

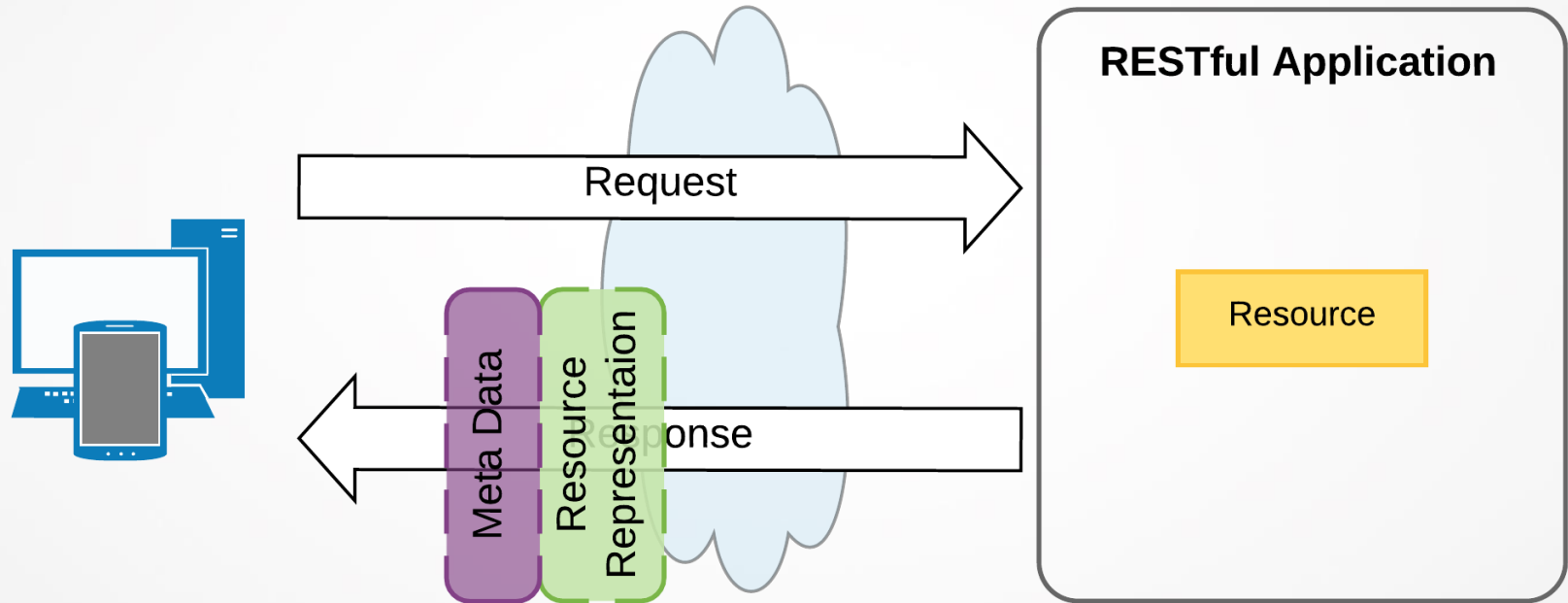
# Komunikacja pomiędzy systemami

Usługi sieciowe oparte o protokół Restful Webservices

# What is REST?

- **RE**presentational **S**tate **T**ransfer
- Architectural style
- Resource-based
- Goals
  - Visibility
  - Reliability
  - Scalability
  - Performance
  - Simplicity (more reliable)
  - Portability
  - Modifiability

# REST - an architectural style



# REST – constraints

- Client-server architecture
- Statelessness
- Cacheability
- Layered system
- Uniform interface
  - Resource identification in requests
  - Resource manipulation through representations
  - Self-descriptive messages
  - Hypermedia as the engine of application state (HATEOAS)
- Code on demand (optional)

# Uniform Interface - Http

- HTTP methods
  - OPTIONS, **GET**, HEAD, TRACE, **PUT**, **DELETE**, **POST**

Method	Safe	Idempotent	Cacheable
GET	Yes	Yes	Yes
PUT	No	Yes	No
DELETE	No	Yes	No
POST	No	No	Yes
PATCH	No	No	No

# Self-descriptive message – HTTP status codes

- **1xx** (Informational)
  - Communicates transfer protocol-level information
- **2xx** (Success)
  - The action requested by the client was received, understood and accepted
  - 200 (OK), 201 (Created), 202 (Accepted), 204 (No Content)
- **3xx** (Redirection)
  - The client must take additional action to complete the request
  - 301 (Moved Permanently), 303 (See Other), 304 (Not Modified), 307 (Temporary Redirect)
- **4xx** (Client errors)
  - The error seems to have been caused by the client
  - 400 (Bad Request), 401 (Unauthorized), 403 (Forbidden), 404 (Not Found), 409 (Conflict)
- **5xx** (Server errors)
  - The server is aware that it has erred or is incapable of performing the request
  - 500 (Internal Server Error), 503 (Service Unavailable)

