Seminar #6

Miroslav Blaško

1 Goal

Get experience with implementation and testing of REST web services using the Spring framework.

2 Getting Ready

- Ensure you have available the software stack installed during the first lab with Netbeans 8.2
- Clone the GIT repo `https://gitlab.fel.cvut.cz/ear/seminar-rest`
- Open project in Netbeans 8.2 and resolve any warnings/errors reported by Netbeans
- Install Postman plugin (`https://www.getpostman.com/`) into your Chrome browser. (As an alternative for Firefox users i suggest HttpRequester plugin available at `https://addons.mozilla.org/firefox/addon/httprequester/`))

3 Becoming Familiar

The project `seminar-rest` is modified excerpt of a bigger project introduced in previous seminars – the reporting tool project for safety management in the czech aviation industry.

3.1 Understanding REST interface for reports

REST interface for management of *reports* is defined in the file `ReportController.java`. The interface allows to create *revisions* of a *report*. Whenever new *revision* of a *report* is created, its older *revisions* become read-only. All *revisions* of same *report* are organized in one *chain* of revisions. The interface allows to reference individual *reports*, *revisions* or *chains of revisions*.

4 Test the REST interface

There are JUnit tests within the project that are verifying correct behaviour of REST API. We will use them to figure out how the API should work and then test in on the running application by a browser. This can be done as follows:

- **pick scenarios for interaction with REST API** – Clean and build the project. Run all tests and pick at least three different scenarios where tests of REST interface are **not** failing. Each scenario should be using different HTTP method (e.g. GET, POST, PUT). Suggested scenarios:
  - `ReportControllerTest.getReportReturnsNotFoundForUnknownId()`
  - `ReportControllerTest.createReportReturnsLocationOfNewInstance()`. You can test effect of the operation again by "/reports" or "/reports/{id}"
  - `ReportControllerTest.updateReportPassesReportToServiceForUpdate`. You can test effect of the operation again by "/reports" or "/reports/{id}".
- **deploy the application** – Deploy the application to a Tomcat server. In order to do it we will might need to disable tests, which can be done in a "run configuration" within your IDE[1]. Alternatively you can use maven through command-line (e.g. "mvn package -DskipTests").
- **test selected scenarios from a browser** – Test selected scenarios on running application using Postman plugin of your Chrome browser. Note different HTTP headers (e.g. Accept, Content-Type, Location) and HTTP status codes (e.g. 200 OK, 404 Not Found).

5 Repair failing tests

Run all tests of the project again. Some of the tests are failing mostly due to missing/wrong annotations, other ones require little coding. Fix all tests and deploy the application. You can find all relevant annotations within *org.springframework.web.bind.annotation* package[2].

Log in to the application to see how it uses corrected REST API. Navigate to *"Dashboards/Create report"*. Try modifying a report and play with *"Save"* and *"Create new revision"* buttons to create multiple revisions of the report.

Use Chrome development tools to see how is your browser communicating with your application. This can be done by pressing *F12* in Chrome, navigating to *Network* section and pushing a button (e.g. *"Create new revision"*).

Moreover, you can check correct solution for this excercise. It is located in git repository of this project, within the branch `seminar/rest/solution`.

[1] In *Netbeans* it should be available through "$PROJECT/Properties/Actions/Run Project/Add/skip tests"

[2] online at `http://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/package-summary.html`

6 Implement additional services

Spring REST controllers defined in `ReportController.java` and `PersonController.java` can be easily extended to create friendlier interface. Argue which of the HTTP methods would be useful for following paths:

- /persons/{username}/reports
- /persons/{username}/reports/chains
- /reports/chains
- /reports/{id}/chain
- /reports/{id}/revisions

Which of the paths above are suitable for HTTP methods GET, POST, and DELETE ? Pick most appropriate and implement it. Create JUnit tests to check correctness of the implementation.