REST-ful API Design with Spring

Ben Hale, Pivotal

https://github.com/nebhale/spring-one-2013

What is REST?

- REpresentational State Transfer
- An architectural style for designing distributed systems
- Not a standard, but rather a set of constraints
  - Client/Server, 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

- URI's identify resources
- HTTP verbs describe a limited set of operations that can be used to manipulate a resource
  - GET
  - DELETE
  - PUT
  - POST
  - less used other verbs
- Headers help describe the messages

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

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

```
DELETE /games/1
```

PUT

- Requests that the entity passed by 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

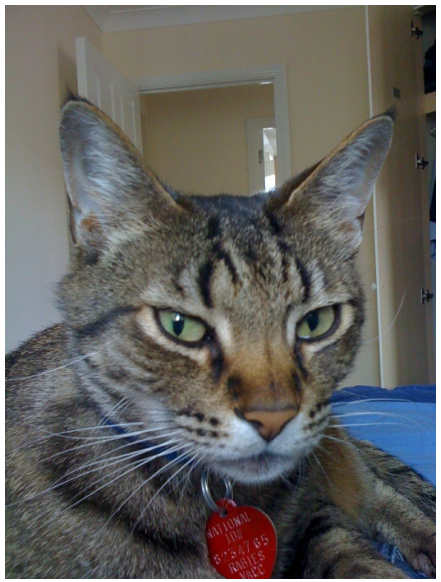
```
PUT /games/1/doors/2
{ “status”: “SELECTED” }
```

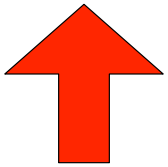
POST

- 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 are 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 us a resource identifier (URI) of the newly created game

```
POST /games
```

List the current state of all Doors

- Needs to return us a collection that represents the state of all doors in the game
- Design doesn't have 'Door 1', 'Door 2', '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 desired state

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

Open a Door

- 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

```
PUT /games/1/doors/3
{“status”: “OPEN”}
```

List the final state 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 MVC Implementation

Status Codes

- Status codes are an indicator of 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`
  - –Sever 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 a built-in knowledge of how to navigate and manipulate the model
- Instead 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 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 describe what it wants with `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 client can recognize them
  - `<link rel=”stylesheet” href=”...”/>`
  - `{“rel”: “doors”, “href”: “...”}`

Spring HATEOAS Implementations

Testing

- Testing of web APIs isn't easy
  - In container, end-to-end, string comparison, etc.
  - Out of container, Java objects, bypassing much of Spring's magic
- What we want is out of container, but with as much of Spring as we can get

Spring MVC Testing

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

Spring MVC Testing

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 Python

springone 2GX

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
- Easy testing
  - –Out-of-container, but full Spring Stack

Learn More. Stay Connected.

https://github.com/nebhale/spring-one-2013

Talk to us on Twitter: @springcentral
Find session replays on YouTube: spring.io/video