Hypermedia As the Engine of Application State (HATEOAS)

SoftUni Team

Technical Trainers

Software University

Software University

https://softuni.bg

Table of Contents

Have a Question?

Software University

sli.do

#java-web

What is HATEOAS

Hypermedia As the Engine of Application State

- **HATEOAS** is a constraint of the REST application architecture
- Keeps the RESTful style architecture **unique from most other network application** architectures
- Uses **hypermedia** to describe what future actions are available to the client
- Allowable actions are derived in the API based on the current application state and returned to the client as a **collection of links**

Hypermedia As the Engine of Application State (2)

- Client uses these **links to drive further** interactions with the API
- Tells the client what **options** are **available** at a given point in time.
  - Doesn't tell them how each link should be used or exactly what information should be sent
- It is conceptually the same as a **web user browsing** through web pages by clicking the **relevant hyperlinks** to achieve a final goal

HATEOAS Example

HATEOAS Example (1)

- Simple response **without** using **HATEOAS**
  - We have a simple REST controller that returns entity in JSON format to the client

```
{ "id" :2, "name": "Pesho", "age":12 }
```

HATEOAS Example (2)

- **Using HATEOAS**


```
{ "id":2,"name":"Pesho","age":12,"
  _links":{
   "self":{"href":"http://localhost:8080/students/2"},
   "delete":{"href":"http://localhost:8080/students/delete/2"},
  "update":{"href":"http://localhost:8080/students/update/2"},
  "orders":{"href":"http://localhost:8080/orders/allByStudentId/2"}
     } }
```

Rel & Href

- **rel** - describes the **relationship** between the Student resource and the URL
  - In example above – self, update, delete …
  - **describes** the **action** that's performed with the link
  - It's important that this **value** is intuitive as it **describes** the purpose of the link
- **href** - the **URL** used to perform the action described in rel

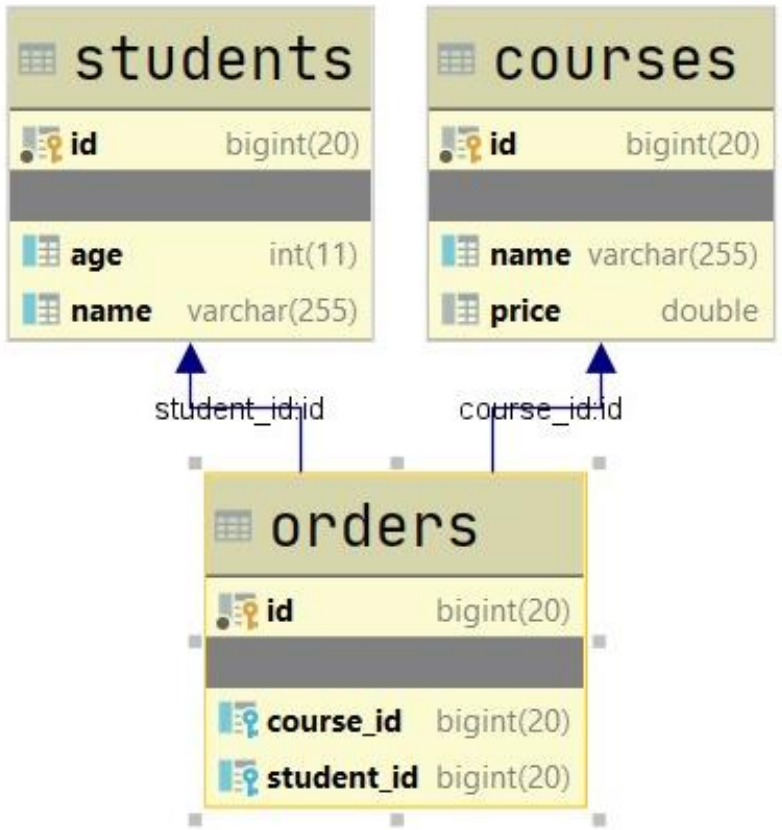
Implement HATEOAS in Spring

Using HATEOAS in Spring Framework

- Adding hypermedia links to RESTful responses is something you could implement on your own, but …
- **Spring HATEOAS** makes it **very easy**

```
<dependency>
        <groupId>org.springframework.hateoas</groupId>
        <artifactId>spring-hateoas</artifactId>
        <version>1.1.0.RELEASE</version>
</dependency>
```

