Linnéuniversitetet

The Web as an Application Platform (1DV527)

Web APIs – part II

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

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
  - Final Thoughts

Linnéuniversitetet

Web APIs – part II

- Metadata Design
- Representation Design
- Client Concerns
- Final Thoughts

Metadata Design: HTTP Headers

- Metadata is provided through the entity headers in HTTP requests and responses
  - Three categories of information:
    1. Information about a requested resource
    2. About the representation carried by the message
    3. About intermediary caches

Rules on Metadata Design

- **Rule 1:** Content-Type must be used
  - Clients and servers rely on this header's value to tell them how to process the sequence of bytes in a message's body.
  - Format: Content-Type := type "/" subtype *[";" parameter]

- Example types:
  - application: application/javascript, application/pdf, application/json, application/xml, application/zip
  - audio: audio/mpeg, audio/x-wav
  - image: image/gif, image/jpeg, image/png, image/tiff, image/svg+xml
  - text: text/css, text/csv, text/html, text/plain, text/xml
  - video: video/mpeg, video/mp4, video/quicktime, etc.

Rules on Metadata Design

- **Rule 2:** Content-Length should be used
  - Gives the size of the entity-body in bytes
  - Good for client to know if it has read the correct number of bytes
  - Client can make a HEAD request to find the size of entity-body
  - Format: Content-Length := "Content-Length" ":" 1*DIGIT

Rules on Metadata Design

- **Rule 3:** Last-Modified should be used in responses
  - Applies to response messages only
  - Value is a timestamp, i.e., the most recent time the state of a resource changed
  - Should always be supplied in response to GET requests

Rules on Metadata Design

- **Rule 4:** ETag should be used in responses
  - An identifier for a specific version of a resource (i.e., fingerprints)
  - If the resource at a given URL changes, a new ETag value must be generated
  - Always send in response to GET requests

- Format: ETag: "<etag_value>”

- If-None-Match *vs.* If-Match request header
  - With ETag and If-Match headers, it is possible to detect mid-air edit collisions

Rules on Metadata Design

- **Rule 5:** Location must be used to specify the URI of a newly created resource
  - The value is a URI that identifies a newly created resource

- Format: Location: <url>

Rules on Metadata Design

- **Rule 6:** Cache-Control, Expires, and Date response headers should be used to encourage caching
  - Caching is an essential feature of HTTP
  - Can be done in the API's server network, content delivery networks (CDNs), or the client's network

- Example: Cache-Control: max-age=60, must-revalidate

- For HTTP/1.0: Date: Tue, 15 Jan 2020 08:00:00 GMT
  Expires: Thu, 01 Feb 2020 16:00:00 GMT

Rules on Metadata Design

- **Rule 7:** Cache-Control, Expires, and Pragma response headers may be used to discourage caching
  - If a response must not cached, add Cache-Control headers with the value no-cache and no-store

- Example: Cache-Control: no-cache

- For HTTP/1.0 Pragma: no-cache and Expires: 0

Rules on Metadata Design

- **Rule 8:** Caching should be encouraged
  - *no-cache* prevents any cache from serving cached responses
  - Instead, use a small value of max-age (allows clients to fetch cached copies for at least a short while)

Rules on Metadata Design

- **Rule 9:** Expiration caching headers should be used with 200 (“OK”) responses
  - Set expiration caching headers in responses to successful GET and HEAD requests

Rules on Metadata Design

- **Rule 10:** Custom HTTP headers must not be used to change the behavior of HTTP methods
  - Use custom headers for informational purposes only
  - Implement clients and servers so that they do not fail when they do not find expected custom headers
  - If the information is required for the correct interpretation of the request/response, put it in the body

Registered Media Types

- Internet Assigned Numbers Authority (IANA)
  - http://www.iana.org/assignments/media-types
  - Registries included below
    - application
    - audio
    - font
    - example
    - image
    - message
    - model
    - multipart
    - text
    - video
  - Vendor-Specific Media Types: use subtype prefix “vnd” and owned or controlled by a “vendor”.

Rules on Media Type Design

- **Rule 11:** Application-specific media types should be used
  - Example: Response from GET http://api.soccer.restapi.org/players/2113 is in JSON
  - The Content-Type header field has value "*application/json*" → syntax of body content
  - No information on the semantics and structure of the player representation
    - Use a descriptive media type: application/wrml
    - Example: application/wrml;

      format="http://api.formats.wrml.org/application/json";

      schema="http://api.schemas.wrml.org/soccer/Player"

Rules on Media Type Design

- **Rule 12:** Media type negotiation should be supported when multiple representations are available
  - Allow clients to negotiate for a given format and schema by submitting an Accept header with the desired media type.
  - Example: Accept: application/wrml;
    format="http://api.formats.wrml.org/text/html";
    schema="http://api.schemas.wrml.org/soccer/Team"

Rules on Media Type Design

- **Rule 13:** Media type selection using a query parameter may be supported
  - Support media type selection via the accept query parameter with a value
  - Mirrors that of the Accept HTTP request header
  - Example:

    GET /bookmarks/mikemassedotcom?accept=application/xml

  - Note: Never include .xml, .json, or .txt to the URI's path

Linnéuniversitetet

Web APIs – part II

- Metadata Design
- Representation Design
- Client Concerns
- Final Thoughts

Representation Design

- Entity body conveys the state of a requested resource.
  - Often a text-based format, e.g., XML and JSON

- **Rule 14**: JSON should be supported for resource representation
  - Other than the standard format for a given resource type (e.g., image/jpeg for image resources), you should use the JSON format to structure its information.

Representation Design

- **Rule 15**: JSON must be well-formed
  - The JSON object is an unordered set of name-value pairs.
  - Define names as strings always surrounded by double quotes.
  - Example:

```
{
    "firstName" : "Osvaldo",
    "lastName" : "Alonso",
    "firstNamePronunciation" : "ahs-VAHL-doe",
    "number" : 6,
    "birthDate" : "1985-11-11"
}
```

Representation Design

- **Rule 16**: XML and other formats may optionally be used for resource representation
  - Clients should express their desired representation using media type negotiation

Hypermedia Representation Design

- Like the Web's HTML-based hyperlinks and forms, REST APIs use hypermedia within representations.
  - Links indicate the associations and actions available for a given resource
  - API clients can programmatically navigate using a uniform link structure

Hypermedia Representation Design

- **Rule 17**: A consistent form should be used to represent links
  - A single link does not typically stand alone as a request or response message body's content.
  - Example:

```
{
    "href" : Text <constrained by URI or URI Template syntax>,  ❶
    "rel"  : Text <constrained by URI syntax>,  ❷
    "requestTypes"  : Array <constrained to contain media type text elements>, ❸
    "responseTypes" : Array <constrained to contain media type text elements>, ❹
    "title" : Text   ❺
}
```

```
{
    "href" : "http://api.soccer.restapi.org/players/2113",
    "rel"  : "http://api.relations.wrml.org/common/self"
}
```

Hypermedia Representation Design

- **Rule 18**: A consistent form should be used to advertise links
  - An API must offer clients a consistent way to easily discover the available links within a representation.
  - Representations should include a structure -- *links* -- to contain all of the links that are available in the resource's current state.
  - Example:

```
{
  "firstName" : "Osvaldo",
  "lastName" : "Alonso",
  "links" : {   ❶
    "self" : {
      "href" : "http://api.soccer.restapi.org/players/2113",
      "rel"  : "http://api.relations.wrml.org/common/self"
    },
    "parent" : {
      "href" : "http://api.soccer.restapi.org/players",
      "rel"  : "http://api.relations.wrml.org/common/parent"
    },
    "team" : {  ❷
      "href" : "http://api.soccer.restapi.org/teams/seattle",
      "rel"  : "http://api.relations.wrml.org/soccer/team"
    },
    "addToFavorites" : {
      "href" : "http://api.soccer.restapi.org/users/42/favorites/{name}",  ❸
      "rel"  : "http://api.relations.wrml.org/common/addToFavorites"
    }
  }
}
```

Hypermedia Representation Design

- **Rule 19**: A self link should be included in response message body representations
  - A response message body should include a link named *self*.
  - In self link relation, the *href* value identifies a resource equivalent to the containing resource.

Linnéuniversitetet

Web APIs – part II

- Metadata Design
- Representation Design
- Client Concerns
- Final Thoughts

Linnéuniversitetet

Client Concerns

- There are a set of REST API design principles to address common client concerns.
  - Versioning
  - Security
  - Response Representation Composition
  - Processing Hypermedia
  - JavaScript Clients

Client Concerns: Versioning

- A REST API is composed of interlinked resources, known as resource model.
  - Version of each resource is conveyed through its representational form and state.

- **Rule 20:** New URIs should be used to introduce new concepts
  - Each character in a resource's URI contributes to its identity
  - The version of a REST API, or its resources, should not be signified in a URI
  - Example:

  http://www.restful.com/api/v2/france/paris/meauseums/louvre/arts/mona-lisa

  vs.

  http://www.restful.com/api/france/paris/meauseums/louvre/arts/mona-lisa

Client Concerns: Versioning

- **Rule 21:** Entity tags should be used to manage representational state versions
    - Rule 4: Rule: ETag should be used in responses

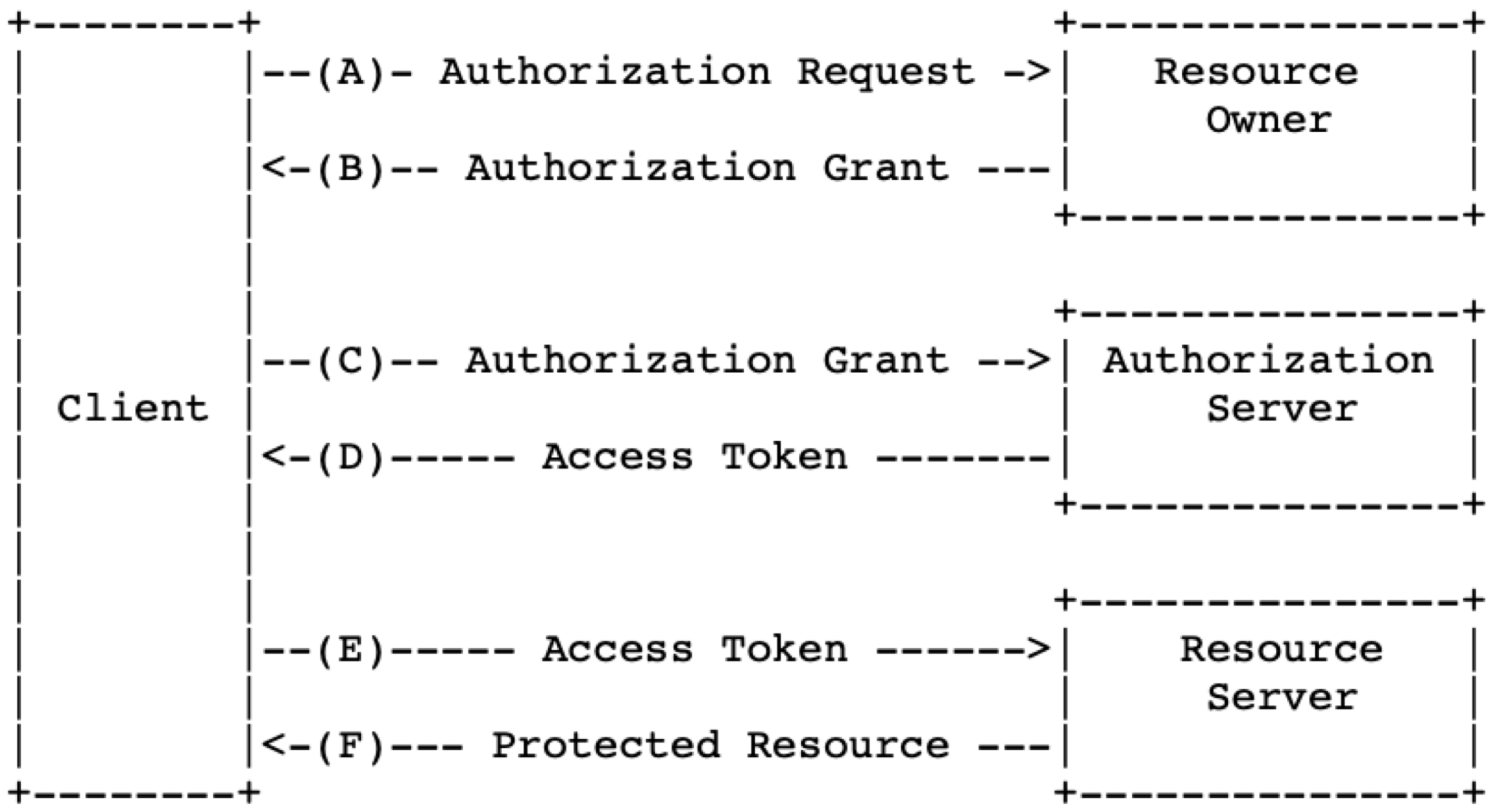
- Entity tag values for each resource are REST API's most fine-grained versioning system.

Client Concerns: Security

- Protecting of a REST API's sensitive resources

- **Rule 22:** OAuth may be used to protect resources
  - OAuth (Open Authorization) is an open standard to provide secure authorization using a consistent approach for all clients
  - Allows users to share their private resources (e.g., photos or contacts from facebook) with another site without having to disclose their confidential username or password.

Client Concerns: Security: OAuth Protocol Flow



Client Concerns: Response Representation Composition

- Enable clients to tune responses to meet their needs, while allowing the REST API to maintain a consistent resource model design.

- **Rule 23:** The query component of a URI should be used to support partial responses
  - Use the query component to trim response data with the fields parameter.

```
# Request
GET /students/morgan?fields=(firstName, birthDate) HTTP/1.1  ❶
Host: api.college.restapi.org
```

```
# Response
HTTP/1.1 200 OK
Content-Type: application/wrml;
              format="http://api.formats.wrml.org/application/json";
              schema="http://api.schemas.wrml.org/college/Student";
              fields="(birthDate, firstName)"  ❷

{
    "firstName" : "Morgan",  ❸
    "birthDate" : "1992-07-31"
}
```

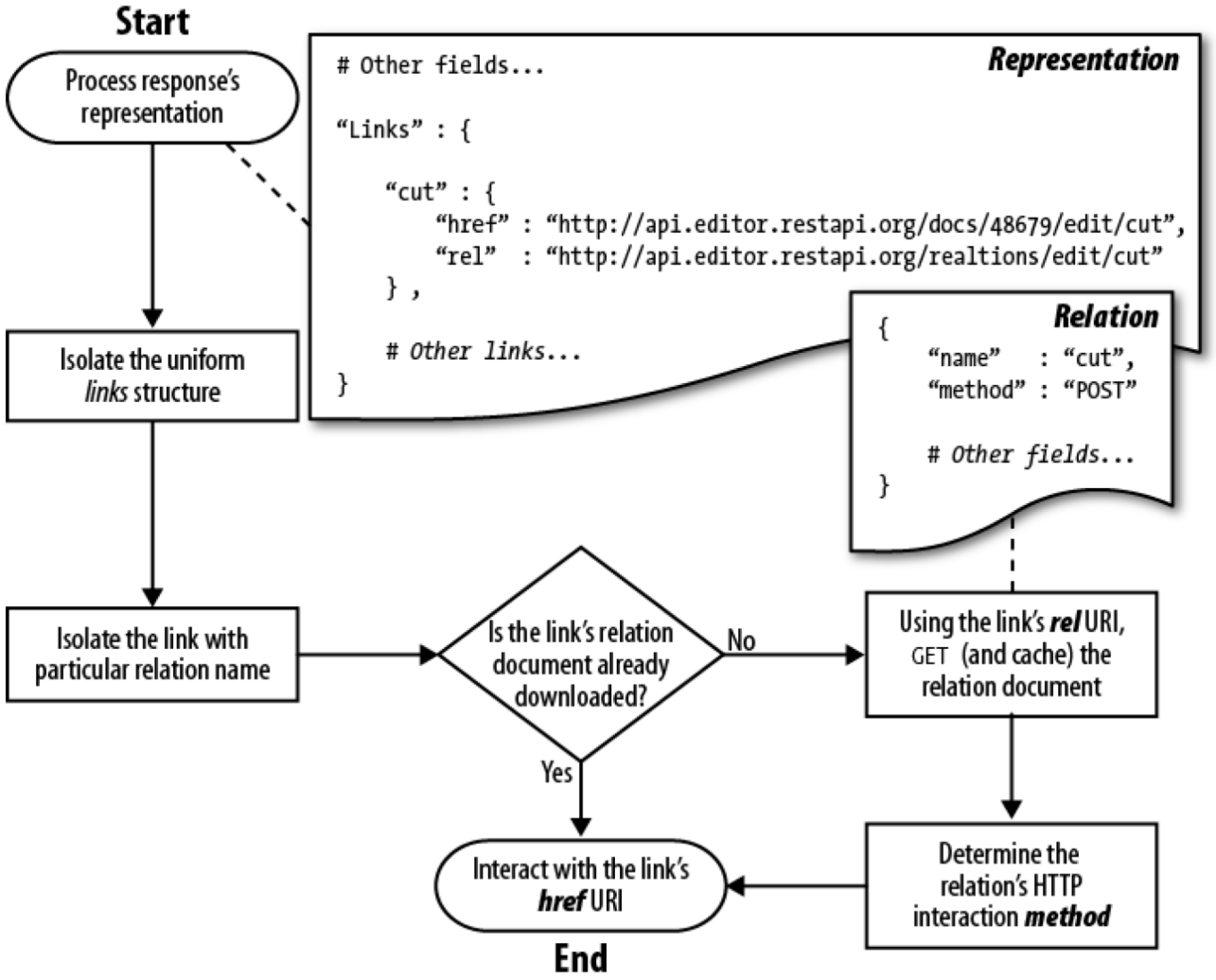
Client Concerns: Response Representation Composition

