REST

Why REST?

- Communication between services or with the (JavaScript) frontend
- Alternative to SOAP with XML

REST fact sheet

- REpresentational State Transfer
- Introduced by Roy Fielding
  - Dissertation in 2000
  - An architectural style for distributed systems
- HTTP is an example of REST

RESTful web services

- Services implemented conform the REST principles
- Mostly based at HTTP

The REST hype

- More public web APIs
  - Google, Amazon, Flickr etc.
- Popularity of lightweight web frameworks
  - Rails / Grails
- People are tired of WSDL
- XML is not always the best format

Everything is a resource

- A list of books
- A product
- A list of search results
- An order

Representation of a resource

- XML
- JSON
- HTML

Representation of a resource

```
<product>
  <productId>1004</productId>
  <name>Product A</name>
</product>

{"product": {"productId":"1004","name":"ProductA"}}

<p class="product">
  <span class="productId">1004</span>
  <span class="name">Product A</span>
</p>
```

HTTP content negotiation

- A client can ask for specific formats
- The accept header
  - Accept: "application/xml"

Dynamic resources

- A resource can be 'static'
  - A record in your database
  - A file
- A resource can be 'dynamic'
  - Calculated results
  - Generated data

RESTful properties

- Uniform Interface
- Addressability
- Connectedness
- Statelessness

Uniform Interface

Method	**Description**
GET	Retrieve a resource representation
PUT	Add or modify a resource with a specified URI
HEAD	GET without body: “Does this resource exists?”
POST	Overloaded: implementation may vary. Might generate a new URI

Possibilities

- These are best practices
- It's possible to implement a GET to work as a POST etc.
- GET should be safe / idempotent

Addressability

- /products
- /product/{id} => /product/10
- /products?color=red
- /search?q=jax-rs

Each resource has a Unique Resource Identifier (URI)

Connectedness

- Navigate from one resource to another
- Clients do not generate URIs
- One of the most important WEB concepts
  - Hyperlinks

Not connected

```
<searchresult>
   <product name="Product 1"/>
   <product name="Product 2"/>
   <product name="Product 3"/>
   <product name="Product 4"/>
</searchresult>
```

How do I get product information?

Connected

Linked to more information

```
<searchresult>
   <product name="Product 1" url="http://myservice/product/1"/ >
   <product name="Product 2" url="http://myservice/product/2"/ >
   <product name="Product 3" url="http://myservice/product/3"/ >
   <product name="Product 4" url="http://myservice/product/4"/ >
</searchresult>
```

RESTful web services in Spring

- Web Services are implemented using controllers
- Familiar Spring MVC programming model

@ResponseBody

- The object returned is converted using a HttpMessageConverter
    - Jaxb2RootElementHttpMessageConverter
    - MappingJacksonHttpMessageConverter
    - StringHttpMessageConverter
    - ...

@ResponseBody

```
@XmlRootElement
public class Book {

@RequestMapping(method = RequestMethod.GET, value = "books",
  headers = "accept=application/xml")
public @ResponseBody BookList listBooksXml() {
  List<Book> books = bookCatalog.listBooks();
  return new BookList(books);
}
```

List of elements

- Don't return a list of elements
- Wrap the list of elements in a wrapper object

Choosing handlers

- How to offer data both as XML and HTML?
  - use the HTTP accept header
  - use a different extension
  - use a request parameter
  - use content negotiation

Choosing handlers

```
@RequestMapping(method = RequestMethod.GET
  value = "books.xml")

@RequestMapping(method = RequestMethod.GET
  value = "books",
  headers = "accept=application/xml")

@RequestMapping(method = RequestMethod.GET
  value = "books",
  params = "contentType=application/xml"
```

POM dependencies

```
<dependency>
  <groupid>org.springframework.boot</groupid>
  <artifactid>spring-boot-starter-web</artifactid>
</dependency>
```

HelloWorld

```
@RestController
public class HelloWorldController {
  @RequestMapping("/hello")
  public String helloWorld() {
    return "Hello world";
  }
}
```

Available on http://localhost:8080/hello

Book and Course

```
public class Book {
  private String name;

public class Course {
  private String location;

  private Book book;
```

Course controller

```
@RestController
public class CourseController {


  @RequestMapping("/course")
  public Course helloWorld() {
    Book book = new Book("Core Spring");
    Course course = new Course("Veenendaal", book);
    return course;
  }
}
```

Available on http://localhost:8080/course

RESTful clients

- Use RESTful web services using a template

POM dependencies

```
<dependency>
  <groupid>org.springframework.boot</groupid>
  <artifactid>spring-boot-starter-web</artifactid>
</dependency>
<dependency>
  <groupid>com.fasterxml.jackson.core</groupid>
  <artifactid>jackson-databind</artifactid>
</dependency>
```

Consume REST endpoint

```
@SpringBootApplication
public class Application {
  public static void main(String[] args) {
    RestTemplate restTemplate = new RestTemplate();
        Course course = restTemplate.getForObject(
                        "http://localhost:8080/course", Course.class);
    System.out.println(course.toString());
  }
}
```

Sending data to a REST endpoint

```
@SpringBootApplication
public class Application {
  public static void main(String[] args) {
    RestTemplate restTemplate = new RestTemplate();

    ...create  changedCourse, newCourse instances

    restTemplate.put(
         "http://localhost:8080/course", changedCourse);
    restTemplate.postForLocation(
         "http://localhost:8080/course", newCourse);
  }
}
```

Documentation

- Could document your API with XML
- Or use alternatives such as Swagger

Swagger

- View the API as a website
- Minimal configuration
- Works automatically for the REST endpoints
- Possibility to add documentation on the endpoint with annotations

POM dependencies

```xml
<dependency>
  <groupid>org.springframework.boot</groupid>
  <artifactid>spring-boot-starter-web</artifactid>
</dependency>

<!-- Swagger -->
<dependency>
  <groupid>io.springfox</groupid>
  <artifactid>springfox-swagger2</artifactid>
  <version>2.6.1</version>
</dependency>
<dependency>
  <groupid>io.springfox</groupid>
  <artifactid>springfox-swagger-ui</artifactid>
  <version>2.6.1</version>
</dependency>
```

Config

```
@Configuration
@EnableSwagger2
public class SwaggerConfig {
    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
                 .select()
                 .apis(RequestHandlerSelectors.any())
                 .paths(PathSelectors.any())
                 .build();
    }
}
```

View the website at http://localhost:8080/swagger-ui.html

Optional documentation

```
@RestController
public class SwaggerCourseController {
  @ApiOperation(notes = "This method allows you to retrieve
      a fixed course", value = "Retrieve course")
      @ApiResponses({
          @ApiResponse(code = 200, message = "Everything went ok.",
                       response = Course.class),
          @ApiResponse(code = 404, message = "Course not found.")
  })
  @RequestMapping("/swaggercourse")
  public Course helloWorld() {
    Book book = new Book("Core Spring");
    Course course = new Course("Veenendaal", book);
    return course;
  }
}
```