# Programmable Web Project
# Part 2: Programmable Web
Spring 2019

- **Services and APIs**
- **Programmable Web**

1

---

2

# SERVICES AND APIS

2

1

---

# 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**)

3

---
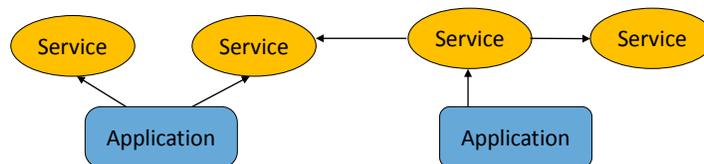
# Web services (II)

- Web services are not prepared to human consumption (in contrast to websites).

  - Web services require an architectural style to provide **clear and unambiguous interaction** (clearly defined interfaces), because there's no smart human being on the client end to keep track.

4

---
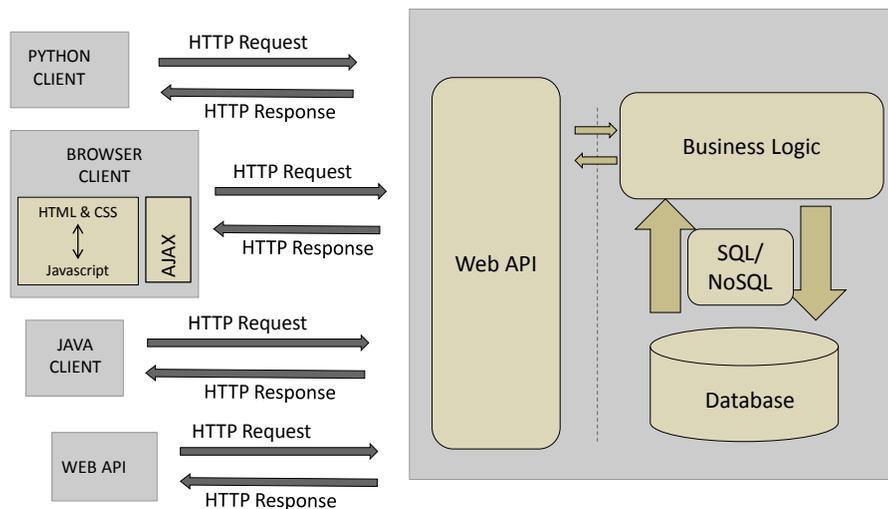
7

# 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

7

8

# Web APIs Examples.

- **Flickr** Web API can be used to retrieve and upload photos from/to the Flickr sharing service. Pictures can be filtered using multiple criteria.
  https://www.flickr.com/
  https://www.flickr.com/services/api/

- **Blurb!** is a web application that makes easy design, publish, market and sell professional-quality books.
  http://www.blurb.com/flickr

- **Glimmr** is a Flickr viewer for Android. It uses Flickr API to collect data.
  https://play.google.com/store/apps/details?id=com.bourke.glimmr

- Much more in http://www.programmableweb.com

8

# Architectural styles

- RPC

- REST
  - CRUD
  - Hypermedia (HATEOAS)

