Aplikacje internetowe (Web Applications)

Katedra Mikroelektroniki i Technik Informatycznych
Politechniki Łódzkiej
ul. Wólczanska 221/223 budynek B18, 90-924 Łódź

dr inż. Rafał Kotas

Agenda

dr inż. Rafał Kotas, rkotas@dmcs.pl

1. REST API – introduction and constraints
2. HTTP – methods and status codes
3. RESTful API example
4. Spring Boot – REST example, Postman

1

REST API – introduction and constraints

Definition
Resource-based
Representations
Constraints

REST - „REpresentational State Transfer”

Web services provide interoperability between computer systems on the Internet. REST-compliant web services allow the requesting systems to access and manipulate textual representations of web resources by using a uniform and predefined set of stateless operations.

REST - „REpresentational State Transfer”

Roy Fielding defined REST in his 2000 PhD dissertation "Architectural Styles and the Design of Network-based Software Architectures" at UC Irvine. He developed the REST architectural style in parallel with HTTP 1.1 of 1996–1999, based on the existing design of HTTP 1.0 of 1996.

Resource-based

- things vs. actions
- nouns vs. verbs
- versus SOAP (Simple Object Access Protocol)
- identified by URIs
- separate from their represantions

Representations

- The way the resources get manipulated
- The resource state transferred between client and server
- Typically JSON or XML
- Example:
    - Resource: person
    - Service: GET contact information
    - Representation: name, address, phone number – JSON or XML format

JSON Example

JavaScript Object Notation

```
{"employees":[
    { "firstName":"John", "lastName":"Doe" },
    { "firstName":"Anna", "lastName":"Smith" },
    { "firstName":"Peter", "lastName":"Jones" }
]}
```

XML Example

Extensible Markup Language

```
<employees>
    <employee>
        <firstName>John</firstName> <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName> <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```

Uniform Interface

- Defines the interface between client and server
- Simplifies the architecture
- Fundamental to RESTful design
- Main assumptions:
    - HTTP methods (GET, PUT, POST, DELETE)
    - URIs (resource name)
    - HTTP response (status, body)

Stateless

- Server contains no client state
- Each request contains enough information to process the message
- Any session state is held on the client side

dr inż. Rafał Kotas, rkotas@dmcs.pl

Client-Server

- A disconnected system – no direct connections to DB, assets or resources
- Separation of concerns
- Uniform interface – the link between the client-server

dr inż. Rafał Kotas, rkotas@dmcs.pl

Cacheable

- Server responses (representations) are cacheable
  - implicitly
  - explicitly
  - negotiated

Layered system

- Client can not assume direct connection to server
- Software and/or hardware intermediaries between client and server
- Improves scalability

dr inż. Rafał Kotas, rkotas@dmcs.pl

Code on demand

- Server can transfer logic to client
- Client executes logic
- Examples: Java applets, JavaScript technologies
- The only optional constraint

Summary

- Violating any constraint other than „Code on demand” means service is not strictly RESTful; it is only designed in RESTful like fashion (example: OAUTH2)
- Compliance with REST constraints allows: Scalability, Simplicity, Modifiability, Visibility, Portability, Reliability

2

HTTP – methods and status codes

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, and hypermedia information systems.

HTTP is the foundation of data communication for the World Wide Web.

HTTP – request and response

dr inż. Rafał Kotas, rkotas@dmcs.pl

```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

bookId=12345&author=Tan+Ah+Teck
```

Request Line

Request Headers

Request Message Header

A blank line separates header & body

Request Message Body

