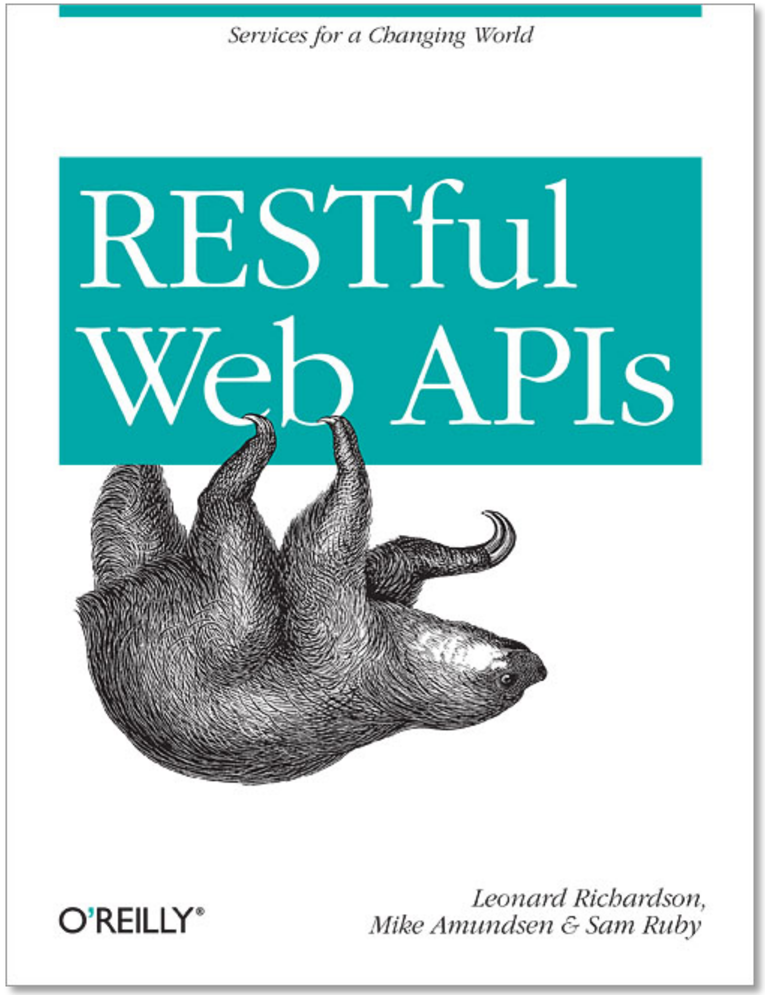
The Web as an Application Platform (1DV527)