# Richardson Maturity Model

Glory of REST



Level 3: Hypermedia Controls

Level 2: HTTP Verbs

Level 1: Resources

Level 0: The Swamp of POX



<https://martinfowler.com/articles/richardsonMaturityModel.html>

# REST implementation - JAX-RS 2.1

- Java API for RESTful Web Services
- API specification
- Goals
  - POJO-based
  - HTTP-centric
  - Format independence
  - Container independence
  - Inclusion in Java EE
- Implementations
  - Jersey (Reference)
  - RestEasy
  - Apache CXF



# Uniform interface - JAX-RS

- Resource identification in requests
  - `@ApplicationPath`, `@Path`, `@PathParam`, `@QueryParam`
  - `@GET`, `@POST`, `@PUT`, `@DELETE`, `@HEAD`, `@OPTIONS`, `@PATCH`
- Resource manipulation through representations
  - `@Consumes`
- Self-descriptive messages
  - `@Produces`
  - http response code (2xx, 3xx, 4xx, 5xx)
- Hypermedia as the engine of application state (HATEOAS)
  - `javax.ws.rs.core.Link`

# REST implementation – Spring Boot 5.0

- Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade application
- Goals
  - To avoid complex XML configuration in Spring
  - To develop a production ready applications in an easier way
  - To reduce the development time and run the application independently
  - Offer an easier way of getting started with the application (*SpringInitalizr*)
- Includes Embedded Servlet Container (Apache Tomcat) for stand-alone applications (*.jar*) but may be packaged as a *.war*

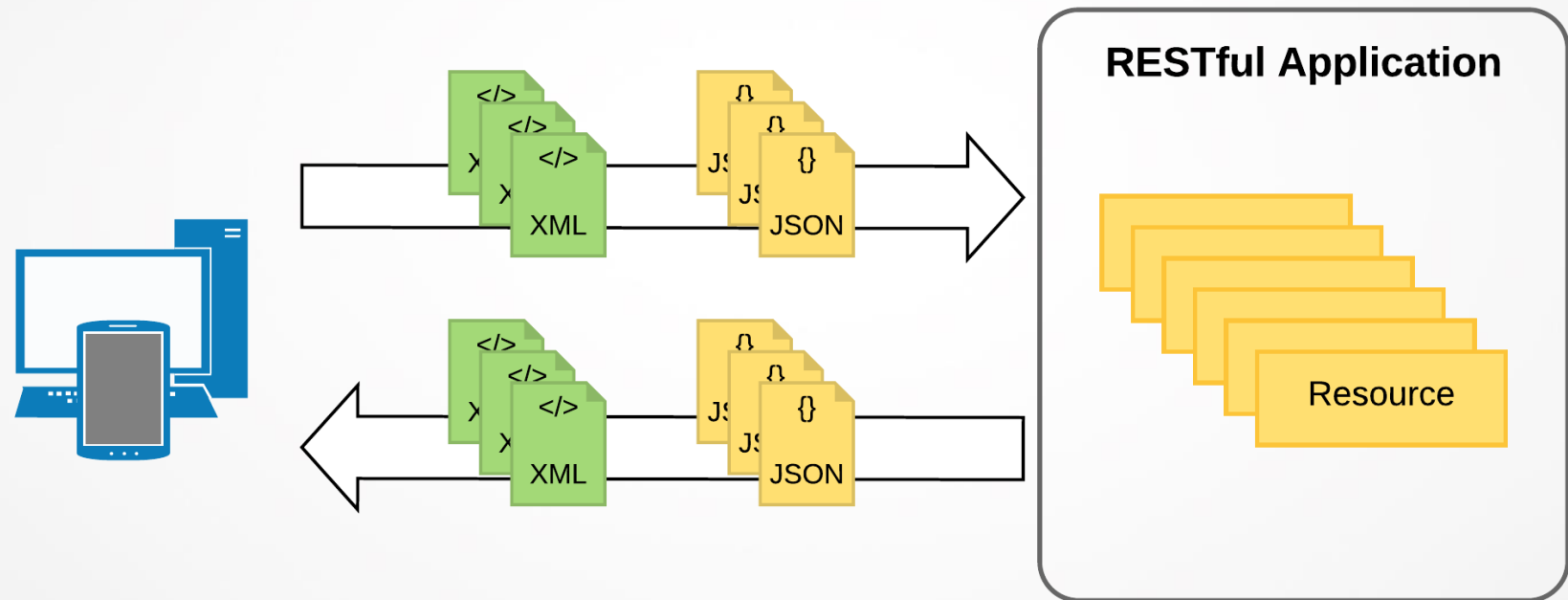
# Uniform interface - Spring Boot 5.0

- Resource identification in requests
  - `@RestController`, `@RequestMapping`, `@GetMapping`, `@PostMapping`, `@PutMapping`, `@DeleteMapping`, `@RequestBody`, `@ResponseBody`, `@PathVariable`, `@RequestParam`
- Resource manipulation through representations
  - `@RequestMapping(consumes = "...")`
- Self-descriptive messages
  - `@RequestMapping(produces = "...")`
  - http response code (2xx, 3xx, 4xx, 5xx)
- Hypermedia as the engine of application state (HATEOAS)
  - `org.springframework.hateoas.Link`

