02267: Software Development of Web Services

Week 8

Hubert Baumeister
huba@dtu.dk

Department of Applied Mathematics and Computer Science
Technical University of Denmark

Fall 2015

Recap

- BPEL:
  - Doing things in parallel: the flow construct
  - Transactions in BPEL: compensatiion handler
- RESTful Web services
  - Introduction and simple example

Contents

REST = Representational State Transfer

- REST is an architecture style defined by Roy Fielding, one of the authors of HTTP 1.0 and 1.1
- Characteristics
  - Client-server
  - Stateless
    - The server does not hold any application state (e.g. state regarding a session); all the information needs to be provided by the client

    $\rightarrow$ improves scalability
  - Cachebale
  - Uniform Interface
    - use of standard methods, e.g. GET, POST, ...
    - use of standard representation, e.g. MIME types

What are RESTful Web Services?

Concepts

- Resource: Identified by URI: `http://localhost:8080/sr/resources/students/123`
- Representation: Mime types, e.g. test/plain, application/xml, application/json, /
- Uniform Interface: HTTP Verbs: GET, POST, PUT (PATCH), DELETE
- Hyperlinks: Business logic (next steps in the business process) $\rightarrow$ next week

Student Registration

- Classical Web Services focus on services (= operations)
- RESTful Web Services focus on resources

URI Path	Resource Class	HTTP Methods
/institute	InstituteResource	GET, PUT
/institute/address	InstituteResource	GET, PUT
/students	StudentsResource	GET, POST
/students/{id}	StudentResource	GET, PUT, DELETE

- CRUD: Create, Read, Update, Delete

Student Registration: Institute Resource

Institute resource

- When accessing `http://localhost:8080/sr/webresources/intitute` using GET, ”DTU” is returned
- Accessing the same URL using PUT with a string, the name of the institution is changed

```
URI Path                          | Resource Class    | HTTP Methods
----------------------------------------------------------------------
/institute                        | InstituteResource | GET, PUT
```

- Bottom up development:
  1. JAX-RS annotated client
  2. WADL (Web Application Description Language)
  3. Client using Jersey

Java implementation: JAX-RS annotations

```
package ws.dtu;

import javax.ws.rs.GET;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.core.MediaType;

@Path("institute")
public class InstituteResource {

    private static String name = "DTU";

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String getInstituteName() {
        return name;
    }

    @PUT
    @Consumes(MediaType.TEXT_PLAIN)
    public void setInstituteName(String input) {
        name = input;
    }

    @Path("reset") // Resets the name for testing purposes
    @PUT
    public void reset() {
       name = "DTU";
    }
}
```

Accept header in HTTP Req text/plain

Body of HTTP request

RESTful Client using Jersey

Note that the video on the Web page uses an older version of the library
Use the code presented in these slides instead

```
import javax.ws.rs.client.*;
import javax.ws.rs.core.MediaType;
...

public class StudentRegistrationTest {

 Client client = ClientBuilder.newClient();
 WebTarget r =
  client.target("http://localhost:8070/sr/webresources/institute");

 @Test
 public void testGetInstituteName() {
  String result = r.request().get(String.class);
  assertEquals("DTU",result);
 }

 @Test
 public void testPutInstituteName() {
  String expected = "Technical University of Denmark";
  r.request().put(Entity.entity(expected, MediaType.TEXT_PLAIN));
  assertEquals(expected,r.request().get(String.class));
 }

 @Before
 public void resetInstituteName() {
  r.path("reset")
    .request()
    .put(Entity.entity("", MediaType.TEXT_PLAIN));
 }
```

Contents

Student Registration extended

- Classical Web Services focus on services (= operations)
- RESTful Web Services focus on resources
  - Two types of resources: students (the list of all students) and student (one student)
  - Basic operations like register student, search student, access student data, update student data and delete student data is mapped to the appropriate HTTP method

URI Path	Resource Class	HTTP Methods
/institute	InstituteResource	GET, PUT
/institute/address	InstituteResource	GET, PUT
/students	StudentsResource	GET, POST
/students/{id}	StudentResource	GET, PUT, DELETE

