Workshop 2: A Service Based Approach, RESTful Web Services with JAX-RS and AngularJS

DAT076/DIT126 Chalmers/Göteborgs Universitet
Joachim von Hacht

Objectives

The goal for this workshop is the same as the previous but here we use a service based approach.

You need the following tools and skills;

- Tools as before and GlassFish 4.x (replacing Tomcat)
- JavaScript core language (no APIs) and AngularJS (very basic Jasmine for testing)
- Java EE RESTful Web Services (JAX-RS aka Jersey, runtime bundled with GlassFish)
- Optional: Cache control, conditional GET's and PUT's and OAuth

Please, have a look at the code samples, everything you need should be there. Also links in slides for more detailed information.

Functionality

To see the intended functionality watch the video (same as for previous workshop)

GlassFish

GlassFish is our REST server (Application Server)

1. Go to Services tab > Servers > Mark GlassFish > Properties. Inspect (select HttpMonitor!)
2. Services tab > Servers > Mark GlassFish > Start
3. Mark GlassFish > View Domain Admin Console. A Web page should show up, note port. This is the administration tool for GlassFish. Peek around! GlassFish is a very complex piece of software …
4. Close the admin console and stop the server.

The Backend

The workshop is contained in a single Maven project for both front- and back-end. We'll start with the back-end (server-side).

1. Download the code skeleton from the course page and open it in NetBeans. Inspect. Make sure the dependency on the shop model is present and working.
2. Check that GlassFish is selected for run
3. It's should be possible to run the code (no functionality will work)!
4. The only back-end class that you need to complete (possibly modify) is the ProductCatalogueResource (PCR).
5. The PCR is a JAX-RS root resource class responsible for exposing the ProductCatalogue to clients. It should have the same methods as the IEntityContainer (in shop) but with different input and output types. The return type should be “Response” for all methods. PCR converts to the correct type before and after the call to ProductCatalogue (in Shop). The methods must of course be mapped to URLs/HTTP-methods and have the correct produces/consumes annotations.
6. Implement each method in PCR and use cURL (or similar) to issue HTTP requests to the service. See Test Packages > curl_test.txt.

Warning: There are annotations with the same names but from different packages! You must select the correct ones. For now just use the annotations from java.xml.bind and javax.ws.rs! Watch out!
Tip: You must use the GenericEntity class as a wrapper before adding Lists (or other generics) to Response.
Tip: You can't return primitive types. Use JsonObject as a wrapper.

The Frontend

Now we'll develop a “single page” AngularJS/JavaScript client for the service.
Tip: To debug the running JavaScript use Chrome/Developer Tools, Firefox/Firebug or similar.

1. The files to work with are in Web Pages/shop/app/js and Web Pages/shop/app/partials/products.
2. Start with the services.js, an AngularJS service module. Add a factory that will create the service object (named ProductResourceProxy or similar). The service object should have the same methods as PCR. It will act as a proxy for the back-end resource class. The service only responsibility is to issue AJAX requests on the PCR. Let all methods return promises (calling methods on $http will yield a promise).
3. (Optional) Create some Jasmin tests to test the service (or do it in some other way, debugger...). Use the ... /shop/test folder.

4. Continue in parallel with the $routeProvider (in app.js), products.html and controllers.js. Start out with a very simple page and controller. Let the controller do some basic call on the service. Call chain: URL -> routeProvider -> control -> service -> back to control setting return value as a $scope variable - > page, using ng-directives (and {{ ... }}) to loop over result in variable.
5. Add list pagination.
6. Master-Detail: Complete product-detail.html (for edit and delete) and product-new.html. Create controls for each.

Have fun … (optional)

Add Cache control, conditional GET and updates and/or OAuth authorization ...

Conditional GETs/updates

1. Refactor ProductsCatalogueResource and rename it to ProductsCatalogueResourceCond. Change @Path to /cond.
2. Modify the new resource. Add conditional gets and updates for relevant methods.
3. Check with cURL (new tests need to be created). See code samples.

Authentication and Authorization

1. Add some authorization using some OAuth provider. Refer to the code samples on how to use Twitter or Google.

NetBeans Project structure