COMP30220 Distributed Systems Practical

Lab 4: RESTful Distribution

Work individually. Submit your code on csmoodle.ucd.ie by the deadline given on moodle.

Scenario:

After returning from an industry expo on new trends in enterprise systems, you manager has again changed their minds about how to implement the quotation system. This week, they have decided that the system should be RESTful rather than a web service. As a result, you are tasked with the evolution of the system to a RESTful web services model.

Your starting point for this new version is a third version of the code that takes account of some of the changes you introduced when implementing the web services solution: namely that the `Quotation` and `ClientInfo` interfaces are now classes. Again, use the sample client info that has been provided in the `test.Main` class.

A key difference between web services and RESTful services is the focus on applying standard CRUD operations to resources identified by URLs. Further, create operations are often to be differentiated from read operations. This means that generating a quotation should be separated from reading a quotation. In the REST model, creating a resource typically results in a response that provides the primary key of the resource and a url to access the newly created resource. The problem statement below breaks down the task of moving to RESTful services in to manageable steps.

Problem 1: Generating a quotation **Total: 50%**

The first task is to implement the quotation services as RESTful services. To do this, we are going to create a restlet that wraps around a quotation service. We will then use the same restlet implementation for all of the quotation services.

Lets start with some design. Our quotation service is going to need a standard URL both for accessing existing quotations and creating new quotations. Let's choose the following URL template:

`http://<host:port>/<service-id>/quotation[/<quotation-ref>]`

Note that the /<quotation-ref> part is optional and should only be used to refer to specific quotations (that you have already created).

To implement this, we need to create a restlet application called `QuotationApplication` that has 3 fields: (i) a reference to a `QuotationService` implementation, (ii) a Map whose keys are quotation reference numbers and whose values are quotations, and (iii) an instance of the Gson Parser. This application will implement 2 inline restlets: (i) to handle quotation creation ("/quotation"), and (ii) to handle quotation retrieval ("/quotation/{reference}"). This service implementation is basically the same as the Student Record service we covered in the course notes and you should study this for more details.

To deploy the service, you will instantiate this application 3 times – one for each quotation service, for example:

```java
public class QuotationServer {
      public static void main(String[] args) throws Exception {
            Component component = new Component();
            component.getServers().add(Protocol.HTTP, 8182);
            component.getDefaultHost().
                  attach("/afq", new QuotationApplication(new AFQService()));
            component.start();
      }
}
```

The response from the POST /quotation operation should call generateQuotation(…); store the created quotation in the Map, using the reference as the key; and return both the key and the URL required to retrieve the quotation:

{ "quotation-ref" : "<reference>", "url" : "<quotation-url>"}

Remember, to make this easy, you should create Java objects that represent the JSON you want to return and use Gson to translate between Java and Json (and vice versa).

NOTE: To get the part of the URL that identifies a restlet, you can use the following code inside of the handle(…) method:

```
request.getRootRef().toString()
```

For the above QuotationServer example, this would give:

```
http://localhost:8182/afq
```

Problem 2: Adding the Vetting Service **Total: 30%**

Create a `VettingRestlet` class that implements the vetting service as a RESTful service. To do this, you will need to change the way this service works. Instead of performing the check on the server, a RESTful implementation should retrieve the points from the server (i.e. the points map is treated like the quotations map in problem 1). The check should then be carried out on the client side. For the purposes of this worksheet, you only need to implement support for the GET method. Points Resources should be accessible via the following URL:

```
http://<host:port>/vetting/{license-number}
```

The service should return something of the form:

```
{ “licenseNumber” : “<license-number>”, “points”:<points> }
```

You can attach restlets directly to the component in the same way that you can attach applications.

NOTE: To work with this class, you will need to change either: (1) perform a url encoding of the license number or (2) change the format of the licence numbers so that the slash (“/”) is not used (this is a delimiter in a URL so the license number cannot be passed in its raw form as part of the url). More marks will be given for encoding the url over the changing of the format (because this change could not happen in real life).

Problem 3: Updating the BrokerService **Total: 20%**

Modify the Broker Service (a new class called `RestletBroker` should be created) to work with the RESTful interfaces developed in problems 1 and 2. You will have to change the implantation of this class more significantly than in previous worksheets. I recommend creating a private vetClient(…) method in the broker implementation so that the change to the getQuotations() method are minimal.

Test your solution by modifying the Main class to work with the new broker.