- A particular student is accessed by its id, e.g. /students/123

Example: Student Registration Implementation

- A resource path corresponds to one implementation class
- StudentsResource (note the plural)

```
import javax.ws.rs.*;

@Path("students")
public class StudentsResource {
    @POST
    @Consumes(MediaType.APPLICATION_XML)
    @Produces(MediaType.TEXT_PLAIN)
    public String registerStudent(Student st) { ... }

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String findStudent(@QueryParam("name") String name) { ... }
}
```

students?name=Jim

- @QueryParam allows access to a query parameter, e.g. students?name=Nielsen
- Conversion from and to XML if XmlRootElement() is used

```
@javax.xml.bind.annotation.XmlRootElement()
public class Student { ... }
```

Example: Student Registration Implementation

Class StudentResource

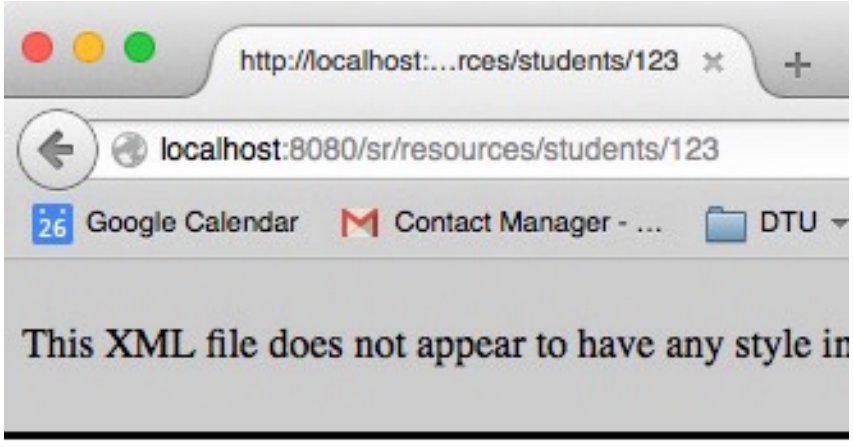
```
@Path("students/{id}")
public class StudentResource {
    @GET
    @Produces(MediaType.APPLICATION_XML)
    public Student getStudent(@PathParam("id") String id){ .. }

    @PUT
    @Produces(MediaType.APPLICATION_XML)
    public Student setStudent(@PathParam("id") String id, Student std)

    @DELETE
    @Produces(MediaType.TEXT_PLAIN)
    public String deleteStudent(@PathParam("id") String id,
                                Student std)
      { .. }
}
```

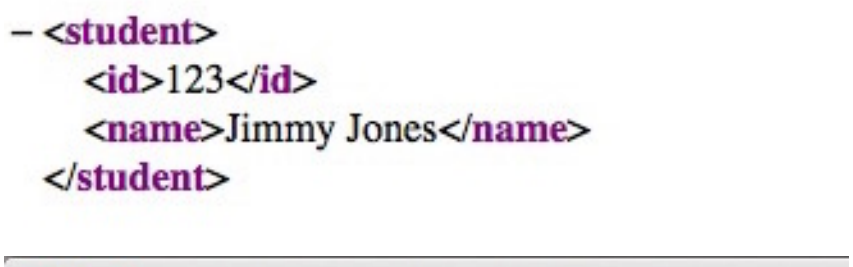
- id in the path is replaced by the actual id of the student
- @PathParam(”id”) allows to map that path component to a paramter in the Java method

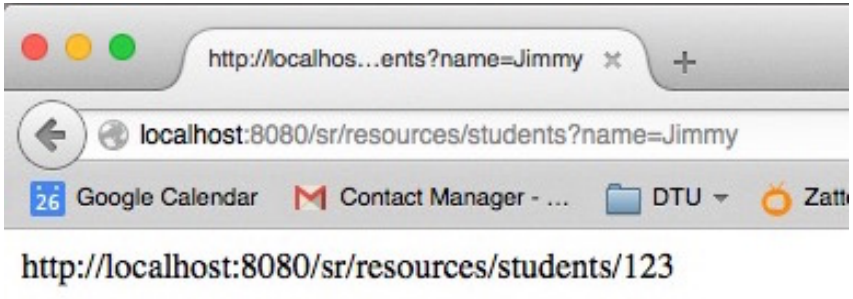
Example: Student Registration Usage

Through the browser

- Advantage: Fast way to test a REST services
- Disadvantage: Only works with GET operation







Example: Student Registration Usage

- CURL (`http://curl.haxx.se/`: sending HTTP requests via the command line

- Search for students

```
curl http://localhost:8080/sr/resources/students\?name=Fleming
```

- Register a new student

```
curl -X POST \
     --data "<student><id>123</id><name>Jimmy Jones</name></studen
     http://localhost:8080/sr/resources/students
```

- Accessing the data of 345

```
curl http://localhost:8080/sr/resources/students/345
```

- Updating the data of student 456

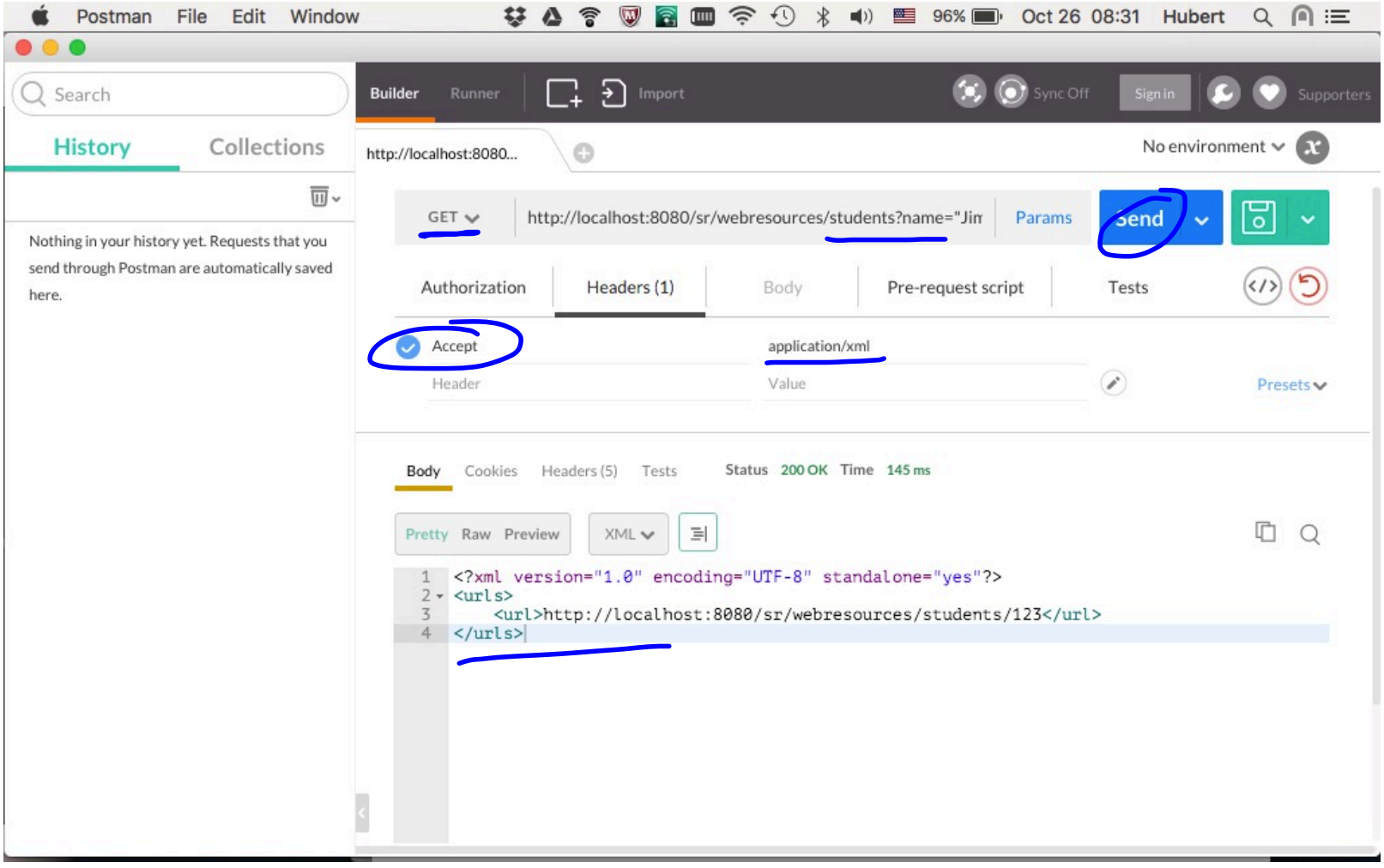
```
curl -X PUT \
     --data "<student><name>Jimmy Jones</name></student>" \
     http://localhost:8080/sr/resources/students/456
```

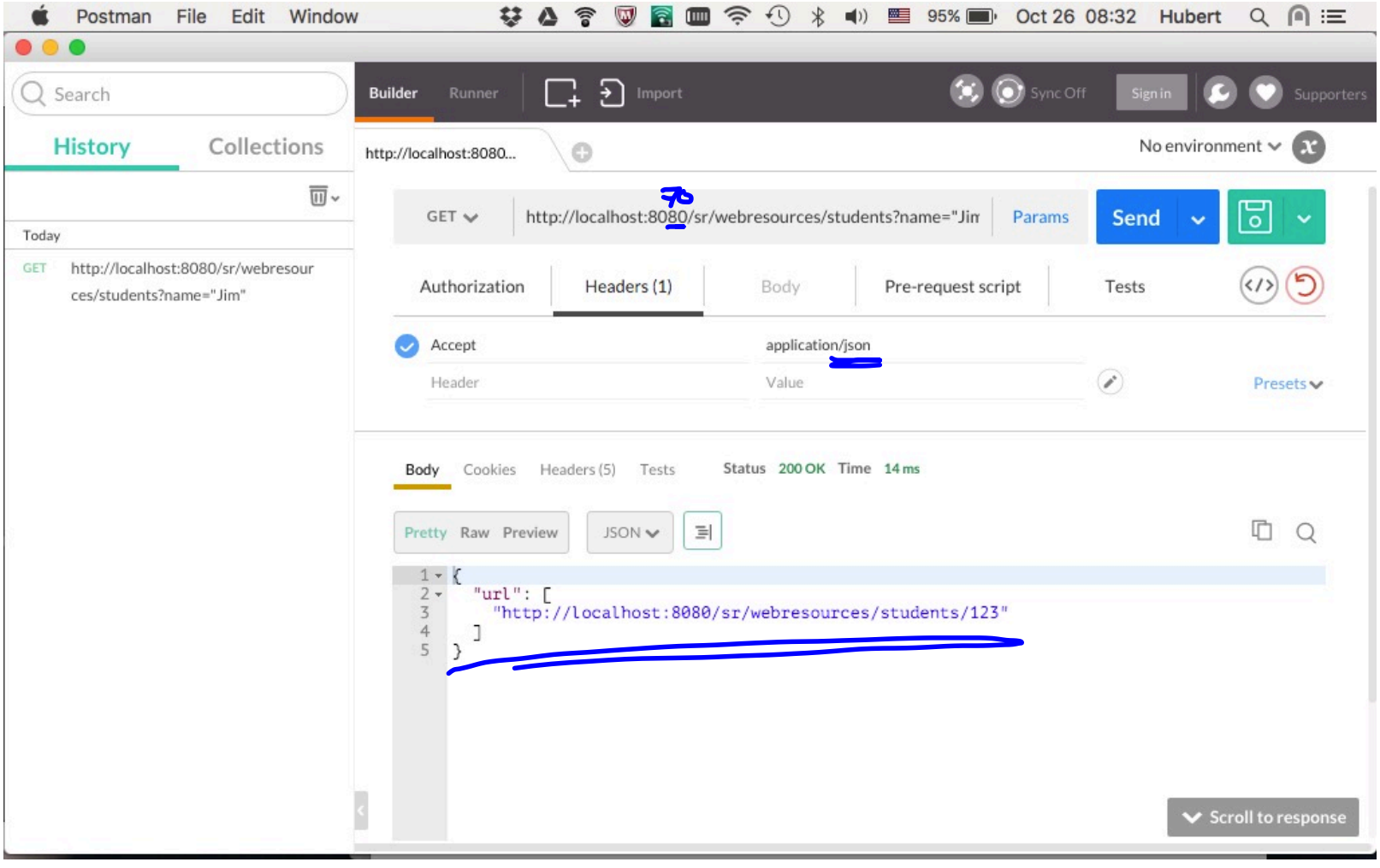
- Deleting an item

```
curl -X DELETE http://localhost:8080/sr/resources/students/456
```

Using Postman plugin in Google Chrome

- Apps::Web Store::Postman


Postman File Edit Window
96% Oct 26 08:31 Hubert
Search
History
Collections
Nothing in your history yet. Requests that you send through Postman are automatically saved here.
Builder
Runner
Import
Sync Off
Sign in
Supporters
http://localhost:8080...
No environment
GET
http://localhost:8080/sr/webresources/students?name="Jir
Params
Send
Authorization
Headers (1)
Body
Pre-request script
Tests
Accept
application/xml
Header
Value
Presets
Body
Cookies
Headers (5)
Tests
Status 200 OK Time 145 ms
Pretty Raw Preview
XML
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2 <urls>
3 <url>http://localhost:8080/sr/webresources/students/123</url>
4 </urls>


Postman File Edit Window
95% Oct 26 08:32 Hubert
Search
History
Collections
Today
GET http://localhost:8080/sr/webresources/students?name="Jim"
Builder
Runner
Import
Sync Off
Sign in
Supporters
http://localhost:8080...
No environment
GET
http://localhost:8080/sr/webresources/students?name="Jim
Params
Send
Authorization
Headers (1)
Body
Pre-request script
Tests
Accept
application/json
Header
Value
Presets
Body
Cookies
Headers (5)
Tests
Status 200 OK Time 14 ms
Pretty Raw Preview
JSON
{
"url": [
"http://localhost:8080/sr/webresources/students/123"
]
}
Scroll to response

Example: Student Registration Usage I

```
Client c = ClientBuilder.newClient();
WebTarget r =
   c.target("http://localhost:8070/sr/webresources")
```

- Search for a student

```
String url = r.path("students")
             .request()
             .queryParam("name","Fleming")
             .get(String.class)
```

- Register a new student

```
Student student = new Student();
student.setName(..);
...
String id = r.path("students")
.request()
.post(Entity.entity(student,MediaType.MediaType.APPLICATION_XML),
      String.class;
```

↳ return data

Example: Student Registration Usage II

- Accessing the data of 345

```
Student s = r.path("students").path("345").get(Student.class);
```

- Updating the data of student 456

```
Student student = new Student();
...
Student s = r.path("students").path("456")
        .request()
        .put(Entity.entity(student,MediaType.APPLICATION_XML),
             Student.class);
```

- Deleting an item

```
String res = r.path("456").request().delete(String.class);
```

Contents

Multiple Representations

registerStudent: data provided as XML or JSON

```
@POST
@Consumes("application/xml")
@Produces("text/plain")
public String registerStudentXML(Student st)  {
   return registerStudent(st);
}

@POST
@Consumes("application/json")
@Produces("text/plain")
public String registerStudentJSON(Student st) {
   return registerStudent(st);
}

public String registerStudent(Student student) {
  student.setId(nextFreeStudentId());
  students.put(student.getId(),student);
  String url = "http://localhost:8080/sr/resources/students/";
  return url + student.getId();
}
```

HTTP Request

JSON

```
POST /sr/resources/students HTTP/1.1
Accept: */*
Content-Type: application/json
User-Agent: Java/1.6.0_37
Host: localhost:8070
Connection: keep-alive
Transfer-Encoding: chunked
21
{"id":"133","name":"Jimmy Jones"}
0
```

XML

```
POST /sr/resources/students HTTP/1.1
Accept: */*
Content-Type: application/xml
User-Agent: Java/1.6.0_37
Host: localhost:8070
Connection: keep-alive
Transfer-Encoding: chunked
6e
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
   <student>
      <id>133</id>
      <name>Jimmy Jones</name>
   </student>
```

decides which method to call

≙ Consumes

Client Code

```
Student student = new Student();
student.setId("133");
student.setName("Jimmy Jones");
String res = target
          .request()
          .post(Entity.entity(student, MediaType.APPLICATION_JSON),
                String.class);
```

Entity.entity(. . . ,MediaType. . .) sets the Content-Type header of the HTTP request

Multiple Representations

- getStudent: data returned as XML or JSON

```
@GET
@Produces("application/xml")
public Student getStudentXML(@PathParam("id") String id) {;
   return getStudent(id);
}

@GET
@Produces("application/json")
public Student getStudentJSON(@PathParam("id") String id) {
   return getStudent(id);
}

public Student getStudent(String id) {
  return students.get(id);
}
```

HTTP Request

XML

```
GET /sr/resources/students/123 HTTP/1.1
Accept: application/xml
User-Agent: Java/1.6.0_37
Host: localhost:8070
Connection: keep-alive
```

```
HTTP/1.1 200 OK
...
Content-Type: application/xml
...

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
   <student>
      <id>123</id>
      <name>Jimmy Jones</name>
   </student>
```

@Produces annotation

JSON

```
GET /sr/resources/students/123 HTTP/1.1
Accept: application/json
User-Agent: Java/1.6.0_37
Host: localhost:8070
Connection: keep-alive
```

Client Code

```
Student student = target.path("123")
                    .request()
                    .accept(MediaType.APPLICATION_XML).
                    .get(Student.class);
```

accept(MediaType...) sets the Accept header of the HTTP request

Contents

JSON: JavaScript Object Notation

Syntax: `http://www.json.org`

- Basically: key value pairs or records or structs
- Simple datatypes: object, string, number, boolean, null, arrays of values

Eg. object

```
{ "name" : "Johan",
  "id" : 123 }
```

Nested objects

```
{ "name" : "Johan",
  "id" : 123,
  "address" : { "city": "Copenhagen", "street": "Frørup Byvej 7" }
}
```

Eg. array of objects

```
[ { "name" : "Johan", "id" : 123 },
  { "name" : "Jakob", "id" : 423 } ]
```

JSON Advantages

- lightweight

```
{ "name" : "Johan", "id" : 123 }
```

- Compare to

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<student>
  <name>Johan</name>
  <id>123</id>
</student>
```

- Easy to use with Javascript, e.g.

```
var student = eval("("+ jsonString + ")")
```

  - → But: Security risk
  - → Better

