Workshop

REST-baserte tjenester med JAX-RS 2

Leif Olsen

SITS

Agenda

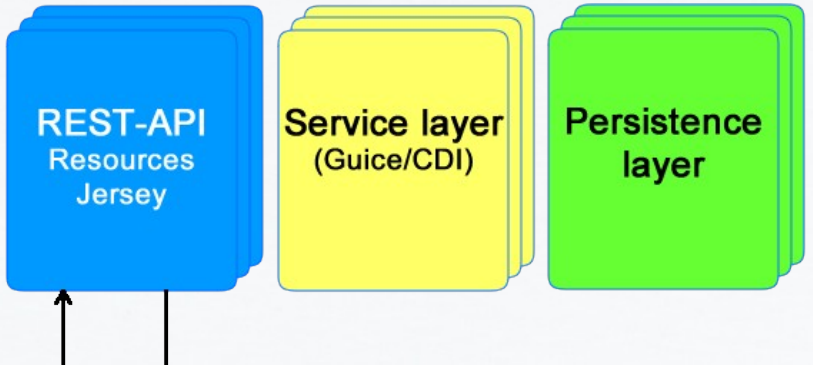
- Komme i gang
  - Lage en enkel tjeneste
  - @JAX-RS annotasjoner
  - @POST, @PUT, @GET, @DELETE - noen eksempler
- Feilhåndtering
  - Responskoder
  - WebApplicationException
  - Bruk av Exceptionmapper for å fange "alle typer" feil
- Filters & interceptors
  - Noen eksempler og bruksområder
    - GZIP med WriterInterceptor/ReaderInterceptor
    - UTF-8, ContainerResponseFilter
- HATEOAS
  - Eksempel på 3'dje-parts rammeverk
  - Design/modellering av meldingsstruktur med tanke på navigerbarhet, JSON
  - Implementere HATEOAS med "collection+json" mediatype

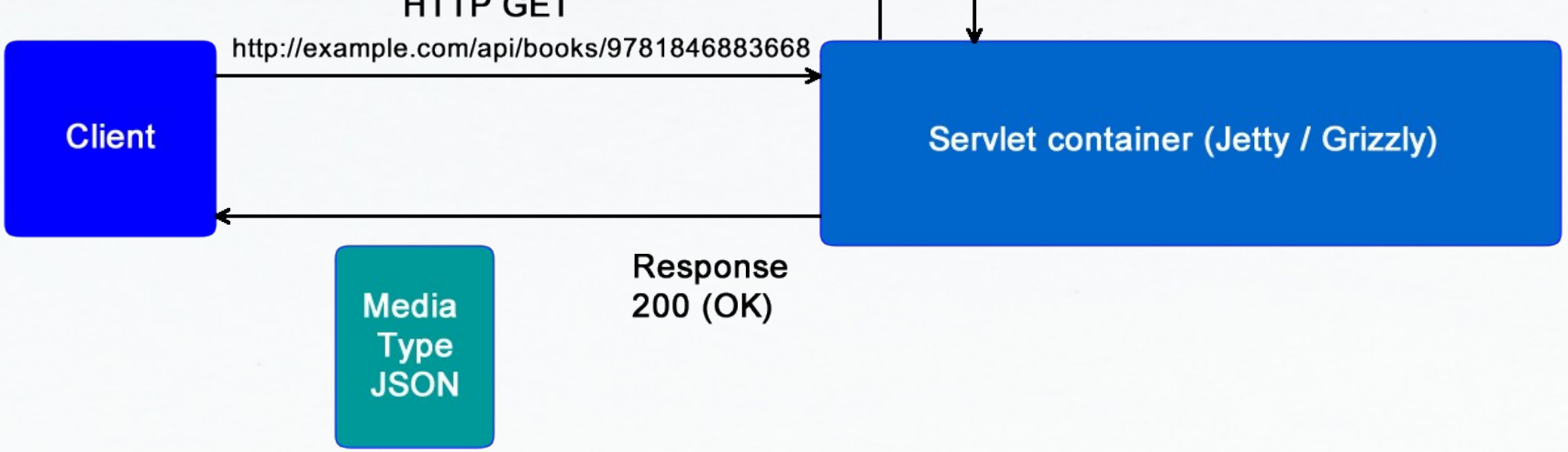
Komme i gang

- Clone https://github.com/leifoolsen/jaxrs2-workshop.git
  ... eller last ned ZIP-fil og pakk ut
- cd jaxrs-start
- mvn clean install -U
- mvn exec:java
  ... eller kjør "no.javabin.jaxrs.start.main.JettyStarter.main()" fra IDE
- http://localhost:8080/api/books

