ET2503: "Mobile Services"

Laboratory exercises : Invoking Services

Blekinge Institute of Technology
School of Computing

Name: _____ Acronym: _____

Examiner: _____ Date: _____ Pass: _____

Contents

1 Introduction

1.1 Goals and ambitions

After having completed the following exercises you will be able to:

- Create your own SOAP client.
- Construct URLs for accessing services using REST principles.
- Access a web-based service using both SOAP and REST.

1.2 Recommended reading

The information needed to complete the exercises is easily found on the World Wide Web. The SOAP protocol is described at `http://www.w3.org/TR/soap/` and there is some information about REST in `http://rest.blueoxen.net/cgi-bin/wiki.pl?FrontPage`. not available 2011-09-12 00:35

1.3 Prerequisites

You will need rudimentary skills in navigating the UNIX system, launching programs and listing directories. Being able to use either emacs or vi is a definite advantage, as is previous contact with other markup languages, such as HTML and Standardised General Markup Language (SGML), as well as basic programming skills.

If you feel that your skills in these areas are somewhat weak, there is plentiful with information on the web on these topics and more.

1.4 Formatting

The text uses different typefaces to distinguish between user input and the output of commands. The italic Courier typeface signifies user input, whereas regular Courier typeface denotes the output from a command. It is also used for the source code listings.

1.5 Exercises

In this lab, you will create and manage a todo list using both SOAP and REST requests and responses. The data stored in the todo list is an unique id, a note, time for the event, priority and acronym.

SOAP is a protocol specification for exchanging structured information in the implementation of Web Services. REST is a set of rules that an architecture should conform to. Because it does not go into enough detail to define an actual architecture we call it an architectural style. The Web is an example of a real architecture that roughly follows the REST rules.

1.6 Languages and tools

You are allowed to use any programming language of your choice to solve the exercises. You are also allowed (and recommended) to use external software libraries for accessing the services. Choose a language you are comfortable with and that will simplify your work.

1.7 Deliverables

The following items need to be provided by you to pass the exercise:

1. The source code to all tasks in the lab. The source files *must* be named according to the following format: `acro_lab2_exY_taskZ.XXX`, where `acro` should be replaced by your studen acronym, `Y` by the corresponding *exercise* number, `Z` by the corresponding *task* number, and `XXX` by the file type. For example, a C program solving task 2 of exercise 1 might be named `abcd10_lab2_ex1_task2.c`, and a Python program might be called `xyzw10_lab2_ex1_task2.py`.

2. Any necessary information on how to compile and run your code. This information should be placed in a *text file* called `README`.

3. The HTTP requests and responses for each task, saved to *text files*. These should be named `acro_lab2_exY_taskZ_req.txt` and `acro_lab2_exY_taskZ_res.txt` for the requests and responses respectively. `acro`, `Y`, and `Z` should be replaced as indicated previously.

4. A short document giving answers to any questions posed in the exercises. The only acceptable formats are pure text or PDF. In particular, Word documents are not permissible! The document should be named '`acronym_report_lab2.txt`' or '`acronym_report_lab2.pdf`', where you should replace 'acronym' with your own acronym. An example would be: `abcd10_report.txt`.

Delivery of the files should be in a zip or tar.gz file, in which the files should be placed in a sub-directory called `acro_et2503_lab2`. The file itself should be called `acro_et2503_lab2.zip` or `acro_et2503_lab2.tar.gz`. Again, you should replace 'acro' with your own acronym.

If the archive is corrupt (i.e., if it is not possible to extract all files), if the correct name format is not used, or the archive does not contain a directory, you will fail examination.

Information on delivery dates will be disseminated separately.

1.8 Examination

During the lab examination, make sure your software is loaded and configured on the student account before presenting the lab. There is only a certain amount of time alotted for each examination, and the lab examiner will not wait for those who must copy/configure the software from removable media. Make sure your software is running without any problems. There will be no lab support, i.e. the lab examiner will not help solving potential problems with your code.

Lab assistance will be provided via email, and, preferably, using the course forums on IT's Learning.

1.9 Code of Ethics

The exercises are to be performed, written and implemented *individually*.

Sometimes it is necessary to ask an advisor or friend for help. This is permitted, provided that:

- You give due credit in the relevant files by placing a comment block at the top of the file.
- You fully understand the part which you did not solve yourself.
- Only a minor (5-10 %) part of the exercises have been solved in this manner.

Any other form of cooperation, copying and/or sharing of solutions will be considered a breach of the code of ethics and will warrant loss of credit and/or additional exercises.

2 REST exercises

The methods used in this lab will follow the CRUD concept, i.e., Creating, Reading, Updating and Destroying data.

2.1 REST API description

The general format of a REST request, as used for this small service, is:

`http://lol.comlab.bth.se:8080/?method=XXXX&param1=paramval1&param2=paramval2`

XXXX represents a specific method, and is used to perform one of the functions below. `param` and `paramval` represent the various parameters to the REST methods.

- To read from the todo list use either the `read` method, which takes `name` as an argument, or the `read_one_day` method that takes both `name` and a `date` as argument. The response will be a list with events.
- To create a new entry, use the `create` method, which takes the arguments `text`, `time`, `prio` and `user`. The response of this method will be a message, indicating if the operation was executed successfully or not.
- To delete or destroy an entry use the `delete` method with the arguments `id` and `name`. You obtain the `id` by using the methods `read` or `read_one_day`. The response of a `delete` will be a message indicating whether the operation was executed successfully or not.
- To change an entry in the todo list use the `update` method with the arguments `id`, `user`, `text`, `time`, `prio`. The response from this method will be a message, indicating if the operation was executed successfully or not.

A more concise list of REST calls is:

- `read(name)`
- `read_one_day(name)`
- `create(name,text,time,prio)`
- `delete(id,name)`
- `update(id,name,text,time,prio)`

2.1.1 Return Value Description

The return value of the `read` and `read_one_day` methods is a list of lists. Each ToDo returned will be represented by a single list, and if there is more than one ToDo returned, it will be placed in a list together with the other ToDos. The lists are encapsulated in parentheses, "`()`", and elements in the list are separated by commas "`,`".

The general format of a response is thus: `( TODO1 , TODO2, ...)`.

The general format for a single ToDo element is `(id, text,priority,date)`. `id` is an integer, always suffiex with the letter `L`. `text` and `priority` are strings, encapsulated in apostrophes "`'`". `date` is another list, prepended by `datetime.datetime`, of the format `(year,month,day,hour,minute)`.

Example: Typing the following URL in a web browser:

`http://lol.comlab.bth.se:8080/?method=read_one_day&name=abcd10&date=2010-05-17`

will yield a result in the following format:

```
((1L, 'meeting with my boss', '1', datetime.datetime(2010, 5, 17, 09, 0)),
(2L, 'dentist', '1', datetime.datetime(2010, 5, 17,15, 0)))
```

This means that there are two entries in the user abcd10's todo for 2010-05-17. The first with id=1, the note says "meeting with my boss", and the priority of the event is 1. The time for the event will be 09:00. The second entry, dentist also has priority 1 and the time for this event will be 15:00.

2.2 Exercise 1

Create and view your todo list. The answers (both the URL created and the response) should be placed in the report deliverable.

Task 1

Create a entry in the todo list with your acronym as argument to create.

Task 2

Read your todo list, using read with your acronym as argument.

Task 3

Create some more entries in the todo list with your acronym on different days.

2.3 Exercise 2

Manage your todo list. The answers (both the URL created and the response) should be placed in the report deliverable.

Task 1

Change (update) the first event.

Task 2

Delete the last event.

3 SOAP Exercises

In this exercise you will create your own SOAP client. A SOAP client allows you to access services exposed using SOAP messages. It performs dynamic bindings and executes methods at remote web services.

3.1 Access Methods

There is a SOAP server running on `http://lol.comlab.bth.se:8090`. The following SOAP methods are defined:

- `getTodoList(acronym)` → List of TodoData()
- `getTodoOneDay(acronym, date)` → List of TodoData()
- `createTodo(acronym, time, note, priority)` → String
- `updateTodo(id, acronym, time, note, priority)` → String
- `deleteTodo(acronym, id)` → String

The arguments to these SOAP requests are as follows:

```
acronym - String
note - String
date - String or datetime
time  - String or datetime
priority - Integer (between 0 and 10)
id - Integer
```

The response from the get-methods will be a list of entries in the todo list. The other methods will respond with a message indicating whether they have succeeded or not.

