

Understanding HTTP and REST

Oxford University
Software Engineering
Programme
January 2018



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

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



```
127.0.0.1:40816: clientconnect
127.0.0.1 GET http://localhost:8080/
host: localhost:8000
accept-encoding: gzip, deflate
user-agent: Python-httpplib2/0.9.2 (gzip)
```

```
<< 200 OK 13B
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
content-length: 13
ETag: W/"d-95lxyDUPrXs/bUPZHxxiWQ"
Date: Mon, 20 Jun 2016 09:19:05 GMT
Connection: keep-alive

{
  "random": 63
}
```

```
127.0.0.1:40816: clientdisconnect
```



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)

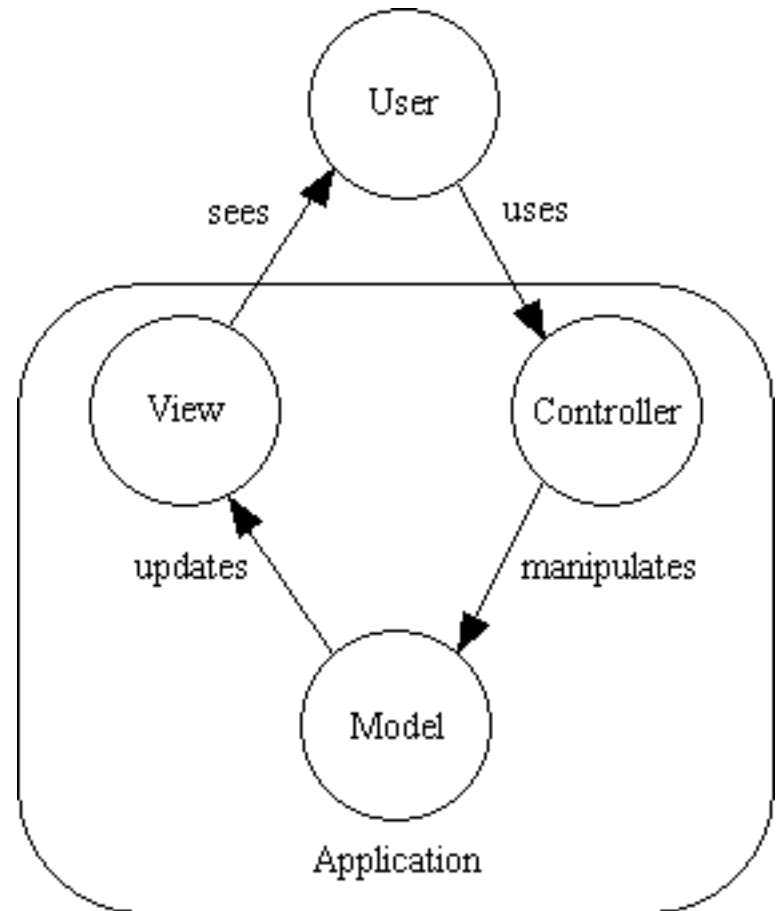
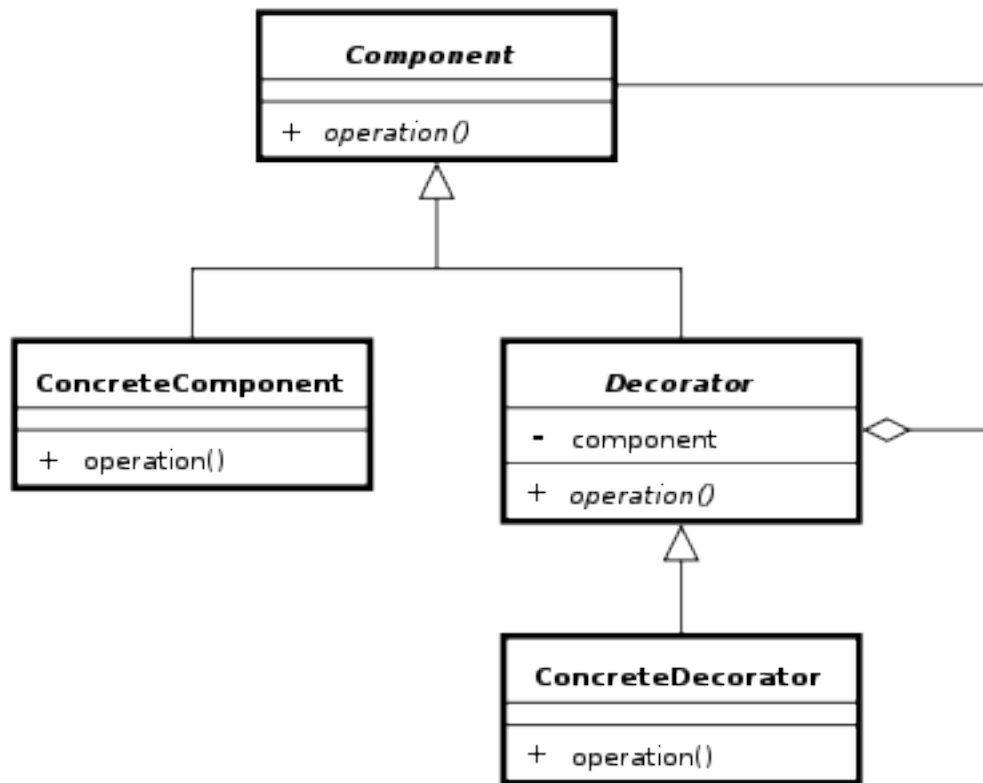


URIs, URNs, URLs

- 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

Glory of REST



Level 3: Hypermedia Controls

Level 2: HTTP Verbs

Level 1: Resources

Level 0: The Swamp of POX



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



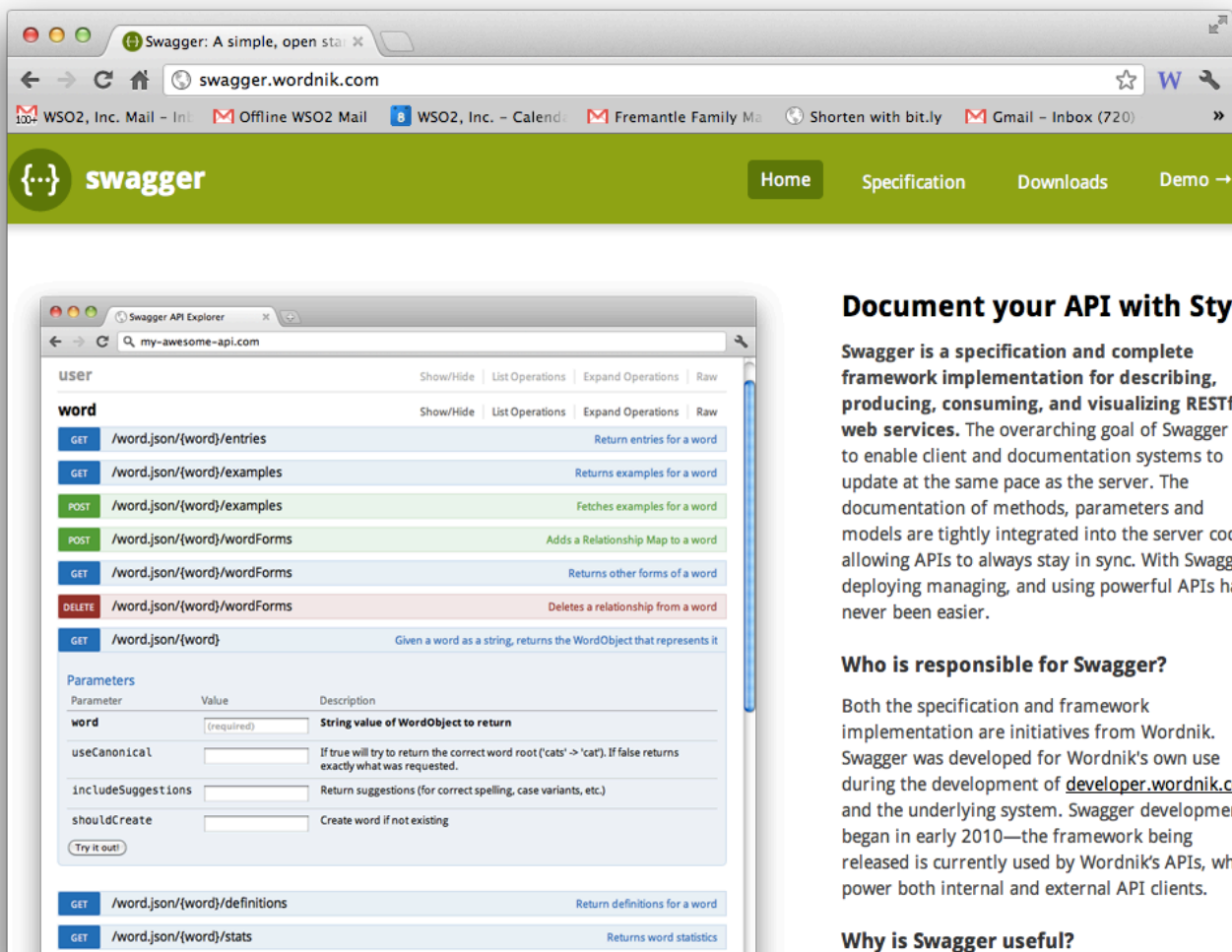
- Responses contain URIs that link to further resources



Beginning



REST description



The screenshot displays the Swagger API Explorer interface for a service named 'my-awesome-api.com'. The interface is divided into two main sections: a top navigation bar and a main content area. The top bar includes the Swagger logo, a 'Home' button, and links to 'Specification', 'Downloads', and 'Demo'. The main content area is titled 'Swagger API Explorer' and shows a list of API endpoints under the 'word' resource. The endpoints are categorized by HTTP method (GET, POST, DELETE) and color-coded. The 'word' resource is expanded, showing a list of endpoints with their descriptions. Below the endpoints, there is a 'Parameters' section with a table of parameters and their descriptions. The parameters include 'word' (required), 'useCanonical', 'includeSuggestions', and 'shouldCreate'. The 'word' parameter is highlighted, and its description is 'String value of WordObject to return'. The 'useCanonical' parameter is described as 'If true will try to return the correct word root ('cats' -> 'cat'). If false returns exactly what was requested.' The 'includeSuggestions' parameter is described as 'Return suggestions (for correct spelling, case variants, etc.)'. The 'shouldCreate' parameter is described as 'Create word if not existing'. There is a 'Try it out!' button next to the parameters. The bottom of the interface shows a list of endpoints with their descriptions.

Document your API with Style

Swagger is a specification and complete framework implementation for describing, producing, consuming, and visualizing REST web services. The overarching goal of Swagger is 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 code, allowing APIs to always stay in sync. With Swagger, deploying, managing, and using powerful APIs has 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.com and the underlying system. Swagger development began in early 2010—the framework being released is currently used by Wordnik's APIs, which 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)
```


Return codes

- Good RESTful design means proper use of return codes...
 - Why?



HTTP return codes

| | | |
|-------------|-----------------------------------|---|
| | 100 Continue | |
| | 101 Switching Protocols | |
| Successful | 200 OK | Everything is normal |
| | 201 Created | |
| | 202 Accepted | |
| | 203 Non-Authoritative Information | |
| | 204 No Content | |
| | 205 Reset Content | |
| | 206 Partial Content | |
| Redirection | 300 Multiple Choices | Update your URL, this has moved for good. |
| | 301 Moved Permanently | |
| | 302 Found | |
| | 303 See Other | |
| | 304 Not Modified | |
| | 305 Use Proxy | |
| | 306 Unused | |
| | 307 Temporary Redirect | This is temporarily moved, don't update your bookmarks. |



Client Error Codes

Client Error

| | |
|-----------------------------------|--|
| 400 Bad Request | Server didn't understand the URL you gave it. |
| 401 Unauthorized | Must be authenticated |
| 402 Payment Required | Not used really |
| 403 Forbidden | Server refuses to give you a file, authentication won't help |
| 404 Not Found | A file doesn't exist at that address |
| 405 Method Not Allowed | |
| 406 Not Acceptable | |
| 407 Proxy Authentication Required | |
| 408 Request Timeout | Browser took too long to request something |
| 409 Conflict | |
| 410 Gone | |
| 411 Length Required | |
| 412 Precondition Failed | |
| 413 Request Entity Too Large | |
| 415 Unsupported Media Type | |
| 416 Request Range Not Satisfiable | |
| 417 Expectation Failed | |

Server Error Codes

Server Error

500 Internal Server Error

Something on the server didn't work right.

501 Not Implemented

502 Bad Gateway

503 Service Unavailable

Too busy to respond to a client

504 Gateway Timeout

505 HTTP Version Not Supported



Summary

- Basic REST concepts:
 - Use the right VERB
 - Use the right return code
 - Use well defined media types
 - Resource representation
 - Use hyperlinks for HATEOAS

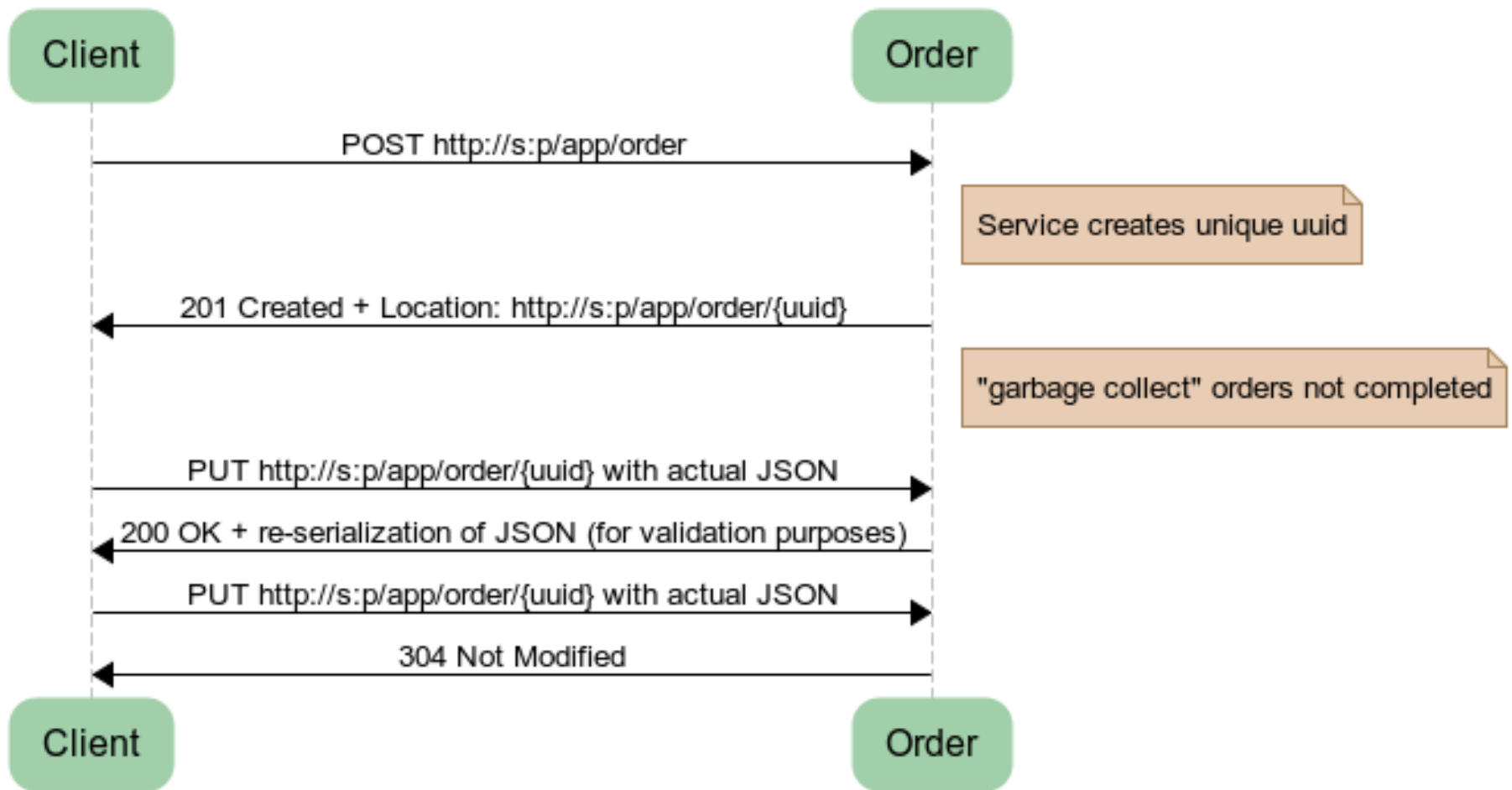


Our sample Purchase service



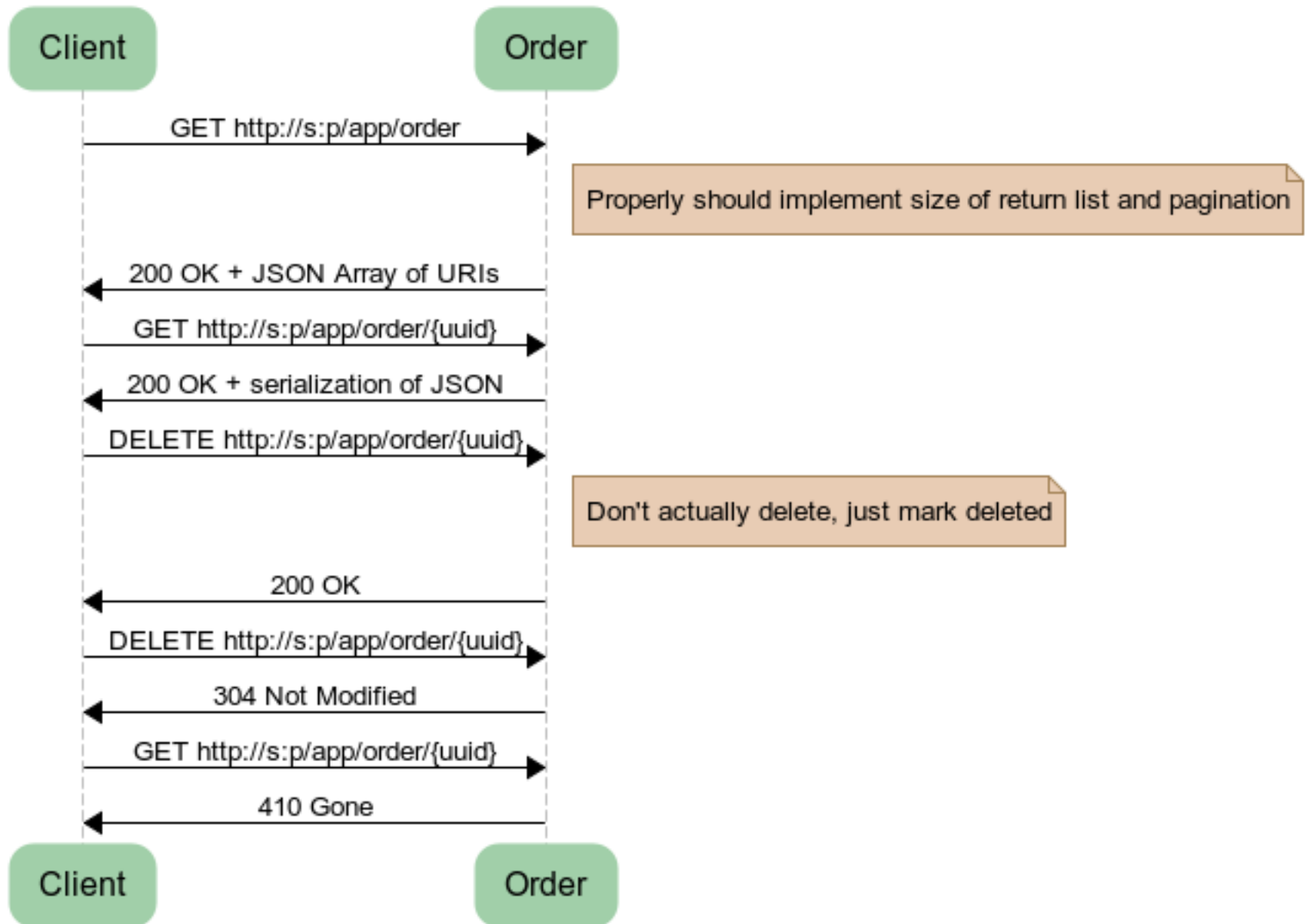
© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

Order API - Create an Order



www.websequencediagrams.com

Order API - Deal with an Order



Questions?



© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>