

Programmable Web Project


Part 2: Programmable Web

Spring 2019

- Services and APIs
- Programmable Web

Iván Sánchez Milara

Programmable Web Project. Spring 2019.




2

SERVICES AND APIS

Iván Sánchez Milara

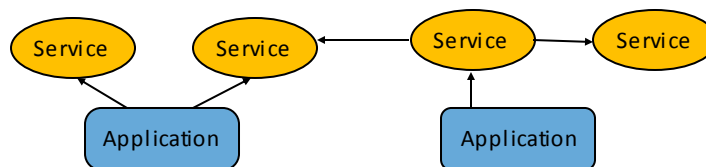
Programmable Web Project. Spring 2019.



3

Web services (I)

- Web services are logical units with **clearly defined interfaces (API)**:
 - what **functionality** they perform and
 - which **data formats** they accept and produce
- They are **application independent**
 - services can be used by other services and applications.
- Web service can incorporate the functionality of other services (**composite service**)



Iván Sánchez Milara

Programmable Web Project. Spring 2019.



3

5

Web APIs

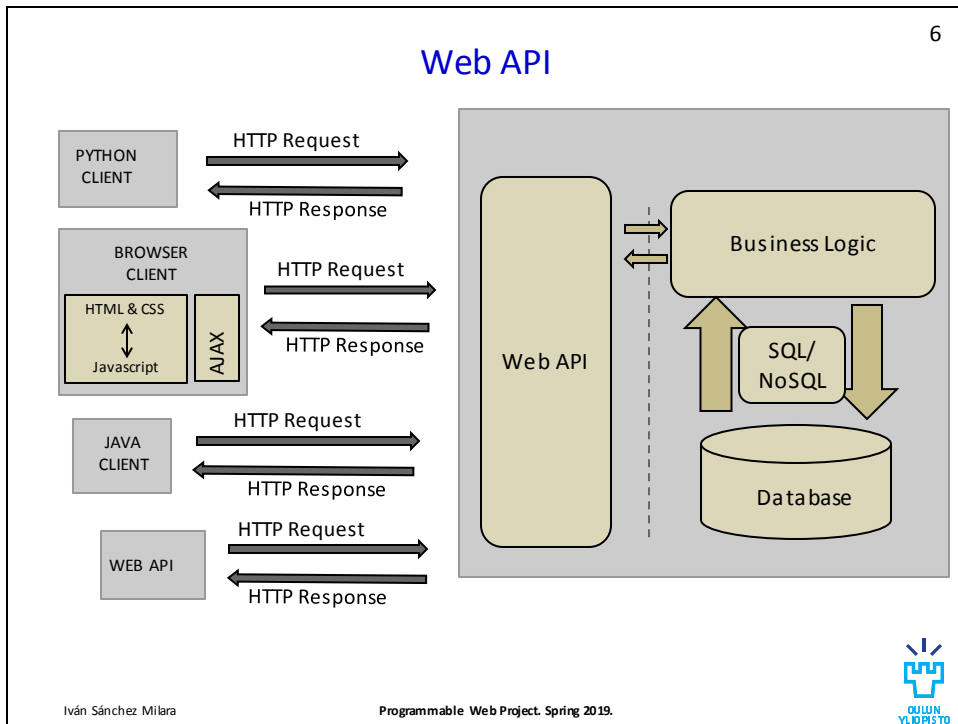
- **Application Programming Interfaces**
- Defines how the service functionality is exposed by means of one or more endpoints:
 - Protocol semantics
 - Application semantics
- **Nowadays, web service word is in disuse => We use Web API instead**

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



5




6

7

Website vs Web API

- Gist:
 - Github tool that allows sharing code and applications
 - Website at: <https://gist.github.com/>
 - API at <https://developer.github.com/v3/gists/>
 - Gist clients: <https://gist.github.com/defunkt/370230>
 - For instance, Sublime Text client: <https://github.com/condemil/Gist>

Iván Sánchez Milara Programmable Web Project. Spring 2019.



OJUN YUBISTO

7

9

Architectural styles

- RPC
- REST
 - CRUD
 - Hypermedia (HATEOAS)

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



9

10

RPC-style Web APIs

- RPC: Remote procedure call**
 - A method or subroutine is executed in another address space, without the programmer explicitly encoding the details of the remote interaction.
- An RPC-style Web API accepts an envelope full of data from its client, and sends a similar envelope back.
 - The method and the scoping information are kept inside the envelope, or on stickers applied to the envelope.
- Every RPC-style Web API defines a brand new vocabulary: method name, method parameters
- Some examples:
 - XML-RPC
 - SOAP.

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



10

11

RPC

```
<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
      <value><i4>40</i4></value>
    </param>
  </params>
</methodCall>
```

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



11

12

REST (Representational State Transfer)

- **Architectural style proposed by Roy Thomas Fielding.**
http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf
 - Does not define an architecture but requirements for the architecture
- **Representation**
 - Resource-oriented: operates with resources.
 - **Resource:** Any piece of information that can be named. Identified generally by URL
- **State:**
 - value of all properties of a resource at the certain moment.
- **Transfer:** State can be transferred
 - Clients can:
 - 1) retrieve the state of a resource and
 - 2) modify the state of the resource
 - UNIFORM interface

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



12


13

REST APIs

- CRUD
 - Most extended approach. Majority of Web APIs nowadays
 - Not follow strictly REST principles
 - More on this next lecture
- Hypermedia
 - Follows strictly REST principles

Iván Sánchez Milara

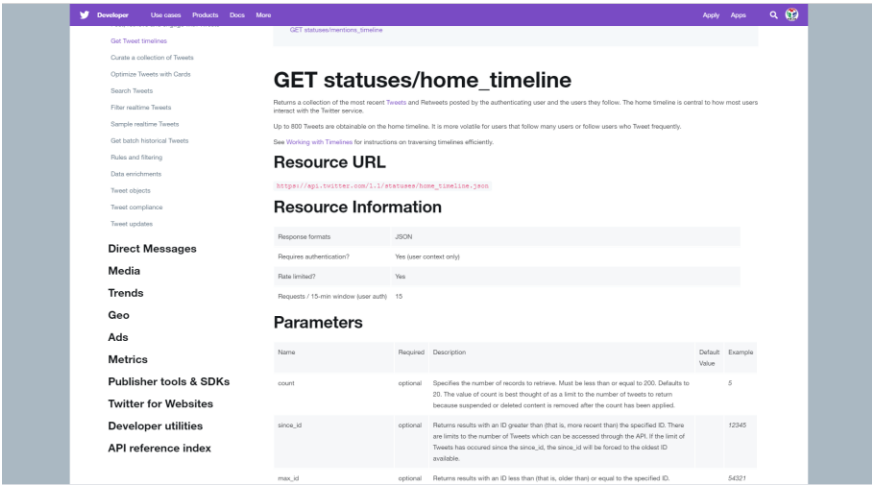
Programmable Web Project. Spring 2019.



13

14

Twitter API




The screenshot shows the Twitter Developer API documentation for the endpoint `GET statuses/home_timeline`. It includes a sidebar with navigation links, a description of the endpoint, the resource URL, response information (JSON, authentication required, rate limit), and a table of parameters.

Name	Required	Description	Default Value	Example
count	optional	Specifies the number of records to retrieve. Must be less than or equal to 200. Defaults to 20. The value of count is best thought of as a limit to the number of tweets to return because suspended or deleted content is removed after the count has been applied.	5	
since_id	optional	Returns results with an ID greater than (that is, more recent than) the specified ID. There are limits to the number of Tweets which can be accessed through the API. If the limit of Tweets has occured since the since_id, the since_id will be forced to the oldest ID available.		12345
max_id	optional	Returns results with an ID less than (that is, older than) or equal to the specified ID.		54321

<https://developer.twitter.com/en/docs.html>

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



14

15

What about current Web applications (RPC or CRUD)?