```
HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1>My Home page</h1>
```

Status Line

Response Headers

Response Message Header

A blank line separates header & body

Response Message Body

dr inż. Rafał Kotas, rkotas@dmcs.pl

GET – The GET method requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect. **(SAFE AND IDEMPOTENT)**

POST – The POST method requests that the server accept the entity enclosed in the request as a new subordinate of the web resource identified by the URI. The data POSTed might be, for example, an annotation for existing resources; a message for a bulletin board, newsgroup, mailing list, or comment thread; a block of data that is the result of submitting a web form to a data-handling process; or an item to add to a database. **(NOT SAFE AND NOT IDEMPOTENT)**

PUT – The PUT method requests that the enclosed entity be stored under the supplied URI. If the URI refers to an already existing resource, it is modified; if the URI does not point to an existing resource, then the server can create the resource with that URI. **(NOT SAFE BUT IDEMPOTENT)**

DELETE – The DELETE method deletes the specified resource. **(NOT SAFE BUT IDEMPOTENT)**

PATCH – The PATCH method applies partial modifications to a resource. **(NOT SAFE AND NOT IDEMPOTENT)**

HEAD – The HEAD method asks for a response identical to that of a GET request, but without the response body. This is useful for retrieving meta-information written in response headers, without having to transport the entire content.

OPTIONS – The OPTIONS method returns the HTTP methods that the server supports for the specified URL. This can be used to check the functionality of a web server by requesting '*' instead of a specific resource.

dr inż. Rafał Kotas, rkotas@dmcs.pl

CONNECT – The CONNECT method converts the request connection to a transparent TCP/IP tunnel, usually to facilitate SSL-encrypted communication (HTTPS) through an unencrypted HTTP proxy.

TRACE – The TRACE method echoes the received request so that a client can see what (if any) changes or additions have been made by intermediate servers.

Informational 1XX

100 - Continue

101 - Switching Protocols

110 - Connection Timed Out

Successful 2XX

200 - OK

201 - Created

202 - Accepted

204 - No content

Redirection 3XX

301 - Moved Permanently

302 - Found

304 - Not Modified

Client Error 4XX

400 - Bad Request

401 - Unauthorized

403 - Forbidden

404 - Not Found

405 - Method Not Allowed

dr inż. Rafał Kotas, rkotas@dmcs.pl

Server Error 5XX

500 - Internal Server Error

503 - Service Unavailable

3

RESTful API example

RESTful API example

dr inż. Rafał Kotas, rkotas@dmcs.pl

URL	GET	PUT
https://restapi.example.com/resources	**List** the URIs and perhaps other details of the collection's members.	**Replace** the entire collection with another collection.
https://restapi.example.com/resources/item_1	**Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	**Replace** the addressed member of the collection, or if it does not exist, create it.

RESTful API example

dr inż. Rafał Kotas, rkotas@dmcs.pl

URL	PATCH	POST	DELETE
https://restapi.example.com/resources	Not used	**Create** a new entry in the collection.	**Delete** the entire collection.
https://restapi.example.com/resources/item_1	**Update** the addressed member of the collection.	Not used	**Delete** the addressed member of the collection.

4

Spring Boot - REST example

Spring Boot – REST example, Postman

dr inż. Rafał Kotas, rkotas@dmcs.pl

GET | localhost:8080/restApi/contacts | Send | Save

Params | Authorization | Headers (1) | Body | Pre-request Script | Tests | Cookies | Code | Comments (0)

Query Params

KEY	VALUE	DESCRIPTION	••• Bulk Edit
Key	Value	Description	

Body | Cookies | Headers (3) | Test Results

Status: 200 OK | Time: 28 ms | Size: 331 B | Download

Pretty | Raw | Preview | JSON


