SOEN 387 Web-based Enterprise Application Design

Stuart Thiel

Concordia University
Department of Computer & Software Engineering

Fall, 2015

Outline

REST

Starting with Idempotence

- A property whereby subsequent occurrences of the same thing make no further changes beyond the first occurrence.
- What happens when an image loads up?
- Should that be Idempotent?
- What else should be Idempotent

HTTP Methods for Idempotence

- GET is suitable for this behaviour
- . . . what about the other methods
- PUT, DELETE, POST?
- If GET is idempotent, what does that imply about those others?

Is GET Idempotent?

- Actually, no
- That's just a convention, and a **VERY** good idea
- So how do we do it?

Idempotence

- A nonce is a good start for determining if a request has happened before
  - . . . unfortunate alternate meaning for that word
  - . . . also lots of effort for this
- Just make no changes on GET requests

Requesting Idempotence

- So we access certain pages for GET and other pages for the other methods?
- Maybe we can rethink how we make requests?

Introducing REST

- REpresentational State Transfer
  - Client-Server
  - Stateless
  - Cacheable
  - Layered Sytem
  - Code-on-Demand
  - Uniform Interface

Do We Need Stateless?

- Technically yes. . .
- People waffle/qualify or ignore Stateless, usually to work for stuff like logging in while discouraging heavy caching within the server's session

Do We Need Code-on-Demand?

- Technically yes. . .
- Code-on-Demand has been tacked on.
- This one is officially optional
- It's done, but I'm certain that wasn't always there, and feels like a bandaid to justify client-side programming that might otherwise not fit the purists definition of REST

Do We Need Uniform Interface?

- Technically yes. . .
- Uniform Interface is generally fine and we've all been doing it
- "Hypermedia As The Engine of Application State" (HATEOAS) suggests more than passing around ids, URLs are a convention
- Simpler to allow the client to know the fixed entry points to application which means it knows how to stick resource ids into URLs

Pure REST?

- It's like documentation, don't use it just to use it
- You have to take some time to understand why those rules exist, why people feel strongly about them
- But realize some people are zealots and in the end you have to be practical
- The main idea of REST is brilliant, but use it as conventient
- Understand that you will be shown to not understand things properly over time, so be ready to learn and adjust

Practical REST

- Each Domain Object has a URL pattern
- /SpaceTime/Player/1
  - GET views the player
  - POST updates the player
  - DELETE deletes the player
- /SpaceTime/Player/
  - GET lists all players
  - PUT could register players
- What else could we add?

HTTP Method Purists

- What's the problem? I use GET for reads and POST for everything else
- What's good about using the specific methods?
- Simple support for authorizing Use Cases based on role at the application level
- Matches well with CRUD approach to Resources

When HTTP Methods Fail

- How about accepting a Challenge in SpaceTime?
- HTTP Methods are dumb for this
- I and others advocate attaching verbs to the resource URL
- /SpaceTime/Invite/1/Decline
- Could that be done with a post on the Invite resource?
- Which is more flexible?
- Why not always use verbs instead of Methods?

AJAX and REST

- I know we're skipping most of the Front End
- You'll note the tests actually pull ids from responses
- AJAX does exactly the same, it just then draws pretty html/etc
- More importantly, RESTful interfaces have nice support for pulling small chunks of data
- . . . support for caching commonly requested small chunks
- e.g. upcoming calendar events json/xml can be a separate call that is cached
- GREATLY simplifies each request by breaking them up this way
- GREATLY simplifies server-load
- If you can handle the adventure that is server-side caching and need it, you get even more power