Example App DB



- Our example app have small base with some relations between entities
  - We have **Students**, **Orders** and **Courses**

Prepare Controllers to Work

- If we implementing **RepresentationModel <T>** we can added links directly to our entity
- We need two methods from **WebMvcLinkBuilder**, that's why we must import them

```
import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.linkTo;
import static org.springframework.hateoas.server.mvc.WebMvcLinkBuilder.methodOn;
```

Main Work in Controller (1)

```
...     // Without implementing RepresentationModel<T>

Optional<Student> studentOpt =
        this.studentRepository.findById(id);

    return studentOpt
        .map(s -> ResponseEntity.ok(
                EntityModel.of(s, getStudentLinks(s))))
                .orElse(ResponseEntity.notFound().build());


        //continue to next slide

...
```

Main Work in Controller (2)

```
... // Without implementing RepresentationModel<T>
private Link[] getStudentLinks(Student student) {
    Link self = linkTo(methodOn(StudentsController.class)
        .getStudent(student.getId()))
        .withSelfRel();
    Link orders = linkTo(methodOn(StudentsController.class)
                .getAllOrdersByStudentId(student.getId()))
        .withRel("orders");
    return List.of(self, orders).toArray(new Link[0]);
}
```

Main Work in Controller (3)

```
... // Implementing RepresentationModel<T>
Student student = this.studentService.findById(id);
        student.add(linkTo(methodOn(StudentsController.class)
                .getStudent(student.getId()))
                                        .withSelfRel());
         student.add(linkTo(methodOn(StudentsController.class)
                .deleteStudent(student.getId()))
                                        .withRel("delete"));
        //continue to next slide
...
```

Main Work in Controller (4)

```
... // Implementing RepresentationModel<T>
        student.add(linkTo(methodOn(StudentsController.class)
                .updateStudent(student.getId(), student))
                        .withRel("update"));
        student.add(linkTo(methodOn(OrdersController.class)
                .findAllOrdersByUserId(student.getId()))
                        .withRel("orders"));

        return ResponseEntity.ok(student);
...
```

Benefits of Using HATEOAS

- **URL structure** of the API can be **changed without affecting** clients
  - If the URL structure is changed in the service, clients will automatically pick up the new URL structure via hypermedia
- Hypermedia APIs are **explorable**
- Guiding clients toward the next step in the workflow by **providing** only the **links** that are **relevant** based on the current application state

Negatives of Using HATEOAS

- Adds **extra complexity** to the API, which affects to:
  - **developer** needs to handle the **extra work** of adding links to each response
  - **more complex** to **build** and **test** than a vanilla CRUD REST API
  - **clients** also have to deal with the **extra complexity** of **hypermedia**

HAL Explorer

HAL Explorer

- To use **HAL Explorer** we need to add **dependency**

```
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-rest-hal-explorer</artifactId>
</dependency>
```

HAL Explorer Example (1)

