Richardson’s Maturity Level 2

- Also known as CRUD 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
Hypermedia within our application?

• HATEOAS: Hypermedia as the engine of application state

• Roy Fielding’s view:

  The simultaneous presentation of information and controls such that the information becomes the affordance through which the user obtains choices and selects actions.

• A simpler wording:

  Include links into the resource representation. Let the user (human or program) continue the execution by following one of those links.
Hypermedia in the Web

<a href="http://www.example.org/search" title="View search page">Search</a>

<img src="http://www.example.org/images/logo" title="Company logo"/>

<form method="get">
  <label>Search term:</label>
  <input name="query" type="text" value=""/>
  <input type="submit"/>
</form>

<form method="post" action="http://www.example.org/keywords'">
  <label>Keywords:</label>
  <input name="keywords" type="text" value=""/>
  <input type="submit"/>
</form>
Hypermedia in the Web

Lazy loading (the user must explicit follow the link)

Eager loading (the browser downloads the image without asking anything to the user)
Hypermedia in the Web

```html
<form method="get">
  <label>Search term:</label>
  <input name="query" type="text" value=""/>
  <input type="submit"/>
</form>

GET /books/?query=rest HTTP/1.1
Host: www.example.org

POST /keywords HTTP/1.1
Host: www.example.org
Content-Type: application/x-www-form-urlencoded

keywords=SOAP+REST+WS

<form method="post" action="http://www.example.org/keywords">
  <label>Keywords:</label>
  <input name="keywords" type="text" value=""/>
  <input type="submit"/>
</form>
```
H-Factors

• “The H Factor of a media-type is a measurement of the level of hypermedia support and sophistication of a media-type” M. Amundsen

1. Link Support
   • [LE] Embedding links
   • [LO] Outbound links
   • [LT] Templated queries
   • [LN] Non-Idempotent updates (POST)
   • [LI] Idempotent updates (PUT/DELETE)

2. Control Data Support
   • [CR] Control data for read requests (Accept header)
   • [CU] Control data for update requests (Content-Type header)
   • [CM] Control data for interface methods (HTTP verb)
   • [CL] Control data for links (link relations)
H-Factor of popular REST media types

https://gtramontina.github.io/h-factors/
Example: PO creation (BuildIt)

POST /orders

http://rentit.com/rest

BuildIt

HTTP/1.1 201 Created
Location: /orders/1253

RentIt

<purchaseOrder>
  <start>7-10-2016</start>
  <end>11-10-2016</end>
  <plant>
    <name>Excavator</name>
  </plant>
</purchaseOrder>
Example: RentIt’s view

HTTP/1.1 200 Ok

GET /orders/1253

http://rentit.com/rest

<purchaseOrder>
  <startDate>7-10-2016</startDate>
  <endDate>11-10-2016</endDate>
  <cost>1500.00</cost>
  <plant>
    <sku>exc1253ab98</sku>
    <name>Excavator</name>
  </plant>
</purchaseOrder>

<link rel="accept" href="/orders/1253/accept" method="POST"/>
<link rel="reject" href="/orders/1253/accept" method="DELETE"/>
Our example application
Our example application

Integration layer (REST adapter)

Application

Domain model
Resource life cycle: Purchase order

PurchaseOrderService
- createPurchaseOrder()
- acceptPurchaseOrder()
- rejectPurchaseOrder()
- resubmitPurchaseOrder()
- extendRentalPeriod()
- acceptRentalPeriodExtension()
- rejectRentalPeriodExtension()
Hypermedia API

<table>
<thead>
<tr>
<th>Verb</th>
<th>URI template</th>
<th>Relation</th>
<th>Current state</th>
<th>New state</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/orders</td>
<td>createPO</td>
<td>Pending confirmation</td>
<td></td>
<td>Submit partial representation</td>
</tr>
<tr>
<td>POST</td>
<td>/orders/{oid}/accept</td>
<td>acceptPO</td>
<td>Pending confirmation</td>
<td>Open</td>
<td>Use empty body</td>
</tr>
<tr>
<td>DELETE</td>
<td>/orders/{oid}/accept</td>
<td>rejectPO</td>
<td>Pending confirmation</td>
<td>Rejected</td>
<td></td>
</tr>
</tbody>
</table>
# Full Purchase Order API