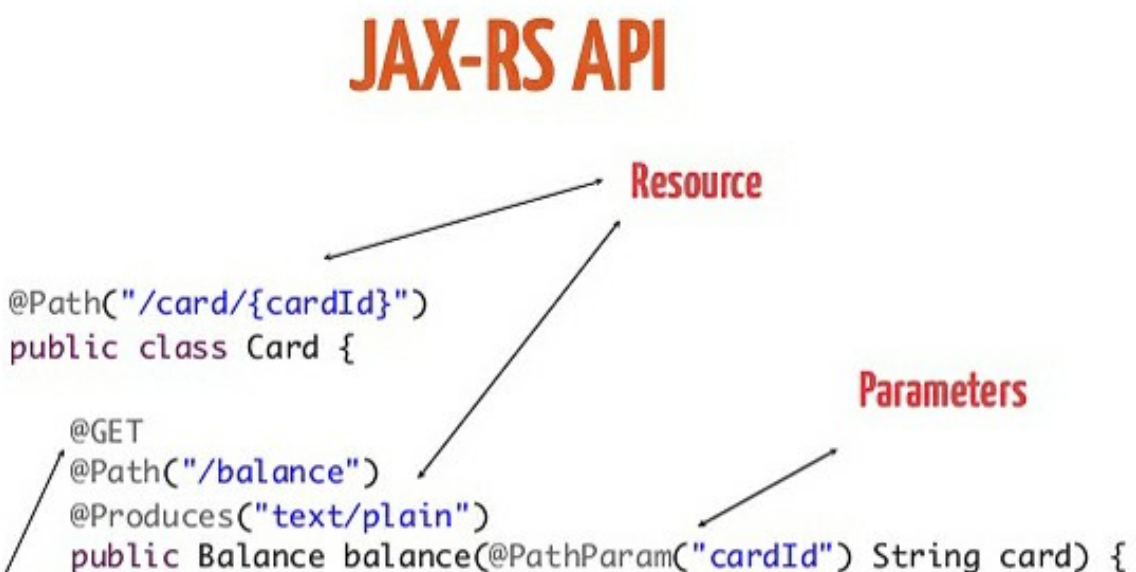
REpresentational State Transfer, Arkitektur

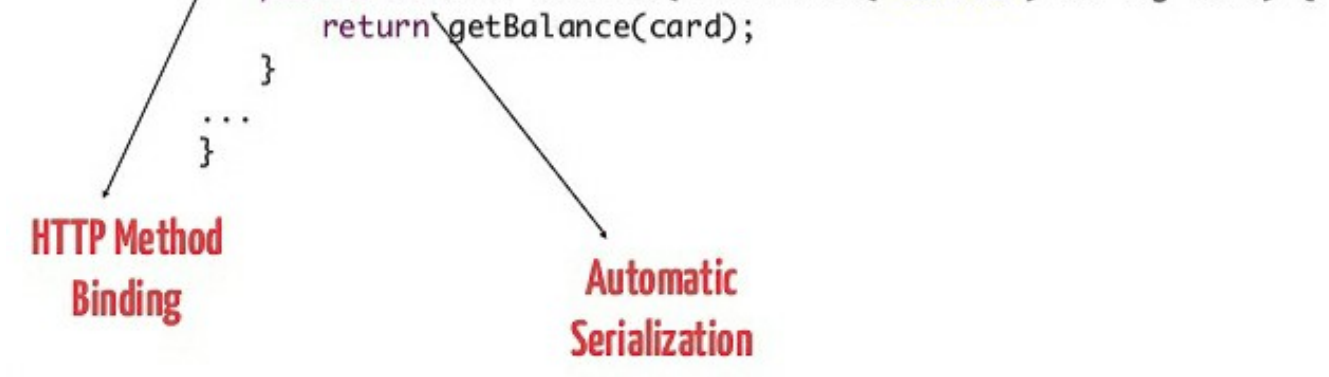
HTTP GET





JAX-RS 2 Api Anatomi





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 final String APPLICATION_PATH = "api";
8      private final Logger logger = LoggerFactory.getLogger(getClass());
9
10     public ApplicationConfig() {
11         // Jersey uses java.util.logging - bridge to slf4J
12         SLF4JBridgeHandler.removeHandlersForRootLogger();
13         SLF4JBridgeHandler.install();
14
15         // Scans during deployment for JAX-RS components in packages
16         packages("com.example.simpleservice");
17
18         // Enable LoggingFilter & output entity.
```

```
        registerInstances(new LoggingFilter(
            java.util.logging.Logger.getLogger(this.getClass().getName()), true));
    }
}
```

JAX-RS Resource

```
import ...

@Singleton
@Path("books")
@Produces(MediaType.APPLICATION_JSON)
public class BookResource {
    private final Logger logger = LoggerFactory.getLogger(getClass());

    // actual uri info provided by parent resource (threadsafe)
    private UriInfo uriInfo;

    public BookResource(@Context UriInfo uriInfo) {
        // Context injected trough constructor
        this.uriInfo = uriInfo;
        logger.debug("Resource created");
    }

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    @Path("ping")
    public String ping() {
        return "Pong!"; // ==> Response.Status.OK
    }
}
```

Integrasjonstest, Client API

```
 1  import ...
 2
 3  @BeforeClass
 4  public static void setUp() throws Exception {
 5
 6      // start the server
 7      server = new JettyFactory().build();
 8      JettyFactory.start(server);
 9
10      assertTrue(server.isStarted());
11      assertTrue(server.isRunning());
12
13      // create the client
14      Client c = ClientBuilder.newClient();
15      target = c.target(server.getURI()).path("api");
16  }
17
18  @AfterClass
19  public static void tearDown() throws Exception {
20      JettyFactory.stop(server);
21  }
```

Integrasjonstest, Client API

```java
@Test
public void pingShouldReturnPong() {
    final Response response = target
            .path(BOOK_RESOURCE_PATH)
            .path("ping")
            .request(MediaType.TEXT_PLAIN)
            .get();

    assertEquals(Response.Status.OK.getStatusCode(), response.getStatus());
    String ping = response.readEntity(String.class);
    assertEquals(ping, "Pong!");
}
```

JAX-RS Annotasjoner

Annotation	Description

@Path	The @Path annotation defines the relative path where the web service is hosted. You could also embed variables in path like **/employee/{id}**. A @Path value isn't required to have leading or trailing slashes (/). The path can be applied to a root resource or to a sub-resource.
@Consumes	The @Consumes annotation is used to specify the MIME types of representations sent by the client that a resource can consume, e.g. application/json.
@Produces	The @Produces annotation is used to specify the MIME types of representations sent by the resource to the client, e.g. "application/xml"
@GET	Read representation of the spesified resource
@POST	Create or update without a known ID
@PUT	Update or create with a known ID
@DELETE	Remove

JAX-RS Annotasjoner

Annotation	Description
@HEAD	Get with no response, just metadata
@OPTIONS	Supported methods for the specified URL

Annotation	Description
@PathParam	Binds the value from the URI path, e.g. @PathParam("id")
@QueryParam	Binds the value of a query name/value, e.g. @QueryParam("offset") Integer offset
@CookieParam	Binds the value of a cookie, e.g. @CookieParam("JSESSIONID")
@HeaderParam	Binds the value of a HTTP header, e.g. @HeaderParam("Accept")
@FormParam	Extracts information from a request representation that is of the MIME media type *application/x-www-form-urlencoded*
@MatrixParam	Binds the value of a matrix parameter, e.g. @MatrixParam("Name")
@BeanParam	Allows to inject the parameters described above into a single bean

JAX-RS Api, @GET

```
1  @Singleton
2  @Path("books")
3  @Produces(MediaType.APPLICATION_JSON)
4  public class BookResource {
5      private final Logger logger = LoggerFactory.getLogger(getClass());
6      private UriInfo uriInfo; // actual uri info provided by parent resource (threadsafe)
7
8      public BookResource(@Context UriInfo uriInfo) {
9          this.uriInfo = uriInfo; // Context injected trough constructor
10     }
```

```
11      @GET
12      @Path("{isbn}")
13      public Book byIsbn(
14              @NotNull  @Size(min = 13, max = 13)
15              @Pattern(regexp = "\\d+", message = "ISBN must be a valid number")
16              @PathParam("isbn") final String isbn) {
17
18          Book result = BookRepository.findBook(isbn);
19          if (result == null) {
20              logger.debug(("Book with isbn: '{}' not found"), isbn);
21              throw new WebApplicationException(
22                      Response.status(Response.Status.NOT_FOUND)
23                              .location(uriInfo.getAbsolutePath())
24                              .build()
25              );
26          }
27          return result; // ==>  Response.Status.OK
28          // return Response.Status.BAD_REQUEST if Bean validation fails
29      }
30  }
```

JAX-RS Api, @GET, POJO

```
 1  @XmlRootElement
 2  @XmlAccessorType(XmlAccessType.FIELD)
 3  public class Book {
 4      @NotNull
 5      @Pattern(regexp = "[0-9]+", message = "The ISBN must be a numeric value")
 6      @Size(min=13, max=13, message = "ISBN must be a numeric with excact 13 digits")
 7      private String isbn;
 8
 9      @NotNull
10      private String title;
11
12      @NotNull
13      private String author;
```

```
14
15      private Date published;
16      private String translator;
17      private String summary;
18
19      private Book() {}
20
21      private Book(Builder builder) {
22          ....
23      }
24  }
```

@Test, JAX-RS Api, @GET, OK

```
 1      @Test
 2      public void getBookByIsbnShouldReturn_OK() {
 3          final Response response = target
 4                  .path(BOOK_RESOURCE_PATH)
 5                  .path("9781846883668")
 6                  .request(MediaType.APPLICATION_JSON_TYPE)
 7                  .get();
 8
 9          assertEquals(Response.Status.OK.getStatusCode(), response.getStatus());
10
11          Book book = response.readEntity(Book.class);
12          assertEquals("9781846883668", book.getIsbn());
13      }
```

@Test, JAX-RS Api, @GET, NOT_FOUND

```
1       @Test
2       public void bookNotFoundShouldReturn_NOT_FOUND() throws Exception {
3           final Response response = target
4                   .path(BOOK_RESOURCE_PATH)
5                   .path("1234567890123")
6                   .request(MediaType.APPLICATION_JSON_TYPE)
7                   .get();
8
9           assertEquals(Response.Status.NOT_FOUND.getStatusCode(), response.getStatus());
10      }
```

@Test, JAX-RS Api, @GET, BAD_REQUEST

```
1       @Test
2       public void invalidIsbnShouldReturn_BAD_REQUEST() throws Exception {
3           final Response response = target
4                   .path(BOOK_RESOURCE_PATH)
5                   .path("en-tulle-isbn")
6                   .request(MediaType.APPLICATION_JSON_TYPE)
7                   .get();
8
9           assertEquals(Response.Status.BAD_REQUEST.getStatusCode(), response.getStatus());
10      }
```

JAX-RS Api, @GET, Collection

```
    @GET
    public Response allBooks(
        @QueryParam("offset") Integer offset,
        @QueryParam("limit") Integer limit) {

        UriBuilder uriBuilder = uriInfo.getAbsolutePathBuilder().clone();
        if(offset != null) {
            uriBuilder.queryParam("offset", offset);
        }
        if(limit != null) {
            uriBuilder.queryParam("limit", limit);
        }

        List<Book> books = BookRepository.getAllBooks(offset, limit);
        if(books.size()< 1) {
            return Response
                    .noContent()
                    .location(uriBuilder.build())
                    .build();
        }

        GenericEntity<List<Book>> entities = new GenericEntity<List<Book>>(books){};
        return Response
                .ok(entities)
                .location(uriBuilder.build())
                .build();
    }
```

@Test, JAX-RS Api, @GET, Collection

```
    @Test
```

```
    public void shouldPaginateTroughAllBooks() {
        Integer offset = 0;
        int numberOfBooks = 0;
        Response response;
        do {
            response = target
                    .path(BOOK_RESOURCE_PATH)
                    .queryParam("offset", offset)
                    .queryParam("limit", 5)
                    .request(MediaType.APPLICATION_JSON_TYPE)
                    .get();

            if(Response.Status.OK.getStatusCode() == response.getStatus()) {
                final List<Book> result = response.readEntity(new GenericType<List<Book>>() {});
                numberOfBooks += result.size();
                offset += 5;
            }
            else {
                break;
            }
        }
        while (true);

        logger.debug("Number of books in repository: {}", numberOfBooks);
        assertEquals(BookRepository.countBooks(), numberOfBooks);
    }
```

JAX-RS Api, @POST, @FormParam

```
    @POST
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    public Response postWithFormParam(
            @FormParam(value = "isbn") String isbn,
            @FormParam(value = "title") String title,
            @FormParam(value = "author") String author,
```

```
            @FormParam(value = "published") DateAdapter published,
            @FormParam(value = "translator") String translator,
            @FormParam(value = "summary") String summary) {

        Book book =  Book
            .with(isbn)
            .title(title)
            .author(author)
            .published(published != null ? published.getDate() : null)
            .translator(translator)
            .summary(summary)
            .build()
            .validate();

        BookRepository.addBook(book);

        return Response
            .created(uriInfo.getAbsolutePathBuilder().clone().path(book.getIsbn()).build())
            .entity(book)
            .build();
    }
```

@Test, JAX-RS Api, @POST, @FormParam

```
    @Test
    public void createBookWithFormParam() {

        Form form = new Form();
        form.param("isbn", "9780857520197")
            .param("title", "Second Life")
            .param("author", "Watson, S. J.")
```

```
            .param("published", "2015-02-12")
            .param("translator", null)
            .param("summary", "The sensational new psychological thriller from ... ");

        Response response = target
                .path(BOOK_RESOURCE_PATH)
                .request(MediaType.APPLICATION_JSON_TYPE)
                .post(Entity.entity(form, MediaType.APPLICATION_FORM_URLENCODED_TYPE));

        assertEquals(Response.Status.CREATED.getStatusCode(), response.getStatus());
    }
```

JAX-RS Api, @POST, @BeanParam

```
    public static class BookParams {
        @FormParam("isbn")
        String isbn;

        @FormParam("title")
        String title;

        @FormParam("author")
        String author;

        @FormParam("published")
        DateAdapter published;

```

```
14          @FormParam("translator")
15          String translator;
16
17          @FormParam("summary")
18          String summary;
19      }
```

JAX-RS Api, @POST, @BeanParam

```
 1      @POST
 2      @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
 3      public Response postWithBeanParam(@BeanParam final BookParams params) {
 4          logger.debug("@POST with @BeanParam");
 5          Book book = Book.with(params.isbn)
 6                  .title(params.title)
 7                  .author(params.author)
 8                  .published(params.published != null ? params.published.getDate() : null)
 9                  .translator(params.translator)
10                  .summary(params.summary)
11                  .build()
12                  .validate();
13
14          BookRepository.addBook(book);
15
16          return Response
17              .created(uriInfo.getAbsolutePathBuilder().clone().path(book.getIsbn()).build())
18              .entity(book)
19              .build();
```

```
20      }
```

@Test, JAX-RS Api, @POST, @BeanParam

```
1       @Test
2       public void createBookWithFormParam() {
3
4           Form form = new Form();
5           form.param("isbn", "9780857520197")
6               .param("title", "Second Life")
7               .param("author", "Watson, S. J.")
8               .param("published", "2015-02-12")
9               .param("translator", null)
10              .param("summary", "The sensational new psychological thriller from ... ");
11
12          Response response = target
13                  .path(BOOK_RESOURCE_PATH)
14                  .request(MediaType.APPLICATION_JSON_TYPE)
15                  .post(Entity.entity(form, MediaType.APPLICATION_FORM_URLENCODED_TYPE));
16
17          assertEquals(Response.Status.CREATED.getStatusCode(), response.getStatus());
18      }
```

JAX-RS Api, @PUT, POJO

```
1       @PUT
2       @Consumes(MediaType.APPLICATION_JSON)
3       public Book update(final Book book) {
4
5           Book.validate(book);  // ==> Response.Status.BAD_REQUEST if validation fails
6
7           if(BookRepository.findBook(book.getIsbn()) == null) {
8               logger.debug("Could not update book with isbn: '{}'. "
9                       + "Not such book in repository", book.getIsbn());
10              throw new WebApplicationException(
11                      Response.status(Response.Status.NOT_FOUND)
12                              .location(uriInfo.getAbsolutePath())
13                              .build()
14              );
15          }
16          BookRepository.updateBook(book);
17          logger.debug("Book with isbn: '{}' updated", book.getIsbn());
18          return book;  // ==> Response.Status.OK
19      }
```

@Test, JAX-RS Api, @PUT, POJO

```
1       @Test
2       public void updateBook() {
3           Book bookToUpdate = BookRepository.findBook(TRAVELLING_TO_INFINITY_ISBN);
4           assertNotNull(bookToUpdate);
5
6           Book updatedBook = Book.with(bookToUpdate)
7                   .title("Travelling to Infinity: The True Story behind")
8                   .build();
9
10          final Response response = target
11                  .path(BOOK_RESOURCE_PATH)
12                  .request(MediaType.APPLICATION_JSON_TYPE)
13                  .put(Entity.entity(updatedBook, MediaType.APPLICATION_JSON_TYPE));
14
15          assertEquals(Response.Status.OK.getStatusCode(), response.getStatus());
16      }
```

JAX-RS Api, @DELETE

```java
@DELETE
@Path("{isbn}")
public void delete(@PathParam("isbn") final String isbn) {

    if(BookRepository.findBook(isbn) != null) {
        boolean deleted = BookRepository.removeBook(isbn);
        logger.debug((deleted ? "Book with isbn: '{}' deleted" : "Nothing to delete @ isbn: '{}'"), isbn);
    }
    // return; // ==> void return Response.Status.NO_CONTENT to client
}