```
[
    {
        "id": 2,
        "firstname": "Jan",
        "lastname": "Kowalski",
        "email": "jkowalski@email.pl",
        "telephone": "123456789"
    },
    {
        "id": 1,
        "firstname": "Rafał",
        "lastname": "Kotas",
        "email": "rkotas@dmcs.pl",
        "telephone": "987654321"
    }
]
```

Spring Boot – REST example

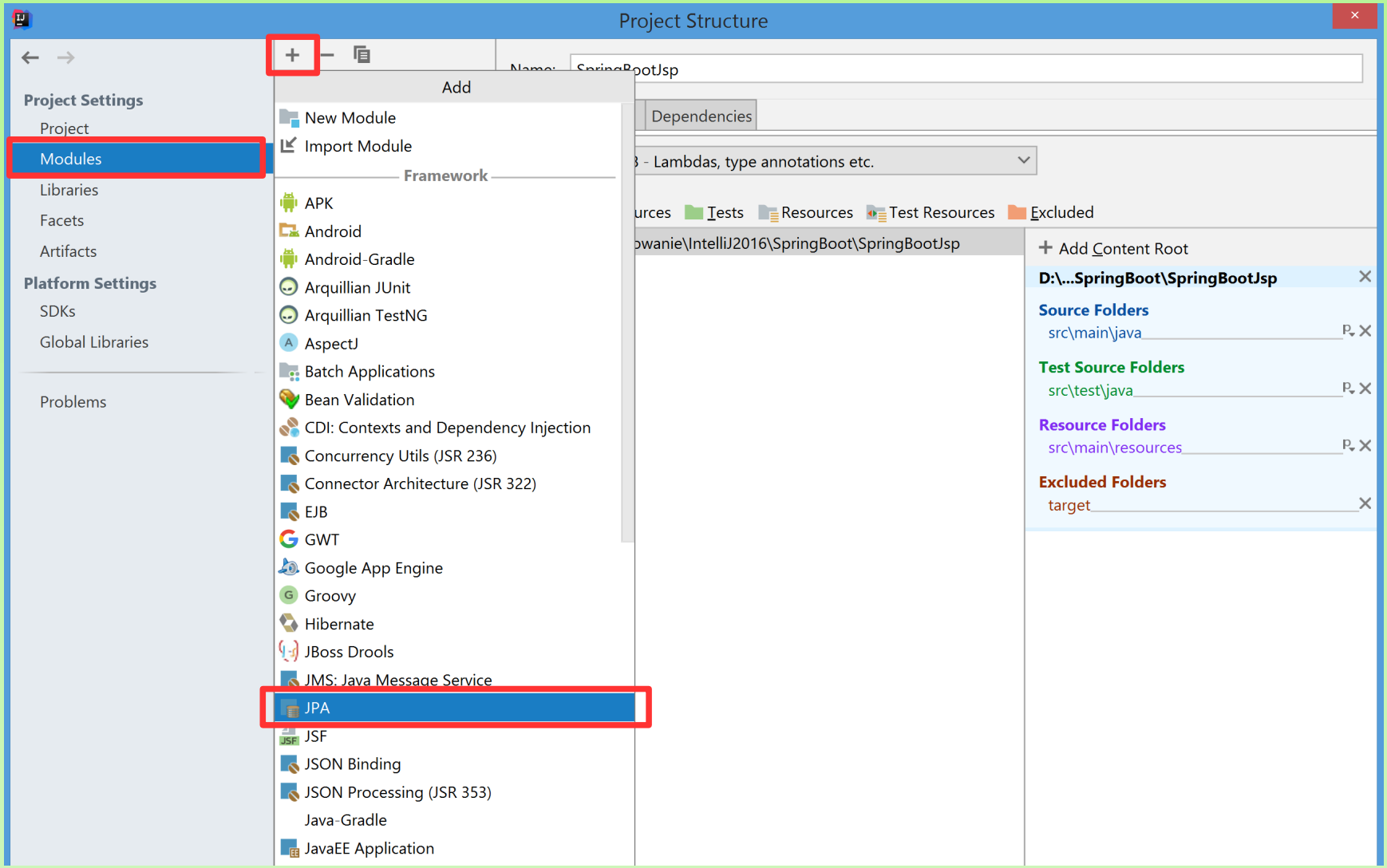
dr inż. Rafał Kotas, rkotas@dmcs.pl

rkotas

