Understanding HTTP and REST

Oxford University
Software Engineering Programme

June 2016

World Wide Web

- navigating document collections
- multimedia documents
- hypertext cross-references
- hypertext markup language
- (HTML)
- hypertext transfer protocol
- (HTTP)
- Tim Berners-Lee at CERN, 1989–1992

HTTP

- two-way transmission of requests and responses
- layered over TCP
- essentially stateless (but. . .)
- standard extensions for security

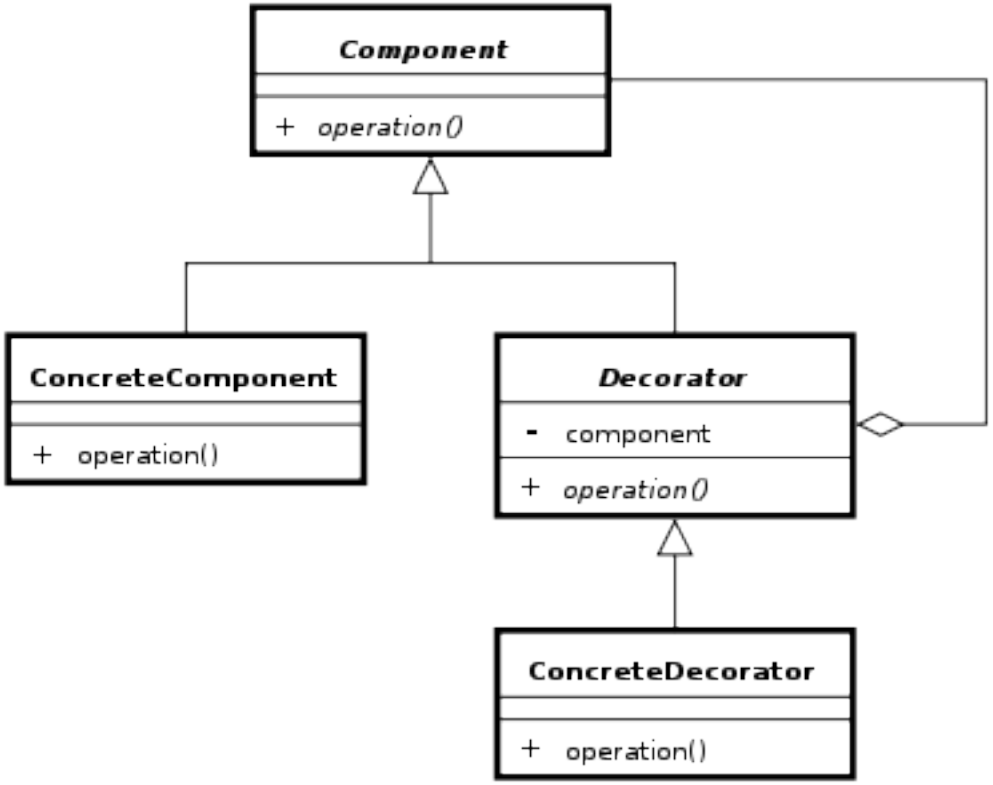
HTTP “Verbs”

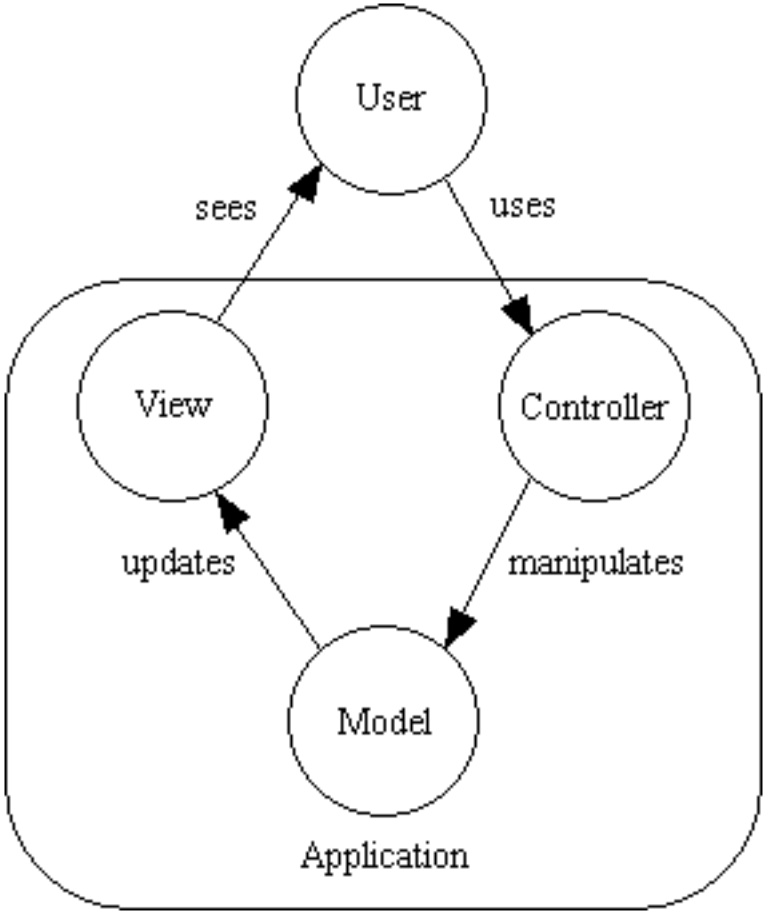
- GET uri
  - read a document; should be “safe”
- PUT uri, data
  - create or modify a resource; should be idempotent
- POST uri, data
  - create a subordinate resource
- DELETE uri
  - delete a resource; should be idempotent
- (also HEAD, TRACE, OPTIONS, CONNECT and now PATCH)

URIs, URNs, URLs

- uniform resource identifier (URI)
  - uniform resource locator (URL)
  - uniform resource name (URN)
- http://fremantle.org/hello
  - Is it a URI? URL? URN?

Examples of Design Patterns





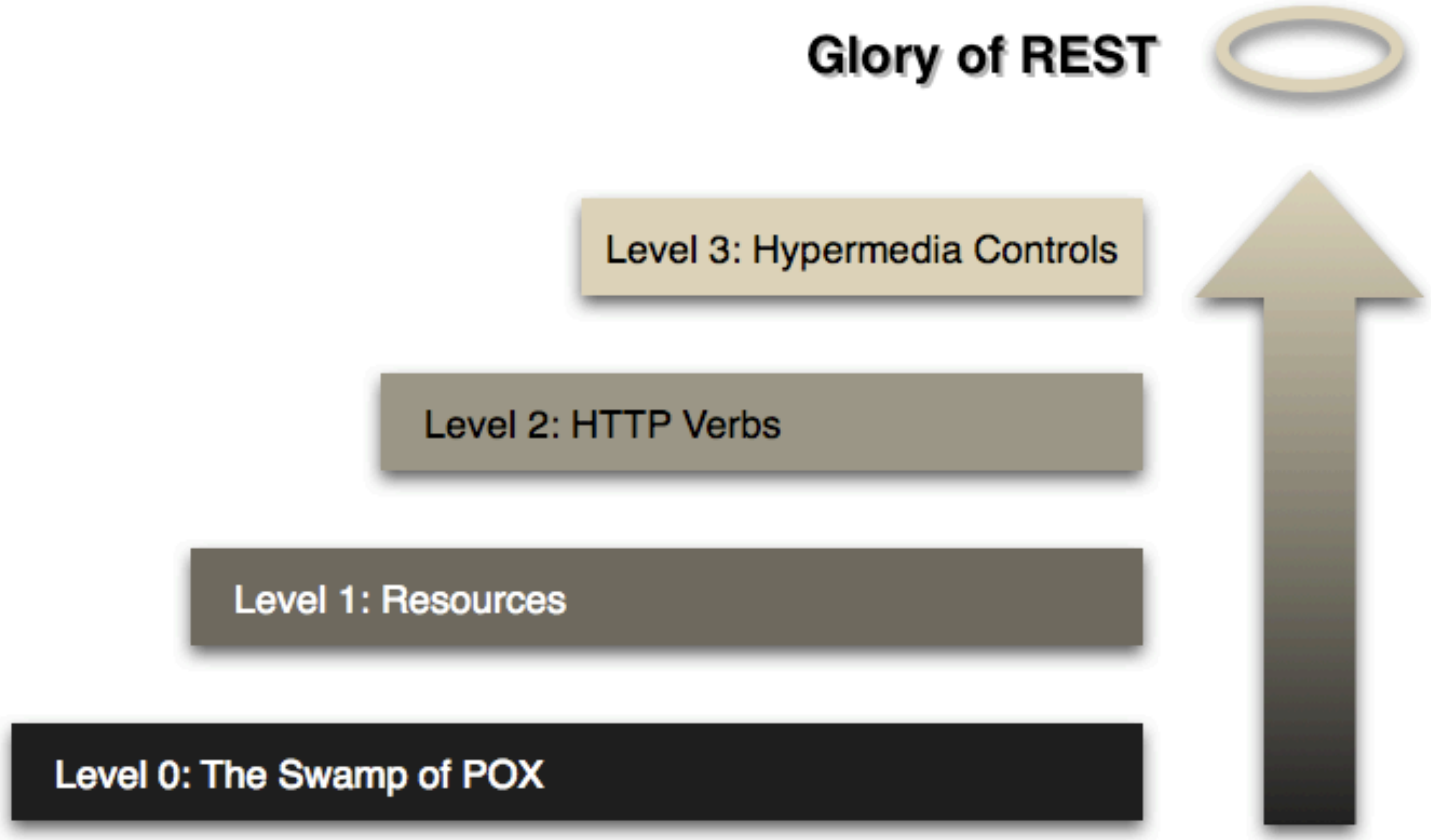
REST is a design pattern

Also characterized as an ***Architectural Style*** (aka an architecture design pattern)

Resource Oriented Architecture

- Resource-oriented architecture
  - after Richardson & Ruby, RESTful WS
  - action identified in HTTP method, not in payload
  - scoping information in URI

Richardson's Maturity Model



HTTP good bad and ugly

- Good
  - GET reports/open-bugs HTTP/1.1
    - in contrast to RPC-style interaction
- Bad

  -
    ```
    POST /rpc HTTP/1.1
    Host: www.upcdatabase.com
    <?xml version="1.0”>
     <methodCall>
     <methodName>lookupUPC</methodName> …
    </methodCall>
    ```
- Ugly
  - http://www.flickr.com/services/rest?method=search&tags=cat

PUT vs POST

- PUT vs POST
  - creation by either PUT to new URI or POST to existing URI
  - typically, create a subordinate resource with a POST to its parent
- use PUT when client chooses URI; use POST when server chooses
- successful POST returns code 201 ‘Created’ with Location header
- (POST also sometimes used for form submission, but this can be non-uniform)

Resource Representations and States

- Interact with services using representations of resources.
  - An XML representation
  - A JSON representation
- An object referenced by one URI can have different formats available. Different platforms need different formats.
  - A mobile application may need JSON
  - A Java application may need XML.
- Utilize the Content-Type header
  - And the Accept: header
- Communicate in a stateless manner
  - Stateless applications are far more scaleable

Hypertext as the Engine of Application State

- Resources are identified by URIs

↓

- Clients communicate with resources via requests using a
- standard set of methods

↓

- Requests and responses contain resource representations
- in formats identified by media types

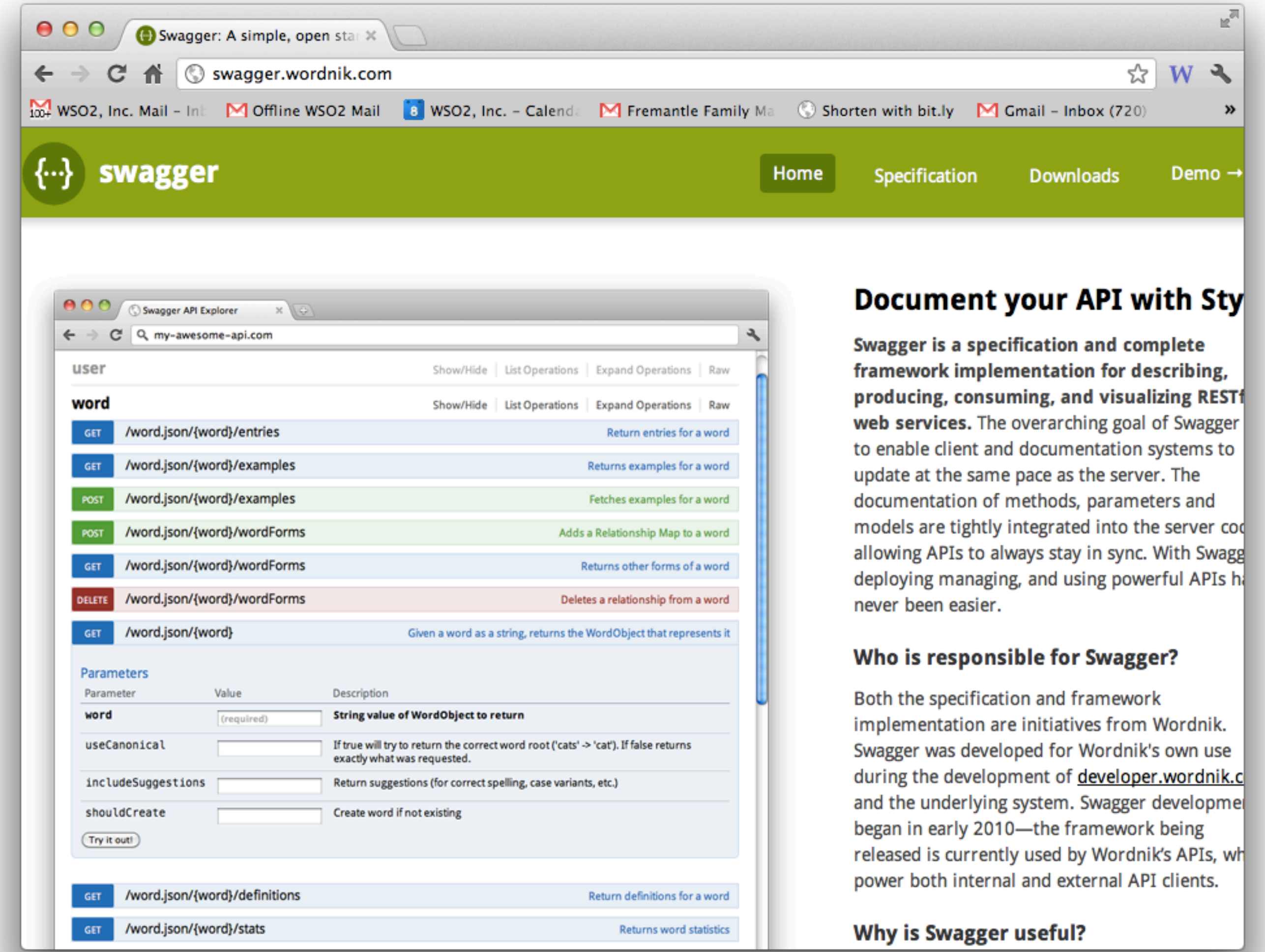
↓

- Responses contain URIs that link to further resources

↓

Beginning

REST description



user

word

GET	/word.json/{word}/entries	Return entries for a word
GET	/word.json/{word}/examples	Returns examples for a word
POST	/word.json/{word}/examples	Fetches examples for a word
POST	/word.json/{word}/wordForms	Adds a Relationship Map to a word
GET	/word.json/{word}/wordForms	Returns other forms of a word
DELETE	/word.json/{word}/wordForms	Deletes a relationship from a word
GET	/word.json/{word}	Given a word as a string, returns the WordObject that represents it

Parameters

Parameter	Value	Description
word	(required)	String value of WordObject to return
useCanonical		If true will try to return the correct word root ('cats' -> 'cat'). If false returns exactly what was requested.
includeSuggestions		Return suggestions (for correct spelling, case variants, etc.)
shouldCreate		Create word if not existing

GET	/word.json/{word}/definitions	Return definitions for a word
GET	/word.json/{word}/stats	Returns word statistics

Document your API with Sty

Swagger is a specification and complete framework implementation for describing, producing, consuming, and visualizing RESTf web services. The overarching goal of Swagger to enable client and documentation systems to update at the same pace as the server. The documentation of methods, parameters and models are tightly integrated into the server cod allowing APIs to always stay in sync. With Swagg deploying managing, and using powerful APIs ha never been easier.

Who is responsible for Swagger?

Both the specification and framework implementation are initiatives from Wordnik. Swagger was developed for Wordnik's own use during the development of developer.wordnik.c and the underlying system. Swagger developme began in early 2010—the framework being released is currently used by Wordnik's APIs, wh power both internal and external API clients.

Why is Swagger useful?

JSON

- A simple notation that originated in JavaScript

```
var x = {a:1, b:2, c:3}
```

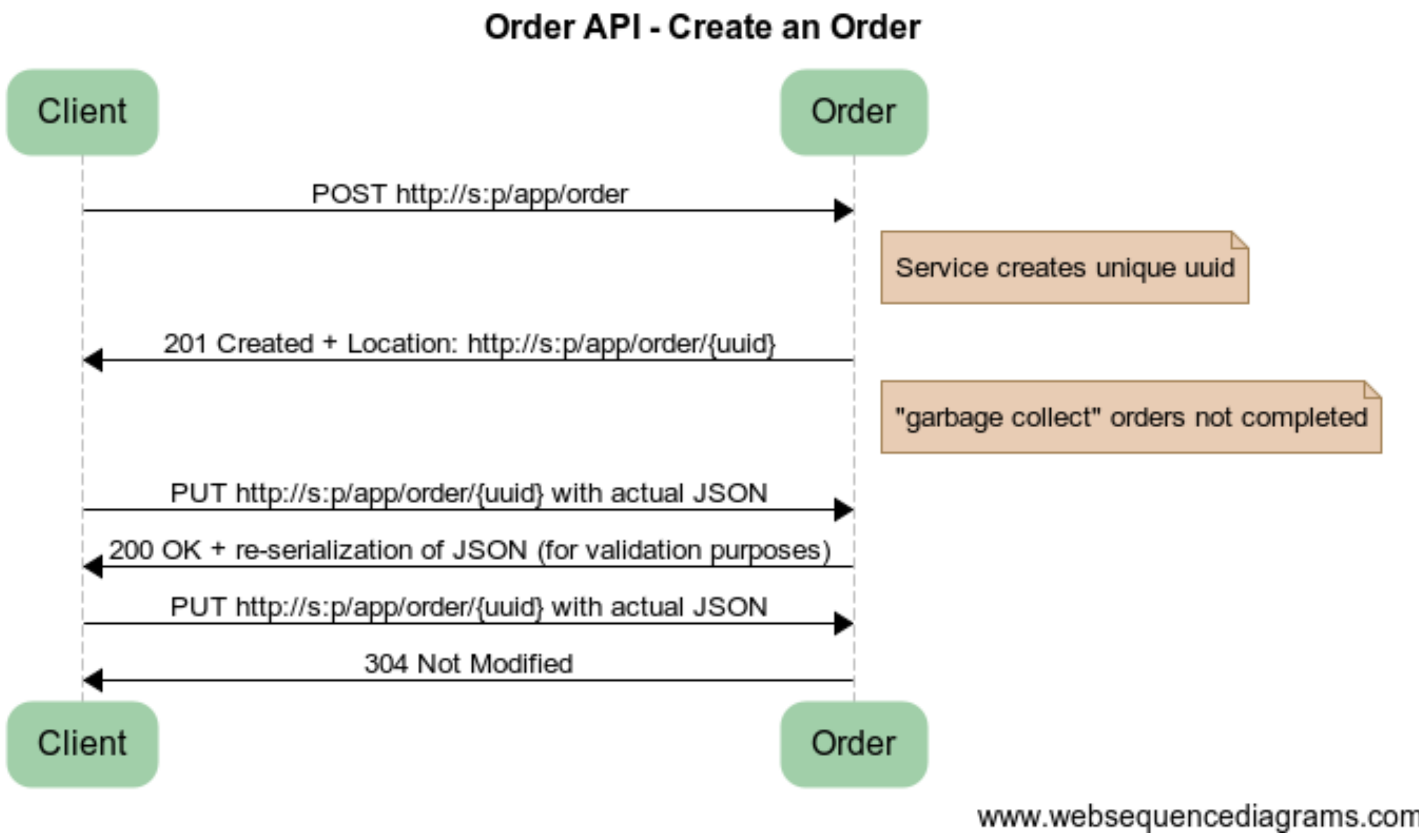
- equivalent to:

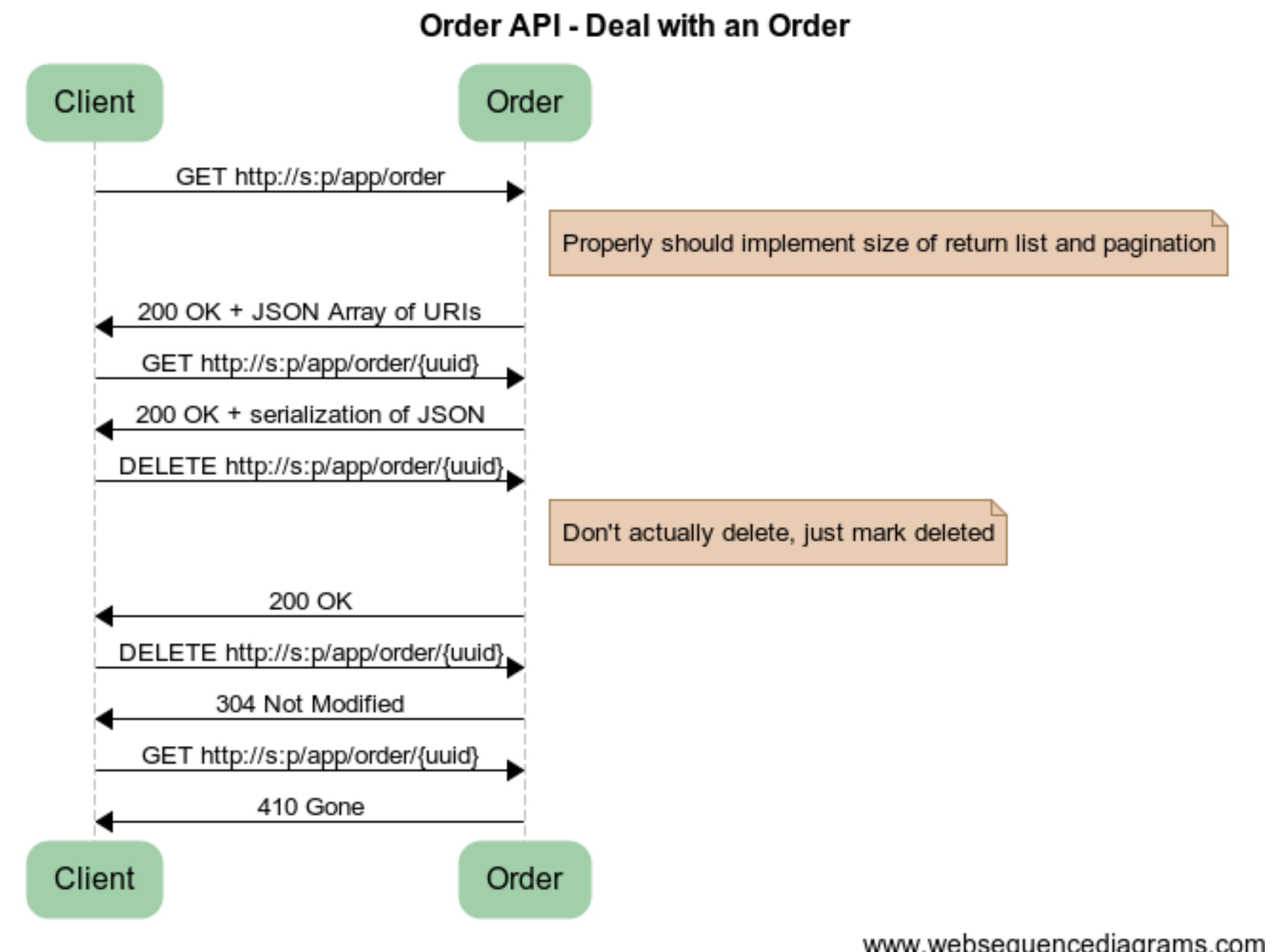
```
x.a = 1; x.b = 2; x.c = 3
```

- Can be done “dynamically”

```
var x = “{a:1, b:2, c:3}”
// imagine this actually
// comes from a webserver
var z = eval(‘(‘+x+’)’)
assert(z.a == 1)
```

Our sample Purchase service


Order API - Create an Order
Client
Order
POST http://s:p/app/order
Service creates unique uuid
201 Created + Location: http://s:p/app/order/{uuid}
"garbage collect" orders not completed
PUT http://s:p/app/order/{uuid} with actual JSON
200 OK + re-serialization of JSON (for validation purposes)
PUT http://s:p/app/order/{uuid} with actual JSON
304 Not Modified
Client
Order
www.websequencediagrams.com


Order API - Deal with an Order
Client
Order
GET http://s:p/app/order
Properly should implement size of return list and pagination
200 OK + JSON Array of URIs
GET http://s:p/app/order/{uuid}
200 OK + serialization of JSON
DELETE http://s:p/app/order/{uuid}
Don't actually delete, just mark deleted
200 OK
DELETE http://s:p/app/order/{uuid}
304 Not Modified
GET http://s:p/app/order/{uuid}
410 Gone
Client
Order
www.websequencediagrams.com

Questions?