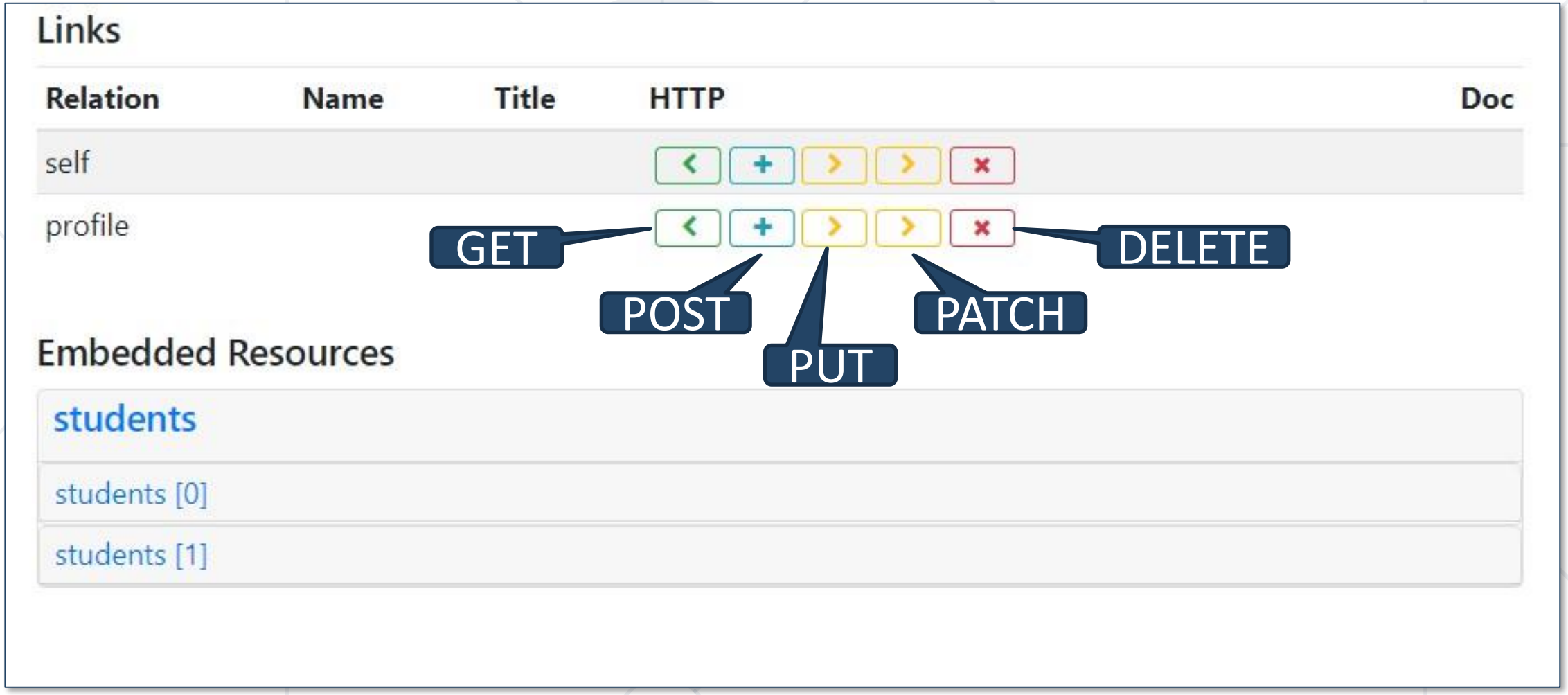
HAL Explorer Theme Layout About

Edit Headers http://localhost:8080/students Go!

JSON Properties


```
{
  "page": {
    "size": 20,
    "totalElements": 2,
    "totalPages": 1,
    "number": 0
  }
}
```

HAL Explorer Example (2)



HAL Explorer Example (3)

Response Status

200 (OK)

Response Headers

connection	keep-alive
content-type	application/hal+json
date	Mon, 06 Jul 2020 07:16:19 GMT
keep-alive	timeout=60
transfer-encoding	chunked
vary	Origin, Access-Control-Request-Method, Access-Control-Request-Headers

HAL Explorer Example (4)

