Creating REST-ful, Hypermedia-based Micro-services with Spring Boot

Ben Hale, Pivotal

What is a Micro-service?


```
@RestController class App { @RequestMapping def ok() { "OK" } }
```

- Not necessarily small by code, rather small by scope or ambition
- A pattern for achieving high cohesion, separation of concerns, and loose coupling.
- Given these goals, why is being REST-ful a good choice for a micro-service?

What is REST?

- REpresentational State Transfer

- An architectural style for designing distributed systems
  - Client/Server, Stateless, Uniform Interface, etc.
  - Highly-cohesive, loosely coupled services

- Not a standard, but rather a set of constraints
  - Flexibility allows it to be adaptable to many types of services

- Not tied to HTTP, but most commonly associated with it

The Uniform Interface

- Flexible principals (no hard and fast rules)
  - Identification of resources
  - Manipulation of resources
  - Self-describing messages
  - Hypermedia as the engine of application state

HTTP's Uniform Interface

- URI's identify resources
  - `/games/0`

- HTTP verbs descried a limited set of operations that can be used to manipulate a resource
  - `GET`, `DELETE`, `PUT`, `POST`, ...

- Headers describe the messages
  - `Content-Type: application/json`

GET

- Retrieve Information
- Must be safe and idempotent
  - Can have side-effects, but since the user doesn't expect them, they shouldn't be critical to the operation of the system
- `GET` can be conditional or partial
  - `If-Modified-Since`
  - `Range`

```
GET /games/1
```

DELETE

- Requests that a resource be removed
- The resource **doesn't** have to be removed immediately
  - Removal may be a long running task

```
DELETE /games/1
```

PUT

- Requests that the entity passed, be stored at the URI
- Can be used to create a new entity or modify an existing one
  - Creation of new entities is uncommon as it allows the client to select the id of the new entity

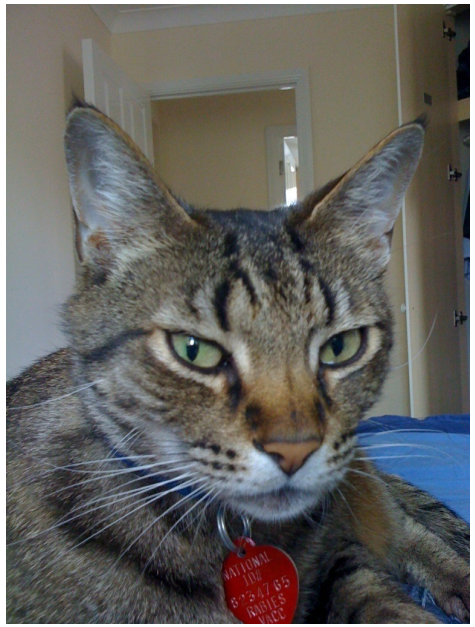
```
PUT /games/1/doors/2
{ “status”: "SELECTED" }
```

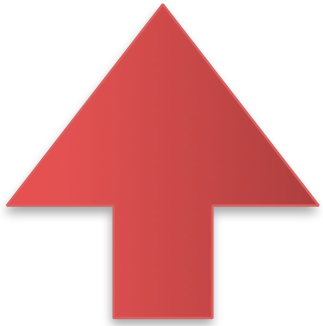
POST

- Requests that the resource at a URI do *something* with the enclosed entity
- What that something is could be almost anything
  - Create, Modify, …
- The major difference between POST and PUT is what resource the request URI identifies

```
POST /games
```

Let's Make A Deal!





Interaction Model

- *Create* `Game`
- *List* the current state of all `Doors`
- *Select* a Door
  - the `Game` will open one of the other non-bicycle `Doors`
- *Open* one of the two remaining `Doors`
- *List* the outcome of the `Game`
- *Delete* the `Game`

Create a `Game`

- Well-known entry point
- Doesn't require any input other than requesting that a `Game` be created
- Needs to return to us a resource identifier (URI) of the newly created `Game`

```
POST /games
```

List the current state of all `Doors`

- Needs to return to us a collection that represents the sate of all doors in the game
- Design doesn't have 'Door 1', 'Door 2', and 'Door 3', just three doors

```
GET /games/0/doors
[{ "status": "CLOSED" }, {…}, {…}]
```

Select a Door

- There is no HTTP verb `SELECT`, so how do we represent the selection of a door?
- Request a resource mutation that leaves the resource in the desired state

```
PUT /games/0/doors/1
{ "status": "SELECTED" }
```

Open a Door

- Like `SELECT`, we want to require a mutation to the desired state
- Since the same (or same type of) resources is being modified, we re-use the same payload

```
PUT /games/0/doors/3
{ "status": "OPENED" }
```

List the outcome of the `Game`

- Needs to return an object that represents the state of the `Game`

```
GET /games/0
{ "status": "WON" }
```

Destroy the Game

- No input required
- No output required

```
DELETE /games/0
```

Spring Boot and Spring MVC Implementation

Status Codes

- Status codes indicate the result of the server's attempt to satisfy the request
- Broadly divided into categories
  - 1XX: Informational
  - 2XX: Success
  - 3XX: Redirection
  - 4XX: Client Error
  - 5XX: Server Error

Success Status Codes

- `200 OK`
  - Everything worked
- `201 Created`
  - The server has successfully created a new resource
  - Newly created resource's location returned in the `Location` header
- `202 Accepted`
  - The server has accepted the request, but it is not yet complete
  - A location to determine the request's current status can be returned in the `Location` header

Client Error Status Codes

- `400 Bad Request`
  - Malformed syntax
  - Should not be repeated without modification
- `401 Unauthorized`
  - Authentication is required
  - Includes a `WWW-Authenticate` header
- `403 Forbidden`
  - Server has understood but refuses to honor the request
  - Should not be repeated without modification

Client Error Status Codes

- `404 Not Found`
  - The server cannot find a resource matching a URI
- `406 Not Acceptable`
  - The server can only return response entities that do not match the client's `Accept` header
- `409 Conflict`
  - The resource is in a state that is in conflict with the request
  - Client should attempt to rectify the conflict and then resubmit the request

Spring MVC Exception Handling

What is HATEOAS?

- Hypermedia As the Engine of Application State

- The client doesn't have built-in knowledge of how to navigate and manipulate the model
- Instead the server provides that information dynamically to the client
- Implemented using media types and link relations

Media Types

- A resource can be represented in different ways
  - JSON, XML, etc.
- A client doesn't know what a server is going to send it
- A server doesn't know what a client can handle
- Content types are negotiated using headers
  - Client describes what it wants with the `Accept` header
  - Server (and client during `POST` and `PUT`) describes what it is sending with `Content-Type` header

Link Relations

- A client cannot be expected to know what a resource is related to and where those relations are located
- The server describes these relations as part of its payload
- Link has two parts
  - `rel`, `href`
- `rel` values are “standardized” so the client can recognize them
  - `<link rel="stylesheet" href="..."/>`
  - `{ "rel": "doors", "href": "..." }`

Spring HATEOAS Implementation

Using the API

- Designed, implemented, and (hypothetically) tested, but can you actually use the API?

- Goals
  - Single URL
  - Link traversal
  - Content Negotiation

Consuming the Game with Ruby

Round Up

- API Design Matters
  - URIs represent resource, not actions
  - HTTP verbs are general, but can be used in ways that make anything possible

- Implementation isn't rocket science
  - Spring Boot
  - Spring MVC
  - Spring HATEOAS

Creating REST-ful, Hypermedia-based Micro-services with Spring Boot

https://github.com/nebhale/spring-one-2014