```
49        <dependency>
50            <groupId>org.springframework.boot</groupId>
51            <artifactId>spring-boot-starter-data-jpa</artifactId>
52        </dependency>
53
54        <dependency>
55            <groupId>org.postgresql</groupId>
56            <artifactId>postgresql</artifactId>
57            <scope>runtime</scope>
58        </dependency>
59
60        <dependency>
61            <groupId>com.fasterxml.jackson.dataformat</groupId>
62            <artifactId>jackson-dataformat-xml</artifactId>
63            <version>2.9.5</version>
64        </dependency>
65
66    </dependencies>
67
68    <build...>
76
77 </project>
```

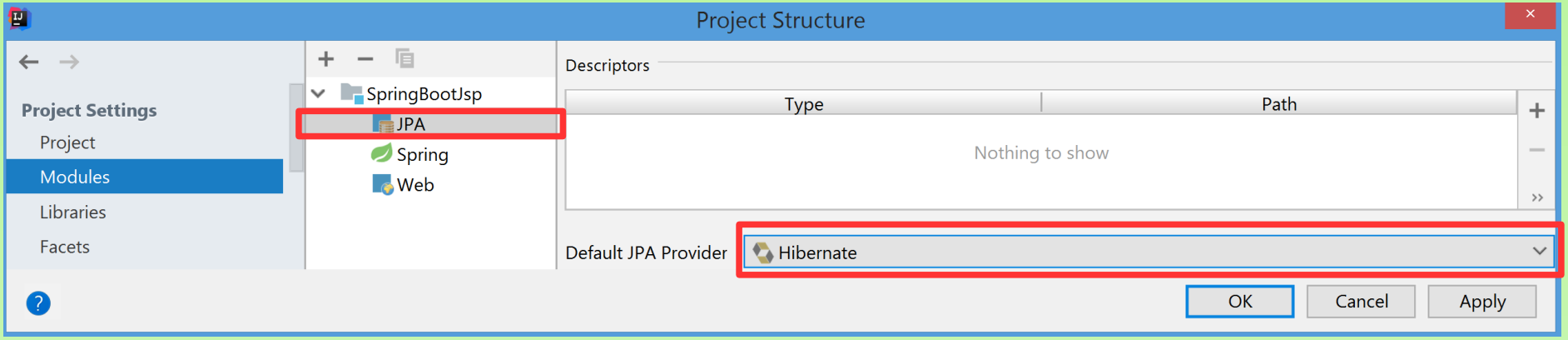
Spring Boot – REST example

dr inż. Rafał Kotas, rkotas@dmcs.pl



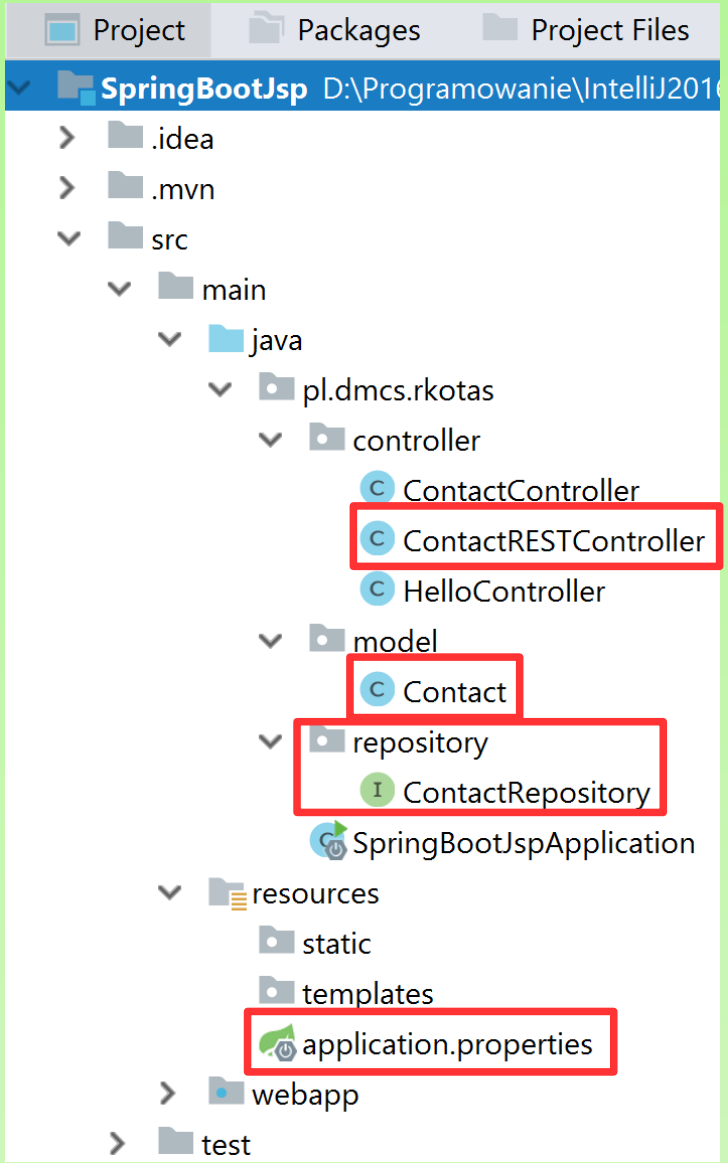
Spring Boot – REST example

dr inż. Rafał Kotas, rkotas@dmcs.pl



Spring Boot – REST example

dr inż. Rafał Kotas, rkotas@dmcs.pl



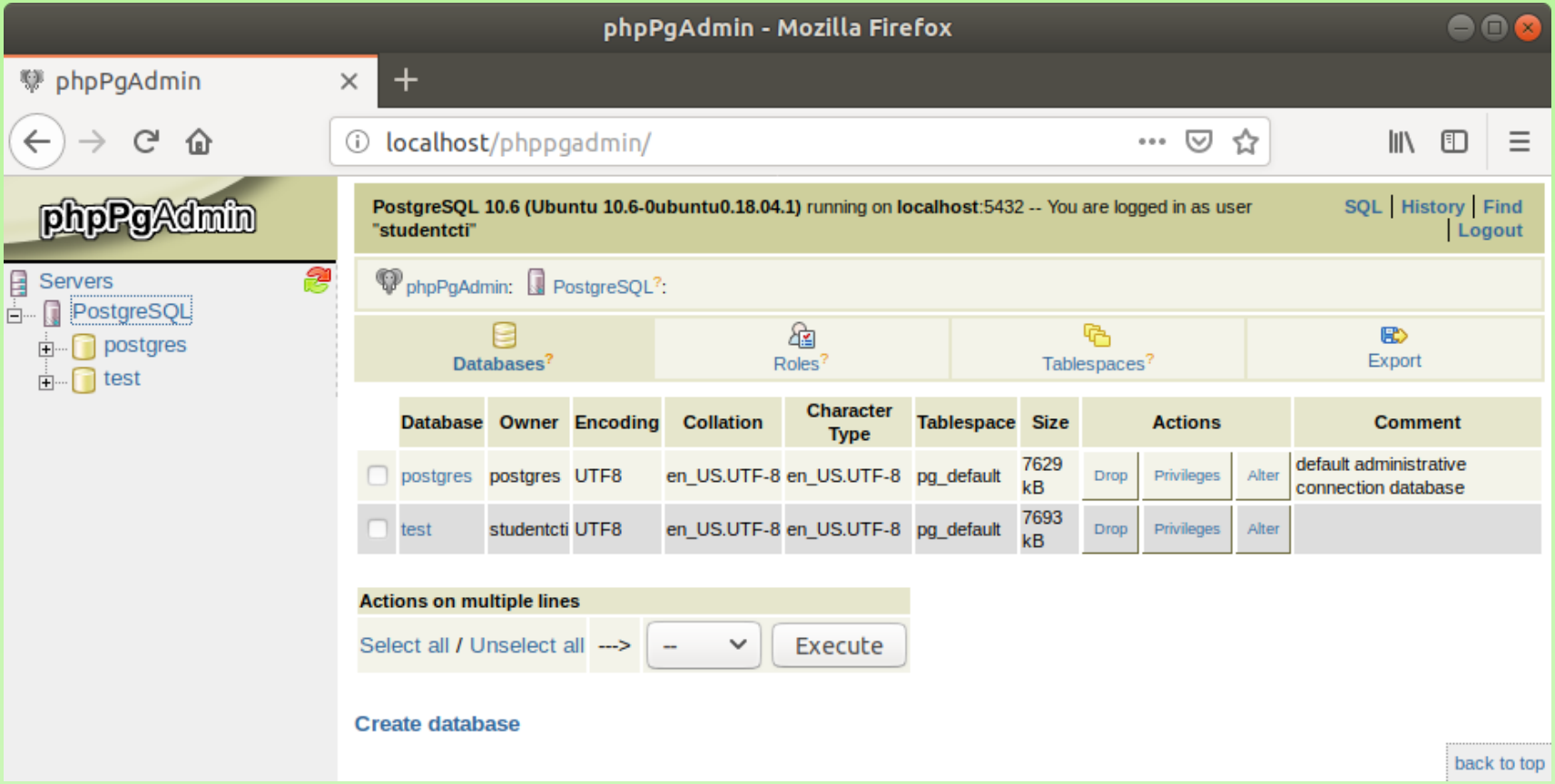
Spring Boot – REST example

dr inż. Rafał Kotas, rkotas@dmcs.pl

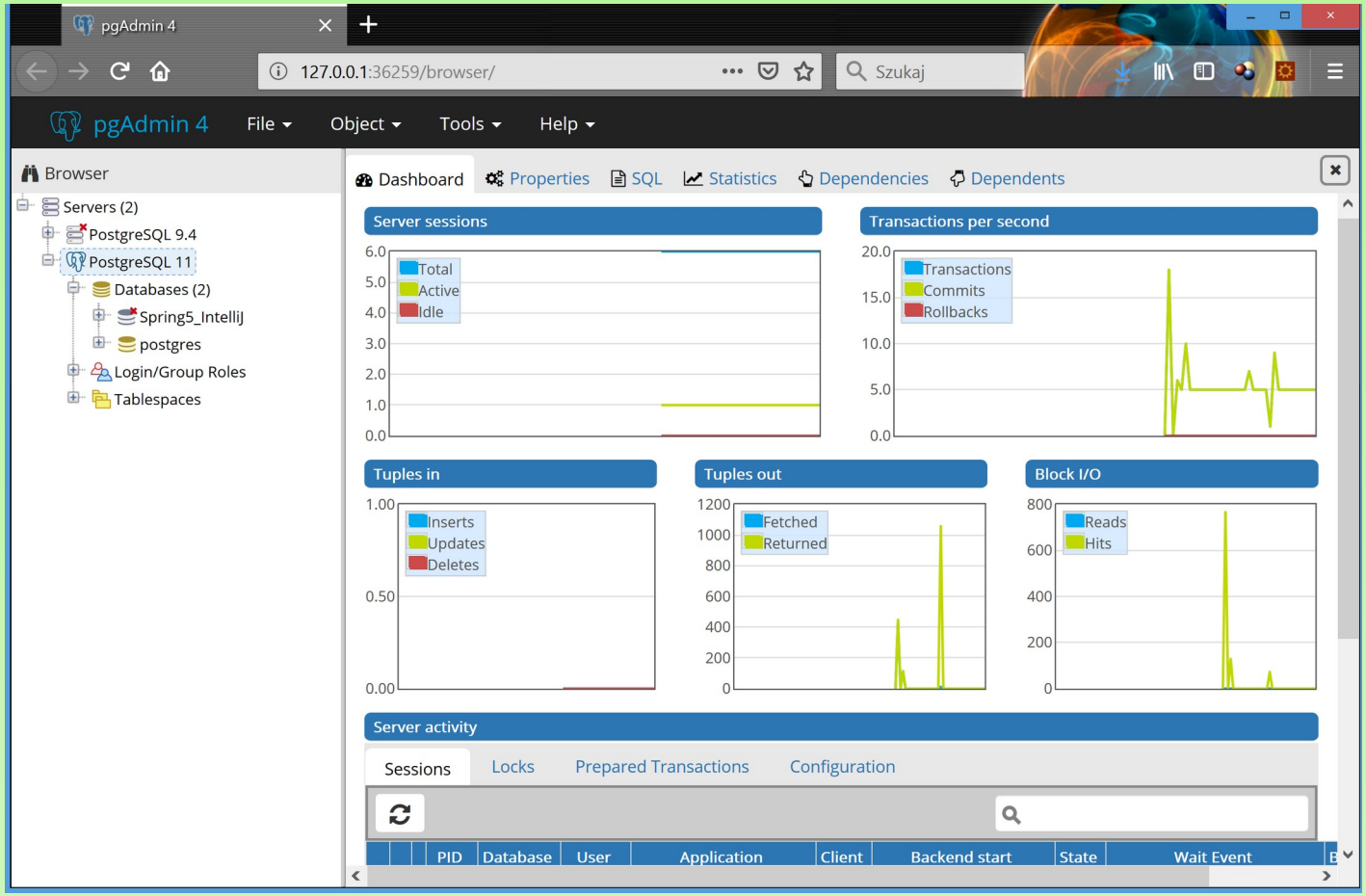
application.properties

```
spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.jsp

# DataSource settings: set here your own configurations for the database
# connection. In this example we have "netgloo_blog" as database name and
# "root" as username and password.
spring.datasource.url = jdbc:postgresql://localhost:5432/test
spring.datasource.username = studentcti
spring.datasource.password = lab301cti

# Keep the connection alive if idle for a long time (needed in production)
spring.datasource.testWhileIdle = true
spring.datasource.validationQuery = SELECT 1

# Show or not log for each sql query
spring.jpa.show-sql = true

# Hibernate ddl auto (create, create-drop, update)
spring.jpa.hibernate.ddl-auto = update

# Naming strategy
spring.jpa.hibernate.naming-strategy = org.hibernate.cfg.ImprovedNamingStrategy

# Use spring.jpa.properties.* for Hibernate native properties (the prefix is
# stripped before adding them to the entity manager)

# The SQL dialect makes Hibernate generate better SQL for the chosen database
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect

spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
```

To manage database on Ubuntu in 301 CTI laboratory run **localhost/phppgadmin** in web browser.
Then create your own database.



To manage database on Windows run **pgAdmin4**.
Then create your own database.



Spring Boot – REST example

dr inż. Rafał Kotas, rkotas@dmcs.pl

Contact.java

```
3   import javax.persistence.Entity;
4   import javax.persistence.GeneratedValue;
5   import javax.persistence.Id;
6
7   @Entity
8   public class Contact {
9
10      @Id
11      @GeneratedValue
12      private long id;
13      private String firstname;
14      private String lastname;
15      private String email;
16      private String telephone;
17
18      public long getId() { return id; }
21
22      public void setId(long id) { this.id = id; }
```

ContactRepository.java

