COMP28112 Exercise 2: Wedding Planner
Duration: 3 sessions

1 Introduction

You need to arrange, for the earliest time possible, a wedding. You are required to make two reservations — a hotel to host and a band to play at the reception. The aim of this exercise is for you to experience the inherent difficulties in building distributed applications that need to cope with concurrency, message delays and failures. Enjoy!

The hotel and the band both advertise slots 1 to 200 where slot 1 represents the earliest slot available while slot 200 represents the latest one.

As you need the band to play at the reception, the reservation for the hotel and the band must match — that is, if you reserve slot i for the hotel then you must also reserve slot i for the band. Note that the hotel and band restrict a user to hold at most two reservations at any one time.

For these lab sessions, you will write a client, using HTTP and XML, that competes with other clients (your fellow classmates) to reserve the hotel and the band.

The hotel and the band both provide a reservation service on the Web. They receive XML messages over HTTP and allow you to send in *requests* to reserve a slot, cancel a reservation, query to find available slots and query to find slots reserved by you. After your client has submitted a request your client needs, at some later time, to fetch the outcome of your request.

You will need to contend with the service being unavailable to receive your messages, with the fact that both the hotel and the band may not process your messages in the order in which they are received and delays before messages are processed. When your client tries to retrieve the results from a request, the message may not be available so your client will need to try again later. The services have been configured to randomly make themselves unavailable to incoming requests and to introduce arbitrary delays in processing requests. However, once the server acknowledges the receipt of your message you may assume your message will not be lost.

Finally, your client should not overwhelm the server with requests and should never send more than one request per second.

2 Session 1

2.1 Reserve a slot

Write a simple client using the language of your choice (C, C++, C#, Java, Perl, Python, Ruby, ...) to send a reservation request to the band (or the hotel).

To send a reservation request to the band you need to perform a `http put` operation with an XML message to the URL

`http://jewel.cs.man.ac.uk:3020/queue/enqueue`

If you wish to send your requests to the hotel then send your `put` request to

`http://jewel.cs.man.ac.uk:3010/queue/enqueue`

The put message must have the following headers:

- `Content-Type => 'application/xml'` and `Accept => 'application/xml'`.

The XML message (body of the `http put`) must be of the form shown in Listing 1. You must ensure that the triple (`request_id`, `username`, `password`) forms a unique identifier — that is, each message can be uniquely identified by `request_id`, `username` and `password`.

You need a strategy to ensure all your requests are unique.

Listing 1: Reservation request

```
<reserve>
  <request_id>id</request_id>
  <username>username</username>
  <password>password</password>
  <slot_id>slot_id</slot_id>
</reserve>
```

There are two possible responses from the server — "message received" or "the server is unavailable to receive your message". When the message is successfully received you will receive a 200 response code (for a list of defined http response codes see `http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html`). The body of the response will contain an `URI` that you will use later to fetch the outcome from your request – see Listing 2). If the service is unavailable to process your `put` request then you will receive, in the response header, a 503 response code.

Listing 2: Message received

```
<msg_uri>http://jewel.cs.man.ac.uk:3010/queue/msg/987</msg_uri>
```

Use a web browser and point to `http://jewel.cs.man.ac.uk:3010/queue/list` and you should see your message and its current state — *pending* or *processed*.

Your next task is to find out if your reservation request was successful. All you need to do is simply send a `http get` to the message's URI with your username password (you just used) appended to the end of the string — e.g. from 2, the URI is:

`http://jewel.cs.man.ac.uk:3010/queue/msg/987?username=name&password=pwd`

You can also use this URI in any web browser to view your message.

The possible responses when you retrieve your message are:

- successfully retrieved your message — `http` response code 200;

- failed as your request has not been processed — `http` response code 404;
- failed as the service is unavailable — `http` response code 503;
- failed as you supplied an invalid username/password — `http` response code 401.

If you are successful in retrieving your message then the body of the response will tell you the outcome of your request and possible outcomes are:

- the slot you requested has been reserved for you — see Listing 3 and note how we reuse the http response code to communicate the result of your request;
- the reservation failed as it is not free — see Listing 4;
- the reservation failed as you already hold the maximum permitted number of reservations — see Listing 5;
- the reservation failed as you supplied an invalid username password — see Listing 6;
- the reservation failed as the slot you requested does not exist — see Listing 7;
- the request is invalid — see Listing 8

Open the URL `http://jewel.cs.man.ac.uk:3010/booking` using your web browser to see your reservations.

Listing 3: Successful Reservation

```
<response>
  <code>200</code>
  <body><reserve>Reserved slot 1</reserve></body>
</response>
```

Listing 4: Slot is not free

```
<response>
  <code>409</code>
  <body>Slot 1 is not free.</body>
</response>
```

Listing 5: Maximum permitted number of reservations

```
<response>
  <code>409</code>
  <body>
    Reservation failed, you already hold the maximum
    permitted number of reservations - 2
  </body>
</response>
```

Listing 6: Invalid username password

```
<response>
  <code>401</code>
  <body>Reservation failed due to an invalid username password
  </body>
</response>
```

Listing 7: Slot does not exist

```
<response>
  <code>403</code>
  <body>Slot 450 does not exist</body>
</response>
```

Listing 8: Invalid request

```
<response>
  <code>510</code>
  <body>Invalid Request</body>
</response>
```

2.2 Cancel a reservation

Extend your client to request the cancellation of a reservation. This time, the body of the message should take the form shown in Listing 9

Listing 9: HTTP body for cancellation a reservation

```
<cancel>
  <request_id>request_id</request_id>
  <username>username</username>
  <password>password</password>
  <slot_id>slot_id</slot_id>
</cancel>
```

You will receive exactly the same set of responses as before. However, the result of processing your request is a little different to before. The possible responses are:

- The reservation has been cancelled — code 200;
- The cancellation failed as the slot was not reserved by you — code: 409;
- The cancellation failed as the username password was invalid — code: 401.

2.3 Finding free slots

Extend your client to find free slots. This time, the XML document you need to send is of the form shown in Listing 10 while the response will look like the XML shown in Listing 11. You can also point your web browser to
`http://jewel.cs.man.ac.uk:3020/booking/available` to see the list of free slots.

Listing 10: Query availability

```
<availability>
  <request_id>request_id</request_id>
  <username>username</username>
  <password>password</password>
</availability>
```

Listing 11: Query — Availability

```
<response>
  <code>200</code>
  <body>
  <availability>
    <slot_id>4</slot_id>
    <slot_id>12</slot_id>
    ...
    <slot_id>15</slot_id>
  </availability>
  </body>
</response>
```

2.4 Slots reserved by you

You can also send a request to find all the slots that are reserved by you. The XML is of the form shown in Listing 12.

Listing 12: Query — Reservations

```
<bookings>
  <request_id>request_id</request_id>
  <username>username</username>
  <password>password</password>
</bookings>
```

The response is of the form shown in Listing 13.

Listing 13: Bookings

```
<response>
  <code>200</code>
  <body>
  <bookings>
    <slot_id>1</slot_id>
    <slot_id>5</slot_id>
  </bookings>
  </body>
</response>
```

2.5 Skeleton Java Client

If you decide to write your client in Java then you can write it by extending the skeleton Java code located in $COMP28112/ex2/java-client/. The file README.txt provides information regarding configuration.

Documentation to the Java skeleton is also provided via doc/index.html in that directory.

The code provides two sample clients:

- ClientReserve: client prepares a reservation request, sends the request using `http put` and then displays the result of the request;
- ClientGetMsg: client uses `http get` to retrieve and display a message that is identified by its URI.

You will need to write the missing code in XMLRequest.java and develop your client. You should start on Session 2 as soon as you have submitted Session 1.

3 Sessions 2 & 3 — Reserving identical slots (as early as possible) for the hotel and the band

Now the fun and games start. You now need to design and implement a strategy to reserve the same numbered slot for both the hotel and the band — that is, if you reserve slot 10 for the band then you also need to reserve slot 10 for the hotel. Remember, you are permitted to have at most two reservations for the hotel and two reservations for the band.

This strategy should try to reserve the earliest available matching slots. Note, we will sometimes block slots from reservations and then release them during the labs.

N.B. It is quite easy, by having a number of clients fire requests at the server as quickly as they can, for this to turn into a denial of service attack! It is strongly recommended that client code which loops round interrogating the server should include a deliberate delay (of say one second) to reduce the danger of this. Code which ignores this advice will be penalised in the assessment.

4 Assessment

Session 1 tasks	
Working client that checks for free slots	1
Working client that reserves a slot	1
Working client that cancels a slot	1
Working client that retrieves bookings	1
A simple strategy to generate unique identifiers for requests	1
Sessions 2 & 3 tasks	
Sort out current bookings	1
Check availability	1
Book earliest common slot	3
Recheck at least once for better bookings	2
Add always an one second delay	1
Fully working and checking, and explanation	2

To get full marks for Sessions 2 & 3 you need to have a fully operational program (client) that implements a strategy for booking matching slots that relies on availability, does not give up when a booking request fails, and will cancel unmatched bookings. Furthermore, your strategy should find the earliest matching slot, not any matching slot; and, once a matching slot is found, it should also check for a better booking (remember, the availability data for the band and the hotel is out-of-date as soon as it is received!); you need to make sure that you cancel bookings after obtaining better ones, but you need to maintain always at least one booking; also, your code should not give up if no matching slots are available. Finally, there must be a delay of at least 1 second between successive requests to the server (even if it is a retry).

In the above, to get full marks you should always assume that slot availability changes frequently and messages may be delayed or lost!

Remember that you are not allowed to give to anybody your username and password (even more you should not put username/password or any code online). If you ignore this or the 1-second-delay rule you may be penalized during marking.

5 Submission

You need to submit your solution to session 1. To do this, copy all your code into a single file called `part1` in your `ex2` directory. To submit your code after sessions 2 and 3, please copy all your code from sessions 2 and 3 into the file `ex22/part2`. Note: markers will insist that the creation times of the two files are later than the last modification times on any of the components! Of course, you always need to submit your exercise before the respective deadlines!