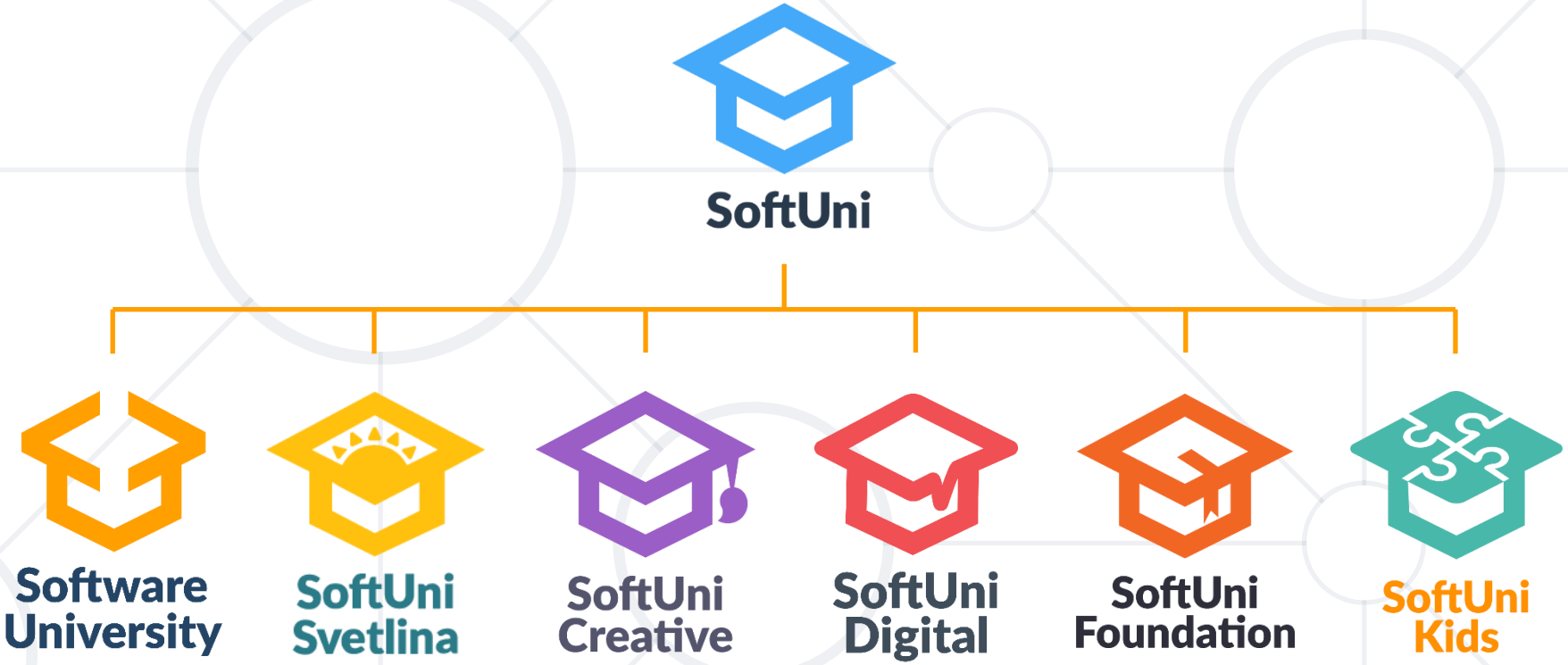
Response Body


```
{
  "_embedded": {
    "students": [
      {
        "name": "Gosho",
        "age": 1,
        "_links": {
          "self": {
            "href": "http://localhost:8080/students/1"
          },
          "student": {
            "href": "http://localhost:8080/students/1"
          },
          "orders": {
            "href": "http://localhost:8080/students/1/orders"
          }
        }
      },
      {
        "name": "Pesho",
        "age": 12,
```

Summary

- **HATEOAS**
  - What is HATEOAS
  - HATEOAS Examples
  - Implementing it in Spring
- **HAL Explorer**
  - Working with HAL Explorer

Questions?



SoftUni Diamond Partners

Software University

DRAFT KINGS

INDEAVR
Serving the high achievers

Coca-Cola HBC Bulgaria

SCHWARZ

LIDL

Kaufland

SCHWARZ Produktion

prezero

Postbank
Решения за твоето утре

POKERSTARS | CAREERS

SUPER HOSTING .BG

SOFTWARE GROUP

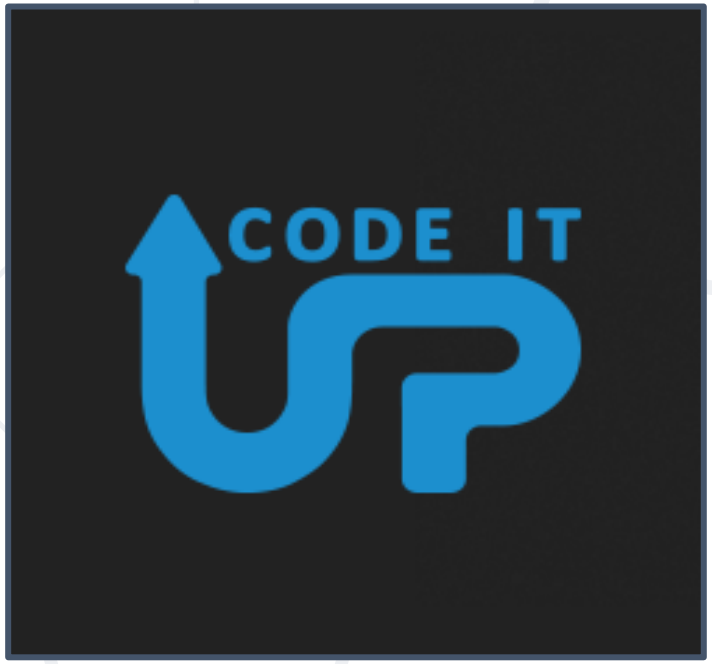
SmartIT

PHAR VISION

createX

taulia

Educational Partners



VIRTUAL RACING SCHOOL

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg