

# Understanding HTTP and REST

Oxford University  
Software Engineering Programme  
April 2021



© 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-95lxuDUPrXs/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)

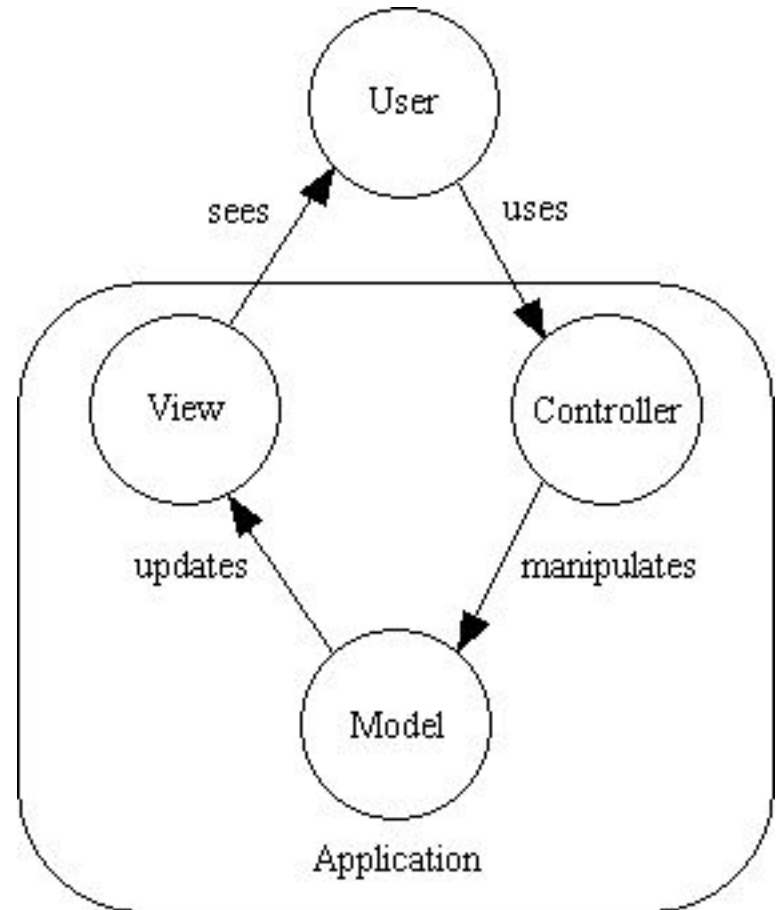
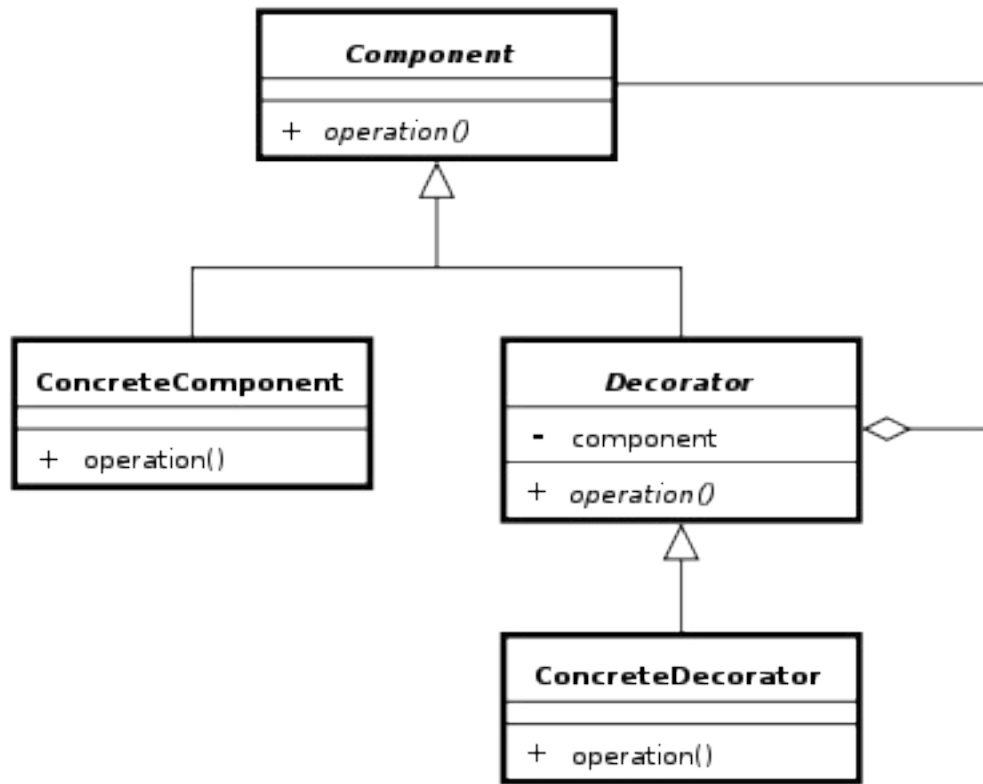


# More VERBS

- HEAD - just get the metadata
- OPTIONS - which verbs are supported?
- PATCH - just send the updates



# Examples of Design Patterns



# REST is a design pattern

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





# Doesn't REST just mean using HTTP?



© 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/>

# REST

- **Roy Fielding**, a principal author of HTTP
- PhD thesis *Architectural Styles and the Design of Network-based*
- Subsequent article *Principled Design of the Modern Web Architecture* (ACM TOIT 2:2, 2002)
- Richardson & Ruby, *RESTful Web Services* architectural patterns of the web

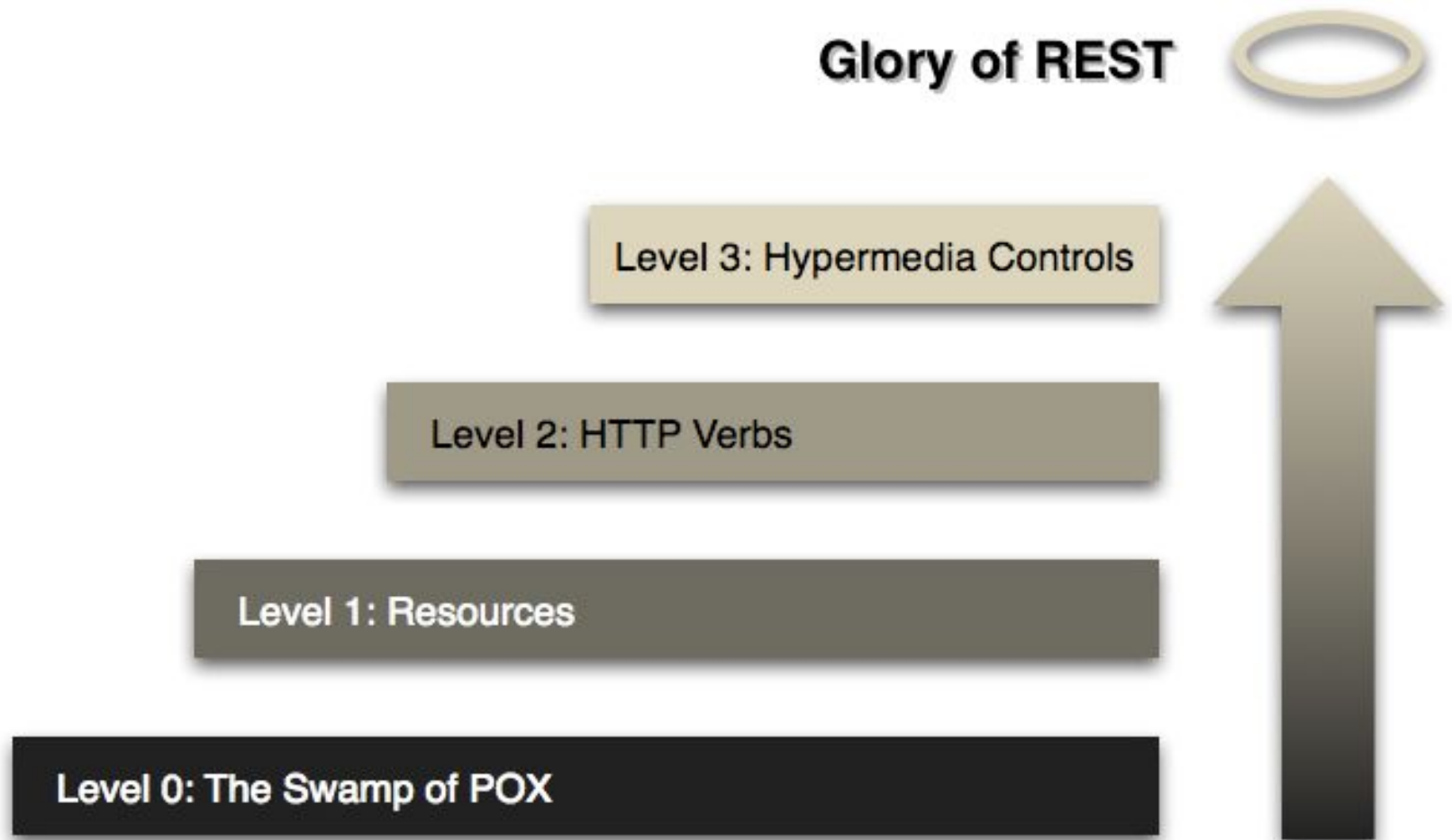


# Core ideas of REST

- “Treat HTTP seriously”
- Every “object” has a unique URL
- Use the correct “VERB”:
  - GET, POST, PUT, DELETE
- Use content-types properly
- Use good HTTP return codes
- Use hyperlinks



# 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&tag=s=cat>

# Core HTTP Verbs

- GET
  - get a representation of a resource
  - no side effects or updates to the resource
  - cacheable & idempotent
- PUT
  - update a resource
  - idempotent
- POST
  - create a new resource
- DELETE
  - remove a resource
  - idempotent



# URLs for resources

<http://mybank.com/account/11002123>

→

{

balance: 1100.10,

transactions:

“/account/11002123/transactions”

}



# Hyperlinks

<http://mybank.com/account/11002123/transactions/1>→

{

from: "<http://otherbank.com/ac/50893432/>",

to: "/account/11002123"

amount: 34.12,

date: "1/1/2021"

}





# PUT vs POST

- creation by either PUT to new URI or POST to existing URI
  - use PUT when client chooses URI;
  - use POST when server chooses
    - typically, create a “subordinate” resource with a POST to its parent
- successful POST returns code 201 ‘Created’ with Location header



# POST example

POST /account/11002123/transactions

```
{  
  to: "https://otherbank.com/ac/88819999",  
  amount: "23.11",  
  date: "1/1/2021"  
}
```

returns

201 Created

Location: /account/11002123/transactions/2



# 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.
  - 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



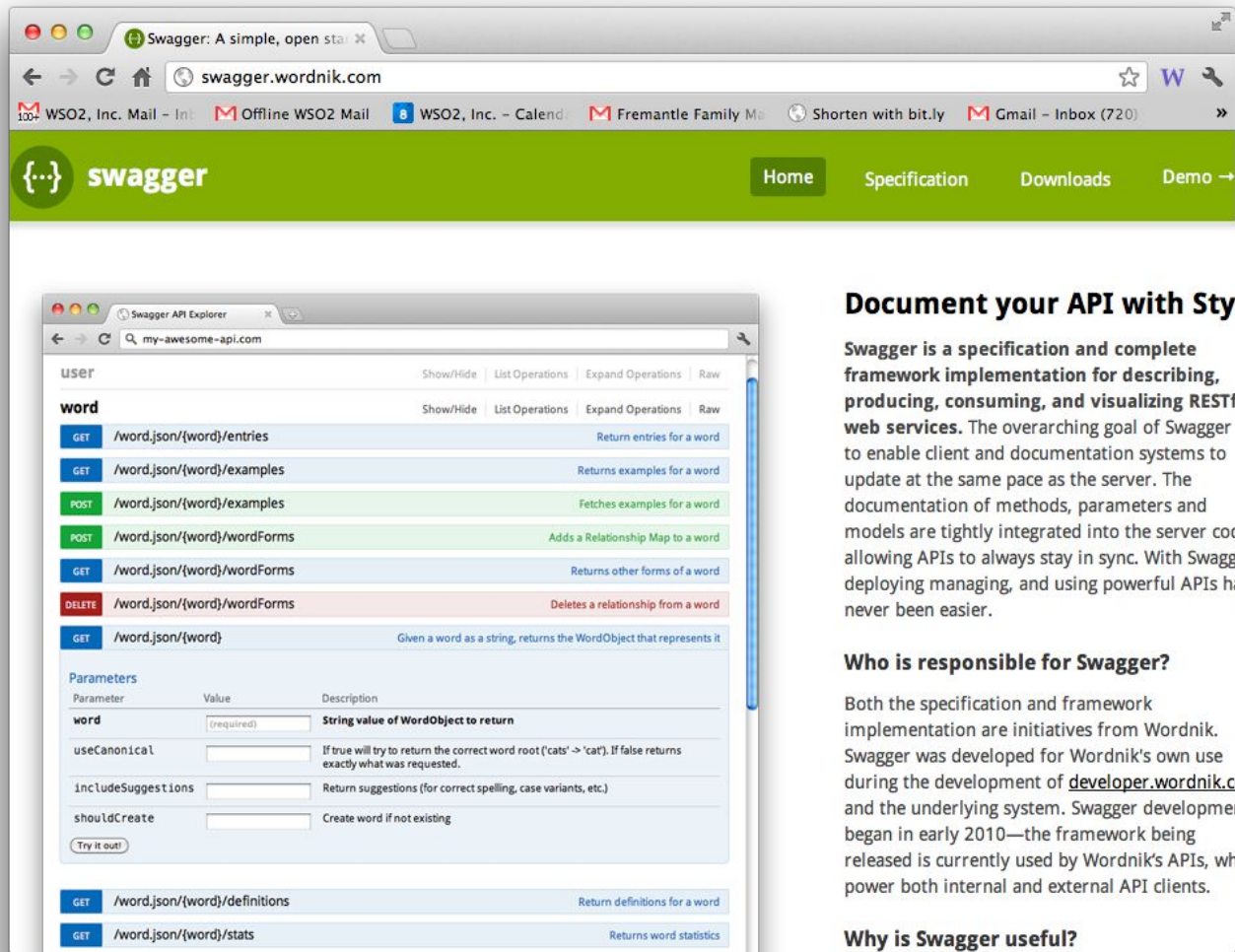
- Responses contain URIs that link to further resources



Beginning



# REST description more later!



The image shows a browser window displaying the Swagger website (swagger.wordnik.com) and a separate window showing the Swagger API Explorer for a sample API.

**Swagger Website:**

- Header: { } swagger
- Navigation: Home, Specification, Downloads, Demo
- Text: Document your API with Style
- Paragraph: 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.
- Section: Who is responsible for Swagger?
- Text: 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](http://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.
- Section: Why is Swagger useful?

**Swagger API Explorer:**

my-awesome-api.com

**word**

| Method | Path                        | Description   |
|--------|-----------------------------|---|
| 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               | <input type="text" value="(required)"/> | String value of WordObject to return   |
| useCanonical       | <input type="checkbox"/>                | If true will try to return the correct word root ('cats' -> 'cat'). If false returns exactly what was requested. |
| includeSuggestions | <input type="checkbox"/>                | Return suggestions (for correct spelling, case variants, etc.)   |
| shouldCreate       | <input type="checkbox"/>                | Create word if not existing  |

[Try it out!](#)

**word**

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

# 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



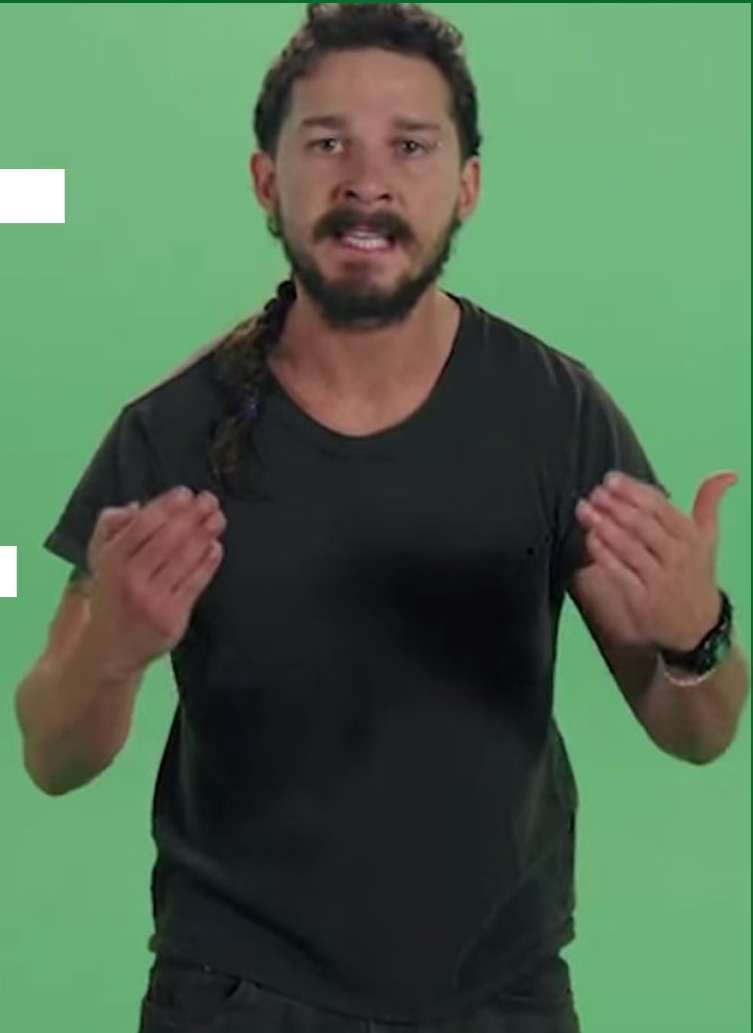
# Implementing REST



© 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/>

Just do it?!

JUST  
DO IT



# A good answer if you already know

- HTTP coding
- JSON
- etc



# Why use a framework?

- Routing
  - Separate logic for different verbs, paths, content-types
- Cacheing and content negotiation
- Data format manipulation
  - Translation to/from JSON
- Readability



# Labs approach

Express + tsoa

- Built in routing, controllers, structure
- Caching, etags
- Decorations/annotations
- Swagger generation
- Security / Authentication



# Clients?

- Typically HTTP clients
- OpenAPI/Swagger can help you generate client code automatically
- Multiple choices again
  - Python: httplib2, http.client, requests
  - Node: http, axios, superagent, etc
  - Java: JAXRS, Apache HTTPClient, builtin



# Summary

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



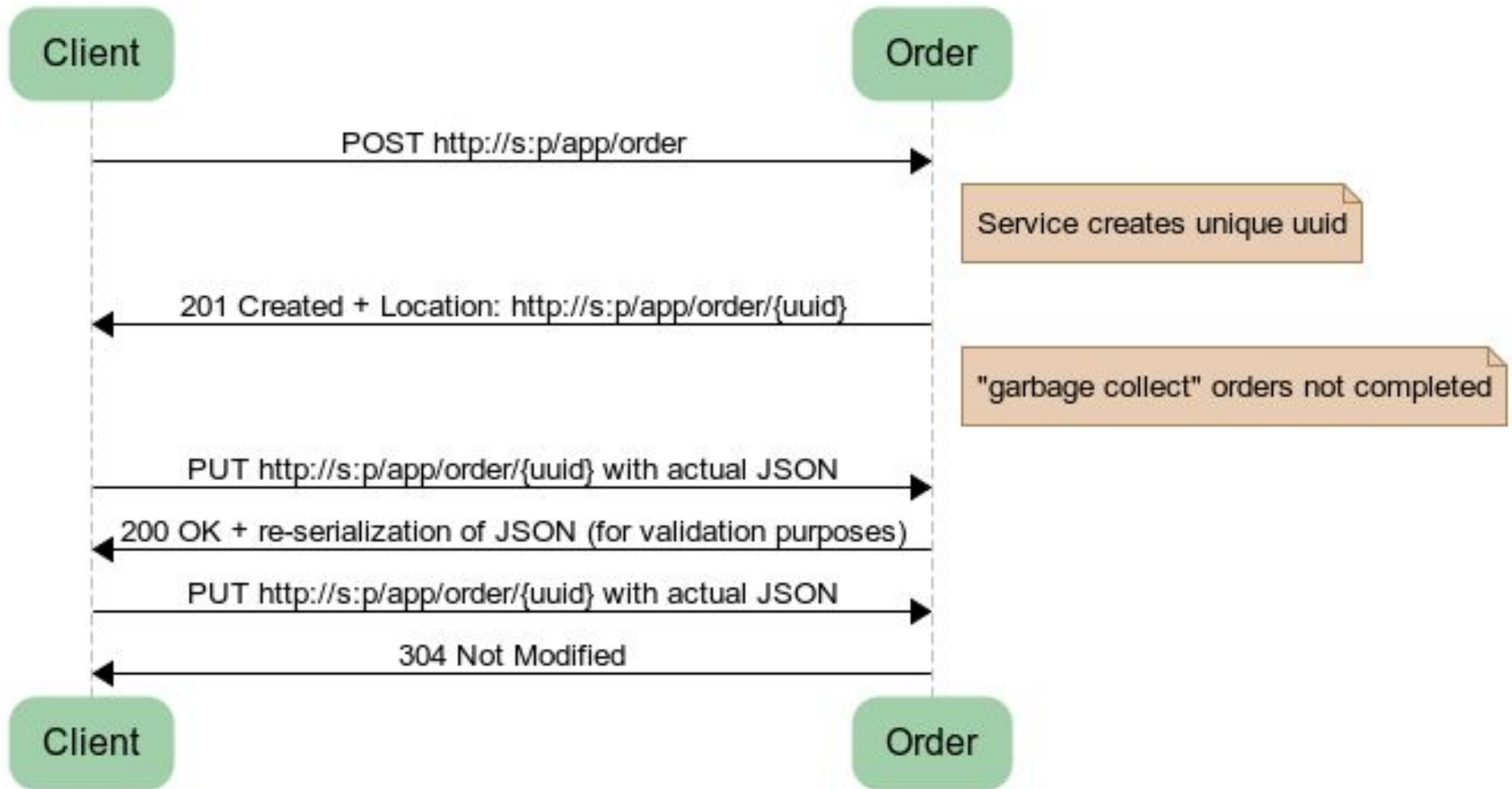


# Our sample Purchase service



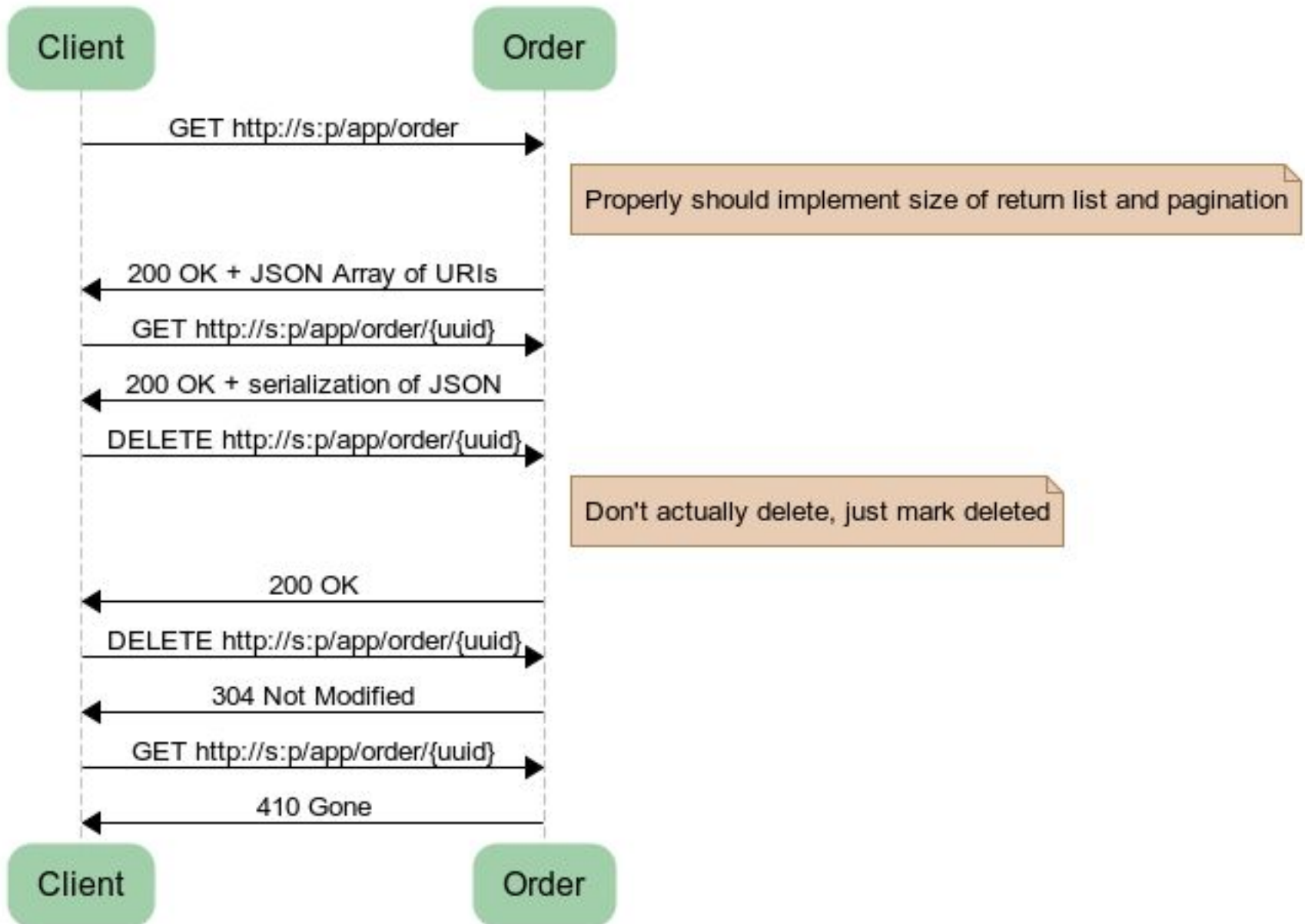
© Paul Fremantle 2016 except where credited elsewhere. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

## 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/>