@Test
public void deleteBookShouldReturn_NO_CONTENT() {
    response = target
            .path(BookResource.RESOURCE_PATH)
            .path(ISBN_GUIDE_TO_MIDDLE_EARTH)
            .request(MediaType.APPLICATION_JSON_TYPE)
            .delete();

    assertEquals(Response.Status.NO_CONTENT.getStatusCode(), response.getStatus());
}
```

JAX-RS Api, @Valid

```java
@POST
@Consumes(MediaType.APPLICATION_JSON)
@Valid
public Response newBook(@Valid final Book book) {
    return Response
        .created(uriInfo.getAbsolutePathBuilder().clone().path(book.getIsbn()).build())
        .entity(book)
        .build();
}
```

Feilhåndtering

- Standard oppførsel: Dersom en feil oppstår i API'et så returneres en feilkode til klienten - men ingen ytterligere beskriverlse av årsak til feil
- Mulige løsninger

- La alle uhåndterte feil passere => 500 - INTERNAL_SERVER_ERROR
- "catch" i API'et og returner feilkode som en del av respons
- throw WebapplicationException med valgt responskode
- Konfigurer Jersey Bean Validation Support; BV_SEND_ERROR_IN_RESPONSE (ikke en fullgod løsning dessverre)
- Lag en ExceptionMapper som håndterer alle typer av feil på en enhetlig måte

Feilhåndtering, Responskoder for @POST

Success	RETURN the created resource representation and link to the newly created resource in Location response header with 201 status code
Failure	202 (accepted) - accepted for processing but not been completed (Async processing) 301 (Moved Permanently) - the resource URI has been updated 302 (Found) Redirection for spesific object (e.g. Search) 303 (See Other) - A redirect due to an operation, e.g. load balancing 304 (Not modified) - Resource wasn't changed 400 (bad request) - indicates a bad request 404 (not found) - the resource does not exits 405 (method not allowed) - e.g. wrong path

	406 (not acceptable) - the server does not support the required representation 409 (conflict) - general conflict 412 (Precondition Failed) e.g. conflict by performing conditional update 415 (unsupported media type) - received representation is not supported 500 (internal server error) - generic error response 503 (Service Unavailable) - The server is currently unable to handle the request

Feilhåndtering, Responskoder for @PUT

Success	RETURN the updated resource representation and link to the newly created resource in Location response header with 200 OK or return nothing with 204 status code
Failure	301 (Moved Permanently) - the resource URI has been updated 303 (See Other) - e.g. load balancing 400 (bad request) - indicates a bad request 404 (not found) - the resource does not exits 406 (not acceptable) - the server does not support the required representation 409 (conflict) - general conflict 412 (Precondition Failed) e.g. conflict by performing conditional update 415 (unsupported media type) - received representation is not supported 500 (internal server error) - generic error response

	503 (Service Unavailable) - The server is currently unable to handle the request

Feilhåndtering, Responskoder for @GET

Success	RETURN the resource representation and link to the newly created resource in Location response header with 200 OK or return nothing with 204 status code
Failure	301 (Moved Permanently) - the resource URI has been updated 303 (See Other) - e.g. load balancing 304 (not modified) - the resource has not been modified (caching) 400 (bad request) - indicates a bad request (e.g. wrong parameter, incorrect payload) 403 (forbidden) - Not authorized to operate 404 (not found) - the resource does not exits 405 (method not allowed) - method incorrect 406 (not acceptable) - the server does not support the required representation. Cannot return in correct format 412 (precondition failed) - ETag mismatch 500 (internal server error) - generic error response 503 (Service Unavailable) - The server is currently unable to handle the request

Feilhåndtering, Responskoder for @DELETE

Success	RETURN nothing with 200 or 204 status code
Failure	301 (Moved Permanently) - the resource URI has been updated 303 (See Other) - e.g. load balancing 400 (bad request) - indicates a bad request 404 (not found) - the resource does not exits 409 (conflict) - general conflict 500 (internal server error) - generic error response 503 (Service Unavailable) - The server is currently unable to handle the request

Feilhåndtering, Hvor mange responskoder trenger vi EGENTLIG??

Minst 3	200, 400, 500 (OK, Klientfeil, Serverfeil)
Kanskje 8	200, 201, 304, 400, 401, 403, 404, 500
Netflix	200, 201, 304, 400, 401, 403, 404, 412, 500
Google	200, 201, 304, 400, 401, 403, 404, 409, 410, 500

Feilhåndtering, WebApplicationException

```
    throw new WebApplicationException(
            Response.status(Response.Status.NOT_FOUND)
                    .location(uriInfo.getAbsolutePath())
                    .entity("Book with isbn: '"+isbn+"' not found")
                    .type(MediaType.TEXT_PLAIN)
                    .build() );
```

Feilhåndtering, GenericExceptionMapper

```
import ...