- Need excessive documentation
 - Exhaustive description of required protocol: HTTP methods, URLs ...
- When an application API changes, clients break and have to be fixed
 - For instance a change in the object model in the server or the URL structure => change in the client.
- Clients need to store a lot of information
 - Protocol semantics
 - Application semantics
 - Application workflow

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



15

16

Web vs Programmable Web

- The **Programmable Web** use the same technologies and communication protocols as the WWW in order to cope with current problems.
- Current differences
 - The data is not delivered necessarily for human consumption (M2M)
 - Nowadays an **specific client** is needed per application at least until we solve the problems derivated from the **semantic challenge**
 - A client can be implemented using any programming language
 - Data is encapsulated and transmitted using any serialization languages such as **JSON, XML, HTML, YAML**

Iván Sánchez Milara

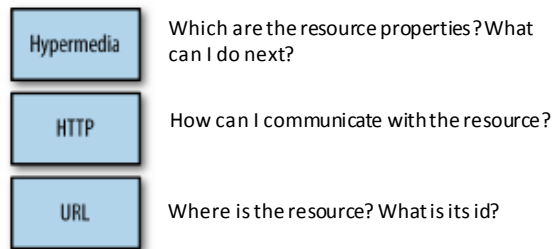
Programmable Web Project. Spring 2019.



16

17

Programmable Web



Web:

- Targeted to humans
- One client

Programmable Web:

- Targeted to machines
- Heterogeneous clients

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



17

Programmable Web Project Part 3: RESTful Web APIs

Spring 2018

- ROA Principles
- RESTful Web APIs
- Designing RESTful Web APIs
- Resource Oriented design vs hypermedia driven design

Iván Sánchez Milara

Programmable Web Project. Spring 2019.

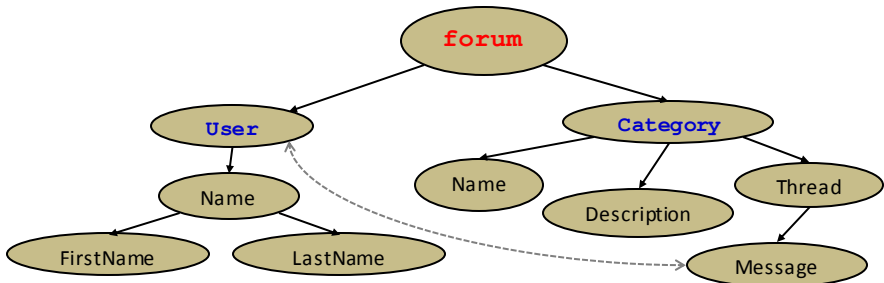


19

FORUM EXAMPLE



Forum resource representation



24

INTRODUCTION TO ROA

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



24

25

REST (Representational State Transfer)

- Architectural style proposed by Roy Thomas Fielding.
http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf
- Representation
 - Resource-oriented: operates with resources.
- State:
 - value of all properties of a resource at the certain moment.
- Transfer: State can be transferred
 - Clients can:
 - 1) retrieve the state of a resource and
 - 2) modify the state of the resource

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



25

26

REST Constraints

- Client-server architecture
- Stateless
- Cacheability
- Layered system
- Code on demand
- Uniform interface

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



26

28

ROA Introduction

- **Resource Oriented Architecture (ROA)**
 - Architecture for creating Web APIs
 - It conforms the REST design principles
 - **Base technologies: URLs, HTTP and Hypermedia**
- Richardson and Ruby [1] call the Web services with a ROA architecture **RESTful Web services**
 - In the course we call them **RESTful Web APIs**
- RESTful Web services work in a similar way as the Web (“Programmable Web”)
 - However not necessarily for human consumption

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



28

29

ROA. Resources and manipulation

- **Resource:**

- Anything important enough to be referenced as a thing itself
 - For example: List of the libraries of the city of Oulu, the last software version of Windows, the relation between two friends, the result of factorizing a number
- Each resource is identified by a unique id:
 - Uniform Resource Identifier (URI)

- We operate with resources **representations:**

- **State of the resource** at certain time
- Encoded in JSON, HTML or other hypermedia formats

