REST API's Error Handling

1) Error Handling with JAX RS - Quotes

Prerequisites: This exercise requires that you have completed the exercise "REST with JAX RS – Quotes"

In this exercise we are going to complete the API-description given in the original exercise with a description of the responses for **Error Scenarios** as sketched below:

Method	URI	
GET	`api/quote/{id}`	Returns the quote with the given id as: `{"quote" : "Quote text"}` If the no quote with the given id exist: returns (1)
GET	`api/quote/random`	Returns a random quote as: `{"quote" : "Quote text"}` If no quotes exists: returns (2)
POST	`api/quote`	Creates the quote supplied with the request as: `{"quote" : "Quote text"}` Response: `{"id": newId, "quote": "Quote text"}`
PUT	`api/quote/{id}`	Changes the quote with the given id to the text given with the request as: `{"quote" : "Quote text"}` Response: `{"id": newId, "quote": "Quote text"}` If the no quote with the given id exist: returns (1)
DELETE	`api/quote/{id}`	Deletes the quote with the given ID Response: `{"quote" : "Quote text"}` If the no quote with the given id exist: returns (1)

- For all exceptions not included above return: (3)
- For request for non-existing services (i.e.: `api/jfskjfajf`) return: (4)

(1): {"code": 404, "message": "Quote with requested id not found"}
(2): {"code": 404, "message": "No Quotes Created yet"}
(3): {"code": 500, "message": "Internal server Error, we are very sorry for the inconvenience"}
(4): {"code": 404 "The page/service you requested does not exist"}
For all errors, the HTTP Response Status Code must be set to the value given in code.

Tasks

Server side (See slides for info about the ExceptionMapper Class):

1) Implement a QuoteNotFoundException and throw the exception with a description as described above.
2) Test and reflect on the result
3) Implement a QuoteNotFoundExceptionMapper class that transforms the exception into a sufficient JSON reply (as described above). Test and verify that both the correct status code and response is generated.
4) Implement a generic ExceptionMapper (Throwable) that should build the response described in (3)
5) Test the mapper implemented above by throwing a NullPointerException or a similar RuntimeException in the GET method for an id=5 (don't forget to remove this code again ;-)
6) Implement an ExceptionMapper that maps `NotFoundException`[1] into the response given in (4)
7) Test the Mapper with a URL like "/api/quote/raaandom"
8) Implement and test the missing error handling in the REST methods as described above

[1] The exception the container will throw for a non-existing URL

9) Answer the question: *Why is it important that the HTTP Response Status Code is set to an error Code (status >=400) when it is also set in the JSON response?*
   You might want to monitor the response with Chrome Developer Tools to answer this question, or see the following question.
10) Change your client code so that errors are reported to clients (see hints below).

Hint: Use a Bootstrap alert to present the error: http://getbootstrap.com/components/#alerts

In the `fail` or `error` handler of your Ajax request you can get the response JSON like:

```
.fail(function (error) {
            var json = error.responseJSON;
            $("#error").show().html(json.message);
          });
```

2) Error Handling with JAX RS - Person

Prerequisites: This exercise requires that you have completed the exercise "REST with JAX RS – Person…."

Again we are going to complete the API-description given in the original exercise, with a description of the responses for **Error Scenarios** as sketched below:

In this exercise, we will include the stack trace into the error-JSON, **but only** when we execute the program in debug-mode, hackers will love to get this info ;-)

Error responses for GET: /person/{id}
{"code": 404, "message": "No person with provided id found", "stackTrace": "…."}

Error responses for POST:
A person must have both a firstName and a lastName
{"code": 400, "message": "First Name or Last Name is missing", "stackTrace": "…."}

Error responses for PUT:
A person must have both a firstName and a lastName
{"code": 400, "message": "First Name or Last Name is missing", "stackTrace": "…."}
{"code": 404, "message": "Cannot edit. Person with provided id does not exist", "stackTrace": "…."}

Error responses for DELETE:
{"code": 404, "message": "Could not delete. No person with provided id exists", "stackTrace": "…."}

For all **RunTimeExceptions**
{"code": 500, "message": "Internal Server Problem. We are sorry for the inconvenience", "stackTrace": "…."}

Request for **non-existing Services**
{"code": 404, "message": "The requested service does not exist", "stackTrace": "…."}

Tasks

Server:

1) Remember, exceptions should **only** be included while we are developing/debugging the system. Include (if not already done) a Standard Deployment Descriptor file (web.xml) into your project and add the following xml:
   ```
   <context-param>
      <description>If true, the stack trace is included with error responses</description>
      <param-name>debug</param-name>
      <param-value>true</param-value>
   </context-param>
   ```
   Stack traces should only be included when this context-parameter is true.
   See the slide "*Exception Handling-3*" to see how to read this value from your code

2) Design two Exception classes; `PersonNotFoundException` and `ValidationErrorException` and `ExceptionMappers` to map these exceptions into JSON-responses as described above.
3) Implement an `ExceptionMapper` that will Map all Exceptions not having their own `ExceptionMapper`
4) Implement an `ExceptionMapper` that will Map `NotFoundExceptions` to the relevant JSON-Response
5) Verify the error-response for "all" possible error conditions.

Client Side (using the REST-API via AJAX)

6) Change your client code so that errors are reported to clients