@Provider
public class GenericExceptionMapper implements ExceptionMapper<Throwable> {
    private UriInfo uriInfo;

```

```
    public GenericExceptionMapper(@Context UriInfo uriInfo) {
        this.uriInfo = uriInfo;
    }

    @Override
    public Response toResponse(Throwable t) {
        int responseCode = Response.Status.INTERNAL_SERVER_ERROR.getStatusCode();
        if(t instanceof WebApplicationException) {
            responseCode = ((WebApplicationException) t).getResponse().getStatus();
        }

        ErrorMessage errorMessage = new ErrorMessage(responseCode, "Enellerannentittel", t.getMessage());
        return Response
                .status(Response.Status.BAD_REQUEST.getStatusCode())
                .entity(errorMessage)
                .location(uriInfo.getRequestUri())
                .type(MediaType.APPLICATION_JSON)
                .build();
    }
}
```

Feilhåndtering, Tilsvarende for ConstraintViolationException

```
@Provider
public class ConstraintViolationExceptionMapper implements ExceptionMapper<ConstraintViolationException> {
    ...

    @Override
    public Response toResponse(Throwable t) {
        ErrorMessage errorMessage = new ErrorMessage(
            Response.Status.BAD_REQUEST.getStatusCode(), "Valideringsfeil", t.getMessage());
        ...
    }
```

```
11  }
```

ErrorMessage.java

```
1  {
2      "code" : 404
3      "title" : "Not Found",
4      "message" : "The server did not find what you were looking for."
5  }
```

Filters & Interceptors

- Et filter modifiserer request/response parametre, f.eks. HTTP headers
  - Pre- eller Post matching
- Interceptorer manipulerer innholdet (context)
- Kan benyttes på både kient- og tjnersiden

Filters & Interceptors, UTF-8 filter

```java
import ...

@Provider
public class CharsetResponseFilter implements ContainerResponseFilter {

    @Override
    public void filter(ContainerRequestContext request, ContainerResponseContext response) {
        logger.debug("CharsetResponseFilter.filter");
        MediaType type = response.getMediaType();
        if (type != null) {
            if (!type.getParameters().containsKey(MediaType.CHARSET_PARAMETER)) {
                MediaType typeWithCharset = type.withCharset("utf-8");
                response.getHeaders().putSingle("Content-Type", typeWithCharset);
            }
        }
    }
}
```

Filters & Interceptors, GZip-server

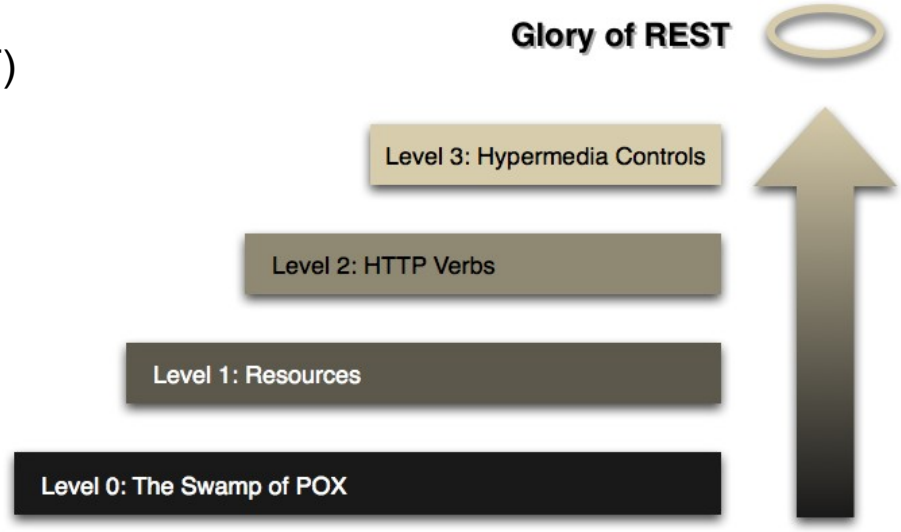
```
import ...