```
var student = JSON.parse(jsonString)
```

Disadvantage

- JSON is not typed
  - → Is { ”name” : ”Johan”, ”id” : 123, ”foo” : ”bar” } a person object?
  - → Each client has to check for himself if all the necessary fields are there

Next Week

- RESTful Services
  - Error Handling
  - Implementing Business Processes
- SOAP based Web services
  - Web service discovery: UDDI, WSIL

Contents

Exam project introduction

TravelGood

- ”TravelGood is a travel agency allowing to manage itineraries, book them, and possibly cancel them”

→ Detailed description on CampusNet

- Business Process
  - Planning phase: get flight/hotel information, add flight/hotels to itineraries
  - Booking phase: book the itinerary
  - Cancel phase: possibly cancel the itinerary
- Trips consists of flights and hotels
  - possible several flights and hotels
  - flights are always one-way
- TravelGood works together with
  - Airline Reservation Agency: LameDuck
  - Hotel Reservation Agency: NiceView
  - Bank: FastMoney
    - will be provided at `http://fastmoney.imm.dtu.dk:8080`

Tasks (I)

1. Write a section on Web services (SOAP-based and RESTful)
2. Show the coordination protocol between the client and the travel agency as a state machine
3. Implement the Web services from the previous slide
   - Define the appropriate data structures
   - Services of LameDuck and NiceView
   - Two implementations for TravelGood
     a. BPEL process
     b. RESTful services

Tasks (2)

4. Create the JUnit tests testing the major behaviour of the planning, booking and cancelling
5. Web service discovery
   - Create WSIL files describing the services each company offers
6. Compare BPEL implementation with RESTful implementation
7. Advanced Web service technology
   - Discuss WS-Addressing, WS-Reliable Messaging, WS-Security, and WS-Policy

General remarks

- The implementation can be done using Java with Netbeans but also using any other Web service technology, e.g. .Net
  - However BPEL must be used for composite Web services
- All your Web services should be running and have the required JUnit tests — or xUnit for other programming languages, e.g NUnit for .Net
- During the project presentations you may be asked to show the running system by executing the clients
- It might be helpful do use some helper services in BPEL processes to do complicated computations.
  - However these helper services are not allowed to call Web services if the helper service is not written in BPEL

Structure of the report

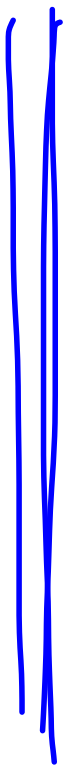
1. Introduction
   - Introduction to Web services
2. Coordination Protocol
3. Web service implementations
   - 3.1 Data structures used
   - 3.2 Airline- and hotel reservation services
   - 3.3 BPEL implementation
   - 3.4 RESTful implementation
4. Web service discovery
5. Comparison between RESTful and SOAP/BPEL Web services
6. Advanced Web service technology
7. Conclusion
8. Who did what

No appendices

Reporting: Web service implementations

- Each Web service should have a short description of what it does and how it is implemented
- BPEL processes
  - graphical representation of the BPEL process in an understandable way (e.g. you can use several diagrams for showing a BPEL process, e.g. one for the overall structure and others for the details)
    - There is a feature in the BPEL designer of Netbeans to hide details
- RESTful services
  - What are resources and why?
  - Mapping to HTTP verbs
  - Representation options and choices

What needs to be delivered?



- Report (report_xx.pdf)
- application_xx.zip
  - Source code projects (e.g. Netbeans projects) of the application so that I can deploy and run the projects
  - Implemented Web services
    - Simple services
    - Complex services (BPEL)
  - Tests

Important: Who did what in the project

- It is important that with each section / subsection it is marked who is responsible for that part of the text.
- There can only be one responsible person for each part of the text.
- In addition, each WSDL, XSD, BPEL, Java, . . . file needs to have an author.
- Who is author of which file should be listed in the section *Who did What* in the report
- Make sure that each member of the group does something of each task (including RESTful WS and BPEL)!
- Note that you all are responsible that the overall project looks good
  - Team work is among the learning objectives
  - Don't think: "I am not responsible for this part, so I don't have to care how this part looks and contributes to the whole"

General tips

- It is likely to get better grades if you all work together and everybody is in fact responsible for every part in the report
- Prioritise high risk task: Don't put off high risk tasks (like doing the BPEL processes) to the end of the project
  - Not so good: XML schema, WSDL file, Simple Web services, BPEL processes, Test
  - Better: work by user stories: book trip successful (Test, XML schema, WSDL file, Simple Web services, BPEL process), book trip with errors (...), cancel trip successful (...), ...
- Try to avoid waiting for each other
  - "I can't do the WSDL files, because I am waiting for the other guy doing the XML schemata first"

Schedule

- Forming of project groups happens now
- Project starts today and finishes midnight on Monday in lecture week 13
- Submission of the two files to the assignment module on CampusNet
- Project presentations will be Tuesday 15 — Friday 18.12: participation is mandatory
- Help
  - Regarding the problem description
  - Technical help
    - Lab sessions before the lecture will continue through the project period and can be used for technical questions and problem clarifications
    - Please also send your Netbeans projects if you write an e-mail
- Detailed task description on CampusNet