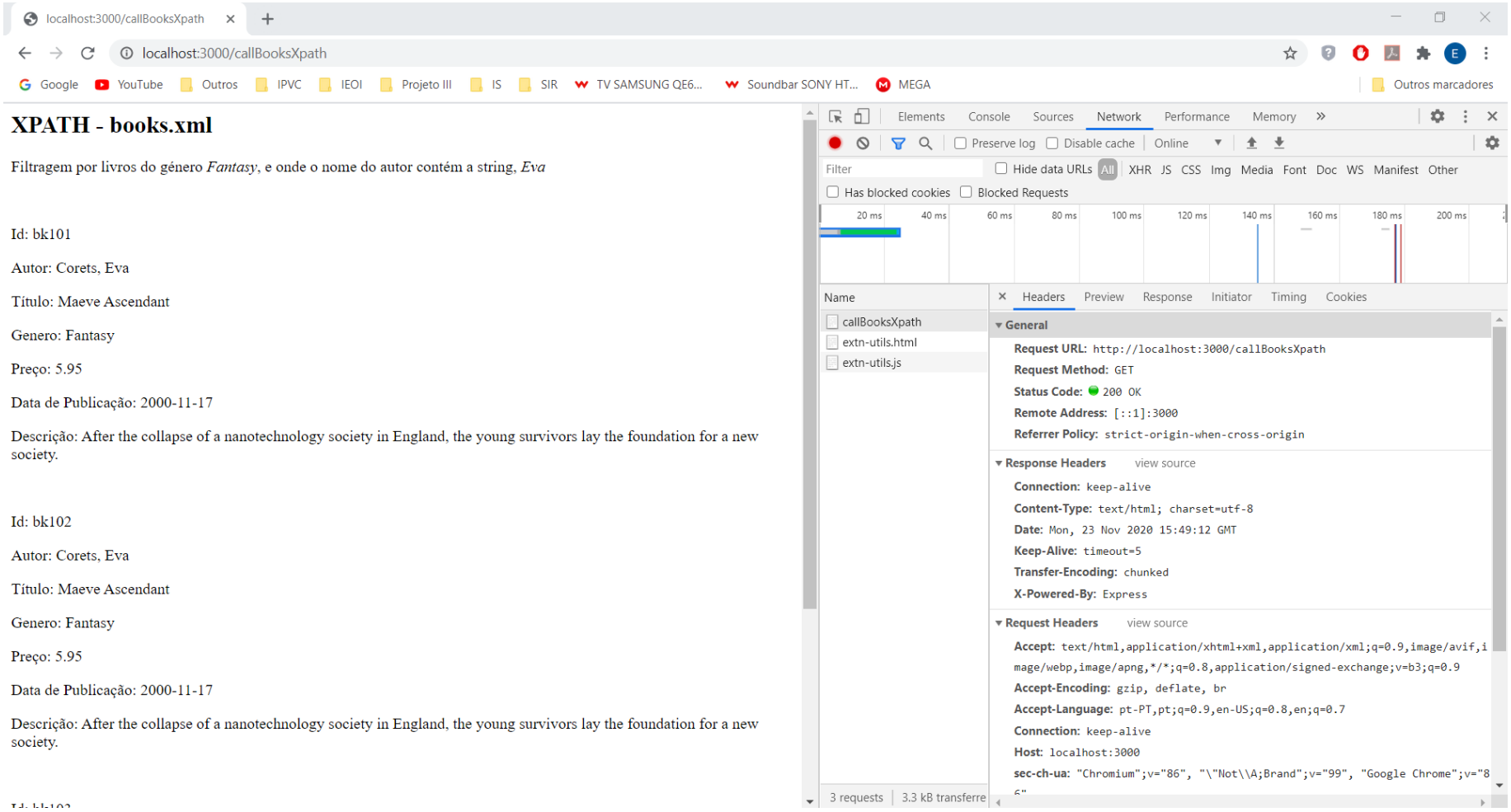
8.4 JavaScript – SOAP web service

- Results obtained calling the webservice to parse the books.xml file.



8.4 JavaScript – SOAP web service

- Results obtained calling the webservice to parse and filter the books.xml file with Xpath.

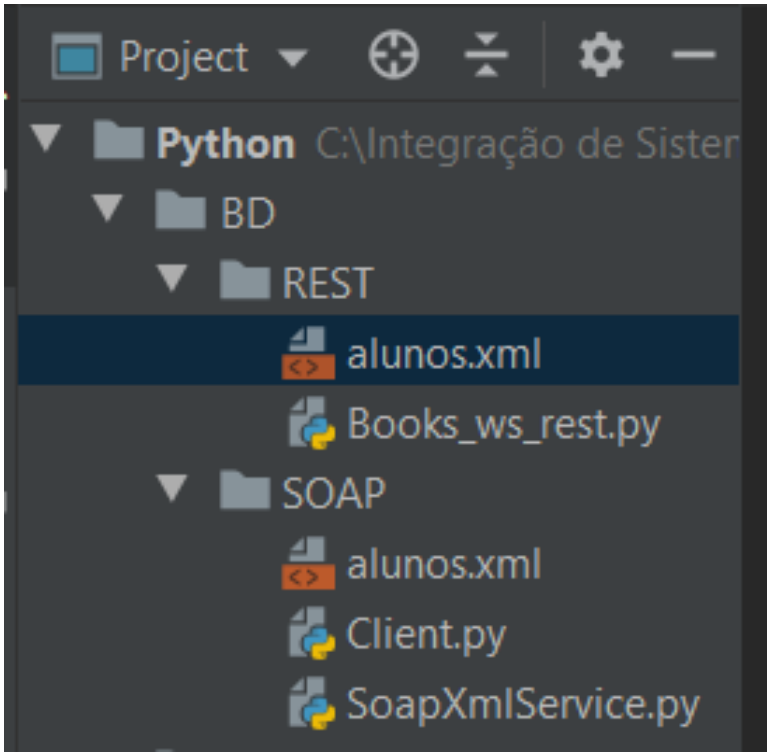


9 Python SOAP and REST web services to read data from MySQL database

- For the REST and SOAP web services involving the access to a MySQL database, the same libraries were used, but additionally it was imported the MySQL connector for Python, allowing the creation of a connection to the database running on the localhost in the port 3306.

- The following slide demonstrates the project's structure, created with the SOAP and REST web services.

9 Python SOAP and REST web services to read data from MySQL database



9.1 Python – REST web service

- Necessary imports used in the web service, definition of the connection to the MySQL database and the first endpoint of the web service.

Books_ws_rest.py

```
from xml.dom import minidom
from flask import Flask, request
from werkzeug.wrappers import Response
import mysql.connector
app = Flask(__name__)

mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="",
  database="is"
)

@app.route('/')
def index():
    html = "<h3>Web Service REST com acesso à base de dados MySQL: <h3>"
    html = html + "<a href='/getBooks/books?type=all'>Obter alunos da BD MySQL - Criar e apresentar ficheiro XML</a><br>"
    return html

```

9.1 Python – REST web service

- Second endpoint with the execution of an SQL query to obtain all the students in the table *alunos,* as well as the creation of the tags to construct the XML file, which is created in the same folder where the *books_ws_rest.py* is defined.
- At the end, the data is returned to the client and printed as XML.

Books_ws_rest.py

```
@app.route('/getAlunos/alunos', methods=['GET'])
def getAlunos():
    selectType = request.args.get('type')
    if selectType.upper() == "ALL":
        mycursor = mydb.cursor()
        mycursor.execute("SELECT * FROM alunos")
        myresult = mycursor.fetchall()
        doc = minidom.Document()
        tag_alunos = doc.createElement('alunos')
        for aluno in myresult:
            id_aluno = str(aluno[0])
            nome = aluno[1]
            tag_aluno = doc.createElement('aluno')

            tag_id = doc.createElement('id')
            content_id = doc.createTextNode(id_aluno)
            tag_id.appendChild(content_id)

            tag_nome = doc.createElement('nome')
            content_nome = doc.createTextNode(nome)
            tag_nome.appendChild(content_nome)

            tag_aluno.appendChild(tag_id)
            tag_aluno.appendChild(tag_nome)

            tag_alunos.appendChild(tag_aluno)

        doc.appendChild(tag_alunos)
        string_xml = doc.toprettyxml(indent="\t")
        caminho_ficheiro = "alunos.xml"
        with open(caminho_ficheiro, "w") as f:
            f.write(string_xml)
```

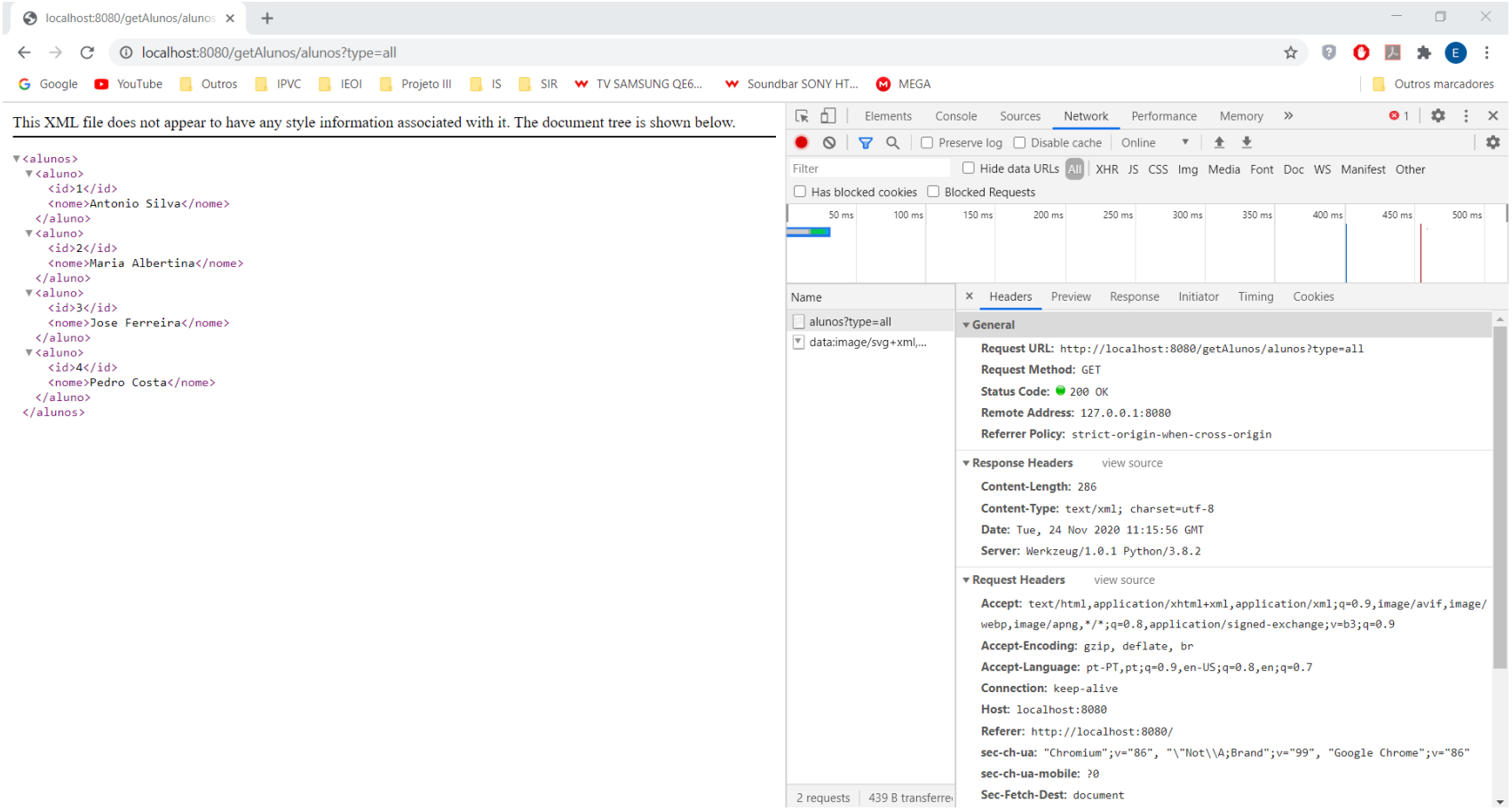
9.1 Python – REST web service

- The image on top demonstrates the response returned to the client, the second, the link to call the web service, and the final image the call registered in the web service when it is called.

```
52          return Response(string_xml, mimetype='text/xml')
53
54  if __name__ == "__main__":
55      app.run(host='localhost', port=8080)
56
```

9.1 Python – REST web service

- Result obtained by calling the web service



9.2 Python – SOAP web service

For the SOAP web service the code is similar to the REST web service, being the use of spyne, creating a server and registering the web service with a *tns,* the only difference in relation to the REST web service, as demonstrated by the following images.

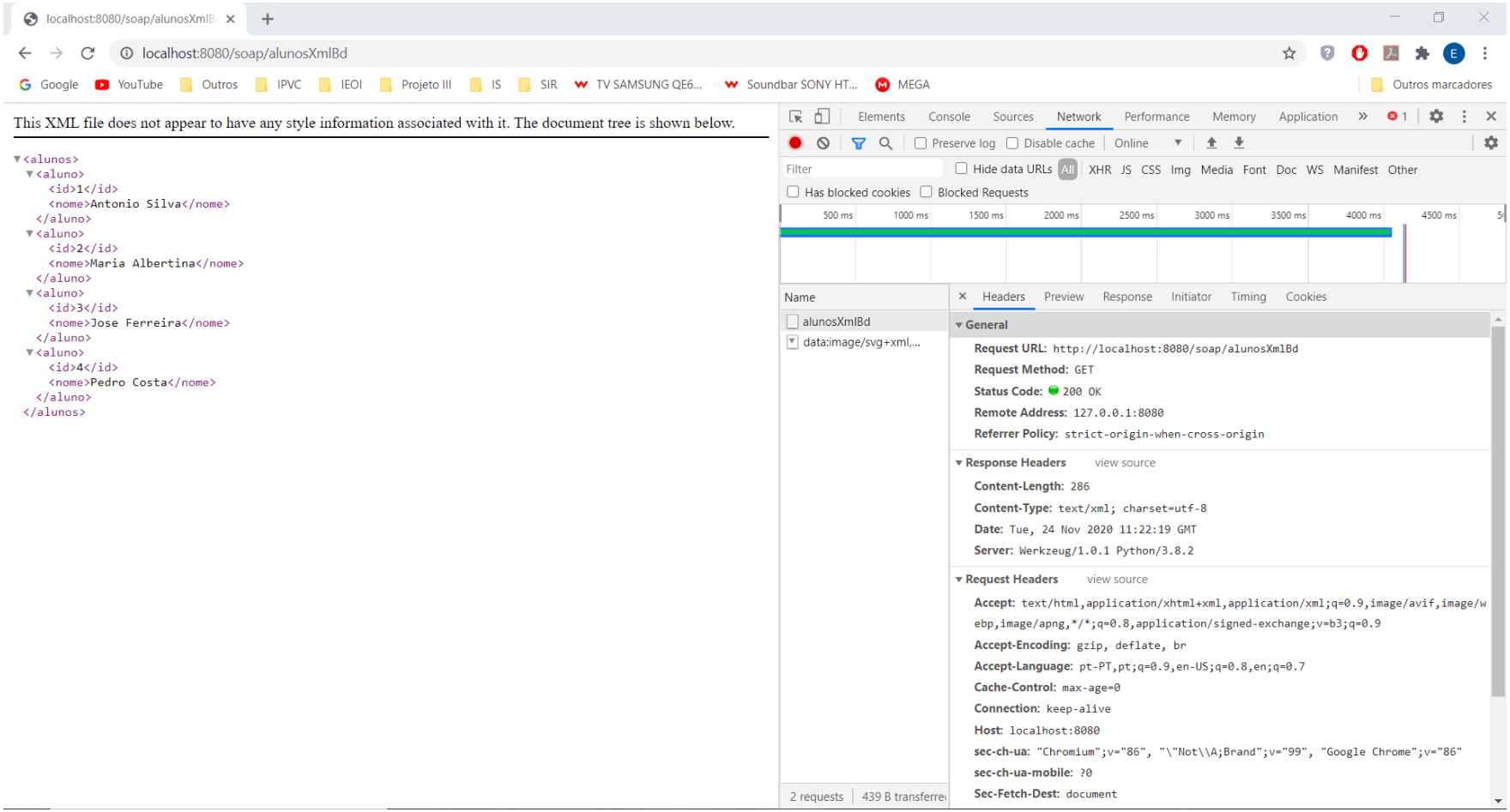
```python
if __name__ == '__main__':
    from wsgiref.simple_server import make_server

    wsgi_app = WsgiApplication(application)
    server = make_server('localhost', 8000, wsgi_app)
    server.serve_forever()
```

SoapXmlService.py | Client.py

```python
            tag_id = doc.createElement('id')
            content_id = doc.createTextNode(id_aluno)
            tag_id.appendChild(content_id)

            tag_nome = doc.createElement('nome')
            content_nome = doc.createTextNode(nome)
            tag_nome.appendChild(content_nome)

            tag_aluno.appendChild(tag_id)
            tag_aluno.appendChild(tag_nome)

            tag_alunos.appendChild(tag_aluno)

        doc.appendChild(tag_alunos)
        string_xml = doc.toprettyxml(indent="\t")
        caminho_ficheiro = "alunos.xml"
        with open(caminho_ficheiro, "w") as f:
            f.write(string_xml)

        return string_xml


application = Application([SoapXmlService],
                          tns='soap.alunos.db',
                          in_protocol=Soap11(validator='lxml'),
                          out_protocol=Soap11()
                          )
```

9.2 Python – SOAP web service

- Result obtained by calling the web service with the client.