```java
package pl.dmcs.rkotas.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import pl.dmcs.rkotas.model.Contact;

@Repository
public interface ContactRepository extends JpaRepository<Contact,Long> {
    Contact findById(long id);
}
```

Spring Boot – REST example

dr inż. Rafał Kotas, rkotas@dmcs.pl

ContactRESTController.java

```java
package pl.dmcs.rkotas.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import pl.dmcs.rkotas.model.Contact;
import pl.dmcs.rkotas.repository.ContactRepository;
import java.util.List;
import java.util.Map;

@RestController
@RequestMapping("/restApi/contacts")
public class ContactRESTController {

    private ContactRepository contactRepository;

    @Autowired
    public ContactRESTController(ContactRepository contactRepository) { this.contactRepository = contactRepository; }

    @RequestMapping(method = RequestMethod.GET/*, produces = "application/xml"*/)
    //@GetMapping
    public List<Contact> findAllContacts() { return contactRepository.findAll(); }

    @RequestMapping(method = RequestMethod.POST)
    //@PostMapping
    public ResponseEntity<Contact> addContact(@RequestBody Contact contact) {
        contactRepository.save(contact);
        return new ResponseEntity<Contact>(HttpStatus.CREATED);
    }
```

Spring Boot – REST example

dr inż. Rafał Kotas, rkotas@dmcs.pl

ContactRESTController.java

```
36      @RequestMapping(value="/{id}", method = RequestMethod.DELETE)
37      //@DeleteMapping("/{id}")
38      public ResponseEntity<Contact> deleteContact (@PathVariable("id") long id) {
39          Contact contact = contactRepository.findById(id);
40          if (contact == null) {
41              System.out.println("Contact not found!");
42              return new ResponseEntity<Contact>(HttpStatus.NOT_FOUND);
43          }
44
45          contactRepository.deleteById(id);
46          return new ResponseEntity<Contact>(HttpStatus.NO_CONTENT);
47      }
48
49      @RequestMapping(value="/{id}", method = RequestMethod.PUT)
50      //@PutMapping("/{id}")
51      public ResponseEntity<Contact> updateContact(@RequestBody Contact contact, @PathVariable("id") long id) {
52          contact.setId(id);
53          contactRepository.save(contact);
54          return new ResponseEntity<Contact>(HttpStatus.NO_CONTENT);
55      }
```

Spring Boot – REST example

dr inż. Rafał Kotas, rkotas@dmcs.pl

ContactRESTController.java

```java
57  @RequestMapping(value="/{id}", method = RequestMethod.PATCH)
58  //@PatchMapping("/{id}")
59  public ResponseEntity<Contact> updatePartOfContact(@RequestBody Map<String, Object> updates, @PathVariable("id") long id) {
60      Contact contact = contactRepository.findById(id);
61      if (contact == null) {
62          System.out.println("Contact not found!");
63          return new ResponseEntity<Contact>(HttpStatus.NOT_FOUND);
64      }
65      partialUpdate(contact,updates);
66      return new ResponseEntity<Contact>(HttpStatus.NO_CONTENT);
67  }
68
69  private void partialUpdate(Contact contact, Map<String, Object> updates) {
70      if (updates.containsKey("firstname")) {
71          contact.setFirstname((String) updates.get("firstname"));
72      }
73      if (updates.containsKey("lastname")) {
74          contact.setLastname((String) updates.get("lastname"));
75      }
76      if (updates.containsKey("email")) {
77          contact.setEmail((String) updates.get("email"));
78      }
79      if (updates.containsKey("telephone")) {
80          contact.setTelephone((String) updates.get("telephone"));
81      }
82      contactRepository.save(contact);
83  }
```

Spring Boot – REST example, Postman

dr inż. Rafał Kotas, rkotas@dmcs.pl

POST | localhost:8080/restApi/contacts | Send | Save

Params | Authorization | Headers (1) | Body | Pre-request Script | Tests | Cookies | Code | Comments (0)

none | form-data | x-www-form-urlencoded | raw | binary | JSON (application/json) | Beautify