- **HTTP requests** are used to manipulate the state of a resource

- **URI:** Identifies the resource to manipulate
<http://www.forum.com/users/Nicky>
 Scope
- **HTTP method:** The action to be performed to manipulate the resource

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



29

30

ROA pillars

Four properties:

- 1) Addressability
- 2) Uniform interface
- 3) Statelessness
- 4) Connectedness

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



30

31

Addressability

- Exposes the interesting aspects of its data set as resources
 - Each resource is exposed using its URI
 - The URI can be copied, pasted and distributed
 - Example:
 - `http://forum.com/users/user1` refers to the information of the user of the Forum
 - I can send this URI by email, and the receiver can access this information by copying this URI into his/her browser

Iván Sánchez Milara

Programmable Web Project. Spring 2019.

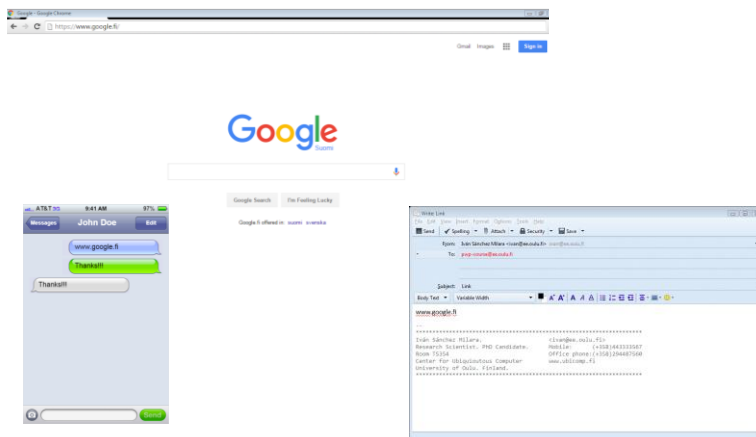


31

32

Addressability in WWW

- The WWW is addressable



Iván Sánchez Milara

Programmable Web Project. Spring 2019.



32

33


Uniform interface (I)

- Every API uses the same methods with the same meanings
 - Without a uniform interface, clients have to learn how each API is expected to get and send information
- ROA uses uniform interface provided by HTTP to act over the resource provided in the URI

Method	Description
GET	Returns the resource representation
PUT	Changes the state of the resource Creates a new resource when the URL is known
POST	Create subordinate resources (no URL known beforehand) Appends information to the current resource state
DELETE	Removes a resource from the server

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



33


35

Uniform interface (II)

- **PATCH** <http://tools.ietf.org/html/rfc5789>
 - Partial edition/modification of a resource
 - Client and server must agree on a new media type for patch documents
 - RFC 6902: proposed standard patch format for JSON.
 - Send a **diff** of the resource representation. Changes to be done to the resource.
 - Content-Type: `application/json-patch+json`
 - ```
[{ "op": "remove", "path": "/a/b/c" }, { "op": "add", "path": "/a/b/c", "value": ["foo", "bar"] }, { "op": "replace", "path": "/a/b/c", "value": 42 }]
```

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



35

36

## Uniform interface (III)

- **URI:** <http://forum.com/messages/msg-3>

```
<msg:Message messageID="msg-3">
 <msg:Title>Edmonton's goalie</msg:Title>
 <msg:Body>Does anyone know where Jussi Markkanen used to play before
 he came to Edmonton Oilers? He was excellent in the Stanley Cup finals
 last season! Too bad they lost...</msg:Body>
 <msg:Sent>2005-09-04T19:22:39+02:00</msg:Sent>
 <msg:SenderIP>217.119.25.162</msg:SenderIP>
 <msg:Registered userID="user-7">
 <user:Nickname>HockeyFan</user:Nickname>
 <user:Avatar file="avatar_7.jpg"/>
 <atom:link rel="self" href="http://forum/users/HockeyFan"/>
 </msg:Registered>
</msg:Message>
```

- **GET:** Retrieves this representation
- **DELETE:** Removes the message with id «msg-3» from the server
- **PUT:** Edits the message with id «msg-3». Title, Body, Sent, SenderIP, and Registered could be modified and **MUST** be included in the request body (The complete representation is sent and it replaces the old one)
- **POST:** Add a response to the message with id «msg-3» (subordinate resource). The body of the request should include the new message

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



36

37

## Uniform interface in WWW

- Only GET and POST supported in HTML
- Rest of HTTP methods supported through Javascript

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



37

38

## Statelessness (I). State concept.

### •Resource state:

- A resource representation that is exchanged between server and client:
  - The values of information items belonging to the resource
  - Links to related resources and future states of applications (including protocol information)
- Same for all the clients making simultaneous requests
- Kept in the server

### •Application state:

- Snapshot of the entire system at a particular instant
  - What I have done till now and what can I do in the future (application workflow)
- Future possible application states are informed in the resource representation sent by the server.
- Kept in the client until it can be used to create, modify or delete a resource

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



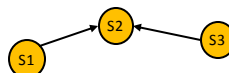
38

39

## Statelessness (II)

### •Every HTTP request happens in complete isolation (STATELESS)

- **Server never operates based on information from previous requests, does not store application state**
  - Eg: In a photo album application if I am in “picture 3” I cannot request the “next picture” but “picture 4”
- Server considers each client request in isolation and in terms of the current resource state
  - Service only needs to care about application state when client is making a request. The rest of the time, it doesn’t even know the client exist.
- Client handles the application workflow



Iván Sánchez Milara

Programmable Web Project. Spring 2019.



39



40

## Statelessness in WWW

- Originally the WWW is statless
  - GET an URL always should return same website
- Multiple applications needs state information (login, last accessed, visited pages)
  - Cookies
  - Session id in URL

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



40

41

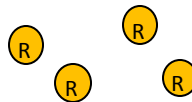
## Connectedness (I)

- Resource representation MUST contain reference (links) to other resources
  - Including the relation among resources and optionally how to access them

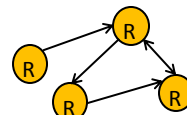
R=Resource



Service exposes everything  
under single URI  
not addressable, not connected



Service is addressable, but not  
connected



Service is addressable  
and connected

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



41

42

## Connectedness (II)

```

<msg:Thread>
 <msg:Message messageID="msg-3">
 <atom:link rel="self" href="http://forum/messages/msg-3"></atom:link>
 <msg:Title>Edmonton's goalie</msg:Title>
 <msg:Registered userID="user-7">
 <user:Nickname>HockeyFan</user:Nickname>
 <user:Avatar file="avatar_7.jpg"/>
 <atom:link rel="self" href="http://forum/users/HockeyFan"/>
 </msg:Registered>
 </msg:Message>
 <msg:Message messageID="msg-7" replyTo="msg-3">
 <atom:link rel="self" href="http://forum/messages/msg-7"/>
 <msg:Title>History</msg:Title>
 <atom:link rel="http://forum/rels/parent-message"
href="http://forum/messages/msg-3"/>
 <msg:Registered userID="user-1">
 <user:Nickname>Mystery</user:Nickname>
 <user:Avatar file="avatar_1.png"/>
 <atom:link rel="self" href="http://forum/users/Mystery"/>
 </msg:Registered>
 </msg:Message>
</msg:Thread>

```

A representation of the message with id «msg-3»

A representation of user with nickname «HockeyFan»

A representation of the parent message of «msg-7»

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



42

43

## Connectedness in WWW

- WWW is connected
  - Access and modification of any resource state: following links or filling forms

```


 See the latest messages

<form action="http://www.youtypeitwepostit.com/messages"
method="post">
 <input type="text" name="message" value=""
required="true" />
 <input type="submit" value="Post" />
</form>

```

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



43

# RESTFUL WEB APIS. HYPERMEDIA.

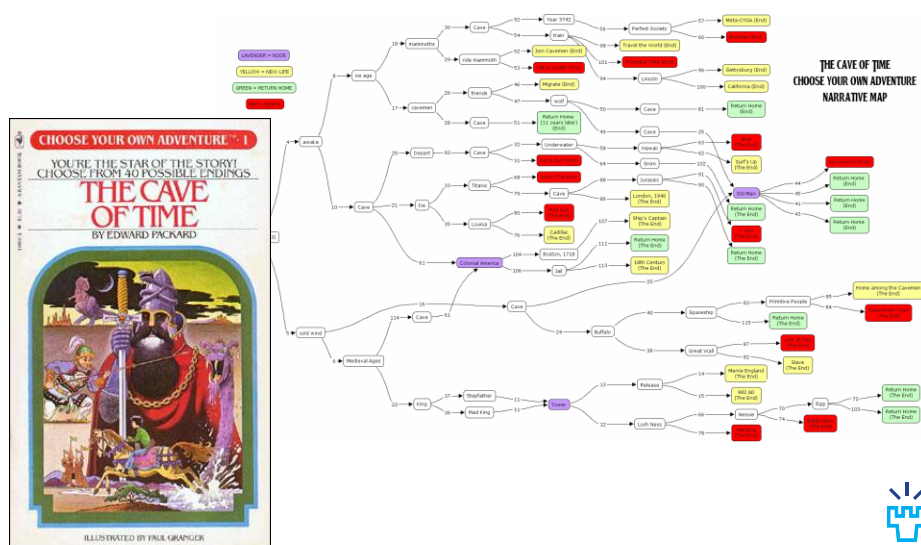
Iván Sánchez Milara

Programmable Web Project. Spring 2019.



## HATEOAS

## Hypermedia As The Engine Of Application State



Iván Sánchez Milara

Programmable Web Project. Spring 2019.

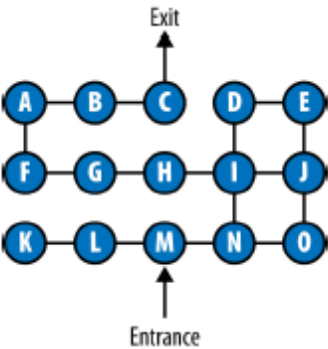


# HATEOAS

- Hypermedia contains:
  - Data
  - **Hypermedia controls**
    - Links
    - Protocol specifications
- Ideally, client just need the entry point to the service
  - The rest of the URI's should be discovered through the **hypermedia controls**
    - Workflow always informed from the server
    - Server informs about possible future states via hypermedia controls
  - Well designed RESTful APIs permit **modifying the server architecture (e.g. URL structure) and data model without breaking the clients**



# HATEOAS



RESTful Web APIs, Richardson, Amundsen and Ruby

```
<maze version="1.0">
 <cell href="/cells/M" rel="current">
 <title>The Entrance Hallway</title>
 <link rel="east" href="/cells/N"/>
 <link rel="west" href="/cells/L"/>
 </cell>
</maze>
```

Server job is to describe mazes so clients can engage with them without dictating the goals



51

## Semantic challenge (I)

- Browser does not understand problems domain.
  - Humans process information coming from the server and decide on future actions
- In M2M this is not possible:
  - Machines NEED to understand the problem domain
  - How can we program a computer to make the decisions about which links to follow?
- **This is the biggest challenge in web API design using hypermedia: bridging the semantic gap between understanding a document's structure and understanding what it means.**

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



51

52

## Semantic Challenge (II)

### Semantic gap

- The gap between the structure of a document and its real-world meaning

#### Protocol semantics

- What kind of actions a client can perform?
- Usually solved using **hypermedia control**

#### Application semantics

- How the representation is explained in terms of real world concepts.
- Same word might have different meanings in different contexts.
  - E.g. **time**:
    - Preparation time if we are using a recipe book
    - Workout duration if we are building a gym agenda
    - Time of the day if we are using a calendar

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



52

53

## Semantic challenge (III)

Two ways of communicating semantics to the client

**Media Types**

**Profiles**

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



53

54

## Media types

- Defines the format of the message
  - Sometimes include protocol and application semantics
- General purpose media-types with hypermedia.
  - Allows personalizing the protocol semantics and application level semantics
  - **HAL, HTML, SIREN, MASON**

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



54

55

## Hypermedia control

- MUST contain the following information.
  - The URI of the remote resource
  - The relation of the current resource with the remote one
  - Usually, protocol information
    - E.g. which method I need to execute / what is the format of the request body.

```

entities : [
 {
 "class" : ["switch"],
 "href" : "/switches/4",
 "rel" : ["item"],
 "properties" : { "position" : { "up" } },
 "actions" : [
 {
 "name" : "flip",
 "href" : "/switches/4",
 "title" : "Flip the mysterious switch.",
 "method" : "POST"
 }
]
 }
]

```

Siren example

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



55

56

## Media Types: Collection+JSON

Mime type: [application/vnd.collection+json](http://amundsen.com/media-types/collection/)

Link: <http://amundsen.com/media-types/collection/>

```

{
 "collection": {
 {
 "version": "1.0",
 "href": "http://www.youtypeitweposit.com/api/",
 "items": [
 { "href": "http://www.youtypeitweposit.com/api/messages/21818525390699506",
 "data": {
 { "name": "text", "value": "Test." },
 { "name": "date_posted", "value": "2013-04-22T05:33:58.930Z" }
 },
 "links": []
 },
 { "href": "http://www.youtypeitweposit.com/api/messages/3689331521745771",
 "data": {
 { "name": "text", "value": "Hello." },
 { "name": "date_posted", "value": "2013-04-20T12:55:59.685Z" }
 },
 "links": []
 },
 "template": {
 "data": {
 { "prompt": "Text of message", "name": "text", "value": "" }
 }
 }
]
 }
 }
}

```

LIST OF HYPERMEDIA FORMATS IN APPENDIX A: Hypermedia formats

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



56

## Profile

- Explains the document semantics that are not covered by its media type.

- A profile describes the exact meaning of each **semantic descriptor**

```
Jenny Gallegos
```

– “A profile is defined to not alter the semantics of the resource representation itself, but to allow clients to learn about additional semantics[...] associated with the resource representation, in addition to those defined by the media type” [RFC 6906]

- Defined in a text document or using a specific description language: ALPS, JSON-LD, RDF-Schema, XMDP



## Twitter API

Developer

Use cases

Products

Docs

More

GET statuses/home\_timeline

Get Tweet timelines

Curate a collection of Tweets

Optimize Tweets with Cards

Search Tweets

Filter realtime Tweets

Sample realtime Tweets

Get batch historical Tweets

Rules and filtering

Data enrichments

Tweet objects

Tweet compliance

Tweet updates

Direct Messages

Media

Trends

Geo

Ads

Metrics

Publisher tools & SDKs

Twitter for Websites

Developer utilities

API reference index

GET statuses/home\_timeline

Return a collection of the most recent Tweets and Retweets posted by the authenticating user and the users they follow. The home timeline is central to how most users interact with the Twitter service.

Up to 500 Tweets are obtainable on the home timeline. It is more volatile for users that follow many users or follow users who Tweet frequently.

See Working with Timelines for instructions on traversing timelines efficiently.

Resource URL

[https://api.twitter.com/1.1/statuses/home\\_timeline.json](https://api.twitter.com/1.1/statuses/home_timeline.json)

Resource Information

Response formats

JSON

Requires authentication?

Yes (user context only)

Rate limited?

Yes

Requests / 15-min window (user auth)

15

Parameters

Name	Required	Description	Default Value	Example
count	optional	Specifies the number of records to retrieve. Must be less than or equal to 200. Defaults to 20. The value of count is best thought of as a limit to the number of tweets to return because suspended or deleted content is removed after the count has been applied.	5	
since_id	optional	Returns results with an ID greater than (that is, more recent than) the specified ID. There are limits to the number of Tweets which can be accessed through the API. If the limit of Tweets has occured since the since_id, the since_id will be forced to the oldest ID available.		12345
max_id	optional	Returns results with an ID less than (that is, older than) or equal to the specified ID.		54321

<https://developer.twitter.com/en/docs.html>





59

## Summary

- REST APIs must be hypertext driven according to Fielding:  
<http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>
- HATEOAS
  - Hypermedia describes the actions that you can perform with the resources.
    - Client does not memorize operations nor workflow. Everything is in the messages
- Documentation reduced drastically: messages are documented by themselves
  - A REST API should spend almost all of its descriptive effort in defining the media type used for representing resources and driving application states
- Easier to create general clients
  - Example: RSS and Atom PUB. Multiple clients can read the same RSS feed.

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



59

60

## DESIGN OF RESTFUL WEB APIS USING ROA

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



60

61

## RESTful Web services design steps

1. Figure out the data set
2. Split the data set into resources
  - Create Hierachy
3. Name the resources with URIs
4. Establish the relations and possible actions among resources
5. Expose a subset of the uniform interface
6. Design the resource representations using hypermedia formats
  1. Define the media types
  2. Define the profiles
7. Define protocol specific attributes
  - E.g. Headers, response code
8. Consider error conditions: What might go wrong?

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



61

88

## HYPERMEDIA DRIVEN DESIGN

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



88

## Resource driven vs Hypermedia driven

### • Resource driven design

- MOST utilized approach nowadays when people talk about REST
- Nouns is the most important

### • Hypermedia driven design

- ACTION is the most important
- Acknowledges that the state transitions are even more important than the state itself.
  - I want to do a thing.
  - Which verbs should I use to do that?
- Previous state transitions will provide 'affordances' that indicates what actions I can perform next and a way of figuring out more information about those affordances if we do not know it already.

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



89

## Design process (I)

1. Evaluate processes
2. Create state machine
3. Evaluate media types
4. Create or choose media types
5. Implementation!
6. Refinements

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



91

92

## Design process (II)

- Documenting a REST API => defining the media types.

*“A REST API should spend almost all of its descriptive effort in defining the media type(s) used for representing resources and driving application state, or in defining extended relation names and/or hypertext-enabled mark-up for existing standard media types. Any effort spent describing what methods to use on what URIs of interest should be entirely defined within the scope of the processing rules for a media type (and, in most cases, already defined by existing media types)”*

**Roy Fielding.** [REST APIs must be hypertext-driven](#)

- The media type is the only sort of contract between the client and the server

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



92

94

## Hypermedia driven APIs examples

- **Skype for business:**
  - <https://msdn.microsoft.com/en-us/skype/ucwa/hypermedia>
- **Paypal** is promoting the use of Hypermedia in their REST API:
  - <https://developer.paypal.com/docs/api/overview/>
  - <https://developer.paypal.com/docs/integration/direct/paypal-rest-payment-hateoas-links/>
- **Amazon AppStream:**
  - <http://docs.aws.amazon.com/appstream/latest/developerguide/api-reference.html>
- **Foxycart:**
  - <https://api.foxycart.com/docs#>
- **Zalando:**
  - <http://zalando.github.io/restful-api-guidelines/index.html>

Iván Sánchez Milara

Programmable Web Project. Spring 2019.



94

## References

1. "RESTful Web Services" by Leonard Richardson and Sam Ruby
2. "RESTful Web APIs" by Leonard Richardson, Mike Amundsen and Sam Ruby
3. "RESTful Web Services Cookbook" by Subbu Allamaraju
4. "REST in practice. Hypermedia and Systems Architecture" by Jim Webber, Savas Parasidis and Ian Robinson.
5. Representational State Transfer (REST), Roy Thomas Fielding. Available at [http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding\\_dissertation.pdf](http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf)
6. "Peer-to-Peer Systems and Applications" Ralf Steinmetz Klaus Wehrle (Eds.)  
Available at <http://www.springerlink.com/content/g6h805426g7t/#section=586017&page=1>
7. ATOM <http://www.ietf.org/rfc/rfc4287.txt>
8. HTTP 1.1 <http://tools.ietf.org/html/rfc2616>
9. JSON <http://www.json.org/>