9.2 Python – SOAP web service

- The image on top demonstrates the requests done on the client code, and the image on the bottom the requests and responses executed by the web service.

```
 * Running on http://localhost:8080/ (Press CTRL+C to quit)
127.0.0.1 - - [24/Nov/2020 11:21:41] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [24/Nov/2020 11:22:11] "GET /soap/alunosXmlBd HTTP/1.1" 200 -
127.0.0.1 - - [24/Nov/2020 11:22:19] "GET /soap/alunosXmlBd HTTP/1.1" 200 -
```

```
C:\Users\eduve\PycharmProjects\testeXPath\venv\Scripts\python.exe "C:/Integração de Sistemas/TP1/ex5_e_6_codigo/Python/BD/SOAP/SoapXmlService.py"
127.0.0.1 - - [24/Nov/2020 11:22:09] "GET /soap.alunos.db?wsdl HTTP/1.1" 200 2402
127.0.0.1 - - [24/Nov/2020 11:22:11] "POST /soap.alunos.db HTTP/1.1" 200 738
127.0.0.1 - - [24/Nov/2020 11:22:17] "GET /soap.alunos.db?wsdl HTTP/1.1" 200 2402
127.0.0.1 - - [24/Nov/2020 11:22:19] "POST /soap.alunos.db HTTP/1.1" 200 738
```

10. Conclusion

To conclude it is possible to state that this practical work has helped to increase our knowledge about the implementation of webservices in JAVA, PHP, JavaScript and Python, as well as utilizing different XML parsers to obtain and read the content of several XML files, with different sizes, and the implications of doing so in large files. This knowledge allowed us to solidify theoretical concepts learned during the classes and apply them practically, such as XPath and the use of its syntax. Additionally it allowed us to use connection methods to a MySQL database running on a local server and obtain data from it and process it in a webservice call.

11. Bibliography and Web References

[1]. TutorialsPoint . 2020. Java XML - Parsers. Available at: https://www.tutorialspoint.com/java_xml/java_xml_parsers.htm;

[2]. PhpMyAdmin Contributors. 2020. Documentation. Available at: https://www.phpmyadmin.net/docs/;

[3]. W3Schools . 2020. XML Validator. Available at: https://www.w3schools.com/xml/xml_validator.asp;

[4]. Arskom Ltd. 2020. What is Spyne?. Available at: http://spyne.io/#inprot=Soap11&outprot=XmlDocument&s=rpc&tpt=WsgiApplication&validator=true;

[5]. LinuxConfig.org . 2020. Deployment of an Example Application into Apache Tomcat Container. Available at: https://linuxconfig.org/deployment-of-an-example-application-into-apache-tomcat-container;

[6]. Python Software Foundation. 2020. Flask-Spyne 0.3.1. Available at: https://pypi.org/project/Flask-Spyne/;

11. Bibliography and Web References

[7]. Onejohi. 2020. Building a simple REST API with NodeJS and Express. Available at: https://medium.com/@onejohi/building-a-simple-rest-api-with-nodejs-and-express-da6273ed7ca9;

[8]. Filestack. 2020. Common problems with large file uploads. Available at: https://blog.filestack.com/thoughts-and-knowledge/common-problems-with-large-file-uploads/;

[9]. Filestack. 2019. How To Upload Large Files. Available at: https://blog.filestack.com/thoughts-and-knowledge/how-to-upload-large-files/;

[10]. James Kleeh. 2019. Code sample: Async parsing. Available at: https://github.com/FasterXML/aalto-xml/wiki/Code-sample:-Async-parsing;

[11]. The PHP Group. 2020. XMLReader. Available at: https://www.php.net/manual/en/book.xmlreader.php;

Instituto Politécnico de Viana do Castelo

Escola Superior de Tecnologia e Gestão

GRADUATION IN INFORMATICS ENGINEERING

Practical Work nº1

Parsing of XML Files using REST and SOAP via JAVA, PHP and PYTHON

- **Course Unit: Integração de Sistemas**

Eduardo Eiras Nº 21484

- **Coordinator: Prof. Doctor Jorge Ribeiro**

Instituto Politécnico de Viana do Castelo

Escola Superior de Tecnologia e Gestão

www.ipvc.pt