```
1 {
2     "firstname": "Kamil",
3     "lastname": "Nowak",
4     "email": "knowak@email.pl",
5     "telephone": "000222444"
6 }
```

Body | Cookies | Headers (2) | Test Results | Status: 201 Created | Time: 2801 ms | Size: 80 B

Params | Authorization | Headers (1) | Body | Pre-request Script | Tests | Cookies | Code | Comments (0)

	KEY	VALUE	DESCRIPTION
☑	Content-Type	application/json	
	Key	Value	Description

Bulk Edit | Presets

Spring Boot – REST example

dr inż. Rafał Kotas, rkotas@dmcs.pl

PUT localhost:8080/restApi/contacts/4 Send Save

Params Authorization Headers (1) Body Pre-request Script Tests Cookies Code Comments (0)

none form-data x-www-form-urlencoded raw binary JSON (application/json) Beautify

```
1 {
2         "firstname": "Karol",
3         "lastname": "Zielony",
4         "email": "KZielony@email.pl",
5         "telephone": "999888777"
6     }
```

Body Cookies Headers (1) Test Results Status: 204 No Content Time: 46 ms Size: 64 B

Spring Boot – REST example

dr inż. Rafał Kotas, rkotas@dmcs.pl

PATCH | localhost:8080/restApi/contacts/2 | Send | Save

Params | Authorization | Headers (1) | Body | Pre-request Script | Tests | Cookies | Code | Comments (0)

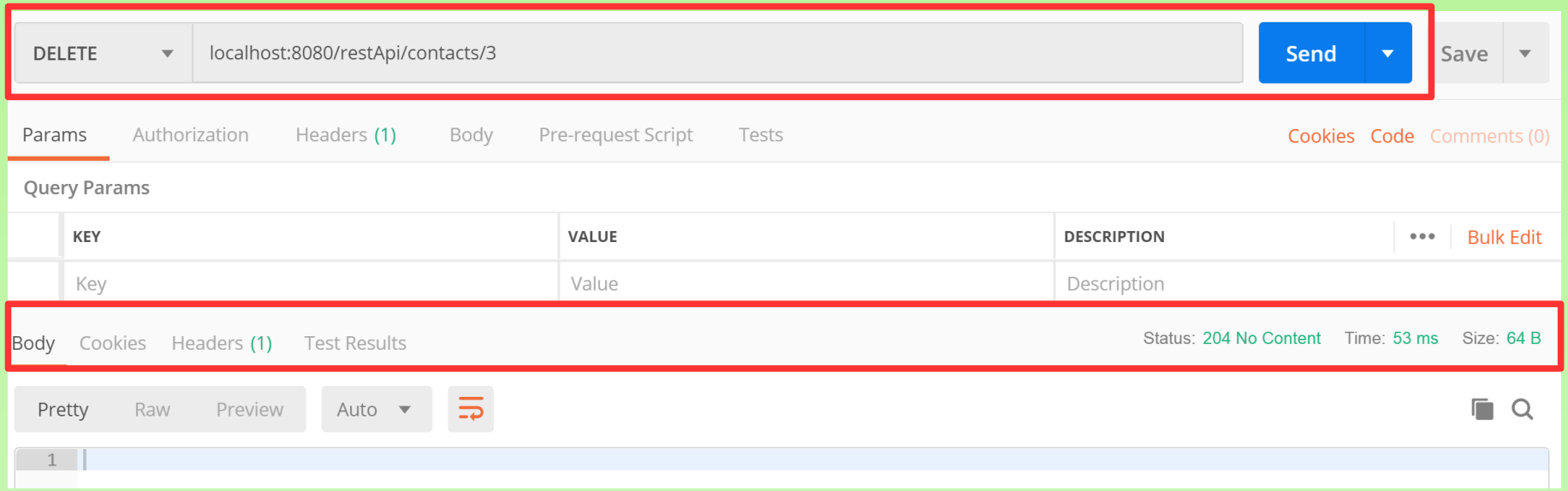
none | form-data | x-www-form-urlencoded | raw | binary | JSON (application/json) | Beautify

```
1 {
2     "firstname": "Karol",
3     "email": "knowakowski@email.pl"
4 }
```

Body | Cookies | Headers (1) | Test Results | Status: 204 No Content | Time: 86 ms | Size: 64 B

Spring Boot – REST example, Postman

dr inż. Rafał Kotas, rkotas@dmcs.pl



To run Postman on Ubuntu in 301 CTI laboratory run in terminal:

/usr/Postman/app/Postman

THE END

fiona.dmcs.pl/~rkotas/ai/angular_v2.zip

Katedra Mikroelektroniki i Technik Informatycznych
Politechniki Łódzkiej
ul. Wólczanska 221/223 budynek B18, 90-924 Łódź

dr inż. Rafał Kotas