Sovelto

JAX-RS

REST-kertaus

- REST = *Representational State Transfer*
- REST on erittäin yksinkertainen web-sovellusten ohjelmointitapa: URL palauttaa dokumentin (usein XML, HTML tai JSON); dokumentin merkitys täytyy sopia palvelun ja kutsujan kesken

- URL = substantiivi
- HTTP-pyyntö määrittää toiminnon:
  - GET: Haku
  - POST: Uuden tiedon syöttö
  - PUT: Päivitys
  - DELETE: Poisto

REST-kertaus: HTTP-statuskoodit

- Tyypillisiä statuskoodeja:
  - 200 OK: Onnistunut GET, joka palauttaa sisältöä
  - 201 CREATED: POST tai PUT loi uutta sisältöä; `Location`-otsikon tulisi palauttaa luodun sisällön URL
  - 204 NO_CONTENT: Onnistunut pyyntö ilman sisältöä, esimerkiksi DELETE tai PUT-päivitys
  - 400 BAD_REQUEST: Virheellinen pyynnön formaatti
  - 401 UNAUTHORIZED: Käyttäjä ei kirjautunut
  - 403 FORBIDDEN: Ei oikeuksia
  - 404 NOT_FOUND
  - 405 METHOD_NOT_ALLOWED: HTTP-pyyntö ei tuettu; `Allow`-otsikon tulisi palauttaa sallitut tyypit
  - 409 CONFLICT: Päivityskonflikti, esimerkiksi PUT ja optimistisen lukituksen virhe; vastaus voi sisältää eri versiot datasta

Yleistä

- JAX-RS = *Java API for RESTful Web Services*
- JSR 311 (JAX-RS 1.0, Java EE 6) ja 339 (JAX-RS 2.0, Java EE 7)
- JAX-RS helpottaa REST-tyyppisen palvelun ohjelmointia
  - Nojautuu pitkälti annotaatioiden käyttöön
  - Paketti: `javax.ws.rs`
- Asiakkaan ohjelmointiin tuli standardi-API versiossa 2.0
  - Aiemmin käytettiin joko luokkaa `java.net.HttpURLConnection`, JAX-RS-toteutuksen mahdollisia apuluokkia tai esimerkiksi Apache HttpClient -kirjastoa

Esimerkki: Palvelu

- Palvelun ohjelmointi:

```
@Path("/rsheimaailma")
public class HeiMaailmaResource {

    @GET
    @Produces("text/plain")
    public String tulostaviesti() {
        return "Hei maailma!";
    }
}
```

- (Palveluluokilla käytetään usein `Resource`-päätettä, vaikka tämä ei olekaan mikään vaatimus)

Esimerkki: Asiakas (2.0)

- Asiakas:

```
Client asiakas = ClientBuilder.newClient();
WebTarget heiMaailmaTarget = asiakas.target(
        "http://localhost:8080/rswebsovellus/rsheimaailma");
String vastaus = heiMaailmaTarget.request().get(String.class);
System.out.println(vastaus);
```

Esimerkki: Asiakas (1.0)

- Asiakas:

```
URL url = new URL(
        "http://localhost:8080/rswebsovellus/rsheimaailma");
HttpURLConnection httpYhteys =
        (HttpURLConnection) url.openConnection();
httpYhteys.connect();
int vastauskoodi = httpYhteys.getResponseCode();
if (vastauskoodi == 200) {
    BufferedReader vastauslukija = new BufferedReader(
            new InputStreamReader(httpYhteys.getInputStream()));
    String rivi = null;
    while ((rivi = vastauslukija.readLine())!= null) {
        System.out.print(rivi);
    }
    httpYhteys.disconnect();
} else {
    System.out.println("Yhteyttä ei saatu");
}
```

Esimerkki: Asiakas (JavaScript)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Esimerkkiasiakas</title>
</head>
<body>
    <h1>Haetaan teksti</h1>
    <p id="kpl">paikka</p>
    <script>
        fetch('http://localhost:8080/rswebsovellus/rsheimaailma')
            .then(function(res){
                return res.text(); // res.json() yleensä toimivampi
            })
            .then(function (teksti){
                document.getElementById('kpl').innerText = teksti;
            }) // .catch() tähän
    </script>
</body>
</html>
```

Asentaminen

- Spesifikaatio ei varsinaisesti ota kantaa asentamiseen, mutta käytännössä toteutukset käyttävät tavanomaista WAR-paketointia
  - Edelleen käytännössä pyynnöt vastaanottaa jokin valmiiksi ohjelmoitu servlet, joka ohjaa kutsut edelleen omille luokille
  - Esimerkki oletustoteutuksen käytöstä (EE 7):

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                        http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
    version="3.1">
  <servlet>
    <servlet-name>javax.ws.rs.core.Application</servlet-name>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>javax.ws.rs.core.Application</servlet-name>
    <url-pattern>/rswebsovellus/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Asentaminen: Oma `Application`-toteutus

- Vaihtoehtona edellisen sivun tyyppiselle `web.xml`:n käytölle on tehdä oma `Application`-luokka:

```
@ApplicationPath("rswebsovellus")
public class HeiMaailmaApplication extends Application {

}
```

Annotaatioita

- Valmiiksi tuetut HTTP-pyyntötyypit ovat:
  - `@GET`
  - `@POST`
  - `@DELETE`
  - `@HEAD`
  - `@OPTIONS`
  - `@PUT`
- Omia pyyntötyyppejä voi tukea annotaatiotyypin `@HttpMethod` avulla, esimerkiksi `@HttpMethod("PATCH")`

Suoritusmetodi

- Metodin tulee olla näkyvyydeltään `public`
- Paluuarvo voi olla:
  - `void`: Kutsujalle tulee statuskoodi 204
  - `javax.ws.rs.core.Response`: Mahdollistaa metadatan lisäämisen vastaukseen (esimerkiksi statuksen vaihdon) ja koodausluokan (`MessageBodyWriter`) etsimisen automaattisesti
    - Esimerkki:

```
public Response tulostaviesti() {
    // Haetaan henkilo-olion tiedot
    return Response.status(Status.CREATED).entity(henkilo).build();
}
```

  - Muu tyyppi (vastaus kirjoitetaan itse)
- Arvon `null` palauttaminen näkyy kutsujalle statuskoodina 204

@Path

- Voidaan asettaa sekä luokalle että metodille
  - Jos annotaatio on määritelty molemmille, osoite on niiden yhdistelmä
- Myös säännölliset lausekkeet on tuettu; esimerkki (arvon lukeminen käsitellään myöhemmin):

```
@GET
@Path("/{muutt:[0-9]*}/regex")
@Produces("text/plain")
public String tulostaviesti() {
```

- URL-esimerkki:
  `http://localhost:8080/.../annotaatio/1234/regex`

Kontekstiluokka

- JAX-RS-palvelu pääsee käsiksi ympäristötietoon annotaation `@javax.ws.rs.core.Context` avulla
- Injektoitavat oliot (kenttä, property tai metodiparametri) ovat `Application`, `UriInfo`, `Request`, `HttpHeaders`, `SecurityContext` tai `Providers`

@PathParam

- Vaihtoehdot parametrin tyypiksi:
  - Merkkijono
  - Primitiivityyppi
  - Mikä tahansa luokka, jolla on merkkijonon hyväksyvä konstruktori
  - Mikä tahansa luokka, jolla on staattinen metodi `String valueOf(String)` tai `String fromString(String)`
  - `List<T>`, `Set<T>` tai `SortedSet<T>` ylläolevista

```
@GET
@Path("/polkuparam/{teksti}")
@Produces("text/plain")
public String tulostaviesti(@PathParam("teksti") String s) {
```

- URL:
  `http://palvelin/.../parametrit/polkuparam/abcdef`

@QueryParam

- Data tulee tavanomaisena pyynnön parametrina
- Esimerkki:

```
@GET
@Path("/pyyntoparam")
@Produces("text/plain")
public String tulostaviesti(@QueryParam("nro") int i) {
```

- URL:
  `http://palvelin/.../parametrit/pyyntoparam?nro=123`

@HeaderParam ja @CookieParam

- Myös otsikoita ja cookieiden sisältöjä on mahdollista käsitellä; esimerkki pyynnön otsikon lukemisesta:

```
@GET
@Path("/otsikkoparam")
@Produces("text/plain")
public String tulostaviesti(
        @HeaderParam("User-Agent") String s) {
```

@MatrixParam

- Matriisiparametrit ovat nimi-arvo-pareja URL-osoitteessa
- Esimerkki:

```
@GET
@Path("/matriisiparam")
@Produces("text/plain")
public String tulostaviesti(
        @MatrixParam("etunimet") String en,
        @MatrixParam("sukunimi") String sn) {
```

- URL:
  `.../matriisiparam;etunimet=Anna Brita;sukunimi=Malli`

@FormParam

- Lomake:

```
<form method="post" action="lomakeparam">
Etunimi: <input type="text" name="etunimi">
<br>
Sukunimi: <input type="text" name="sukunimi">
<br>
<input type="submit">
</form>
```

- Arvojen lukeminen:

```
@GET
@Path("/lomakeparam")
@Produces("text/plain")
public String tulostaviesti(
        @FormParam("etunimet") String en,
        @FormParam("sukunimi") String sn) {
```

@DefaultValue

- Oletusarvoja voi määritellä kaikille @*Xxx*Param-annotaatioille @DefaultValue-annotaatiolla
- Esimerkki:

```
@GET
@Path("/pyyntoparam")
@Produces("text/plain")
public String tulostaviesti(
        @QueryParam("nro") @DefaultValue("123") int i) {
```

`@Encoded`

- Oletusarvoisesti JAX-RS-toteutus purkaa URL-osoitteiden HTML-enkoodauksen seuraavien parametrien tapauksessa: `QueryParam`, `PathParam`, `FormParam` tai `MatrixParam`
- Luokka- tai metodikohtaisella annotaatiolla `@Encoded` tämä voidaan estää

@Consumes

- Annotaatiolla voidaan määrittää, mitä MIME-tyyppejä palvelu osaa käsitellä
  - Oletusarvoisesti kaikki tyypit ovat hyväksyttyjä
  - Jos metodia kutsutaan tyypillä, jota ei ole tuettu, palvelin palauttaa virhearvon ”415 Unsupported Media Type”

Datan palautus

@Produces

- Annotaatiolla voidaan määrittää, mitä MIME-tyyppejä palvelu osaa tuottaa
  - Vastaavasti kuin @Consumes
  - Asiakas kertoo pyynnön Header määreen Accept arvolla minkä muotoista dataa se haluaa
- Myös yksittäisen tai useamman tyypin voi määritellä vakioilla, tai MIME tyypeillä
  - @Produces(MediaType.APPLICATION_JSON)
  - @Produces({"text/plain,text/html"})
- Kuten @Consumes, myös @Produces käy sekä luokalle (oletus kaikille metodeille), että yksittäiselle metodille
- Paluuarvon muokkaus taas automaagisesti esimerkiksi JAXB:n avulla

Response ja ResponseBuilder

- Vastauksen voi rakentaa käyttäen apuluokkaa ResponseBuilder
- Erityisen hyödyllinen se on, mikäli täytyy itse toteuttaa tyypin muokkaus - tai halutaan vain palauttaa statuskoodi

```
@GET
@Path("/kaikkiok")
@Consumes("application/xml")
public Response luoOlio(String content) {
      URI createdUri = ...
      create(content);
      return Response.created(createdUri).build();

return vastaus;
}
```

Paluuarvon käsittely

- Esimerkki paluuarvon `javax.ws.rs.core.Response` käytöstä (luokka etsii itse tehdyn `MessageBodyWriter`-rajapinnan toteutuksen ja käyttää sitä sisällön tulostamiseen):

```
@GET
@Path("/pyyntoparam")
@Produces(MediaType.APPLICATION_JSON)
public Response tulostaviesti() {
    List<String> vastauslista = new ArrayList<String>();
    vastauslista.add("Anna");
    vastauslista.add("Malli");
    ResponseBuilder rakentaja = Response.ok(vastauslista);
    Response vastaus = rakentaja.build();
    return vastaus;
}
```

- Oman MessageBodyWriter luokan toteutus menee yli tämän kurssin, mutta siitä on verkossa paljon tietoa saatavilla