INFO/CS 4302
Web Information Systems

FT 2012
Week 7: RESTful API Design

- Bernhard Haslhofer -
Plan for today...

• RESTful APIs – Architectural principles contd.

• REST API Design

• Real-world REST APIs (Groupwork)

• Questions, Housekeeping, ...
RESTFUL APIS – ARCHITECTURAL PRINCIPLES CONT'D.
The Resource-Oriented Architecture

• A set of design principles for building RESTful Web Services
  – Addressability
  – Uniform interface
  – Connectedness
  – Statelessness
Uniform Interface

• With HTTP we have all methods we need to manipulate Web resources (CRUD interface)
  – **Create** = POST (or PUT)
  – **Read** = GET
  – **Update** = PUT
  – **Delete** = DELETE
Uniform Interface

• **CREATE** a new resource with HTTP **POST**
Uniform Interface

- **READ** an existing resource with HTTP GET

```
GET /movies/1234 HTTP/1.1
Host: example.com

200 OK
<movie ... />
```

```
404 Not Found
```

```
500 Internal Server Error
```
Uniform Interface

- **UPDATE** an existing resource with HTTP **PUT**
Uniform Interface

• **DELETE** an existing resource with HTTP DELETE
The Resource-Oriented Architecture

• A set of design principles for building RESTful Web Services
  – Addressability
  – Uniform interface
  – Connectedness
  – Statelessness
Connectedness

• In RESTful services, resource representations are hypermedia
• Served documents contain not just data, but also links to other resources

HTTP/1.1 200 OK
Date: ...
Content-Type: application/xml

<?xml...>
<movie>
    <title>The Godfather</title>
    <synopsis>...</synopsis>
    <actor>http://example.com/actors/567</actor>
</movie>
The Resource-Oriented Architecture

- A set of design principles for building RESTful Web Services
  - Addressability
  - Uniform interface
  - Connectedness
  - Statelessness
Statelessness

- Statelessness = every HTTP request executes in complete isolation
- The request contains all the information necessary for the server to fulfill that request
- The server never relies on information from a previous request
  - if information is important (e.g., user-authentication), the client must send it again
Statelessness

• This constraint does not say “stateless applications”!
  – for many RESTful applications, state is essential
  – e.g., shopping carts
• It means to move state to clients or resources
• State in resources
  – the same for every client working with the service
  – when a client changes resource state other clients see this change as well
• State in clients (e.g., cookies)
  – specific to client and has to be maintained by each client
  – makes sense for maintaining session state (login / logout)
State in the Application

User Agent

Application

Shopping Carts

Session-Identified State

Resources

Products

Users

© Erik Wilde: http://dret.net/netdret/docs/rest-icwe2010/
Statelessness
Statelessness
Tools and Frameworks

• **Ruby on Rails** - a framework for building RESTful Web applications

• **Restlet** - framework for mapping REST concepts to Java classes

• **Django** - framework for building RESTful Web applications in Python

• **JAX-RC specification** ([http://jsr311.java.net/](http://jsr311.java.net/)) provides a Java API for RESTful Web Services over the HTTP protocol.

• **RESTEasy** ([http://www.jboss.org/resteasy/](http://www.jboss.org/resteasy/)) - JBoss project that provides various frameworks for building RESTful Web Services and RESTful Java applications. Fully certified JAX-RC implementation.
RESTFUL SERVICE DESIGN – IN BRIEF
2 RESTful Service Design

Cesare Pautasso
Faculty of Informatics
University of Lugano, Switzerland

c.pautasso@iieee.org
http://www.pautasso.info

ICWE 2010 Vienna

Web API Design
Crafting Interfaces that Developers Love
Design Methodology

• Identify and name resources to be exposed by the service
  – actors and movies

• Model relationships between resources that can be followed to get more details
  – an actor can play in several movies
  – several actors are playing in a movie

• Define “nice” URIs to address the resources
Design Methodology

• Map **HTTP verbs** to resources
  – e.g., GET movie, POST movie, etc...

• Design and document **resource representations**
  – we want to serve JSON (and XML)
  – the JSON mime-type is application/json

• Implement and deploy Web Service

• Test with cURL or browser developer tools
REST API Design Principles

• Who is the target audience?

