http://github.com/nebhale/devoxx-2013

REST-ful API Design with Spring

Ben Hale
Cloud Foundry Java Experience
Pivotal

What is REST?

- REpresentational State Transfer
- An architectural style for designing distributed systems
- Not a standard, but rather a set of constraints
  - Client/Service, Stateless, Uniform Interface, etc.
- Not tied to HTTP but associated most commonly with it

Uniform Interface

- “… more what you’d call ‘guidelines’ than actual rules”
  - — Captain Barbossa

- Identification of resources
- Manipulation of resources
- Self-descriptive messages
- Hypermedia As The Engine Of Application State

HTTP's Uniform Interface

- URIs identify resources
- HTTP verbs describe a limited set of operations that be used to manipulate a resource
  - `GET`, `DELETE`, `POST`, `PUT`
  - other lesser-used verbs
- Headers describe the messages

GET

```
GET /games/1
```

- 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`

DELETE

```
DELETE /game/1
```

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

PUT

```
PUT /games/1/doors/2
{"status": "selected"}
```

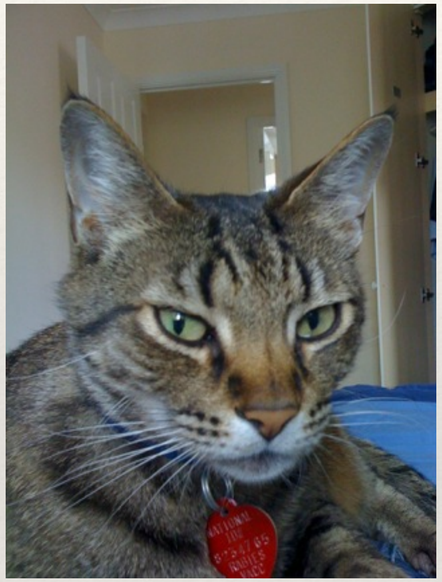
- Requests that the entity passed, be stored at the URI
- Can be used to create a new entity or modify an existing entity
  - Creation of new entities is uncommon as it allows the client to select the id of the new entity

POST

```
POST /games
```

- Requests that the entity at the URI do *something* with the enclosed entity
- What that something is, could be almost anything
  - create, modify
- The major difference between `PUT` and `POST` are what resource the request URI identifies

Let's Make a Deal!



Interaction Model

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

Create a `Game`

```
POST /games
```

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

List the Current State of a `Door`

```
GET /games/0/doors/1
{"status": "CLOSED"}
```

- Needs to return us the status of a given door
- Design ‘Door 1’, ‘Door 2’, ‘Door 3’. Instead it links to three uniquely identifiable doors

Select a `Door`

```
PUT /games/0/doors/2
{"status": "SELECTED"}
```

- 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

Open a `Door`

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

- Like select, we want to request a mutation to the desired state
- Since the same (or same type of) resource is being modified, we re-use the same payload

List the Final State of the `Game`

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

- Needs to return an object that represents the state of the game

Destroy the `Game`

```
DELETE /games/0
```

- No input required
- No output required

Spring MVC Implementation

```
@RestController
@RequestMapping
```

Status Codes

- Status codes indicate the result of the server's attempt to satisfy the request
- Broadly defined 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
- `403 Forbidden`
  - Server has understood, but refuses to honor the request
  - Should not be repeated without modification

Client Error Status Codes

- `404 Not Found`
  - There 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 Response Status

```
ResponseEntity<>
@ResponseStatus
@ExceptionHandler
```

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 user
- Implemented by using media types and link relations

Media Types

- A resource can be represented in different ways
  - JSON, XML, etc.
- A client does not 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 an `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 resources 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

```
linkTo()
.withRel()
```

Testing

- Testing of Web APIs isn't easy
  - In-container, end-to-end, string comparison, etc.
  - Out-of-container, Java object, bypassing much of Spring's magic
- Ideally tests should be out-of-container, but with as much Spring as possible

Spring MVC Testing

- Bootstraps most of Spring's MVC infrastructure so that unit and integration tests exercise the web application end-to-end
- Provides APIs for testing interesting parts of requests and responses

Spring MVC Testing

```
perform()
.andExpect()
jsonPath()
```

Using the API

- Designed, implemented, and tested, but can you actually use this API?
- Goals
  - Single URL
  - Link Traversal
  - Content Negotiation

Consuming the Game in Ruby

`client.rb`
Change URI scheme

Round Up

- API Design Matters
  - URIs represent resources, not actions
  - HTTP verbs are general, but can be used in ways that make anything possible
- Implementation isn't rocket science
  - Spring MVC
  - Spring HATEOAS
- East testing
  - Out-of-container, but full Spring Stack

Q & A

- `http://github.com/nebhale/devoxx-2013`
- Spring Boot
  - `http://projects.spring.io/spring-boot/`
- Spring Data JPA
  - `http://projects.spring.io/spring-data-jpa/`
- Spring Framework
  - `http://projects.spring.io/spring-framework/`
- Spring HATEOAS
  - `http://projects.spring.io/spring-hateoas/`