Iván Sánchez Milara

**Programmable Web Project. Spring 2019.**

# 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.**

# RPC

```xml
<?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

# 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
- **Re**presentation
    - Resource-oriented: operates with resources.
        - **Resource**: Any piece of information that can be named. Identified generally by URL
- **S**tate:
    - value of all properties of a resource at the certain moment.
- **T**ransfer: 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



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

Iván Sánchez Milara          **Programmable Web Project. Spring 2019.**

14

---

15

# Hypermedia driven Web APIs

- Follows strictly Fielding dissertation principles.
  - REST APIs must be hypertext driven: http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven
- Uses Hypermedia as the Engine of the Application State
  - 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.**

15

---

16

# 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.**

16

17

# 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 derived 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.**

17

---

18

# Programmable Web



Hypermedia — Which are the resource properties? What can I do next?

HTTP — How can I communicate with the resource?

URL — Where is the resource? What is its id?
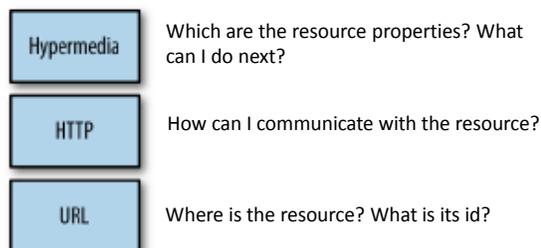
**Web:**
- Targeted to humans
- One client

**Programmable Web:**
- Targeted to machines
- Heterogeneous clients

Iván Sánchez Milara                    **Programmable Web Project. Spring 2019.**

18

# 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

---

20

# Table of contents

1. ROA
   - REST
   - Resources
   - Properties:
     - Addressability
     - Uniform Interface
     - Statelesness
     - Connectedness
       - HATEOAS

2. RESTful Web APIs
   - RESTful and hypermedia
   - Designing RESTful Web APIs

3. Hypermedia driven APIs

Iván Sánchez Milara

**Programmable Web Project. Spring 2019.**

20

21

**FORUM EXAMPLE**

Iván Sánchez Milara                    **Programmable Web Project. Spring 2019.**

21

---

22

## Forum resource representation
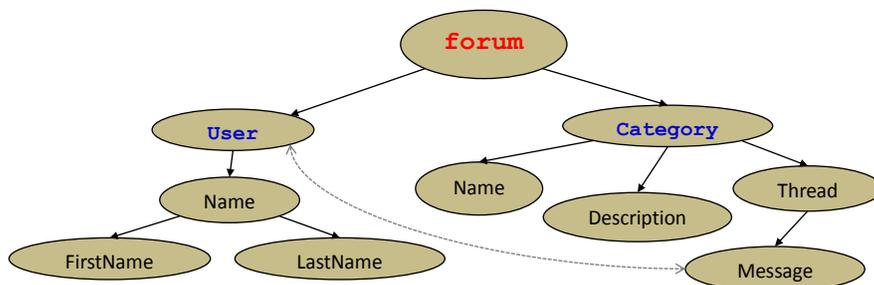


Iván Sánchez Milara                    **Programmable Web Project. Spring 2019.**

22

11

23

# INTRODUCTION TO ROA

**Programmable Web Project. Spring 2019.**

23

---

24

# 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.
- **S**tate:
  - value of all properties of a resource at the certain moment.

- **T**ransfer: 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.**

24

12

# REST Constraints

- Client-server architecture

- Stateless

- Cacheability

- Layered system

- Code on demand

- Uniform interface

Iván Sánchez Milara          **Programmable Web Project. Spring 2019.**

25

# REST

Roy T. Fielding: Understanding the REST Style

https://www.youtube.com/watch?v=w5j2KwzzB-0

Iván Sánchez Milara          **Programmable Web Project. Spring 2019.**

26

27

# 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.**

27

---

28

# 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.**

28

29

# ROA pillars

Four properties:

1) Addressability
2) Uniform interface
3) Statlessness
4) Connectedness

Iván Sánchez Milara                    **Programmable Web Project. Spring 2019.**

29

30

# 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 <u>email</u>, and the receiver can access this information by copying this URI into his/her <u>browser</u>

Iván Sánchez Milara                    **Programmable Web Project. Spring 2019.**

30

31

# Addressability in WWW

- The WWW is addressable
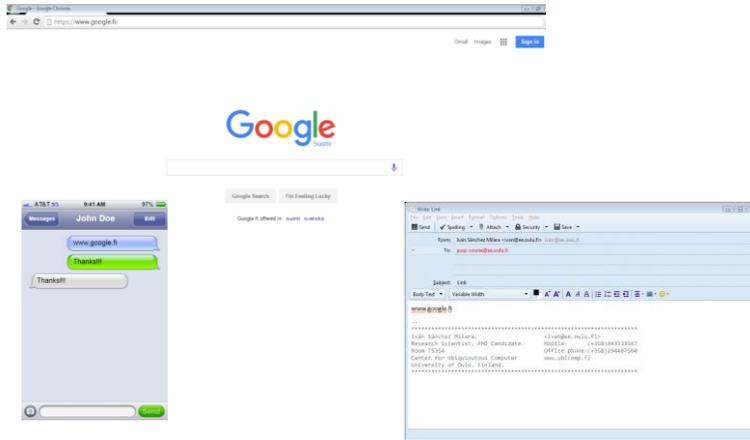


Iván Sánchez Milara          **Programmable Web Project. Spring 2019.**

31

---

32

# 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.**

32

---

# Uniform interface

- **POST**
  - <u>Creates a subordinate resource</u>, that is, a resource existing as a children of another resource
    - Difference with PUT:
      - POST creates new resources when the client does not know their URI
    - Example: A client wants to create a new message in the forum
      - The forum backend generates itself APIs for new messages. Client does not know in advanced.
      - POST HTTP request to **/forum/categories/categoryName**
      - The server creates the message and assigns the URI, e.g., **/forum/messages/message5**
    - The server sends the URI of the new resource back to the client in the HTTP Response headers
  - <u>Appends information to the current resource state</u>
    - Example: Adding lines to a log entry
    - Difference with PUT:
      - POST modifies just part of the resource state

Iván Sánchez Milara                    **Programmable Web Project. Spring 2019.**

33

---

# 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.**

34

35

# 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.**

35

36

# 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.**

36

18

# 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 <u>in the server</u>
- **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 <u>in the client until it can be used to create, modify or delete a resource</u>

Iván Sánchez Milara     **Programmable Web Project. Spring 2019.**

37

# 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
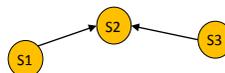
Iván Sánchez Milara     **Programmable Web Project. Spring 2019.**

38

# 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.**

39

# 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
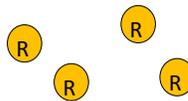
Service is addressable, but not connected
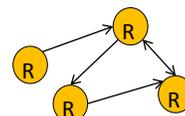
Service is addressable and connected

Iván Sánchez Milara          **Programmable Web Project. Spring 2019.**

40

1/20/2019

## Connectedness (II)

41

> 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*»

```
<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/Mistery"/>
      </msg:Registered>
   </msg:Message>
</msg:Thread>
```

Iván Sánchez Milara

**Programmable Web Project. Spring 2019.**

OULUN YLIOPISTO

41

---

42

## Connectedness in WWW

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

```
<a href="http://www.youtypeitwepostit.com/messages/">
        See the latest messages
</a>


<img rel="icon" src="http://www.example.com/logo.png" />


<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.**

OULUN YLIOPISTO

42

# RESTFUL WEB APIS.
# HYPERMEDIA.

Iván Sánchez Milara                    **Programmable Web Project. Spring 2019.**

43

## Richardson Maturity Model

**Glory of REST**

Level 3: Hypermedia Controls

Level 2: HTTP Verbs

Level 1: Resources

Level 0: The Swamp of POX

Iván Sánchez Milara                    **Programmable Web Project. Spring 2019.**

44
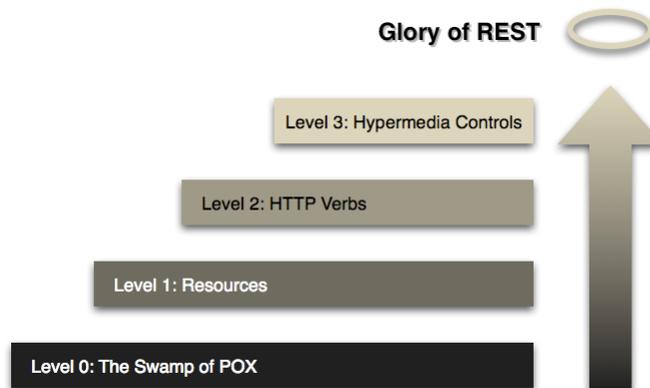
# RESTful and Hypermedia

- PROGRAMMABLE WEB goal:
  - Achieve a machine to machine understanding similar the client-server understanding in the web.
    - E.g. Modifying the object model in the server does not affect the server
- RESTful designers forgot one of the principles of REST:
  - *What needs to be done to make the REST architectural style clear on the notion that hypertext is a constraint? In other words, if the engine of application state (and hence the API) is not being driven by hypertext, then it cannot be RESTful and cannot be a REST API.*
    - ***Roy Fielding.*** REST APIs must be hypertext-driven
- Client does not need to know beforehand workflows or request formats. All that information comes on the server responses.

Iván Sánchez Milara                    **Programmable Web Project. Spring 2019.**

45

# HATEOAS
## Hypermedia As The Engine Of Application State



Iván Sánchez Milara                    **Programmable Web Project. Spring 2019.**

46
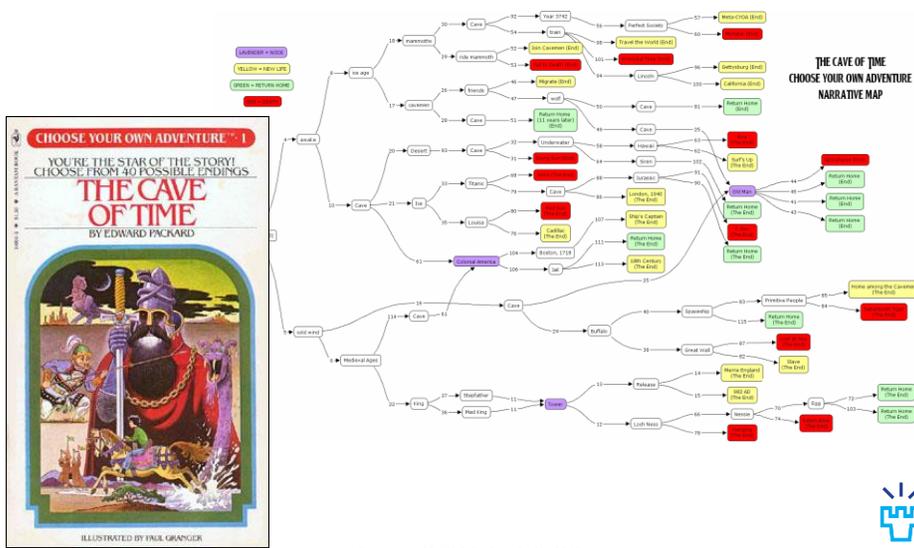
47

# HATEOAS (I)

- 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**

Iván Sánchez Milara          **Programmable Web Project. Spring 2019.**

47

48

# HATEOAS (II)



```
<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>
```
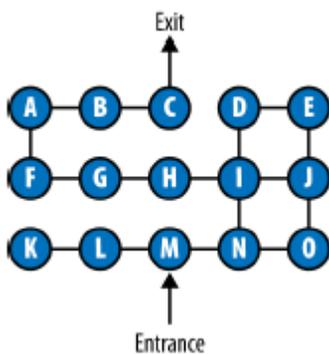
RESTful Web APIs. Richardson, Amundsen and Ruby

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

Iván Sánchez Milara          **Programmable Web Project. Spring 2019.**

48

49

# HATEOAS (III)

- **Hypermedia As The Engine Of Application State**
  - Hypermedia
    - Techniques to integrate content in multiple formats (text, image, audio, video…) in a way that all content is connected and accessible to the user
  - Engine of Application State
    - **Hypermedia:** Core and driving force of the transformation of the application state
    - The server manipulates the client's state by sending a hypermedia "menu" containing options from which the client is free to choose.
    - Hypermedia contains:
      - Data
      - **Hypermedia controls:**
        » Enables the state transitions, guiding clients future requests.
        » Provides protocol semantics: which URL, method, request body is required to perform an application state transition.
        » Server warrantees workflow control. The hypermedia control:
          » Describe relationship among resources
          » Explain who the client should integrate the response into the workflow
        » In HTML `<a>`, `<img>`, `<script>` and Link header are hypermedia control
- **Hypermedia drives the application state**
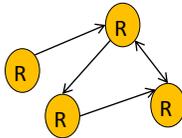
Iván Sánchez Milara          **Programmable Web Project. Spring 2019.**

49

---

50

# 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.**

50

# 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.**

51

# Semantic challenge (III)

Two ways of communicating semantics to the client

**Media Types**                              **Profiles**

Iván Sánchez Milara          **Programmable Web Project. Spring 2019.**

52

# 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.**

53

# 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.**

54

# Media Types: Collection+JSON

Mime type: application/vnd.collection+json
Link: http://amundsen.com/media-types/collection/

```
{ "collection":
    {
        "version" : "1.0",
        "href" : "http://www.youtypeitwepostit.com/api/",
        "items" : [
          { "href" : "http://www.youtypeitwepostit.com/api/messages/21818525390699506",
              "data" : [
                  { "name" : "text", "value" : "Test." },
                  { "name" : "date_posted", "value" : "2013-04-22T05:33:58.930Z" }
              ],
              "links" : []
            },
            { "href" : "http://www.youtypeitwepostit.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 FORMATSIN APPENDIX A: Hypermedia formats

Iván Sánchez Milara                    **Programmable Web Project. Spring 2019.**

55

---

# Profile

- Explains the document semantics that are not covered by its media type.
    - A profile describes the exact meaning of each **semantic descriptor**
    ```
    <span class="fn">Jenny Gallegos</span>
    ```

    – *"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

Iván Sánchez Milara                    **Programmable Web Project. Spring 2019.**

56

57

# Twitter API



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

Iván Sánchez Milara                **Programmable Web Project. Spring 2019.**

57

---

58

# 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.**

58

59

# DESIGN OF RESTFUL WEB APIS USING ROA

Iván Sánchez Milara          **Programmable Web Project. Spring 2019.**

59

---

60

## 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.**

60

61

# Step 1 - Figure out the data set

- Define the concepts that you are going to expose in the Web API
- Describe the relations between them

**Forum example**
- ❑ Forum API permits <u>users</u> to publish new <u>messages</u>
- ❑ Users can post messages to different <u>categories</u>
- ❑ Users can reply to other users' messages
- ❑ Every user has a <u>public profile</u> and a <u>private profile</u>
  - ➤ Every user can check other users' public profiles
  - ➤ A private profile is shown only to that user's friends
- ❑ Users can check the last messages anyone has posted and commented
- ❑ Users can search messages in the forum using several criteria: keywords, user, popularity, date published, date commented, ...

Iván Sánchez Milara                    **Programmable Web Project. Spring 2019.**

61

---

62

# Step 2 - Split the data into resources (I)

- RESTful Web services expose 3 kinds of resources:
  - <u>Predefined one-off resources </u>for important aspects of the application
    - They are usually repository for other resources.
      - Also known as **Collections.**
    - They cannot be deleted and their state cannot be modified directly
      - State only changes by modifying children resources

    **Forum example:**
    List of all users; list of all messages

  - <u>A resource for every object exposed through the service</u>
    - A service may expose many kinds of objects, each with its own resource set
    - Most services expose a large number of these resources

    **Forum example:**
    message categories (eg, Science category); particular users; particular messages

  - <u>Resources representing the results of algorithms applied to the data set</u>
    - Collections of resources, which are usually the results of queries

    **Forum example:**
    List of messages sent by a certain user; messages of a certain category

Iván Sánchez Milara                    **Programmable Web Project. Spring 2019.**

62

63

# Step 2 - Split the data into resources (II)

- Resources are ordered in a hierarchical way
  - Hierarchy can be represent using a graph diagram
  - Consider carefully the hierarchy when resources which represent results of algorithms are involved; what is the result of the action?
- STEPS:
  - Define all possible types of resources the Web service is intended to expose
  - Give a name to each resource type

  > **Forum example:**
  > Some of the resource types are: message, user, category

  - Define the hierarchy
  - Define how those types of resources fit in the hierarchy
  - Take into account the platform you are going to use
    - Some platforms make it easier to create resources in certain way

Iván Sánchez Milara          **Programmable Web Project. Spring 2019.**

63

---

64

# Step 3 - Name the resources with URIs (I)

- Associate each resource type with a URI pattern
  - In a resource-oriented service the URI contains all the scoping info
- Design principles:
  1. <u>URIs should be descriptive</u>
     - The resource and its URI should be naturally and intuitively linked
  2. <u>Every URI designates exactly one resource</u>
     - Two resources can **NOT** share the same URI
     - Two different resources may point to the same data (but they are different resources!!!)
     - Forum Example:
       - At some moment the resources
         `/forum/users/user_id/last_message` and
         `/forum/message/message_1`
         could point to the same data (a forum message),
         but the resources are different!
  3. <u>The same resource can have one or many URIs</u>

Iván Sánchez Milara          **Programmable Web Project. Spring 2019.**

64

# Step 3 - Name the resources with URIs (II)

4. <u>URIs should have a clear structure</u>
   - Variation should be predictable – a client knowing the structure of the service's URI should be able of building URIs
   - Example:
     - **http://forum.com/users/user_1/public_profile**
   - Then, to get the public profile of user_2 the URI should be
     - Correct: **/users/user_2/public_profile**
     - Incorrect: **/get_public_profile/user_2**
   - Use the following convention:
     1) Use path variables to encode hierarchy: **/parent/child**
     2) Use punctuation characters in path variables if there is <u>no hiearchical relation</u>: **/parent/child1;child2**
        » Use commas when the order of the scoping is important
        » Use semicolon in other cases
     3) Use query variables to imply inputs for an algorithm

**Forum example:**
- **http://forum.example.com/Users/user1**
- **http:://forum.example.com/messages/message1;message2**
- **http://forum.example.com/Categories/Science**
- **http://forum.example.com/Users/user1/history?last=5**
  ➢ Returns a list of the last 5 messages posted by user1

Iván Sánchez Milara **Programmable Web Project. Spring 2019.**

65

---

# Step 4 – Establish the relation among resources

- State diagram of the application
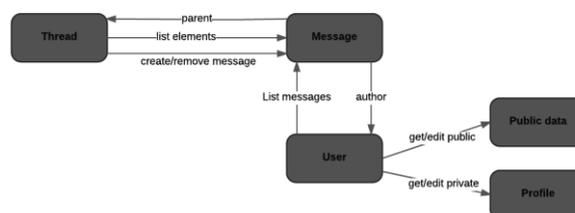


- Will help later to design the hypermedia

Iván Sánchez Milara **Programmable Web Project. Spring 2019.**

66

67

# Step 5 - Expose a subset of the uniform interface (I)

- Explain what happens to each resource when it is exposed to any of the methods of the uniform interface
  - Remember: A resource DOES NOT have to expose all the methods
  - If your resource is read-only, then expose two methods: GET and/or HEAD
  - If your resource can be created or modified you need to implement PUT, POST and/or DELETE
- Avoid creating your own methods (by *overloading* POST)
  - If you think you need an extra method,
    change the verb into a noun and create a resource
  - Example: If you think you need a method named **publish** just create a resource named **publication.** Use the uniform interface operations to modify it (e.g. POST a publication)

Iván Sánchez Milara                    **Programmable Web Project. Spring 2019.**

67

---

68

# Step 5 - Expose a subset of the uniform interface (II)

- **Forum examples:**

– Get all messages from the Sports category

```
GET  http://forum.example.com/Category/Sports
```

– Create a new User

```
PUT  http://forum.example.com/Users/nicky
```

  • User information in the HTTP request body

– Post a message into Science category

```
POST http://forum.example.com/Category/Science/Messages
```

  • Message content and details of the user are in the message body

– Delete the message msg-4

```
DELETE http://forum.example.com/Category/Computers/Messages/msg-4
```

Iván Sánchez Milara                    **Programmable Web Project. Spring 2019.**

68

## Steps 6. Design the resource representation using hypermedia formats

- Assign to each resource representation a format to transfer the resource state between client and server
  - **HYPERMEDIA**
  - The same resource can have different representation formats, but:
    - The server must understand all representations sent by the clients
    - The server must use a representation format the clients can understand
      - A client can ask for a specific format in the URI:
        » Eg: **`http://forum.example.com/users/user_1.xml`**
      - A client can send HTTP headers indicating the formats it accepts:
        » RFC2616 defines the following headers: Accept, Accept-Encoding…

- **NOTE: The resources representations sent from the client does not need to use hypermedia: JSON OR XML IS ENOUGH**

Iván Sánchez Milara          **Programmable Web Project. Spring 2019.**

69

---

70

## Media types (I)

- Domain specific standards
  - Defines application level and protocol level semantics
  - **OpenSearch, SVG, VoiceXML**
- Standard for specific patterns (e.g. collection pattern)
  - Defines protocol level but not application level standards
    - **Collection+JSON, Atom, Odata**
- Microformat and microdata
  - Defines protocol level but not application level
  - Microformat:
    - Extension of HTML4. Allows using the **class** attribute to define semantics
  - Microdata:
    - Extension of HTML5. Use itemprop, itemscope and itemtype attributes to define the semantics
  - Lots in schema.org

- General purpose media-types.
  - Allows personalizing the the protocol semantics and application level semantics
  - **HAL, HTML, SIREN**
- **Be careful with fake hypermedia: XML and JSON**

Iván Sánchez Milara          **Programmable Web Project. Spring 2019.**

70

# Media types (II)

- If you use your own media type be sure that hypermedia controls:
  - The URI of the remote resource
  - The relation of the current resource with the remote one
  - Try to include protocol information
    - E.g. which method I need to execute / what is the format of the request body
- If you are using XHTML:
  - Use **<a>** to have a link to another resource
  - Use **<form>** when you:
    - Include in the URI a query string
    - Represent URIs that follow a certain pattern

71

---

# Domain specific media types. Creating links (I)

- Using xlink (http://www.w3.org/1999/xlink) attributes to create links:
  - Establish a relation between a local XML element and remote resources
    - **xlink:type="simple"** : Simple relation between current XML element and remote resource
    - **xlink:href="uri"** : Provides the path to the linked resource
    - Other voluntary attributes are:
      - **xlink:role** is a URI which indicates the relation between two resources
      - **xlink:title** is a human readable label which describes the link

```
<users xmlns:xlink="http://www.w3.org/1999/xlink>
   <user xlink:type="simple" xlink:href="http://forum/users/axel">
       <nickname>Axel</nickname>
   </user>
   <user xlink:type="simple" xlink:href="http://forum/users/bob">
       <nickname>Bob</nickname>
   </user>
</users>
```

  - More complicated relations can be established using other association types: extended, locator, arc, resource

72

# Domain specific media types. Creating links (II)

- Using atom:link (http://www.w3.org/2005/Atom)
  - Element **<atom:link>** contains attributes to establish a relation between resources:
    - **href**: indicates the URI of the linked resource
    - **rel**: establish the semantic association between the resources. Different values:
      - **self:** the link points to the resource itself
      - More values on next slide
    - **type**: indicates the mime type of the representation

```
<users xmlns:atom="http://www.w3.org/2005/Atom">
    <user>
        <atom:link rel="self" href="http://forum/users/axel">
        <nickname>Axel</nickname>
    </user>
    <user>
        <atom:link rel="self" href="http://forum/users/bob">
        <nickname>Bob</nickname>
    </user>
</users>
```
←――― XML

JSON

```
{users:[
        user:{nickname:"Axel",link:{rel:"self",href=" http://forum/users/axel"}},
        user:{nickname:"Bob",link:{rel:"self",href=" http://forum/users/bob"}}
]}
```

Iván Sánchez Milara                    **Programmable Web Project. Spring 2019.**

# Domain specific media types. Creating links (III)

- Using atom:link (cont)
  - More values for **rel** attribute:
    - **alternate:** alternate representation of the same resource
    - **edit:** clients can edit the resource using this link
    - **related:** the linked resource has certain relation with the current reource
    - **via**: identifies the source for the information of current resource
    - **enclosure**: the link is a resource which contains current resource
    - **previous, next:** previous and next element in a list
    - **first, last:** first and last element of a list
    - Application developer can create application specific relations, expressed as URI
      - Very useful to manage application flow
        Drawback: it is application dependant

Iván Sánchez Milara                    **Programmable Web Project. Spring 2019.**

# Domain specific media types. Creating links (IV)

- **URI templates** permit exposing an unlimited number of resources of the same type using just one URI
  - Parametrize URIs with variables that can be substituted at runtime
    - Variable names are shown between {}
  - Useful for the client to deliver parameters for an algorithm:
    - `http://forum/messages?older_than={timestamp}&maxReturned={max_returned}`
  - And to access a resource from a large set:
    - `http://forum/users/{user_id}`
    - In this case the client should have some knowledge on possible values
- URI templates are generated in the servers
  - They are parts of the links to other resources included in a resource representation; clients can fill the templates
- However, there are no conventions for representing URI templates
- Do not abuse URI templates
  - If you doubt then do not use URI templates
  - Use links when the set of results is known
- Use URI templates for:
  - Documentation
  - To identify resources in servers that accept URI template syntax

Iván Sánchez Milara  **Programmable Web Project. Spring 2019.**

75

---

# Profiles

- A profile must define:
  - **Link relations**:
    - Describing the state transition that will happen if the client triggers a hypermedia control (protocol semantics)
    - Usually implemented as 'rel' attribute
      - http://www.iana.org/assignments/link-relations/link-relations.xhtml
      - Must be documented unless rel attribute is defined by IANA
    - Do not forget to include the method that is utilized

  - **Semantic descriptors**:
    - Describing the meaning of properties in the representation (application semantics)

Iván Sánchez Milara  **Programmable Web Project. Spring 2019.**

76

77

# IANA link relations

- Global register containing about 60 relations.
  - http://www.iana.org/assignments/link-relations/link-relations.xhtml
  - Some useful relations:
    - `collection` and `item` to create collections.
    - `first`, `last`, `next` and `previous` for pagination
    - `replies` to described message thread
    - `latest-version`, `successor-version`, `working-copy` for history of a resource state
    - `edit` and `edit-media` to cover update/delete a resource
- Some document media types defines its own possible relations
- Some profiles include also relations
- If you wanna use your own link relation
  - Use extension relations: *http://mydoma.in/myrelation*

- Microformats Wiki also contains a big set of relations:
  - http://microformats.org/wiki/existing-rel-values
  - DO NOT USE THEM AS SUCH IF YOU HAVE NOT DEFINED THEM IN YOUR PROFILE

Iván Sánchez Milara                **Programmable Web Project. Spring 2019.**

77

78

# Linking to a profile

- Using the **profile** Link relation:
  - RFC 6906 defines a `rel` called **profile**
  - Can be used in any `rel` attribute: `links` (Siren or Collection+Json); `link` defined in HTML, HAL or in the `Link` HTTP header.

```
<html>
  <head>
    <link href="http://microformats.org/wiki/hcard" rel="profile">
```

- Using the **profile** Media Type parameter:
  - Added as parameter in the Content-Type header

  **Content-Type** = application/collection+json;profile=http://myprofile

- Using special purpose hypermedia controls defined in some media types.

Iván Sánchez Milara                **Programmable Web Project. Spring 2019.**

78

## Steps 6. Design the resource representation using hypermedia formats

**Forum example. Message resource.**
Media type: HAL

```
{

  "_links":{
    "self":{"href":"/forum/api/messages/msg-
2/" "profile":"http://atlassian.virtues.fi:8090/display/PWP/Exercise+3#Exercise3-Forum_Message"},
    "collection":{"href":"/forum/api/messages/", "type":"application/vnd.collection+json", "profile":"http://atlassi
an.virtues.fi:8090/display/PWP/Exercise+3#Exercise3-Forum_Message"},
    "author":{"href":"/forum/api/users/AxelW/", "type":"application/hal+json", "profile":"http://atlassian.virtues.f
i:8090/display/PWP/Exercise+3#Exercise3-Forum_User"},
    "in-reply-to":{"href":null "profile":"http://atlassian.virtues.fi:8090/display/PWP/Exercise+3#Exercise3-
Forum_Message"}
  }
  "template" : {
    "data" : [
      {"prompt" : "", "name" : "headline", "value" : "", "required":true},
      {"prompt" : "", "name" : "articleBody", "value" : "", "required":true},
      {"prompt" : "", "name" : "editor", "value" : "", required:false},
      {"prompt" : "", "name" : "author", "value" : "", required:false},
    ]
  }
  "articleBody":"I am using a float layout on my website but I've run into some problems with Internet Explorer. I
have set the left margin of a float to 100 pixels, but IE uses a margin of 200px instead. Why is that? Is this one
of the many bugs in IE?",
  "headline":"CSS: Margin problems with IE",
  "editor":null,
  "author":"AxelW"
}
```

Iván Sánchez Milara          **Programmable Web Project. Spring 2019.**

79

## Step 7. Define protocol specific attributes

- The resource representation is encapsulated in the HTTP request/response message
  - The HTTP body contains the representation
  - The HTTP entity headers contain metadata about the representation e.g. Its media type. Some important headers are:
    - **Content-Type**: mime-type of the representation format
      - A list of mime types can be found in RFC2045 and RFC2046
    - **Content-Length**: size of the body
    - **Accept**: formats a client understands (only in HTTP request)
    - **Accept-Encoding**: encoding accepted for the body
    - Other headers can be used for other purposes:
      - caching, authorization...

Iván Sánchez Milara          **Programmable Web Project. Spring 2019.**

80

## Step 7 - Define protocol specific attributes

• An HTTP response includes a **status code** indicating how the request was processed in the server

    – Headers provide additional information

• Response code + headers indicating success:

    – **GET**

| 200 OK | No headers | Successful request |
|---|---|---|
| 304 Not Modified | No headers | The client must get the resource from the cache |

    – **DELETE**

| 200 OK | No headers | Successful request. The HTTP body might contain a status message |
|---|---|---|

    – **POST** and **PUT**

| 201 Created | Location | Successful creation. Location header indicates the URI of the resouce |
|---|---|---|
| 200 OK | No headers | The resource existed and has been modified. The Body could contain the new resource |
| 301 Moved permanently | Location | The data sent caused the resource URI to be changed |

Iván Sánchez Milara       **Programmable Web Project. Spring 2019.**

81

## Step 8 - Define possible errors

• Define when and how a request could fail

    – Define the error message in the response body. It should be another resource

• Define also the response status codes and the headers of the response:

    – **GET** and **DELETE**

| 404 Not Found | No headers | Resource was not found. HTTP body message might have contained an error message. |
|---|---|---|
| 303 See Other | Location | The resource was not found. Location header provides a related resource. |
| 400 Bad Request | No header | The URI contained some erroneous fields or parameters |

| 415 Unsopported Media Type | No headers | The representation format is not supported by the server |
|---|---|---|
| 409 Conflict | No header | The representation tried to change the resource to a state that is not allowed |
| 400 Bad Request | No header | The resource representation contained an invalid value |

Iván Sánchez Milara       **Programmable Web Project. Spring 2019.**

82

83

# Basic workflow between client and web service



**CLIENT**    **SERVER**

Build the HTTP envelope.
Include method information.
Headers and message
body if needed.

**Send HTTP request**

**Send HTTP response**

❑Check  HTTP response code.
❑ Proceed according to the
response code:
  ▪ process the message body
  ▪ perform needed actions
  ▪ handle exceptions

❑Check the HTTP method
❑Check the headers
❑Parse the message body
❑Perform the requested action
❑Build the response message
  ▪ HTTP response code
  ▪ headers
  ▪ message body

Iván Sánchez Milara            **Programmable Web Project. Spring 2019.**

83

---

# Forum example - GET     84

- Get all messages from the Sports category
  - **HTTP Method:** GET
  - **URI:** http://forum.example.com/Category/Sports
  - **Returns:**
    - On success: 200 OK + XML message body
    - On error: 401 Unauthorized or 404 Not found

**Request HTTP envelope**

```
GET Category/Sports/ HTTP/1.1
Host: forum.example.com
Accept: text/xml
Accept-Encoding: gzip,deflate
Accept-Charset: windows-
1251,utf-8;q=0.7,*;q=0.7
```

**Successful HTTP response envelope**

```
HTTP/1.1 200 OK
Date: Sun, 12 Sep 2010 11:30:12 GMT
Transfer-Encoding: chunked
Content-Type: text/xml;
Content-Length: length;
<?xml version="1.0" encoding="UTF-8"?>
<msg:Thread>
  <msg:Message messageID="msg-3">
    <msg:Registered userID="user-7">
      <user:Nickname>HockeyFan</user:Nickname>
      <user:Avatar file="avatar_7.jpg"/>
    </msg:Registered>
    <msg:Title>Edmonton's goalie</msg:Title>
    <msg:Body>Does anyone know where Jussi...
  (...)
  </msg:Message>
(…)
<msg:Thread>
```

Iván Sánchez Milara     **Programmable Web Project. Spring 2019.**

84

## Forum example - POST

- Post the message into Science category
  - **HTTP Method:** POST
  - **URI:** http://forum.example.com/Category/Science/Messages
  - **Request:** XML message body
  - **Returns:**
    - On success: 201 Created (Location header tells the URI of created message)
    - On error: 400 Bad Request or 409 Conflict

**Request HTTP envelope**

```
POST Category/Science/Messages HTTP/1.1
Host: forum.example.com
Accept: text/xml
Accept-Encoding: gzip,deflate
Accept-Charset: windows-1251,utf-
8;q=0.7,*;q=0.7
Content-Type: text/xml;charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="UTF-8"?>
<msg:Message messageID="" replyTo="msg-1">
  <msg:Anonymous>Science guru</msg:Anonymous>
        (...)
        </msg:Message>
```

**Successful  HTTP response envelope**

```
HTTP/1.1 201 Created
Date: Tue, 19 Sep 2010 06:11:22 GMT
Content-Type: text/xml; charset=iso-8859-1
Content-Length: length
Location:
http://forum.example.com/Category/Science/Messages/msg-4

<?xml version="1.0" encoding="UTF-8"?>
<msg:Message messageID="msg-4" replyTo="msg-1">
  <msg:Anonymous>Science guru</msg:Anonymous>
  <msg:Title>In case</msg:Title>
  <msg:Body>Just in case you can't ...
(...)
</msg:Message>
```

Iván Sánchez Milara          **Programmable Web Project. Spring 2019.**          OULUN YLIOPISTO

## Forum example - DELETE

- Delete  certain message
  - **HTTP Method:** DELETE
  - **URL:** http://forum.example.com/Category/Science/Messages/msg-4
  - **Returns:**
    - On success: 204 No Content
    - On error: 401 Unauthorized or  404 Not Found

**Request HTTP envelope**

```
DELETE
Category/Science/Messages/msg-4
HTTP/1.1
Host: forum.example.com
Accept: text/xml, text/html
Accept-Encoding: gzip,deflate
Accept-Charset: windows-1251,utf-
8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

**Error  HTTP response**

```
HTTP/1.1 404 Not Found
Date: Tue, 19 Sep 2010 06:11:22 GMT
Content-Type: text/html; charset=iso-8859-1
Content-Length: length
Keep-Alive: timeout=15, max=96
Connection: Keep-Alive

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
 <head>
  <title>404 Not Found</title>
 </head>
 <body>
  <h1>Not Found</h1>
  <p>The requested message msg-4 was not found on
this server.</p>
 </body>
</html>
```

Iván Sánchez Milara          **Programmable Web Project. Spring 2019.**          OULUN YLIOPISTO

# HYPERMEDIA DRIVEN DESIGN

Iván Sánchez Milara                    **Programmable Web Project. Spring 2019.**

87

# 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.**

88

# 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.**

89

# 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.**

90

91

# Hypermedia driven APIs examples

92

- **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.**

# 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 Parastidis and Ian Robinson.
5. Representational State Transfer (REST), Roy Thomas Fielding. Available at http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf
6. "Peer-to-Peer Systems and Applications" Ralf Steinmetz KlausWehrle (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/

Iván Sánchez Milara                    **Programmable Web Project. Spring 2019.**