Don’t Hate the HATEOAS

Or How I Learned To Stop Worrying And Love The HATEOAS

Billy Korando, Software Consultant
Keyhole Software

http://bit.ly/1ThsbZI

Thank you sponsors!

Who’s this person talking to me?!

- Software consultant with Keyhole software in Kansas City
- Over 8 years of experience in Java web development
- Worked in insurance, healthcare, and freight shipping

http://bit.ly/1ThsbZI

Before we get started

1. Feel free to ask me a question at any point in the talk
2. If I start going too fast, tell me
3. The only link you need is in the bottom right on each slide

http://bit.ly/1ThsbZI

Agenda

1. Background on REST
2. Moving towards HATEOAS
3. Showing HATEOAS worth with use cases
4. Implementing a HATEOAS service with Spring-Data-REST and Spring-HATEOAS

http://bit.ly/1ThsbZI

REST is more than GET /resource

A success story in constraint

http://bit.ly/1ThsbZI

Key REST Constraints

- Client-Server
- Stateless
- Cacheable
- Uniform Interface
- Layered Architecture

http://bit.ly/1ThsbZI

Client-Server

- Independent evolvability
- Scalability

http://bit.ly/1ThsbZI

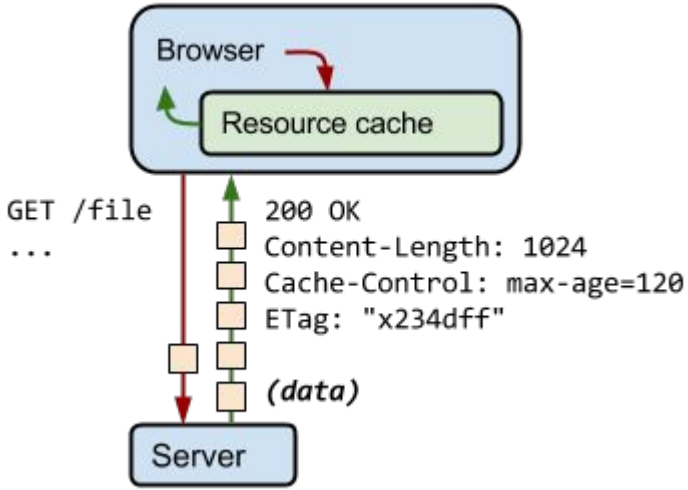
Stateless

- Durability
- Scalability
- Client maintains state (not server)
- Heavier network load

http://bit.ly/1ThsbZI

Cacheable

- Request can be cached by the client to increase client perceived performance and decrease server workload



http://bit.ly/1ThsbZI

Uniform Interface

- Where the look and feel of REST
  - Resources are identified in the URI
  - Resources are manipulated
  - Messages are self-descriptive
  - Hypermedia as the engine of application state (more on that later)
- Basically this is setting up what a REST request and response looks like

http://bit.ly/1ThsbZI

Layered Architecture

- Encapsulation of legacy processes
- Sharing of resources (i.e. a cache)
- Load balancing

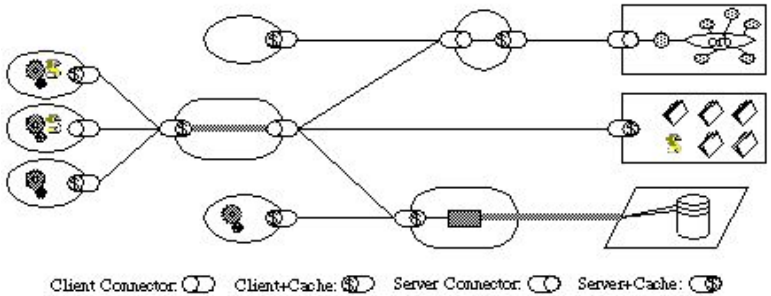


Figure 5-8. REST

http://bit.ly/1ThsbZI

The Richardson Maturity Model

- Level 0 - The Swamp of POX
- Level 1 - Resources
- Level 2 - Proper use of HTTP mechanics
- Level 3 - Hypermedia Controls

http://bit.ly/1ThsbZI

The “Swamp” of POX

Using HTTP as a tunneling mechanism

```
POST: /viewItem
{
      “id”: “1234”
}
Response:
HTTP 1.1 200
{
      “id” : 1234,
      “description” : “TV”
}
POST: /orderItem
{
      “id” : 1,
      “item” : {
            “id” : 1234
      }
}
```

http://bit.ly/1ThsbZI

Resources

- Entities now have their own dedicated endpoint
- Interacting with the RESTful service is done by manipulating a resource

http://bit.ly/1ThsbZI

```
POST: /items/1234
{}
Response:
HTTP 1.1 200
{
      “id” : 1234,
      “description” : “TV”
}
POST: /order/1
{
      “item” : {
            “id” : 1234
      }
}
Response:
HTTP 1.1 200
{
      “id” : 1,
      “items” : [
            “item” : {
                  “id” : 1234
            }
      ]
}
```

http://bit.ly/1ThsbZI

HTTP

- Using the HTTP specification as it has been defined
  - Requests will now use the correct HTTP “verb” (method type)
  - Server will respond with the correct status code

