MTAT.03.229
Enterprise System Integration

Lecture 8: Hypermedia REST with HAL and HAL-Forms

Marlon Dumas
University of Tartu

Based on slides by Greg Turnquist
Recap from previous weeks

• CRUD REST services

• Multiple URIs, multiple HTTP verbs, and HTTP Status Codes

• HTTP verbs
  ◦ GET, POST, PUT, DELETE

• HTTP Status Codes
  ◦ 1xx - Metadata
  ◦ 2xx – Everything’s fine
  ◦ 3xx – Redirection
  ◦ 4xx – Client did something wrong
  ◦ 5xx – Server did a bad thing
Example: PO creation (BuildIt)

```
<purchaseOrder>
  <start>7-10-2016</start>
  <end>11-10-2016</end>
  <plant>
    <name>Excavator</name>
    <plant>
  </plant>
</purchaseOrder>

http://rentit.com/rest

POST /orders/

HTTP/1.1 201 Created
Location: /orders/1253

```

```xml
<purchaseOrder>
  <startDate>7-10-2016</startDate>
  <endDate>11-10-2016</endDate>
  <cost>1500.00</cost>
  <plant>
    <sku>exc1253ab98</sku>
    <name>Excavator</name>
    <plant>
  </plant>
  <links>
    <link rel="accept" href="/orders/1253/accept" method="POST"/>
    <link rel="reject" href="/orders/1253/accept" method="DELETE"/>
  </links>
</purchaseOrder>
```
Example: PO retrieval

GET /orders/1253

HTTP/1.1 200 Ok

```xml
<purchaseOrder>
  <startDate>7-10-2016</startDate>
  <endDate>11-10-2016</endDate>
  <cost>1500.00</cost>
  <plant>
    <sku>exc1253ab98</sku>
    <name>Excavator</name>
  </plant>
  <links>
    <link rel="accept" href="/orders/1253/accept" method="POST"/>
    <link rel="reject" href="/orders/1253/accept" method="DELETE"/>
  </links>
</purchaseOrder>
```

How do we know what inputs are required?
Specifying REST APIs

• Static API specification
  ◦ Option 1: Just write it down in text (make sure you do it rigorously)
  ◦ Option 2: API specs in Apiary, Swagger, REST Docs, Slate
    ◦ Useful to generate documentation, API mockups, and tests

• Hypermedia resource specifications
  ◦ HAL and **HAL-Forms**
  ◦ Alternatives: JSON-LD, Collection+JSON, SIREN, JSON:API
  ◦ Useful for dynamic form generation, crawling, and testing
  ◦ For a comparison: [https://sookocheff.com/post/api/one-choosing-a-hypermedia-format/](https://sookocheff.com/post/api/one-choosing-a-hypermedia-format/)
Hypermedia Application Language

• Proposed IETF specification for representing API resources and their relations with hyperlinks.

• Allows us to represent:
  ◦ The elements of an API resource
  ◦ The hypertext links of an API resource. (“_links” property)
  ◦ Its child (embedded) resources ( “_embedded” property)

• HAL has an XML and a JSON serialization (hal+json)

• HAL Forms extends HAL with the possibility of capturing “templates” to specify operations over a resource and the inputs of these operations (“_template” property)
HAL Forms : Employees Collection

{"_embedded": { embedded/child elements
  "employees": [...] },
"_links": { links
  "self": { "href": http://localhost:8080/employees } },
"_templates": { HAL Form template
  "default": {
    "title": null,
    "method": "post",
    "contentType": "",
    "properties": [
      {"name": "firstName",
       "required": true },
      {"name": "id",
       "required": true },
      {"name": "lastName",
       "required": true },
      {"name": "role",
       "required": true }
    ]
  }
}
HAL Forms: Employee

{"id" : 1,
"firstName" : "Frodo",
"lastName" : "Baggins",
"role" : "ring bearer",
"_links" : {
    "self" : {
        "href" : "http://localhost:8080/employees/1"
    },
    "employees" : {
        "href" : "http://localhost:8080/employees"
    }
},
...
HAL Forms: Employee (continued)