<table>
<thead>
<tr>
<th>Method</th>
<th>URI template</th>
<th>Relation</th>
<th>Current state</th>
<th>New state</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/orders</td>
<td>createPO</td>
<td>Pending confirmation</td>
<td></td>
<td>Submit partial representation</td>
</tr>
<tr>
<td>POST</td>
<td>/orders/{oid}/accept</td>
<td>acceptPO</td>
<td>Pending confirmation</td>
<td>Open</td>
<td>Use empty body</td>
</tr>
<tr>
<td>DELETE</td>
<td>/orders/{oid}/accept</td>
<td>rejectPO</td>
<td>Pending confirmation</td>
<td>Rejected</td>
<td></td>
</tr>
<tr>
<td>PUT</td>
<td>/orders/{oid}</td>
<td>resubmitPO</td>
<td>Rejected</td>
<td>Pending confirmation</td>
<td>Submit new full representation</td>
</tr>
<tr>
<td>DELETE</td>
<td>/orders/{oid}</td>
<td>closePO</td>
<td>Open</td>
<td>Closed</td>
<td></td>
</tr>
<tr>
<td>POST</td>
<td>/orders/{oid}/extensions</td>
<td>extendRentalPeriod</td>
<td>Open</td>
<td>Pending update</td>
<td>Submit new end date</td>
</tr>
<tr>
<td>POST</td>
<td>/orders/{oid}/extensions/{eid}/accept</td>
<td>acceptRPExtension</td>
<td>Pending update</td>
<td>Open</td>
<td>Use empty body</td>
</tr>
<tr>
<td>DELETE</td>
<td>/orders/{oid}/extensions/{eid}/accept</td>
<td>rejectRPExtension</td>
<td>Pending update</td>
<td>Open</td>
<td></td>
</tr>
</tbody>
</table>
Annotated State model

- POST /orders
  - createPO

- POST /orders/{oid}/extensions/{eid}/accept
  - acceptRPEXTENSION

- DELETE /orders/{oid}/extensions/{eid}/accept
  - rejectRPEXTENSION

- POST /orders/{oid}/accept
  - acceptPO

- DELETE /orders/{oid}/accept
  - rejectPO

- PUT /orders/{oid}
  - resubmitPO

- DELETE /orders/{oid}
  - closePO

- POST /orders/{oid}/extensions
  - extendRentalPeriod

- POST /orders/{oid}/accept
  - acceptRPExtension

- PUT /orders/{oid}
  - resubmitPO

- closePO

- rejected

- pending

- confirmation

- open

- pending extension
Implementing the Hypermedia API
Changes on resource classes

In this case, we will use the library “spring-hateoas”. However, the classes required and their implementation are quite similar to those in our previous version.

Note that we will use JSON as the representation format.

Maven dependency

Group id: org.springframework.hateoas
Artifact id: spring-hateoas
Version: 0.19.0.RELEASE
public class PlantResourceAssembler
   extends ResourceAssemblerSupport<Plant, PlantResource> {

   public PlantResourceAssembler() {
      super(PlantRESTController.class, PlantResource.class);
   }

   @Override
   public PlantResource toResource(Plant plant) {
      PlantResource resource;
      resource = createResourceWithId(plant.getId(), plant);
      resource.setName(plant.getName());
      resource.setDescription(plant.getDescription());
      resource.setPrice(plant.getPrice());
      return resource;
   }

   }
}
@RestController
@RequestMapping("/api/orders")
public class PurchaseOrderRESTController {

@Autowired
PurchaseOrderService service;

@RequestMapping(method = RequestMethod.POST, value = "")
public ResponseEntity<PurchaseOrderResource> createPO(@RequestBody PurchaseOrder po)
    throws PlantUnavailableException {
    po.setStatus(POStatus.PENDING_CONFIRMATION);
    service.createPO(po);