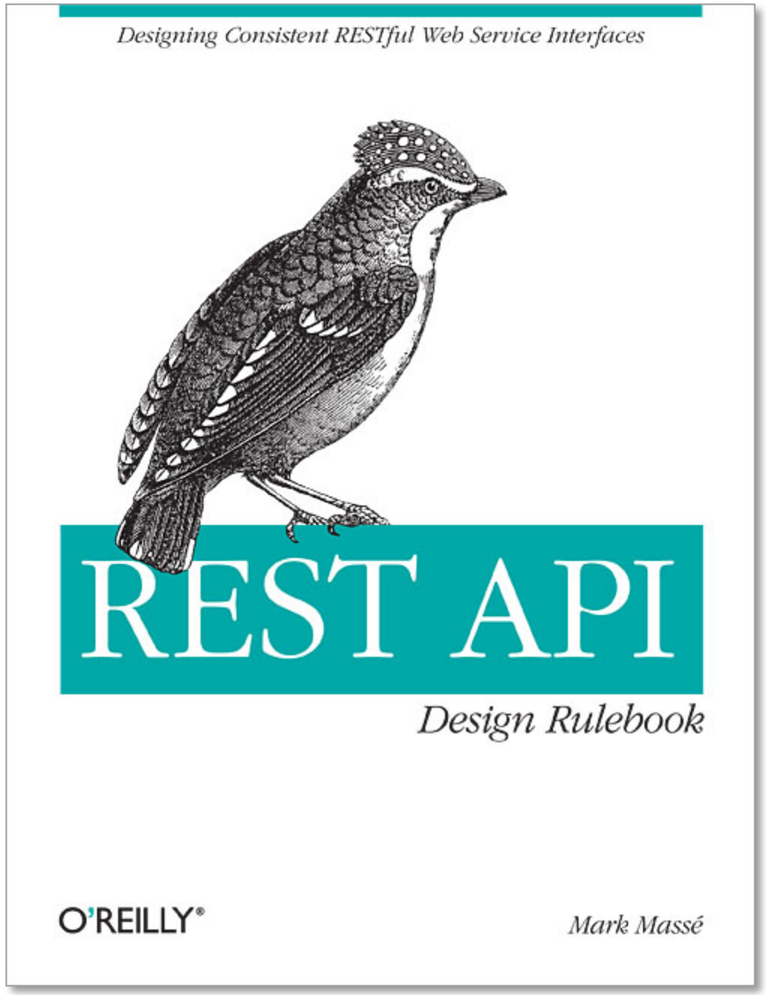
Web APIs – part I

Francis Palma, Ph.D.
francis.palma@lnu.se
Senior Lecturer
Department of Computer Science and Media Technology
Linnaeus University

Linnéuniversitetet

Books and References


Services for a Changing World
RESTful Web APIs
O'REILLY
Leonard Richardson, Mike Amundsen & Sam Ruby



Module Outline

- Web APIs – part I
  - Introduction on REST APIs
  - Identifier Design with URIs
  - Interaction Design with HTTP

- Web APIs – part II
  - Metadata Design
  - Representation Design
  - Client Concerns

Web APIs – part I

- Introduction on REST APIs
- Identifier Design with URIs
- Interaction Design with HTTP

Web APIs – part I

- Introduction on REST APIs
- Identifier Design with URIs
- Interaction Design with HTTP

Introduction on REST APIs

"The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents."

- TimBL

Introduction on REST APIs

- **Web** - European Organization for Nuclear Research, Geneva, Switzerland.
- In December 1990, Tim Berners-Lee started a non-profit software project "WorldWideWeb"
- Berners-Lee invented and implemented:
  - the Uniform Resource Identifier (URI)
  - the HyperText Transfer Protocol (HTTP)
  - the HyperText Mark-up Language (HTML)
  - the first Web server: http://info.cern.ch/
  - the first web browser "WorldWideWeb" (later known as "Nexus")
  - the first WYSIWYG HTML editor

Introduction on REST APIs

- **Web Architecture**
  - Scalability problem
- The scalability of Web is governed by a set of key ***constraints*** (Roy Fielding, 1993).
- The constraints, grouped into 6 categories, referred to as the ***Web's architectural style***.
  - Client-server
  - Uniform interface
  - Layered system
  - Cache
  - Stateless
  - Code-on-demand

Introduction on REST APIs

- **Client–Server**: Web is a client-server based system, separate roles, implemented and deployed independently (separation of concerns).

- **Uniform Interface**: Interactions among the Web components depend on the uniformity of their interfaces. Four (sub)constraints:
  - Identification of ***resources***
  - Manipulation of resources through ***representations***
  - Self-descriptive ***messages***
  - ***Hypermedia*** as the engine of application state (HATEOAS)

- **Layered System:** Enable network-based intermediaries, e.g., proxies and gateways to be deployed between clients and server using the Web's uniform interface. Example uses: enforcement of security, response caching, and load balancing.

Introduction on REST APIs

- **Cache**: Web servers declare the ***cacheability*** of each response's data, thus, reduce the latency, increase the availability and reliability of an application, and control the load of a Web server.

- **Stateless**: Web servers are not required to memorize the state of its client applications, thus, each client must include contextual information in each interaction with the Web server.

- **Code-On-Demand:** Web servers can transfer executable programs (e.g., scripts or plug-ins) to clients. Example include Web browser-hosted technologies like Java applets, JavaScript, and Flash.

Some Concepts

- Architectural constraint
- Cache
- Entity body
- Entity headers
- Representation
- Resource
- Resource identifier
- Resource model
- Resource state representation
- Request message
- Response message
- Stateless
- Uniform interface
- Uniform resource identifier

Web APIs – part I

- Introduction on REST APIs
- Identifier Design with URIs
- Interaction Design with HTTP

Identifier Design with URIs

- REST APIs use Uniform Resource Identifiers (URIs) to address resources.
- http://api.example.restapi.org/68dd0-a9d3-11e0-9f1c-0800200c9a66
- http://api.example.restapi.org/france/paris/louvre/leonardo-da-vinci/mona-lisa
- URI format: URI = scheme "://" authority "/" path ["?" query] ["#" fragment]
  - authority = [userinfo@]host[:port]
  - Defined in RFC 3986
- $\underbrace{\texttt{https}}_{\textbf{scheme}}\texttt{://}\underbrace{\texttt{www.google.com}}_{\textbf{authority}}\texttt{/}\underbrace{\texttt{search}}_{\textbf{path}}\texttt{?}\underbrace{\texttt{client=safari\&rls=en\&q=lnu\&ie=UTF-8\&oe=UTF-8}}_{\textbf{query}}$

Rules of Identifier Design

- Example:

http://api.canvas.restapi.org/shapes/polygons/quadrilaterals/squares

Rule 1: Forward slash separator (/) must be used to indicate a hierarchical relationship

Rules of Identifier Design

- Example:

http://api.canvas.restapi.org/shapes/

http://api.canvas.restapi.org/shapes

Rule 2: A trailing forward slash (/) should not be included in URIs

Rules of Identifier Design

- Example:

http://api.example.restapi.org/blogs/mark-masse/entries/this-is-my-first-post

Rule 3: Hyphens (-) should be used to improve the readability of URIs

Rules of Identifier Design

- Example:

http://api.example.restapi.org/blogs/mark_masse/entries/this_is_my_first_post

http://api.example.restapi.org/blogs/mark_masse/entries/this_is_my_first_post

Rule 4: Underscores (_) should not be used in URIs

Rules of Identifier Design

- Example:

1. http://api.example.restapi.org/my-folder/my-doc
2. HTTP://API.EXAMPLE.RESTAPI.ORG/my-folder/my-doc
3. http://api.example.restapi.org/My-Folder/my-doc

Rule 5: Lowercase letters should be preferred in URI paths

- RFC 3986 defines URIs as case sensitive

Rules of Identifier Design

- Example:

1. http://api.college.restapi.org/students/3248234/transcripts/2005/fall.json
2. http://api.college.restapi.org/students/3248234/transcripts/2005/fall

Rule 6: File extensions should not be included in URIs

Rules of Authority Design

Rule 7: Consistent subdomain names should be used for your APIs

- Example:

$$\text{http://}\underbrace{\text{api}}_{\textbf{sub domain}}.\underbrace{\text{soccer.restapi.org}}_{\textbf{service owner}}$$

Rules of Authority Design

Rule 8: Consistent subdomain names should be used for your client developer portal

- Example:

http://developer.soccer.restapi.org

Resource Modelling

- The URI path conveys a REST API's resource model where each forward slash separated path segment corresponding to a unique resource within the model's hierarchy.

- Example:

  http://api.soccer.restapi.org/leagues/seattle/teams/trebuchet

- Resource modeling establishes API's key concepts.
  - Similar to the data modeling for a relational database schema or modeling of an object-oriented system.

Resource Archetypes

- A REST API is composed of four distinct resource archetypes: *document*, *collection*, *store*, and *controller*.

- **Document:** A document resource is a singular concept, i.e., an object instance or database record.

- Example:

  http://api.soccer.restapi.org/leagues/seattle

  http://api.soccer.restapi.org/leagues/seattle/teams/trebuchet

  http://api.soccer.restapi.org/leagues/seattle/teams/trebuchet/players/mike

Resource Archetypes

- **Collection:** A collection resource is a server-managed directory of resources.
  - Clients may propose to add new resources to collection. But, the collection decide to create a new resource.

  Example:
  http://api.soccer.restapi.org/leagues/seattle/teams
  http://api.soccer.restapi.org/leagues/seattle/teams/trebuchet/players

- **Store:** A store is a client-managed resource repository.
  - On their own, stores do not create new resources, i.e., a store never generate new URIs.

  Example: PUT /users/1234/favorites/alonso

- **Controller:** A controller resource models a procedural concept, i.e., executable functions, with parameters and return values; inputs and outputs.
  - Controller names appear as the last segment in a URI path, with no hierarchical child resources.

  Example: POST /alerts/245743/**resend**

URI Path Design

Rule 9: A singular noun should be used for document names

- Example:

http://api.soccer.restapi.org/leagues/seattle/teams/trebuchet/players/**claudio**

URI Path Design

Rule 10: A plural noun should be used for collection names

- Example:

  http://api.soccer.restapi.org/leagues/seattle/teams/trebuchet/**players**

URI Path Design

Rule 11: A plural noun should be used for store names

- Example:

http://api.music.restapi.org/artists/mikemassedotcom/**playlists**

URI Path Design

Rule 12: A verb or verb phrase should be used for controller names

Example:

1. http://api.college.restapi.org/students/morgan/**register**
2. http://api.example.restapi.org/lists/4324/**dedupe**
3. http://api.ognom.restapi.org/dbs/**reindex**
4. http://api.build.restapi.org/qa/nightly/**runTestSuite**

URI Path Design

Rule 13: Variable path segments may be substituted with identity-based values

- https://tools.ietf.org/html/rfc6570

Example 1:

http://api.soccer.restapi.org/leagues/{leagueId}/teams/{teamId}/players/{playerId}

http://api.soccer.restapi.org/leagues/seattle/teams/trebuchet/players/21

Example 2:

http://api.soccer.restapi.org/games/{gameId}

http://api.soccer.restapi.org/games/3fd65a60-cb8b-11e0-9572-0800200c9a66

URI Path Design

Examples:

```
GET /deleteUser?id=1234
GET /deleteUser/1234
DELETE /deleteUser/1234
POST /users/1234/delete
```

Good Example:

```
DELETE /users/1234
```

Rule 14: CRUD function names should not be used in URIs

URI Query Design

- URI = scheme "://" authority "/" path ["?" query] ["#" fragment]

Example

http://api.college.restapi.org/students/morgan/send-sms

http://api.college.restapi.org/students/morgan/send-sms?text=hello

URI Query Design

Rule 15: The query component of a URI may be used to filter collections or stores

Example:

```
GET /users
GET /users?role=admin
```

URI Query Design

Rule 16: The query component of a URI should be used to <u>paginate</u> collection or store results

Example:

GET /users?pageSize=25&pageStartIndex=50

Web APIs – part I

- Introduction on REST APIs
- Identifier Design with URIs
- Interaction Design with HTTP

Interaction Design with HTTP: Request Methods

- REST APIs embrace all aspects of the HyperText Transfer Protocol 1.1 including its request methods, response codes, and message headers.

- Request Methods
  - **GET** retrieves a representation of a resource state
  - **HEAD** retrieves the metadata associated with a resource state
  - **PUT** adds a new resource to a store or update a resource
  - **DELETE** removes a resource from its parent
  - **POST** creates a new resource within a collection and execute controllers
  - **CONNECT** establishes a tunnel to the server identified by the target resource
  - **OPTIONS** describes the communication options for the target resource
  - **TRACE** performs a message loop-back test along the path to the target resource
  - **PATCH** applies partial modifications to a resource

Interaction Design with HTTP

Rule 17: GET and POST must not be used to **tunnel** other request methods

- abuse of HTTP that misrepresents a message's intent and undermines the protocol's transparency
- make proper use of the HTTP methods

Interaction Design with HTTP

Rule 18: GET must be used to retrieve a representation of a resource

- client’s GET request message may contain <u>only headers but no body</u>

- Example:

```
curl* -v http://api.example.restapi.org/greeting
```

*https://curl.haxx.se

Interaction Design with HTTP

Rule 19: HEAD should be used to retrieve response headers

- without a body
- check whether a resource exists or to read its metadata

- Example:

```
curl --head http://api.example.restapi.org/greeting
```

*https://curl.haxx.se

Interaction Design with HTTP

Rule 20: PUT must be used to update mutable resources

- PUT request message may include a body that reflects the desired changes

Interaction Design with HTTP

Rule 21: POST must be used to create a new resource in a collection

- POST request's body contains the suggested state representation of the new resource to be added to the server-owned collection
- analogous to "posting" a new message on a bulletin board

- Example:

POST /leagues/seattle/teams/trebuchet/players

Interaction Design with HTTP

Rule 22: POST must be used to execute controllers

- invoke the function-oriented controller resources
- may include both headers and a body as inputs
- trigger operations that cannot be mapped to core HTTP methods
- *unsafe* and *non-idempotent*

- Example:

  POST /alerts/245743/resend

Interaction Design with HTTP

HTTP Method	Idempotent	Safe
OPTIONS	Y	Y
GET	Y	Y
HEAD	Y	Y
PUT	Y	N
POST	N	N
DELETE	Y	N
PATCH	N	N

