JAX-RS 2 med Jetty, Jersey + Moxy + JSON

Hvordan komme i gang med REST, fort

Leif Olsen

SITS

REpresentational **State Transfer**, Arkitektur

Hva er REST

REpresentational **S**tate **T**ransfer. REST, er en HTTP-sentrisk protokoll med et enhetlig grensesnitt. Protokollen forholder seg til et begrenset sett med metoder, POST, GET, PUT, DELETE, (HEAD, OPTIONS, PATCH)

http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

- Annotasjonsdrevet
- POJO-basert
- Adresserbare ressurser i et distribuert hypermediasystem
  - ID
  - URI
- Samme ressurs kan returnere resultatet på flere formater (JSON/TEXT/XML ++)
- Velegnet for CRUD-liknende operasjoner
- Tilstandsløs (og distribuerbar)
- Støtter HTTP Caching
- REST er en spesifikasjon - IKKE en standard
- Består av et sett med retningslinjer og *mønsterpraksiser (best practices)*

Nytt i JAX-RS 2

- Client API
  - Kjøkt bl.a. for integrasjonstesting
  - Kommunikasjon mellom separate tjenester
- Async
  - Future eller InvocationCallback
- WATCHES
  - Hypermedia as the Engine of Application State
  - Ingen WATCHES ingen REST
  - Link og Target benyttes for å utvikle hyperlinker mellom tjener og klient
  - Lit over det den hjelp å frik JAX-RS 2
  - Ikke en standard
  - Det er opp til den enkelte applikasjonen å tilby noe fornuftig

Nytt i JAX-RS 2

- Annotations
  - Nye annotasjoner bl.a. for injection
- Bean Validation
  - Nytt er validering av parametre på metodenivå
- Filters and Handlers
  - Filter API: bit med Servlet 3
  - Handler: "Interceptor"
- Content negotiation
  - Utviklende i forholdende annotasjoner for @Accepts og @Produces parametre

JAX-RS 2 Api Anatomi (stjelt fra internettet)

Feilhåndtering

- Klientsiden kan ikke forholde seg til Exception som opptrår på serveren
  - må altid vædere returnert HTTP status
- Man min anånt kaste WebApplicationException med ønsket HTTP-status eller returnere en Response-objekt
- Feil som ikke håndteres fører til at serveren returnerer HTTP-status 500, INTERNAL_SERVER_ERROR, til klienten
- Unntaket er WebApplicationException og ConstraintViolationException
  - WebApplicationException returnerer statuskode angitt i konstruktør
  - ConstraintViolationException returnerer 400, BAD_REQUEST
- Man må selv sørge for at formålig system for overføring av feilmeldinger til klient
- Eventuelt konfigurere egen Exception Mapper som håndterer alle mulige feil og som sørger for å overføre feilmeldinger til klienten på et definert mønsterformat

Feilhåndtering

```
throw new WebApplicationException(
    Response.status(Response.Status.NOT_FOUND)
            .entity("Ansatt med id 0815 finnes ikke")
            .type("text/plain").build();
            .cookie(newCookie("TOKEN","0815"
            .location(URI.create("/ansatt/0815")
            .lastMod(...)
            .tag(...)
            .cacheControl(...)
            .build();
```

HATEOAS - Richardson Maturity Model

HATEOAS: Collection+JSON - Hypermedia Type (Level 3)

```
{ "collection" :
    { "version" : "1.0",
      "href" : "http://example.com/todos/?offset=100?limit=10",
      "links" : [
        { "rel" : "prevpg", "href" : "http://example.com/todos/?
        { "rel" : "nextpg", "href" : "http://example.com/todos/?offset=200?limit=10"
        { "rel" : "profile", "href" : "http://example.com/todos/?offset=100?limit=10"
      ],
      "items" : [
        { "href" : "http://example.com/todos/0789987535?",
          "data" : [
            { "name" : "Description", "value" : "Buy groceries", "prompt" : "Description"
            { "name" : "Completed", "value" : "false", "prompt" : "Completed"
            { "name" : "Priority", "value" : "3", "prompt" : "Priority"
            { "name" : "Due Date", "value" : "2012-12-24T00:00:00", "prompt" : "Due Date"
          ],
          "links" : [
            { "rel" : "parent", "href" : "http://example.com/todos/0789987535?"
            { "rel" : "edit-form", "href" : "http://example.com/todos/0789987535?"
          ]
        }
      ],
      "template" :
        { "data" :
          [ { "name" : "Description", "value" : "", "prompt" : "Description"
            { "name" : "Completed", "value" : "false", "prompt" : "Completed"
            { "name" : "Priority", "value" : "", "prompt" : "Priority"
            { "name" : "Due Date", "value" : "", "prompt" : "Due Date"
          ]
        }
    }
}
```

Hva trenger vi for å komme i gang, fort som f..