3.1.1 TodoData()

TodoData() is a serialized class that represents an entry in the todo list (id, prio, acro, note, date). Serialization is a process of converting a data structure or object into a sequence of bits so that it can be stored in a file, a memory buffer, or transmitted across a network. In the case of SOAP, the sequence of bits is SOAP XML document.

Typically, these SOAP XML documents are not written manually, but are instead generated by specific APIs. APIs are available for most programming languages, such as python, java, C++ or PHP. These languages provides APIs for creating, building and parsing SOAP messages.

Examples of both SOAP requests and responses are provided in Appendix A.

3.1.2 Exercise 1

Create and view your todo list. You are free to choose how to implement your SOAP client. You can build your request from scratch or use an API for this. In both cases everything, even the request and response messages, should be placed in the report deliverable.

Task 1

Create a entry in the todo list with your acronym as argument to create.

Task 2

Read your todo list, using read with your acronym as argument.

Task 3

Create some more entries in the todo list with your acronym on different days.

3.2 Exercise 2

Manage your todo list. Everything created for this should be placed in the deliverable.

Task 1

Change (update) the first event.

Task 2

Delete the last event.

4 Exercise 3

Prepare a short (about 1/2 a4 page) report, comparing and contrasting REST and SOAP. Some points to consider:

- When is REST better suited than SOAP and vice versa?
- Which is the more efficient? Flexible? Well-defined?

A SOAP XML Documents

A.1 getTodoList

Method call example: *getTodoList(mil)*

A.1.1 Request

getTodoList Request

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns="toDoServer.Services">
    <SOAP-ENV:Body>
```

```
        <getTodoList>
            <acro xsi:type="xs:string">abcd10</acro>
        </getTodoList>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

A.1.2 Response

getTodoList Response

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns="Services.Services">
    <SOAP-ENV:Body>
        <getTodoListResponse>
            <getTodoListResult type="tns:TodoDataArray">
                <TodoData>
                    <note xsi:type="xs:string">handla</note>
                    <date xsi:type="xs:string">2010-04-16 14:00:00</date>
                    <acro xs:null="1"/>
                    <prio xsi:type="xs:int">1</prio>
                    <id xsi:type="xs:int">1</id>
                </TodoData>
                <TodoData>
                    <note xsi:type="xs:string">sola</note>
                    <date xsi:type="xs:string">2010-04-18 14:00:00</date>
                    <acro xs:null="1"/>
                    <prio xsi:type="xs:int">1</prio>
                    <id xsi:type="xs:int">3</id>
                </TodoData>
                <TodoData>
                    <note xsi:type="xs:string">date</note>
                    <date xsi:type="xs:string">2010-04-18 00:00:00</date>
                    <acro xs:null="1"/>
                    <prio xsi:type="xs:int">2</prio>
                    <id xsi:type="xs:int">4</id>
                </TodoData>
                <TodoData>
                    <note xsi:type="xs:string">date2</note>
                    <date xsi:type="xs:string">2010-04-18 00:00:00</date>
                    <acro xs:null="1"/>
                    <prio xsi:type="xs:int">2</prio>
                    <id xsi:type="xs:int">5</id>
                </TodoData>
            </getTodoListResult>
        </getTodoListResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

A.2 deleteTodoList

Method call example: *deleteTodoList(mil,3)*

A.2.1 Request

deleteTodo Request

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns="toDoServer.Services">
    <SOAP-ENV:Body>
        <deleteTodo>
            <acro xsi:type="xs:string">abcd10</acro>
            <id xsi:type="xs: int">3</id>
        </deleteTodo>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

A.2.2 Response

deleteTodo Response

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
        <deleteTodoResponse>
            <deleteTodoResult xsi:type="xs:string">DELETED</deleteTodoResult>
        </deleteTodoResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

B Acronyms

REST Representational State Transfer

SOAP Simple Object Access Protocol