...
Possible form for editing/deleting employee

```html
<form method="put" action="http://localhost:8080/employees/1">
  
  <input type="text" id="firstName" name="firstName"/>
  
  <input type="text" id="id" name="id" />
  
  <input type="text" id="lastName" name="lastName" />
  
  <input type="text" id="role" name="role" />
  
  <input type="submit" value="Submit" />

</form>

<form method="delete" action="http://localhost:8080/employees/1">
  
  <input type="submit" value="Submit" />

</form>
```
HAL – Representing “paginated” resources

{
   "_links": {
      "self": { "href": "/api/books?page=7" },
      "first": { "href": "/api/books?page=1" },
      "prev": { "href": "/api/books?page=6" },
      "next": { "href": "/api/books?page=8" },
      "last": { "href": "/api/books?page=17" }
   }
}

"search": {
   "href": "/api/books?query={searchTerms}",
   "templated": true
}

...

HYPERMEDIA REST   LUCIANO GARCÍA-BañueLOS 10
HAL – Representing “paginated” resources (continued)

"_embedded": {
  "book": [
    {
      "_links": {
        "self": { "href": "/api/books/1234" }
      }
    }
    "id": 1234,
    "title": "Hitchhiker's Guide to the Galaxy",
    "author": "Adams, Douglas"
  },
  {
    "_links": {
      "self": { "href": "/api/books/6789" }
    }
  }
  "id": 6789,
  "title": "Ancillary Justice",
  "author": "Leckie, Ann"
  
  
  
  ]
},
"_page": 7,
"_per_page": 2,
"_total": 33
Exercise

• Continuing from last week’s RentIT domain model...
  ◦ Let’s write down an example of a purchase order resource and a purchase order line item example using HAL and HAL-Forms
  ◦ Include operations for changing the PO, cancelling the PO, and marking the PO as dispatched and received (shipment and delivery notifications)
Implementing Hypermedia APIs with Spring Affordances
Example Employees API

```java
@Data
@Entity
@NoArgsConstructor(access = AccessLevel.PRIVATE)
@AllArgsConstructor
class Employee {
    @Id @GeneratedValue
    private Long id;
    private String firstName;
    private String lastName;
    private String role;

    /**
     * Useful constructor when id is not yet known.
     */
    Employee(String firstName, String lastName, String role) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.role = role;
    }
}
```
Example: Employees API (cont.)

interface EmployeeRepository extends CrudRepository<Employee, Long> {
}

@RestController
class EmployeeController {

    private final EmployeeRepository repository;

    EmployeeController(EmployeeRepository repository) {
        this.repository = repository;
    }

    ...
}

Example: Employees API (cont.)

@RestController
class EmployeeController {
    ...
    @GetMapping("/employees/{id}")
    ResponseEntity<Resource<Employee>> findOne(@PathVariable long id) {
        return repository.findById(id)
            .map(employee -> new Resource<>(employee, getSingleItemLinks(employee.getId())))
            .map(ResponseEntity::ok)
            .orElse(ResponseEntity.notFound().build());
    }
    ....
Example: Employees API (cont.)

```java
@PutMapping("/employees/{id}"),
ResponseEntity<?>
updateEmployee(@RequestBody Employee employee,
@PathVariable long id) {

    employee.setId(id);
    Employee updatedEmployee = repository.save(employee);

    Resource<Employee> employeeResource = new Resource<
    (updatedEmployee, getSingleItemLinks(updatedEmployee.getId())));

    try {
        return ResponseEntity
            .created(new
            URI(employeeResource.getRequiredLink(Link.REL_SELF).getHref()))
                .body(employeeResource);
    } catch (URISyntaxException e) {
        return ResponseEntity.badRequest().body("Unable to update " + employee);
    }
```
Example: Employees API

```java
private List<Link> getSingleItemLinks(long id) {
    return Arrays.asList(
        linkTo(methodOn(EmployeeController.class).findOne(id)).withSelfRel()
            .andAffordance(afford(methodOn(EmployeeController.class).updateEmployee(null, id)))
            .andAffordance(afford(methodOn(EmployeeController.class).deleteEmployee(id))),
        linkTo(methodOn(EmployeeController.class).findAll()).withRel("employees"));
}
```

https://spring.io/blog/2018/01/12/building-richer-hypermedia-with-spring-hateoas
https://github.com/spring-projects/spring-hateoas-examples