Eksempelkoden: https://github.com/LeifOlsen/simple-jersi2

POM

```
<dependencies>
    <dependencyMgmt.glassfish.jersey.containers.</groupid>
    <artifactId>jersey-container-grizzlyt-servlet</artifactid>
    <version>2.0</version>
    <dependency>
        <groupid>org.glassfish.jersey.media.</groupid>
        <artifactid>jersey-media-moxy</artifactid>
    </dependency>
</dependencies>
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupid>org.glassfish.jersey.</groupid>
            <artifactid>jersey-bom</artifactid>
            <version>${jersey.version}
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
</dependencies>
```

POM

Disse avhengighetene gir oss fetgende funksjonalitet - uten spesiell konfigurasjon av serveren:

- Annotasjonsbaset konfigurasjon av tjenestene iht. Servlet3-standarden
- Annotasjonsdrevet API
- JSON binding feature from jersey-media-moxy
  - automatisk serialisering/deserialisering mellom Java og JSON/XML
- jersey-media-json-processing
- jersey-bean-validation
- WadlFeature - enables WADL processing
- Jersey-client - a URI-based content negotiation filter.

. I tillegg trenger vi noen avhengigheter for Logback, Guava, Guice, JUnit, Jetty og eventuelle database/driver - men disse er ikke nødvendige for å komme i gang!

Embedded Grizzly

```
1  import ...
2
3  public class EmbeddedHttpSrv {
4      private static final Uri baseUri = URI.create("http://localhost:8080");
5      public static Thread ste = null;
6      private Map<Class<?>,Boolean> jerseyApplicationConfig = new HashMap<Class<?>,Boolean>();
7
8      public static HttpServer startServer() {
9          final ResourceConfig rc = new ResourceConfig();
10         rc.register(JacksonJsonProvider.class);
11
12     public static void main(String[] args) throws IOException {
13         final HttpServer server = startServer();
14         System.out.println("Jersey app started with WADL available at "
15                 + baseUri + "application.wadl\nHit enter to stop it...", getBytes("UTF-8"));
16         System.in.read();
17         server.shutdowNow();
18     }
19 }
```

Web Deployment Descriptor, erstatter web.xml

```
1  import ...
2
3  @WebServlet(loadOnStartup = 1)
4  @ApplicationPath("/api/*")
5
6  public class ApplicationConfig extends ResourceConfig {
7      public static Thread ste=null;
8      private Final Logger logger = LoggerFactory.getLogger(getClass());
9
10     public ApplicationConfig() {
11         // Jersey uses java.util.logging - bridge to slf4j
12         ...
13         packages("no.example.rest.resources");
14         ...
15         // Scan during deployment for rest-ws components in package
16         packages("com.example.singleservice");
17
18         // Enable LoggingFilter & output entity.
19         register(LoggingFilter.class);
20         ...
21         ...java.util.logging.Logger.getLogger(SSEs.getClass(), getName());, true);
22         ...
23     }
24 }
```

JAX-RS Resource

```
1  import ...
2
3  @Singleton
4  @Path("test")
5  @ProduceJsonMediaType.com.sun.jersey.spi.resource.
6  public class TestResource {
7      private Final Logger logger = LoggerFactory.getClass();
8
9      // output urls info provided by parent resource (throwable)
10     private ExtendedUriInfo eui;
11
12     public TestResource(@Context UriInfo aui) {
13         // cannot inject extended interface
14         this.eui = (ExtendedUriInfo) aui;
15         logger.debug("Resource created");
16     }
17
18     @Get
19     @ProduceJsonMediaTypes("TEXT_PLAIN")
20     @Path("/ping")
21     public Response ping() {
22         return "Pong" // <- Response.status.ok
23     }
24 }
```

Integrasjonstest, Client API

```
1  import ...
2
3  public class HomeResourceTest {
4      private static Final String WEB_RESOURCE_PATH = "home";
5      private static ExecutorService server;
6      private static LoggerLogger logger;
7
8      @BeforeClass
9      public static void setup() {
10         // start the server
11         server = JettyWebSrv.startServer();
12
13         // create the client
14         Client c = ClientBuilder.create(ClientConfig.
15         target = c.target(UriBuilder.fromUri(BASE_URI));
16     }
17
18     @AfterClass
19     public static void teardown() {
20         // stop the server
21         server.shutdowNow();
22     }
23
24
25
26
27
28
29
30
31 }
```

```
2  import ...
3
4  public class HomeResourceTest {
5      private static Final String WEB_RESOURCE_PATH = "home";
6      private static ExecutorService server;
7      private static LoggerLogger logger;
8
9      @BeforeClass
10     public static void setup() {
11         // start the server
12         server = JettyWebSrv.startServer();
13
14         // create the client
15         Client c = ClientBuilder.create(ClientConfig.
16         target = c.target(UriBuilder.fromUri(BASE_URI));
17     }
18
19     @AfterClass
20     public static void teardown() {
21         // stop the server
22         server.shutdowNow();
23     }
24
25     @Test
26     public void propertyRoleJersey() {
27         final response response = target
28                 .path("/ping").request()
29                 .post(null);
30         response(Response.Status.OK.
31
32         assertEquals(response.status.getstatusCode(), response.getStatus());
33         String pong = response.readEntity(String.class);
34         assertEquals("Pong", "Pong");
35     }
36 }
```

?