- How the fields query parameter can be used to specify a set of fields that are unwanted

```
# Request
GET /students/morgan?fields=!(address,schedule!(wednesday, friday))  HTTP/1.1 ❶
Host: api.college.restapi.org
```

```
# Response
HTTP/1.1 200 OK
Content-Type: application/wrml;
              format="http://api.formats.wrml.org/application/json";
              schema="http://api.schemas.wrml.org/college/Student";
              fields="!(address, schedule!(friday, wednesday))"  ❷

{
    "firstName" : "Morgan",  ❸
    "birthDate" : "1992-07-31",
    "schedule"  : {
        "monday" : {
            "links" : {
                "firstClass" : {
                    "href" : "http://api.college.restapi.org/classes/math-202",
                    "rel"  : "http://api.relations.wrml.org/college/firstClass"
                },

                # Daily schedule's other links...
            }
        },

        # Schedule's other fields (except friday and wednesday)...
    },

    # Student's other fields (except address)...

    "links" : {
        # Student's links...
    }
}
```

Client Concerns: Processing Hypermedia



Web APIs – part II

- Metadata Design
- Representation Design
- Client Concerns
- Final Thoughts

Final Thoughts

- Microsoft REST API Guidelines
  - https://github.com/Microsoft/api-guidelines

URL Structure

- Humans SHOULD be able to easily read and construct URLs.
  - facilitates discovery and eases adoption

- Good Example:

https://api.contoso.com/v1.0/people/jdoe@contoso.com/inbox

- Bad Example:

https://api.contoso.com/EWS/OData/Users('jdoe@microsoft.com')/Folders('AAMkADdiYzI1MjUzLTk4MjQtNDQ1Yy05YjJkLWNlMzMzYmIzNTY0MwAuAAAAAACzMsPHYH6HQoSwfdpDx-2bAQCXhUk6PC1dS7AERFluCgBfAABo58UAAA=')

URL Length

- HTTP 1.1 message format, defined in RFC 7230, has no length limit on the Request Line
  - 2000 characters
  - Search engines like URLs < 2048 chars
  - IE's maximum URL length is 2083 chars

Resource Creation with POST

- POST operations SHOULD support the Location response header to specify the location of any created resource that was not explicitly named, via the Location header.

- Example:

  POST http://api.example.com/account1/servers

- Response

  201 Created

  Location: http://api.example.com/account1/servers/server321

Standard Request and Response Headers

Header	Type
Authorization	String
Date	Date
Accept	Content type
Accept-Encoding	Gzip, deflate
Accept-Language	"en", "es", etc.
Accept-Charset	Charset type like "UTF-8"
Content-Type	Content type
Prefer	return=minimal, return=representation
If-Match, If-None-Match, If-Range	String

Response Header	Required
Date	All responses
Content-Type	All responses
Content-Encoding	All responses
Preference-Applied	When specified in request
ETag	When the requested resource has an Entity Tag

Response Formats

- Web-based communications highly rely on JSON for its lightweight nature and easy to consume via JavaScript-based clients.
  - JSON property names should be camelCased
  - Services should provide JSON as the default encoding

Linnéuniversitetet

What have we learned so far…

Metadata Design (Rule 1 to 10): Information on Content type, length, caching should be provided.

Media Type Design (Rule 11 to 13): Proper media types should be used and negotiated by the clients.

Representation Design (Rule 14 to 16): JSON or other representations should be well formed.

Hypermedia Representation Design (Rule 17 to 19): Well formed links in response bodies.

Hypermedia Representation Design (Rule 20 to 23): Some client concerns on versioning, security, and hypermedia processing.