Web Services, part 3: Advanced Topics

i.e. topics not directly involved in project 2

REST Web Services in general

From Dr Dobbs:

Addressing Resources (making up URIs for resources)

- REST requires each resource to have at least one URI.
- The job of a URI is to identify a resource or a collection of resources.
- The actual operation is determined by an HTTP verb. The URI itself should not say anything about the operation or action.
- This enables us to call the same URI with different HTTP verbs to perform different operations.
- Our example: GET /…/orders vs. POST /…/orders

URIs vs. URLs

- Dr. Dobbs talks of URIs for REST web services
- But they look like the URLs
- What's the difference?
- From https://www.ietf.org/rfc/rfc3986.txt , the official authority on URIs and URLs:
  - A Uniform Resource Identifier (URI) is a compact sequence of characters that identifies an abstract or physical resource.
  - A URI can be further classified as a locator, a name, or both. The term "Uniform Resource Locator" (URL) refers to the subset of URIs that, in addition to identifying a resource, provide a means of locating the resource by describing its primary access mechanism (e.g., its network "location").

- So a URI might simply be an id with no "substance", nothing "at that address" on the Internet, but a URL should be usable with HTTP to access something
- We can argue that our REST web services are using URLs, not just URIs.

Dr. Dobbs Example

- Suppose we have a database of persons and we wish to expose it to the outer world through a service. A resource person can be addressed like this:

- http://MyService/Persons/1

- This URL has following format: Protocol://ServiceName/ResourceType/ResourceID

- Here are some important recommendations for well-structured URIs:

  – Use plural nouns for naming your collection resources.

  – Avoid using spaces in URIs as they create confusion. Use an _ (underscore) or – (hyphen) instead.

  – A URI is case insensitive. I use camel case in my URIs for better clarity. You can use all lower-case URIs.

    - This is wrong. The hostname part is case insensitive, but the path part is case sensitive on UNIX/Linux web servers because the filesystem is: try http://localhost:8000/cs637/eoneil/ch05_gs_server/api/CATEGORIES, see it fail, but localhost:8000/cs637/eoneil/ch05_gs_server/api/categories succeed

Dr. Dobbs Example

- http://MyService/Persons/1
- Avoid verbs for your resource names until your resource is actually an operation or a process. Verbs are more suitable for the names of operations.
- For example, a RESTful service should not have the URIs

http://MyService/FetchPerson/1 or http://MyService/DeletePerson/1.

- Instead, use GET to http://MyService/Person/1 and DELETE to http://MyService/FetchPerson/1

Query Parameters in REST URIs

- Here is a URI constructed with the help of a query parameter:

  http://MyService/Persons?id=1

- The query parameter approach works just fine and REST does not stop you from using query parameters. However, this approach has a few disadvantages.
  - Increased complexity and reduced readability, which will increase if you have more parameters
  - Search-engine crawlers and indexers like Google can by confused by URIs with query parameters. If you are developing for the Web, this could be a great disadvantage as a portion of your Web service will be hidden from the search engines. See https://support.google.com/webmasters/answer/6080548?hl=en

Query Parameters in REST URIs

- The basic purpose of query parameters is to provide parameters to an operation that needs the additional info. For example, if you want the format of the presentation to be decided by the client. You can achieve that through a parameter like this:

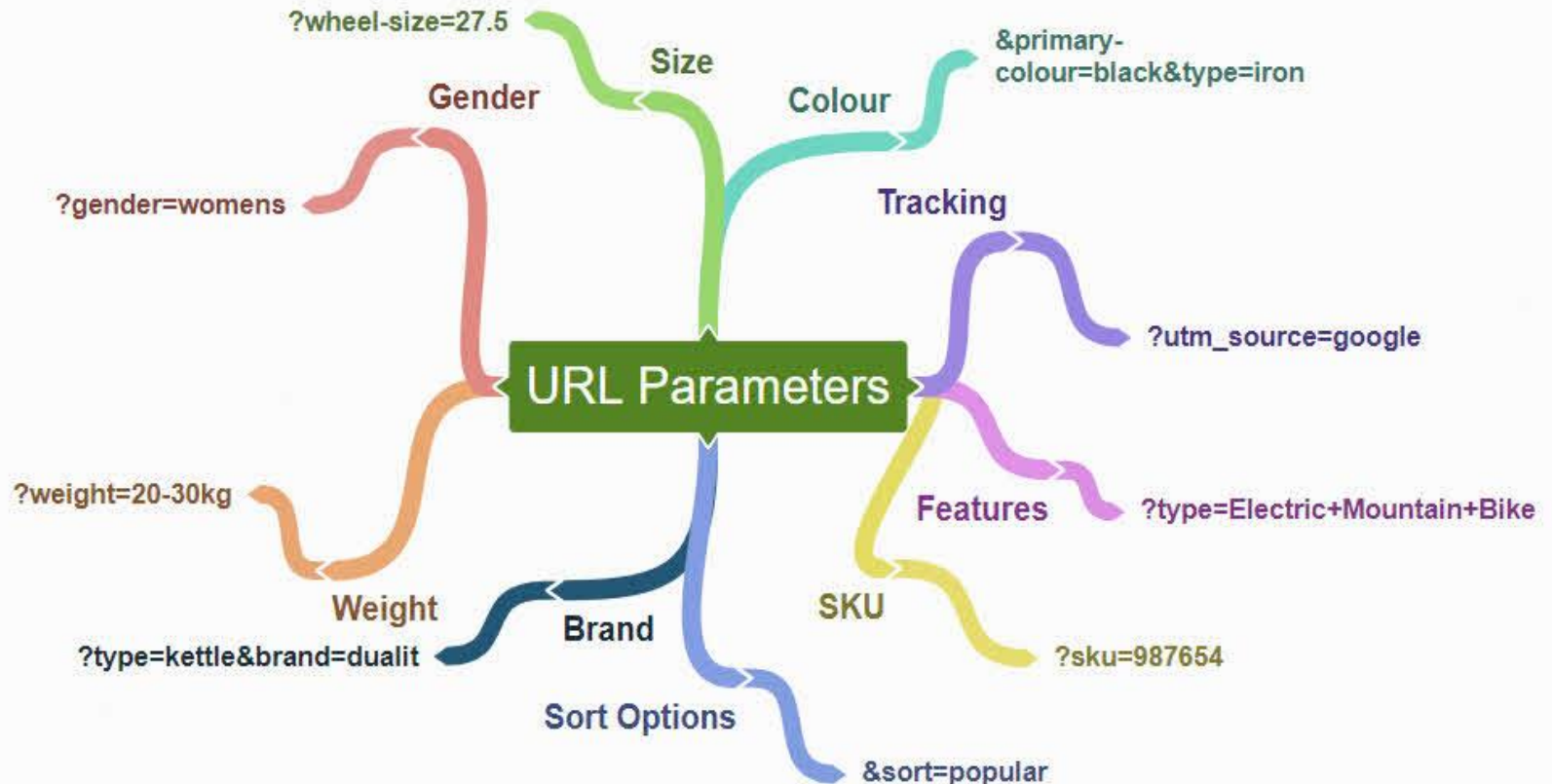http://MyService/Persons/1?format=xml&encoding=UTF8

- Note: There is another way to specify format and encoding using HTTP headers, covered later.

- Recall Flickr example from Purewal Chap 5:

https://api.flickr.com/services/feeds/photos_public.gne?tags=dogs&format=json

URL parameter ideas, from

Example service: fixer.io

- Reports currency conversion rates for 170 world currencies, including Bitcoin, Gold and Silver rates.

- Has free service upon registration (not as immediately updated)

- Example from their docs:

```
https://data.fixer.io/api/latest?access_key=
mykey&base=USD&symbols=GBP,JPY,EUR
```

Fixer request and response: example from their docs

```
https://data.fixer.io/api/latest?access_key=mykey&base=USD&symbols=GBP,JPY,EUR

{
    "success": true,
    "timestamp": 1519296206,
    "base": "USD",
    "date": "2020-12-04",
    "rates": {
        "GBP": 0.72007,    ←one dollar buys .72 pounds
        "JPY": 107.346001,←one dollar buys 107 yen
        "EUR": 0.813399,   ←one dollar buys .81 euro
    }
```

Fixer free example

http://data.fixer.io/api/latest?access_key=9add2f3ab6d0ec54c00ca0272e57f7e4&symbols=GBP,JPY,EUR,USD

{"success":true,"timestamp":1607111346,

"base":"EUR","date":"2020-12-04",
"rates":{"GBP":0.903425,"JPY":126.388524,"EUR":1
"USD":1.21357}}

So 1 dollar buys 1/1.21357 euros, etc.

Similarly, YouTube Data API requires access key, query in parameters

Links Between Resources

- A resource representation can contain links to other resources like an HTML page contains links to other pages.
- The representations returned by the service should drive the process flow as in case of a website.
- When you visit any website, you are presented with an index page. You click one of the links and move to another page and so on. Here, the representation is in the HTML documents and the user is driven through the website by these HTML documents themselves. The user does not need a map before coming to a website. A service can be (and should be) designed in the same manner.
- Let's consider the case in which a client requests one resource that contains multiple other resources. Instead of dumping all these resources, you can list the resources and provide links to them. Links help keep the representations small in size.
- Added by eoneil: However, following lots of links takes many round-trip times. We saw how much info was dumped by one video search…

JSON for Club of Persons

```
{
    "Name": "Authors Club",
    "Persons": [
        {
            "Name": "M. Vaqqas",
            "URI": "http:\/\/MyService\/Persons\/1"
        },
        {
            "Name": "S .Allamaraju",
            "URI": "http:\/\/MyService\/Persons\/12"
        }
    ]
}
```

We need to escape the / in the URL to hold it in a JSON string

JSON for Club

PHP: First build PHP array, then json_encode it:

```
$person1 = ['Name'=> 'M. Vaqqas',
    'URI'=>'http://MyService/Persons/1'];

$person2 = ['Name'=> 'S .Allamaraju',
                'URI'=>'http://MyService/Persons/12'];

$club = array('Name'=>'Authors Club',
                'Persons'=>[$person1, $person2]);

$json = json_encode($club,JSON_PRETTY_PRINT);
```

JS: build JS, call stringify:

```
let person1 = {"Name": 'M. Vaqqas', "URI": http://MyService/Persons/1};

let person2 = …

let club = {"Name": 'Authors Club', "Persons":[person1, person2]};

let clubJSON = JSON.stringify();
```

URI Templates

Dr. Dobbs example:

http://MyService/Persons/{PersonID}

For pizza2_server:

pizza2_server/api/orders/{orderID}

We saw this syntax in the routes setup under Slim:

$app->get('/orders/{id}', 'getOrder');

$app->put('/orders/{id}', 'updateOrder');

Content negotiation

From Wikipedia:

Content negotiation is a mechanism defined in the HTTP specification that makes it possible to serve different versions of a document (a resource representation) at the same URI, so that user agents (browsers, etc.) can specify which version fit their capabilities the best.

One classical use of this mechanism is to serve an image in GIF or PNG format, so that a browser that cannot display PNG images (e.g. MS Internet Explorer 4) will be served the GIF version.

Content Negotiation

- The user agent provides an Accept [HTTP header](#) that lists acceptable media types and associated quality factors.
- The server is then able to supply the version of the resource that best fits the user agent's needs.
- So, a resource may be available in several different representations. For example, it might be available in different languages or different media types, or a combination.
- For example, a browser could indicate that it would like to see information in German, if possible, else English will do. Browsers indicate their preferences by headers in the request. To request only German representations, the browser would send:

  Accept-Language: de
- Note that this preference will only be applied when there is a choice of representations and they vary by language.

Multiple preferences

As an example of a more complex request, this browser has been configured to accept German and English, but prefer German, and to accept various media types, preferring HTML over plain text or other text types, and preferring GIF or JPEG over other media types, but also allowing any other media type as a last resort:

```
Accept-Language: de; q=1.0, en; q=0.5
Accept: text/html; q=1.0, text/*; q=0.8, image/gif;
   q=0.6, image/jpeg; q=0.6, image/*; q=0.5, */*;
   q=0.1
```

- RFC 7231 does not specify how to resolve trade-offs (such as, in the above example, choosing between an HTML page in English and a GIF image in German).

XML vs. JSON

Client app 1 sends

Accept: application/json

Server sees that, sends response in JSON

Client app 2 sends

Accept: application/xml

Server sends XML

Languages

Browser 1 sends

Accept-Language: da (Danish)

Server sends HTML in Danish if possible

Accept-Language: da, en-gb;q=0.8, en;q=0.7

This means Danish is best for user, and British English (en-gb) is of somewhat lower quality, but better than non-British English. Similarly the app can specify which image formats it wants.

Hypermedia Protocols

- If the links in the webservice results are well enough organized, a client can discover more pages and snake their way through related services.
- This is the basic idea of hypermedia.
- Example from REST in Practice, by Webber et al:
  - Customer POSTs order, response Order rep (in JSON or XML) has link to cancel URI and make-payment URI
  - Customer POST to make-payment returns a Response that has link back to order and link to get-receipt
  - Customer GET to get-receipt returns a Response with link to order
  - Customer GET for Order (get-order) rep just has status (user needs to wait), no links
  - When order ready Order rep back from get-order has status=ready and link to receipt
  - When order done, Order rep has no links.

Some Available Web Services: and SDKs for PHP and JS

Look at Google APIs: all need https:
See translate, gmail, youtube, maps, etc., PHP SDK (Component)
GAPI The Google APIs Client Library for Browser JS: Google Docs, etc.
Google Maps JS API

Amazon AWS (cloud and storage services) SDKs for 9 languages!
PHP SDK (Component) JS in Browser SDK

Amazon S3: First important Web API, from '06 (Wikipedia), for Storage: see deprecated SOAP API, current REST API, has http:// endpoint, but requires https: and authorization for many actions. Basically provides files in the cloud.

Some Available Web Services: and SDKs for PHP and JS, continued

Facebook: graph API overview, shows GET, POST, JSON snippet
PHP SDK (Component) JS SDK

Twitter: REST API for tweets, PHP component for REST JS API

Video on REST APIs with Facebook, Google Maps, Instagram examples, using APIs as of 2014, but has useful ideas. The API tester app in use (at apigee.com) is no longer available. The video points to programmableweb.com for an API directory. That site has guide to tools for testing APIs