• What are we trying to achieve with an API?
REST API Design Principles

• Make application developer as successful as possible

• Primary design principle: “…maximize developer productivity and success” (Mulloy)

• Keep simple things simple

• Take the developer’s point of view
Nouns are good; verbs are bad

• Simple and intuitive base URLs
  – /actors
  – /peopleplayingin80iesmovies

• 2 base URLs per resource
  – /actors (collection)
  – /actors/1234 (specific element in collection)

• Keep verbs out of your base URLs
  – /getAllActors
Nouns are good; verbs are bad

- Use HTTP verbs

<table>
<thead>
<tr>
<th>Resource</th>
<th>POST (create)</th>
<th>GET (read)</th>
<th>PUT (update)</th>
<th>DELETE (delete)</th>
</tr>
</thead>
<tbody>
<tr>
<td>/actors</td>
<td>Create a new actor</td>
<td>List actors</td>
<td>Bulk update actors</td>
<td>Delete all actors</td>
</tr>
<tr>
<td>/actors/1234</td>
<td>Error</td>
<td>Show actor 1234</td>
<td>If exists update actor 1234 Else: error</td>
<td>Delete actor 1234</td>
</tr>
</tbody>
</table>
Plural nouns and concrete names

• Using plural nouns might be more intuitive
  – /movies
  – /actors
• Singular nouns are OK, but avoid mixed model
  – /movie /actor
  – /movies /actor
• Prefer a managable number (12-24) of concrete entities over abstraction
  – /movie /actor /producer /cinema ...
  – /item
Simplify associations

• Relationships can be complex
  – movie -> actor -> pets -> ...
  – URL levels can become deep

• In most cases URL level shouldn’t be deeper than: resource/identifier/resource
  – /actor/1234/movies
  – /movies/1234/actors
Filtering

...sweep complexity behind the ?

/actors?gender=male&age=50
Handling Errors

• Use HTTP status codes
  – over 70 are defined; most APIs use only subset of 8-10

• Start by using
  – 200 OK (…everything worked)
  – 400 Bad Request (..the application did sth. wrong)
  – 500 Internal Server Error (...the API did sth. wrong)

• If you need more, add them
  – 201 Created, 304 Not Modified, 401 Unauthorized, 403 Forbidden, etc..
Handling Errors

• Make messages returned in HTTP body as verbose as possible

```json
{"developerMessage" : "Verbose, plain language description of the problem for the app developer with hints about how to fix it.",

"userMessage":"Pass this message on to the app user if needed.",

"errorCode" : 12345,

"more info": http://example.com/errors/12345"}
```
Versioning

• Never release an API without a version

• Suggested syntax
  – put version number in first path element
  – ‘v’ prefix
  – simple ordinal number
  – /v1/actors

• Maintain at least one version back
Partial responses

• Sometimes you don’t need the entire representation
• Save bandwidth

• Add optional fields in a comma-delimited list
  – /movies?fields=title
Pagination

• It’s almost always a bad idea to return every available resource

• Use limit and offset to allow pagination
  – /movies?limit=20&offset=0

• Include metadata about total number of resources in representation
Actions not dealing with resources

• Certain API calls don’t send resource responses
  – calculate
  – translate
  – convert

• Use verbs and make it clear in the docs

• /convert?from=EUR&to=USD&amount=100
Multiple Formats

• Support for more than one format is recommended
  – JSON default format; XML secondary
  – mapping can be automated

• „Pure“ RESTful approach
  – Accept: application/xml in HTTP Header

• Pragmatic approach
  – /actors.json, /actors.xml
  – /actors/1234.json, /actors/1234.xml

• Mixed approach
  – /actors -> content negotiated depending on Accept header
  – /actors.json -> direct format-specific access
Search

• Global search (across resources)
  – /search?q=godfather

• Scoped search
  – /actors/1234/movies?q=godfather

• Formatted results
  – /search.xml?q=godfather
API subdomain

- Consolidate all API requests under one API subdomain
  - api.example.com
- Developer portal (documentation, etc...)
  - developer.example.com
- Web redirects
  - e.g., redirect browser requests to developer portal
REAL-WORLD REST APIS
Instructions

• Form groups of 5 and choose one Web API
• Answer the following questions (15 min):
  – Which resources are exposed and how are they named?
  – Which HTTP verbs are used and for what purpose?
  – How is error handling implemented? Which HTTP error codes are used?
  – Is filtering, pagination, and search supported? If yes, how?
  – How RESTful is the Web API?
• Create summary slides at: http://bit.ly/info4302-existing-apis
• Be prepared to talk about your findings
Outlook

Social Networking Sites as Walled Gardens by David Simonds
Outlook
Readings

• Tutorial Design Principles, Patterns and Emerging Technologies for RESTful Web Services (Cesare Pautasso and Erik Wilde): http://dret.net/netdret/docs/rest-icwe2010/

• Web API Design – Crafting Interfaces that Developers Love: http://apigee.com/about/api-best-practices