Interaction Design with HTTP

Rule 23: DELETE must be used to remove a resource from its parent

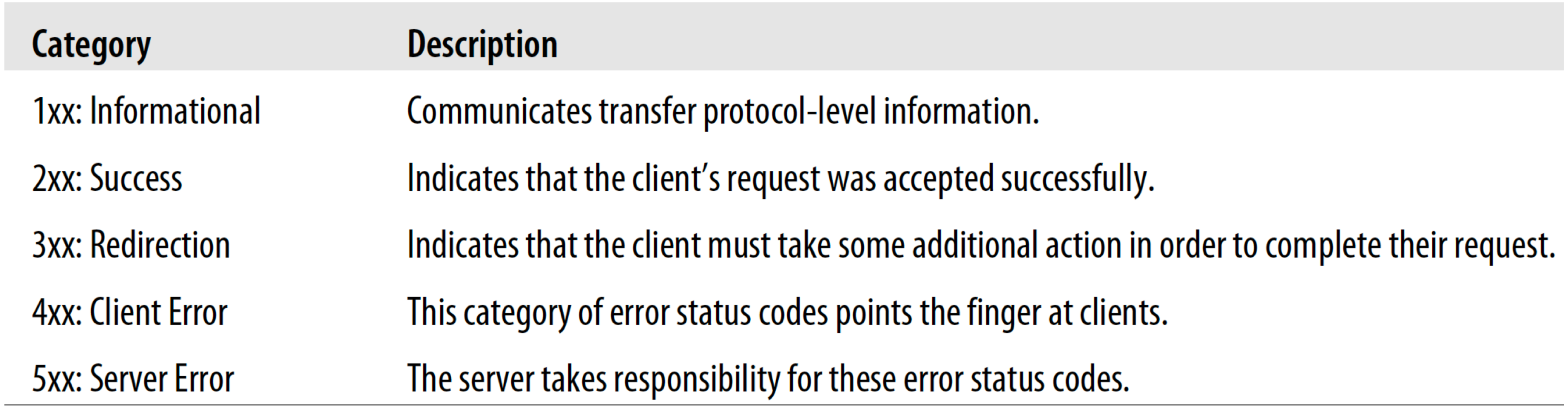
- the resource should no longer be found by clients
- any future attempt to retrieve the resource’s state must result in a 404 (“Not Found”)

- Example:

DELETE /accounts/4ef2d5d0-cb7e-11e0-9572-0800200c9a66/buckets/objects/4321

Interaction Design with HTTP: Response Status Codes

- Status-Line := HTTP-Version SP Status-Code SP Reason-Phrase CRLF
- Example: HTTP/1.1 200 OK



Category	Description
1xx: Informational	Communicates transfer protocol-level information.
2xx: Success	Indicates that the client's request was accepted successfully.
3xx: Redirection	Indicates that the client must take some additional action in order to complete their request.
4xx: Client Error	This category of error status codes points the finger at clients.
5xx: Server Error	The server takes responsibility for these error status codes.

Interaction Design with HTTP

Rule Set 24:

- 200 (“OK”) should be used to indicate nonspecific success
- 200 (“OK”) **must not** be used to communicate **errors** in the response body
- 201 (“Created”) must be used to indicate successful resource creation
- 202 (“Accepted”) must be used to indicate successful start of an asynchronous action
- 204 (“No Content”) should be used when the response body is intentionally empty

Interaction Design with HTTP

Rule Set 25:

- 301 (“Moved Permanently”) should be used to relocate resources
- 303 (“See Other”) should be used to refer the client to a different URI
- 307 (“Temporary Redirect”) should be used to tell clients to resubmit the request to another URI

Interaction Design with HTTP

Rule Set 26:

- **400 ("Bad Request")** may be used to indicate nonspecific failure
- 401 ("Unauthorized") must be used when there is a problem with the client's credentials
- 403 ("Forbidden") should be used to forbid access regardless of authorization state
- **404 ("Not Found")** must be used when a client's URI cannot be mapped to a resource
- 405 ("Method Not Allowed") must be used when the HTTP method is not supported
- 406 ("Not Acceptable") must be used when the requested media type cannot be served
- 415 ("Unsupported Media Type") must be used when the media type of a request's payload cannot be processed

Interaction Design with HTTP

Rule Set 27:

- **500 Internal Server Error**, should be used to indicate API malfunction
- 501 Syntax error in parameters or arguments
- 502 Command not implemented
- 503 Bad sequence of commands
- 504 Command not implemented for that parameter
- 530 Not logged in

So, what have we learned so far?

Quality URI **design** is crucial (Rules 1—8)

Designers should be careful in using **nouns and verbs** for URI path design (Rules 9—16)

HTTP methods should be used according to their **semantics** (Rules 17—23)

Communication between client and server should be meaningful and correct (Rules 24—27)

Linnéuniversitetet

Questions?

Next Lecture on Wednesday Feb 5th 13:15

Web APIs – part II