

FROM '00s TO '20s: FROM RESTful to gRPC

Gianfranco Reppucci





WHAT THIS TALK IS ABOUT

BEFORE REST



**Common Object Request
Broker Architecture**

SOAP

**Simple Object
Access Protocol**

REST: REpresentational State Transfer



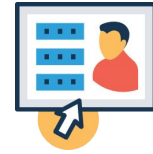
REST PRINCIPLES



**Extension of the
web *resource*
concept**



**Identification of
resources with a
*universal syntax***



**Resource
accessibility via a
*universal interface***

REST ARCHITECTURAL ELEMENTS

DATA ELEMENTS



RESOURCES



URIs



REPRESENTATIONS

CONNECTORS



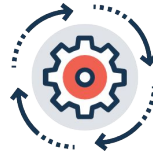
COMPONENTS



REST CONSTRAINTS



Client - Server



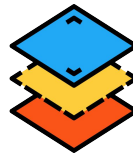
Stateless



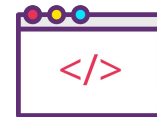
Cacheability



Uniform Interface



Layered System



Code On Demand

The background is a dark gray field filled with a complex network of thin, light gray lines and dots, creating a web-like or molecular structure. Scattered throughout are various geometric shapes, including hexagons and triangles, some of which are slightly more prominent than others. A bright orange rectangular border is centered on the slide, enclosing the text.

**SO, WHAT'S THE
PROBLEM WITH REST?**

RELATIONSHIP BETWEEN URIs AND HTTP VERBS / 1

`https://api.example.com/persons`

GET

List of URIs (w/other details?) of all
the person items in the database

PUT

Replace the entire persons
dataset with a new one

PATCH

Not generally used
ok...

POST

Add a new person
to the dataset

DELETE

Delete the entire persons
dataset

RELATIONSHIP BETWEEN URIs AND HTTP VERBS / 2

<https://api.example.com/persons/123>

GET

Retrieve the person matching
the given identifier "123"

PUT

Replace the addressed
person with a new one

PATCH

Update the addressed person
with the given fields

POST

Not generally used
uhm...

DELETE

Delete the addressed person
from the dataset



REST*ful* IS EVIL

FULL LIST OF HTTP VERBS

OPTIONS

GET

HEAD

POST

PUT

DELETE

TRACE

CONNECT

√ (ツ) √

FULL LIST OF HTTP STATUS CODES / 1



INFORMATIONAL

100, 101, 102, 103



SUCCESS

200, 201, 203, 204,
205, 206, 207, 208,
226



REDIRECTION

300, 301, 302, 303,
304, 305, 306, 307,
308

FULL LIST OF HTTP STATUS CODES / 1

CLIENT ERRORS

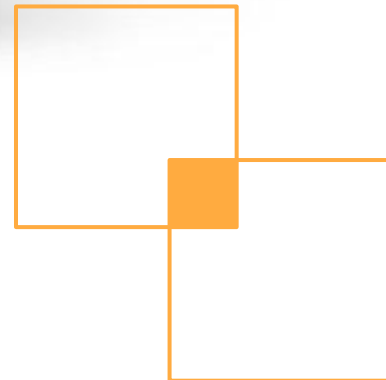
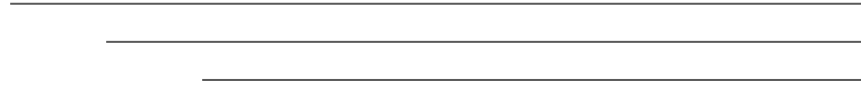
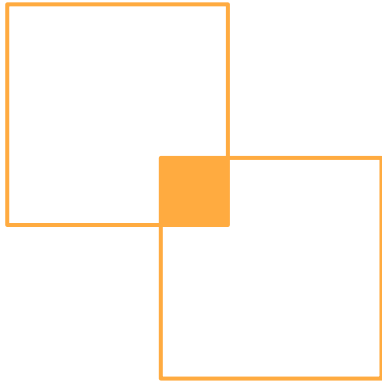
400, 401, 402, 403,
404, 405, 406, 407,
408, 409, 410, 411,
412, 413, 414, 415,
416, 417, **418**, 421,
422, 423, 424, 426,
428, 429, 431, 451

SERVER ERRORS

500, 501, 502, 503,
504, 505, 506, 507,
508, 510, 511

UNOFFICIAL CODES

103, 218, 420, 450,
498, 499, 509, 530,
598, many more...





PROBLEMS OF RESTful APIs

#1

LITTLE AGREEMENT ON WHAT “RESTful” MEANS



HTTP/1.1 Status Codes 400 and 417, cannot choose which



3

I have been referred to here that it might be of better help, I've got a processing file which handles the user sent data, before that, however, it compares the input from client to the expected values to ensure no client-side data change.



I can say I don't know lot about HTTP status codes, but I have made up some research on it, and to choose which one is the best for unexpected input handling. So I came up with:



400 Bad Request: The request cannot be fulfilled due to bad syntax

417 Expectation Failed: The server cannot meet the requirements of the Expect request-header field

Now, I cannot be really sure which one to use, I have seen 400 Bad Request being used alot, however, whatl get from explanation is that the error is due to an unexistent request rather than an illegal input.

On the other side 417 Expectation Failed seems to just fit for my use, however, I have never seen or experimented this header status before.

I need your experience and opinions, thanks alot!

#2

REST VOCABULARY IS NOT *FULLY* SUPPORTED

[Developer](#)[Use cases](#)[Products](#)[Docs](#)[More](#)

🔍 Search all documentation...

Follow, search, and get users

Basics

Accounts and users

Subscribe to account activity

Manage account settings and profile

Mute, block and report users

[Overview](#) [API Reference](#)

API Reference contents ^

[GET followers/ids](#)

[GET followers/list](#)

[GET users/lookup](#)

[GET users/search](#)

[GET users/show](#)

[GET users/suggestions](#)

[GET users/suggestions/:slug](#)

[GET users/suggestions/:slug/me](#)

[POST friendships/create](#)

[POST friendships/destroy](#)

[POST friendships/update](#)

[POST friendships/create](#)

[POST friendships/destroy](#)

[POST friendships/update](#)

Geo

Ads

#3

REST VOCABULARY IS NOT RICH ENOUGH FOR A COMPLETE API

[Fusion](#)[Fusion API](#)[GraphQL](#)[Manage App](#)[Log In](#)[Sign Up](#)

General

[Create App](#)[Email / Notifications](#)[Display Requirements](#)[Terms of Use](#)[FAQ](#)

Yelp Fusion

[Introduction](#)[Business Endpoints](#)[Business Search](#)[Phone Search](#)[Transaction Search](#)[Business Details](#)[Business Match](#)[Reviews](#)[Autocomplete](#)

/businesses/search

This endpoint returns up to 1000 businesses based on the provided search criteria. It has some basic information about the business. To get detailed information and reviews, please use the Business ID returned here and refer to [/businesses/{id}](#) and [/businesses/{id}/reviews](#) endpoints.

Note: at this time, the API does not return businesses without any reviews.

Request

```
GET https://api.yelp.com/v3/businesses/search
```

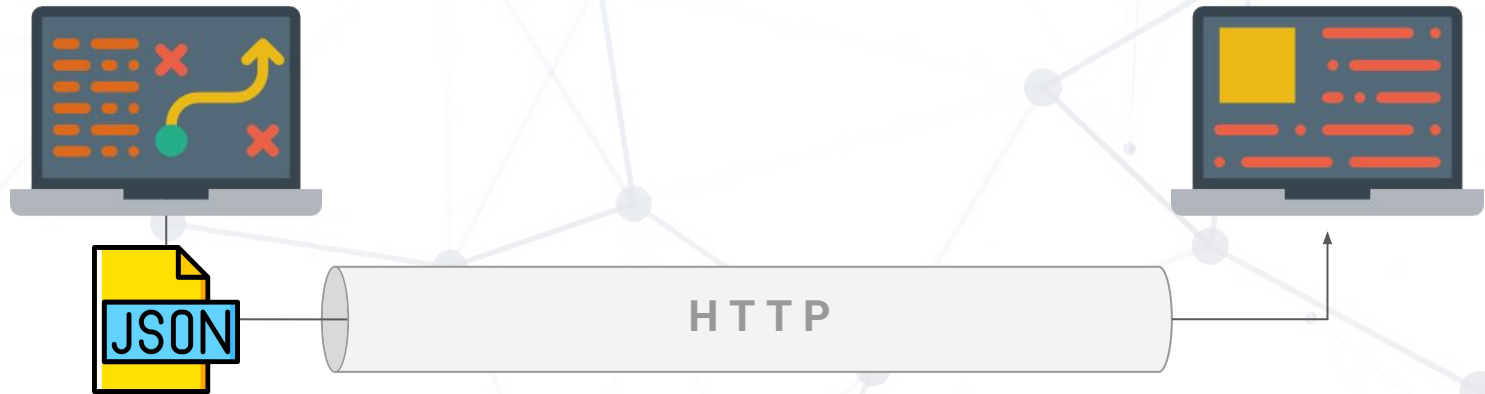
Parameters

These parameters should be in the query string.

Name	Type	Description
term	string	Optional. Search term (e.g. "food", "restaurants"). If term isn't included we search everything. The term keyword also accepts business names such as "Starbucks".
location	string	Required if either latitude or longitude is not provided. Specifies the combination of "address, neighborhood, city, state or zip, optional country" to be used when searching for businesses.
latitude	decimal	Required if location is not provided. Latitude of the location you want to search nearby.
longitude	decimal	Required if location is not provided. Longitude of the location you want to search nearby.
radius	int	Optional. Search radius in meters. If the value is too large, a <code>AREA_TOO_LARGE</code> error may be returned. The max value is 40000 meters (25 miles).
categories	string	Optional. Categories to filter the search results with. See the list of supported categories . The category filter can be a list of comma delimited categories. For example, "bars,french" will filter by Bars OR French. The category identifier should

#4

RESTful APIs ARE TIED TO HTTP



The background is a dark gray field filled with a complex network of thin, light gray lines and dots, creating a sense of a digital or molecular structure. Scattered throughout are various geometric shapes, including hexagons and triangles, some of which are slightly larger and more prominent than others. A bright orange rectangular border is centered on the page, framing the text.

MOVING FORWARD

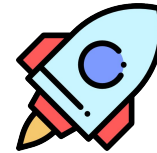
JSON - RPC



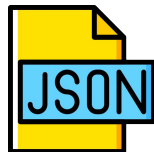
STATELESS



LIGHTWEIGHT



**TRANSPORT
AGNOSTIC**



**USES JSON AS
DATA FORMAT**



**SUPPORTS
NOTIFICATION REQUEST**

JSON - RPC : REQUEST

```
{  
  "jsonrpc": "2.0",  
  "method": "DemoRPCService.CreatePerson",  
  "params": {  
    "name": "Gianfranco",  
    "surname": "Reppucci",  
    "age": 36  
  },  
  "id": 1234567  
}
```


JSON - RESPONSE

```
{  
  "jsonrpc": "2.0",  
  "result": {  
    "id": "bcjsuge8t5ekk4rj6b4g",  
    "name": "Gianfranco",  
    "surname": "Reppucci",  
    "age": 36  
  },  
  "id": 1234567  
}
```

JSON - RPC : NOTIFICATION

```
{  
  "jsonrpc": "2.0",  
  "method": "DemoRPCService.CheckForNewPersons"  
}
```

JSON - RPC : ERROR

```
{  
  "jsonrpc": "2.0",  
  "error": {  
    "code": -32000,  
    "message": "person: invalid name or surname  
given",  
    "data": null  
  },  
  "id": 1234567  
}
```

BATCH REQUESTS



**Useful to aggregate
multiple requests**

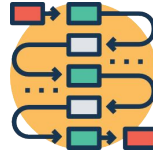


**Server is obliged to respond to
every non-Notification request**

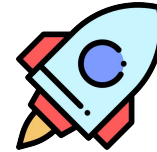
JSON-RPC ADVANTAGES



Readability



**Ease of encoding /
decoding**



**Separation from
transport protocol**

JSON-RPC DISADVANTAGES



**No binary
encoding**



**Ease to mess up
method names**



[**github.com/giefferre/jsonrpc-usage-example**](https://github.com/giefferre/jsonrpc-usage-example)

The background is a dark gray field filled with a complex network of thin, light gray lines and dots, creating a sense of a digital or molecular structure. Scattered throughout are various geometric shapes, including hexagons and triangles, some of which are slightly larger and more prominent than others. A thin, bright orange rectangular border is centered on the page, enclosing the text.