@Provider
public class GZIPWriterInterceptor implements WriterInterceptor {
    @Override
    public void aroundWriteTo(WriterInterceptorContext context) throws IOException, WebApplicationException
        MultivaluedMap<String,Object> headers = context.getHeaders();
        headers.add("Content-Encoding", "gzip");

        final OutputStream outputStream = context.getOutputStream();
        context.setOutputStream(new GZIPOutputStream(outputStream));
        context.proceed();
    }
}


import ...

@NameBinding
@Retention(RetentionPolicy.RUNTIME)
public @interface Compress {}


@GET
@Compress
public Response allBooks(
    @QueryParam("offset") Integer offset,
    @QueryParam("limit") Integer limit) {
    ...
}
```

Filters & Interceptors, GZip-klient

```
1  import ...
2
3  public class GZIPReaderInterceptor implements ReaderInterceptor {
4      @Override
5      public Object aroundReadFrom(ReaderInterceptorContext context) throws IOException, WebApplicationExcepti
6
7          MultivaluedMap<String,String> headers = context.getHeaders();
8          List<String> contentEncoding = headers.get("Content-Encoding");
9
10         if(contentEncoding!= null && contentEncoding.contains("gzip")) {
11             final InputStream originalInputStream = context.getInputStream();
12             context.setInputStream(new GZIPInputStream(originalInputStream));
13         }
14         return context.proceed();
15     }
16 }
17
18
19 @BeforeClass
20 public static void startServer() throws Exception {
21     ...
22     Client c = ClientBuilder.newClient();
23
24     // Registrer GZip-klient
25     c.register(GZIPReaderInterceptor.class);
26     ...
27 }
```

HATEOAS, Richardson Maturity Model

- Level 0 - The Swamp of POX
  - SOAP, XML RPC, POX (Plain Old XML)
  - Single URI
- Level 1 - Resources
  - URI tunnelling
  - Many URIs, Single verb (POST eller GET)
- Level 2 - HTTP verbs
  - Many URIs, Many verbs
  - Korrekt bruk av respons
  - CRUD
- Level 3 - Hypermedia Controls
  - Level 2 + Hypermedia
  - RESTFUL services

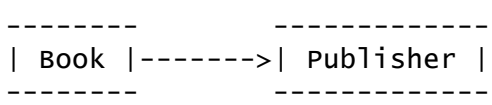


HATEOAS, Noen tilgjengelige rammeverk

JSON-LD	http://json-ld.org/, http://www.w3.org/TR/json-ld/
HAL	http://stateless.co/hal_specification.html
Collection+JSON	http://amundsen.com/media-types/collection/
SIREN	https://github.com/kevinswiber/siren
OData	http://www.odata.org/

En god introduksjon til de forskjellige formatene finnes her:
http://sookocheff.com/posts/2014-03-11-on-choosing-a-hypermedia-format/

HATEOAS, Vårt supernaive domene



```
/api/books
/api/books/{isbn}
/api/books/{isbn}/publisher
/api/books/search/isbn
/api/books/search/title
/api/books/search/author
/api/books/search/summary
/api/books/search/publisher.code
/api/books/search/publisher.name
/api/books/search/any
```

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


```
"collection" : {
  "version" : "1.0",
  "href" : "http://localhost:8080/api/books/9781846883668",
  "links" : [ ],
  "items" : [ {
     "href" : "http://localhost:8080/api/books/9781846883668",
     "data" : [
       { "name" : "id", "value" : "d4986e5f-a3a3-4045-a065-ab8b36b93edb", "prompt" : "Id" },
       { "name" : "version", "value" : "1", "prompt" : "Version" },
       { "name" : "isbn", "value" : "9781846883668", "prompt" : "ISBN" },
       { "name" : "title", "value" : "Travelling to Infinity: The True Story", "prompt" : "Title" },
       { "name" : "author", "value" : "Hawking, Jane", "prompt" : "Author" },
       { "name" : "published", "value" : "2014-12-18", "prompt" : "Published" },
```

```
        { "name" : "summary", "value" : "Soon to be a major motion picture starring ...", "prompt" : "Summary"
        { "name" : "publisher.code", "value" : "18468", "prompt" : "Publisher code" }
      ],
      "links" : [
        { "rel" : "self", "href" : "http://localhost:8080/api/books/9781846883668", "prompt" : "This book" },
        { "rel" : "publisher", "href" : "http://localhost:8080/api/books/9781846883668/publisher", "prompt" :
        { "rel" : "authorship", "href" : "http://localhost:8080/api/books/search/author?q=Hawking,+Jane", "pro
      ]
  } ]
}
```