http://bit.ly/1ThsbZI

HTTP Verbs

	SAFE	**NOT SAFE**
IDEMPOTENT	GET, OPTIONS	DELETE, PUT
NOT IDEMPOTENT		POST

http://bit.ly/1ThsbZI

```
GET: /items/1234
Response:
HTTP 1.1 200
{
    "id" : 1234,
    "description" : "TV"
}

GET: /items/NotReal
Response:
HTTP 1.1 404
```

```
PUT: /orders/1
{
    "items" : [
        "item" : {
            "id" : 1234
        }
    ]
}
```

```
Response:
HTTP 1.1 226
{
    "items" : [
        "item" : {
            "id" : 1234
        }
    ]
}
```

http://bit.ly/1ThsbZI

Why HTTP Isn’t Enough

- Client needs to understand state
  - Changes to how state is determined need to be made in multiple locations
  - Creates deployment dependencies

http://bit.ly/1ThsbZI

Hypermedia Controls

- The server tells us what we can do

http://bit.ly/1ThsbZI

```
GET: /items/1234
Response:
HTTP 1.1 200
{
    "id" : 1234,
    "description" : "FooBar TV",
    "image" : "fooBarTv.jpg",
    "price" : 50.00,
    "link" : {
            "rel" : "next",
            "href" : "/orders"
    }
}
POST: /orders
{
    "id" : 1,
    "items" : [
        {
            "id" : 1234
        }
    ]
}
```

```
Response:
HTTP 1.1 201:
{
    "id" : 1,
    "items" : [
    {
            "id" : 1234
    }
],
links : [
        {
            "rel" : "next",
            "href" : "/orders/1/payment"
        },
        {
            "rel" : "self",
            "href" : "/orders/1"
        }
    ]
}
```

http://bit.ly/1ThsbZI

HATEOAS use case #1

- Admins and common users interact through the same client
  - Both can view items
  - Only admins should be able to update and delete items
  - Users should be able to purchase items

http://bit.ly/1ThsbZI

```
Request:
[Headers]
user: bob
roles: USER
GET: /items/1234
Response:
HTTP 1.1 200
{
    “id” : 1234,
    “description” : “FooBar TV”,
    “image” : “fooBarTv.jpg”,
    “price” : 50.00,
    “links” : [
            {
                “rel” : “next”,
                “href” : “/orders”
            }
        ]
    }
}
```

```
Request:
[ Headers ]
user: jim
roles:ADMIN
GET: /items/1234
Response:
HTTP 1.1 200
{
    “id” : 1234,
    “description” : “FooBar TV”,
    “image” : “fooBarTv.jpg”,
    “price” : 50.00,
    “links” : [
            {
                “rel” : “modify”,
                “href” : “/items/1234”
            },
            {
                “rel” : “delete”,
                “href” : “/items/1234”
            }
        ]
    }
}
```

http://bit.ly/1ThsbZI

HATEOAS use case #2

- Users can sell their own items
  - Users should be able to add, update, and delete their own items
  - Users should not be able to add, update, and delete items for other users

http://bit.ly/1ThsbZI

```
Request:
[Headers]
user: jim
roles: USER
GET: /items/1234
Response:
HTTP 1.1 200
{
    “id” : 1234,
    “description” : “FooBar TV”,
    “image” : “fooBarTv.jpg”,
    “price” : 50.00,
    “links” : [
            {
                “rel” : “modify”,
                “href” : “/items/1234”
            },
            {
                “rel” : “delete”,
                “href” : “/items/1234”
            }
        ]
    }
}
```

```
Request:
[Headers]
user: bob
roles: USER
GET: /items/1234
Response:
HTTP 1.1 200
{
    “id” : 1234,
    “description” : “FooBar TV”,
    “image” : “fooBarTv.jpg”,
    “price” : 50.00,
    “links” : [
            {
                “rel” : “next”,
                “href” : “/orders”
            }
        ]
    }
}
```

http://bit.ly/1ThsbZI

HATEOAS standards

- HAL (Hypertext Application Language) - used by Spring Data REST
- RFC 5988

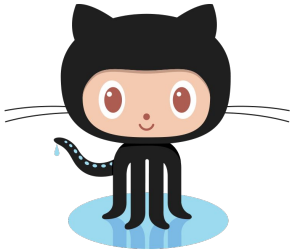
http://bit.ly/1ThsbZI

Consuming HATEOAS

- Traverson - Node
  - https://github.com/basti1302/traverson
- Angular Spring Data REST
  - https://github.com/guylabs/angular-spring-data-rest

http://bit.ly/1ThsbZI

HATEOAS in the wild



Github

Amazon AppStream

Paypal

http://bit.ly/1ThsbZI

Springtime for HATEOAS!

- Implementing a HATEOAS service using Spring-Data-REST and Spring-HATEOAS
- Source code can be found here: https://github.com/in-the-keyhole/hateoas-demo-II

http://bit.ly/1ThsbZI

Additional reading:

This talk in blog form:

https://keyholesoftware.com/2016/02/29/dont-hate-the-hateoas/

https://keyholesoftware.com/2016/05/09/dont-hate-the-hateoas-part-deux-springtime-for-hateoas/

Code:

https://github.com/in-the-keyhole/hateoas-demo-II

http://bit.ly/1ThsbZI

Sources

- https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
- http://martinfowler.com/articles/richardsonMaturityModel.html
- http://www.infoq.com/articles/webber-rest-workflow
- https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html
- http://docs.spring.io/spring-data/rest/docs/2.5.1.RELEASE/reference/html/
- http://docs.spring.io/spring-hateoas/docs/0.19.0.RELEASE/reference/html/

http://bit.ly/1ThsbZI

Keep in touch

Twitter: @KorandoBilly

Email: bkorando@keyholesoftware.com

http://bit.ly/1ThsbZI