Designing RESTful Clients

Howard Dierking
http://codebetter.com/howarddierking

pluralsight
hardcore developer training

Overview

- RESTful clients are harder
- Revisiting the “contract”
- Clients and resources
- Clients representations
- Clients and messages
- Clients and hypermedia

RESTful Clients [can be] Harder

- Lack of tooling support
- More expansive view of contract
- Component interactions specified in human-readable rather than machine-readable documents
- More dynamic relationship between clients and servers

It's All About Tradeoffs

- **Heterogeny**
  - Seamless interoperability between connectors, regardless of language or platform
- **Scalability**
  - Complexity is scoped to a uniform interface and a single communication layer
- **Evolvability**
  - Clients and servers can evolve without making each other unstable
- **Visibility**
  - The state of the system can be determined examining messages
- **Reliability**
  - Rich compensation strategies can be built against a uniform interface and self-describing messages
- **Efficiency**
  - Local and intermediate caches can take load off of origin servers, enabling them to handle more clients
- **Performance**
  - Local and intermediate caches can improve the speed of getting a response
- **Manageability**
  - Layering architectural elements enables the overall system to grow in complexity without impacting an individual layer

Remember the Contract

- Identification of resources
- Manipulation through representations
- Self-descriptive messages
- Hypermedia as the engine of application state (HATEOAS)

Resource[s]

- **Know the resource identifier (URL in HTTP)**
- **Client should only need to know how to construct a single URL**
- **Client can bookmark other resource URLs, but as an optimization**
  - Should expect to return to the entry URL for an updated link if the resource can no longer be found

A REST API should be entered **with no prior knowledge beyond the initial URI** (bookmark) and set of **standardized media types** that are appropriate for the intended audience (i.e., expected to be understood by any client that might use the API). From that point on, **all application state transitions must be driven by client selection of server-provided choices** that are present in the received representations or implied by the user's manipulation of those representations. The transitions may be determined (or limited by) the client's knowledge of media types and resource communication mechanisms, both of which may be improved on-the-fly (e.g., code-on-demand).

~ Roy Fielding

Representations

- **Determining the processor for a response**
- **Ignore what you don’t understand**
- **Content negotiation**
  - Server driven
  - Agent driven

Example Versioning with Content Negotiation

Self-Describing Messages

- **Control Data**

Self-Describing Messages in HTTP

- **Methods**
- **Status codes**
- **Headers**

- **Control data**
  - Caching
  - HTTP status codes

Example: Optimistic Concurrency with ETags

- Implement a simple resource/representation cache with optimistic concurrency using Etags

Example: Client Retry

- **Implement a retry scenario for a 503 with a retry-after response header**

Hypermedia-Aware

- **Link types**
  - Embedded
  - Outgoing
  - Templated
  - Idempotent
  - Non-idempotent
- **Link relations**
- **Link Locations**

Example: Working with forms

Summary