    HttpHeaders headers = new HttpHeaders();
    URI uri = linkTo(PurchaseOrderRESTController.class).slash(po.getId()).toUri();
    headers.setLocation(uri);
    PurchaseOrderResource poRs = assembler.toResource(po);
    ResponseEntity<PurchaseOrderResource> response;
    response = new ResponseEntity<>(poRs, headers, HttpStatus.CREATED);
    return response;
}
}
REST service façade (retrieval)

```java
@Controller
@RequestMapping("/api/orders")
public class PurchaseOrderRESTController {

@RequestMapping(method = RequestMethod.GET, value = "{id}")
public ResponseEntity<PurchaseOrderResource> getPO(@PathVariable Long id) {
    PurchaseOrder po = PurchaseOrder.findPurchaseOrder(id);
    PurchaseOrderResourceAssembler assembler = new PurchaseOrderResourceAssembler();
    PurchaseOrderResource resource = assembler.toResource(po);
    switch (po.getStatus()) {
        case PENDING_CONFIRMATION:
            Method _acceptPO = PurchaseOrderRESTController.class.getMethod("acceptPO", Long.class);
            String acceptLink = linkTo(_acceptPO, po.getId()).toUri().toString();
            resource.add(new ExtendedLink(acceptLink, "acceptPO", "POST"));
            Method _rejectPO = PurchaseOrderRESTController.class.getMethod("rejectPO", Long.class);
            String rejectLink = linkTo(_rejectPO, po.getId()).toUri().toString();
            resource.add(new ExtendedLink(rejectLink, "rejectPO", "DELETE"));
            break;
        default: break;
    }
    ResponseEntity<PurchaseOrderResource> response;
    response = new ResponseEntity<>(resource, HttpStatus.OK);
    return response;
}
```

---

HYPERMEDIA REST  LUCIANO GARCÍA-BAÑUELOS  21
Exception handling
Example

GET /orders/1253

HTTP/1.1 200 Ok
<purchaseOrder>...</purchaseOrder>

HTTP/1.1 404 Not found

HTTP/1.1 401 Unauthorized

HTTP/1.1 500 Internal Server Error
Exception handling & HTTP status codes

• 400 – Bad Request
  ◦ The client has PUT or POST a resource representation that is in the right format, but contains invalid information

• 401 – Unauthorized
  ◦ Proper credentials to operate on a resource were not provided
  ◦ Response WWW-Authenticate header contains the type of authentication the server expects
    ◦ Basic, digest, WSSE, etc.

• 403 – Forbidden
  ◦ The client request is OK, but the server doesn’t want to process it (e.g., Restricted by IP address)
Exception handling & HTTP status codes

• 404 – Not Found
  ◦ The standard catch-all response

• 405 – Method Not Allowed
  ◦ The resource does not support a given method. Use “Allow” header to list the verbs the resource understands (e.g., \texttt{Allow: GET, POST, PUT})

• 409 – Conflict
  ◦ Tried to change the state of the resource to something the server will not allow
Exception handling & HTTP status codes

• 406 – Not Acceptable
  ◦ The server cannot reply with the representation requested by the client
    ◦ HTTP header Accept

• 415 Unsupported Media Type
  ◦ The server cannot consume the resource format used by the client
    ◦ HTTP header Content-type
Exception handling within our REST controller

```java
@RestController
@RequestMapping("/rest/orders")
public class PurchaseOrderRESTController {

    @ExceptionHandler({PlantUnavailableException.class,
                        InvalidHirePeriodException.class})
    public ResponseEntity<String> handleBadRequest(Exception ex) {
        return new ResponseEntity<>(ex.getMessage(), HttpStatus.CONFLICT);
    }

    @ExceptionHandler(EntityNotFoundException.class)
    public ResponseEntity<String> handleNotFound(Exception ex) {
        return new ResponseEntity<>(ex.getMessage(), HttpStatus.NOT_FOUND);
    }
}
```