MOVING *FAST* FORWARD

gRPC



**Open Source Remote
Procedure Call protocol**



**Developed initially
at Google**

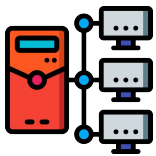


**Uses HTTP/2
for transport**

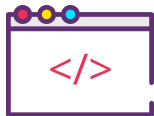


**Protocol Buffers as
Interface Description Language**

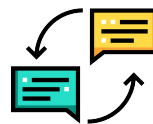
gRPC: PRINCIPLES



Services, not Objects
Messages, not
References



Built-in support for 10
languages across multiple
environments



(Bidirectional)
Streaming



Blocking /
Non Blocking



Cancellation &
Timeout



Flow
Control



Standardized
Status Codes

DEFINITION OF A SAMPLE SERVICE / 1

```
message Person {  
    string id = 1; // Unique ID for this person.  
    string name = 2;  
    string surname = 3;  
    uint32 age = 4;  
}
```

DEFINITION OF A SAMPLE SERVICE / 2

```
message CreatePersonArgs {  
    string name = 1;  
    string surname = 2;  
    uint32 age = 3;  
}
```

```
message ReadPersonArgs {  
    string id = 1;  
}
```

DEFINITION OF A SAMPLE SERVICE / 3

```
service DemoRPCService {  
    rpc CreatePerson (CreatePersonArgs) returns (Person) {}  
    rpc ReadPerson (ReadPersonArgs) returns (Person) {}  
}
```

WRITING A SERVER IN GO / 1

```
type demoRPCServer struct {  
    ...  
}  
  
func (s *demoRPCServer) CreatePerson  
    (ctx context.Context, args *CreatePersonArgs) (*Person, error) {  
    ...  
}  
  
func (s *demoRPCServer) ReadPerson  
    (ctx context.Context, args *ReadPersonArgs) (*Person, error) {  
    ...  
}
```

WRITING A SERVER IN GO / 2

```
func main() {  
    listener, err := net.Listen("tcp", ":1234")  
  
    if err != nil {  
        log.Fatalf("failed to listen: %v", err)  
    }  
  
    grpcServer := grpc.NewServer()  
    RegisterDemoRPCServiceServer(grpcServer, &demoRPCServer{})  
  
    grpcServer.Serve(listener)  
}
```

WRITING A CLIENT IN PYTHON

```
channel = grpc.insecure_channel('localhost:6000')
client = rpcservice.DemoRPCServiceStub(channel)

new_person = client.CreatePerson(
    pb.CreatePersonArgs(
        name='John',
        surname='Doe',
        age=36,
    )
)
```

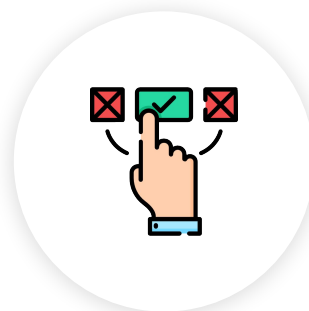



github.com/giefferre/grpc-usage-example

CONCLUSIONS



**REST concepts are solid,
RESTful implementations
aren't**



***RPC alternatives
are valid**



**You can take advantages of
some REST concepts when
developing *RPC services**



**JSON-RPC and gRPC are
modern and they can be
pretty powerful**

The background is a dark gray field filled with a complex network of thin, light gray lines connecting small dots. Scattered throughout are numerous light gray hexagons of varying sizes, some of which are slightly more prominent than others. The overall aesthetic is technical and digital.

JOIN US
careers.cubeyou.com

A blurred background image of a modern office with several people working at desks. A large orange rectangular frame is centered over the image.

THANK YOU

Gianfranco Reppucci



@giefferre

Data Engineer

cube_{you}