# JAX-RS vs. Spring Boot

Spring Annotation	JAX-RS Annotation
n/a	@ApplicationPath("/api")
@RequestMapping(path = "/surveys")	@Path("/surveys")
@RequestMapping(method = RequestMethod.GET)	@GET
@PostMapping	@POST
@DeleteMapping	@DELETE
@RequestBody	n/a
@ResponseBody	n/a
@PathVariable("id")	@PathParam("id")
@RequestParam("lang")	@QueryParam("lang")
@RequestParam(value="name")	@FormParam("name")
@RequestMapping(consumes = {"text/xml"})	@Consumes("text/xml")
@RequestMapping(produces = {"text/xml"})	@Produces("text/xml")

# Data binding



# Java Architecture for XML Binding (JAXB) 2.2

- Is a Java standard (JSR 31, JSR 222)
- Provides an API and tools that automate the mapping between XML documents and Java objects
  - Unmarshall (XML to Java)
  - Marshall (Java to XML)
- Binding may be done by
  - generating Java classes from XML Schema (.xsd)
  - or annotating POJOs (*javax.xml.bind.annotation*).
- Schema validation
- Implementations
  - Jackson
  - EclipseLink MOXy
  - TopLink (Oracle)

# JAXB - Example

```
@XmlRootElement(name = "university", namespace = "pl.fis.lbd2019.jaxrs")
@XmlType(propOrder = {"name", "foundingYear", "rank", "faculties"})
@XmlAccessorType(XmlAccessType.FIELD)
public class UniversityXML
{
    @XmlElement(required = true)
    private String name;

    @XmlElement(name = "founding-year")
    @XmlJavaTypeAdapter(value = LocalDateAdapter.class)
    private LocalDate foundingYear;

    @XmlElementWrapper(name = "faculties")
    @XmlElement(name = "faculty")
    private List<Faculty> faculties;

    @XmlElement(name = "ranking-position")
    private int rank;

    @XmlAttribute
    private boolean verified;
```

...

# JAXB - Example

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:university
  xmlns:ns2="pl.fis.lbd2019.jaxrs" verified="true">
  <name>FIS University of Technology</name>
  <founding-year>2000-06-13</founding-year>
  <ranking-position>1</ranking-position>
  <faculties>
    <faculty>
      <name>Learning by Doing</name>
      <head-of-department>Prof. Joda</head-of-department>
      <email>joda@fis.pl</email>
      <location>Gliwice</location>
    </faculty>
  </faculties>
</ns2:university>
```



# Java API for JSON Binding (JSON-B) 1.0

- Is a Java standard (JSR 367)
- Provides an API that automate the mapping between JSON documents and Java objects
  - serialize (JSON to Java)
  - deserialize (Java to JSON)
- Binding **may** be done/customized by using annotated POJOs
  - *javax.json.bind.annotation*
  - `@JsonbProperty`, `@JsonbDateFormat`, `@JsonbNumberFormat`,  
`@JsonbTransient`, `@JsonbNillable`, `@JsonbTypeAdapter`,  
`@JsonbTypeDeserializer`, `@JsonbTypeSerializer`
- Naming strategies
- Implementations
  - Eclipse Yasson (reference)
  - Apache Johnzon

# JSON-B - Example

```
@JsonPropertyOrder(value = {"name", "founding-year", "ranking-  
position", "faculties"})  
public class UniversityJSON  
{  
    private String name;  
  
    @JsonProperty("founding-year")  
    @JsonDateFormat("yyyy-MM-dd")  
    private LocalDate foundingYear;  
  
    @JsonProperty("faculties")  
    private List<FacultyJSON> faculties;  
  
    @JsonProperty("ranking-position")  
    private int rank;  
  
    private boolean verified;  
  
    ...  
}
```

# JSON-B - Example

```
{  
  "name": "FIS University of Technology",  
  "founding-year": "2000-06-13",  
  "ranking-position": 1,  
  "faculties": [  
    {  
      "name": "Learning by Doing",  
      "head-of-department": "Prof. Joda",  
      "email": "joda@fis.pl",  
      "location": "Gliwice"  
    }  
  ],  
  "verified": true  
}
```

# Validation – Bean Validation 2.0

- Constraints for JavaBeans
- Validation using API or automatically
  - JAX-RS, SPRING MVC, JPA
- Annotation based
  - Cascade validation `@Valid`
  - `@NotNull`, `@NotBlank`, `@Email`, `@Positive`,...
- Extensible by custom constraints
- Java 8 support
  - Optional
  - Date/Time support

# Exception handling – JAX-RS

- `WebApplicationException` - build in exception handled gracefully by the framework
- Provides extension of `WebApplicationException` by a bunch of convenient exceptions for most http error conditions
  - `BadRequestException`, `NotAuthorizedException`, `NotFoundException`, ...
- `ExceptionHandler` – global handler which take care of exceptions pointed by the developer. Use `@Provider` to register the mapper.

# Exception handling – Spring Boot

- Handling standard MVC exceptions by default exception resolver
  - `BindException`, `MethodArgumentNotValidException`
  - no control over the body of the response
- Custom exception resolver by extending `AbstractHandlerExceptionResolver`
  - no control over the body of the response
- Annotating business exception with `@ResponseStatus`
  - no control over the body of the response
- On controller level by defining a handler method annotated with `@ExceptionHandler`
  - Full control over the response
- Global handler which take care of exceptions pointed by the developer
  - Full control over the response
  - It makes good use of the newer RESTful `ResponseEntity` response
  - `@ControllerAdvice`

# Documentation of RESTful API

- Manually
  - Wiki
  - Word
  - LaTeX
- Generated
  - OpenAPI Specification (Swagger)
  - RESTful API Modeling Language (RAML)
  - RESTful Service Description Language (RSDL)
  - Spring REST Docs
  - SpringRestDoc
  - ApiDocJS

# Documentation - Swagger

```
@Api(value = "Restaurant Controller",
    produces = "Provides functionality to operate on simple restaurant")
@RestController
public class RestaurantController
{
    ...

    @ApiOperation(value = "Get restaurant information and its available dishes",
response = RestaurantResource.class)
    @ApiResponses(value = {
        @ApiResponse(code = 200,
            message = "Successfully retrieved restaurant information",
            response = RestaurantResource.class),
        @ApiResponse(code = 404,
            message = "Restaurant not found", response = ErrorResource.class)
    })
    @GetMapping(path = ResourceLink.UriTemplates.RESTAURANT,
        produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
    public RestaurantResource getRestaurant(
        @ApiParam(value = "restaurant identifier", required = true)
        @Valid @NotNull @PathVariable("restaurantId") Long id)
    {
```



# Documentation - Swagger

```
@Api(value = "Restaurant Controller",
    produces = "Provides functionality to operate on simple restaurant")
@RestController
public class RestaurantController
{
    ...

    @ApiOperation(value = "Get restaurant information and its available dishes",
response = RestaurantResource.class)
    @ApiResponses(value = {
        @ApiResponse(code = 200,
            message = "Successfully retrieved restaurant information",
            response = RestaurantResource.class),
        @ApiResponse(code = 404,
            message = "Restaurant not found", response = ErrorResource.class)
    })
    @GetMapping(path = ResourceLink.UriTemplates.RESTAURANT,
        produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
    public RestaurantResource getRestaurant(
        @ApiParam(value = "restaurant identifier", required = true)
        @Valid @NotNull @PathVariable("restaurantId") Long id)
    {
```

# Documentation - Swagger

**restaurant-controller** Restaurant Controller

GET /api/restaurants/{restaurantId} Get restaurant information and its available dishes

Try it out

Name	Description
<b>restaurantId</b> <small>required</small> integer(\$int64) (path)	restaurant identifier

Responses

Response content type application/json;charset=UTF-8

Code	Description
200	Successfully retrieved restaurant information

# Thank You. Any questions?



Piotr Apollo  
Krzysztof Grel

# Additional information

- [https://docs.wildfly.org/16/JavaEE\\_Tutorial.html](https://docs.wildfly.org/16/JavaEE_Tutorial.html)
- [https://dennis-xlc.gitbooks.io/restful-java-with-jax-rs-2-0-en/cn/part1/chapter7/wrapping\\_up.html](https://dennis-xlc.gitbooks.io/restful-java-with-jax-rs-2-0-en/cn/part1/chapter7/wrapping_up.html)
- <https://spring.io/guides/gs/rest-service/>
- <https://start.spring.io/>
- <https://howtodoinjava.com/jaxb/jaxb-annotations/>
- <http://json-b.net/docs/user-guide.html#json-binding-api>
- <https://docs.oracle.com/javase/tutorial/jaxb/intro/index.html>
- <https://javaee.github.io/tutorial/bean-validation002.html>