Programmable Web Project
Part 2: Programmable Web

Spring 2025

WWW technologies (II)

Programmable Web

RESTful Web APIs and HATEOAS

The World Wide Web and technologies

What is the World Wide Web?

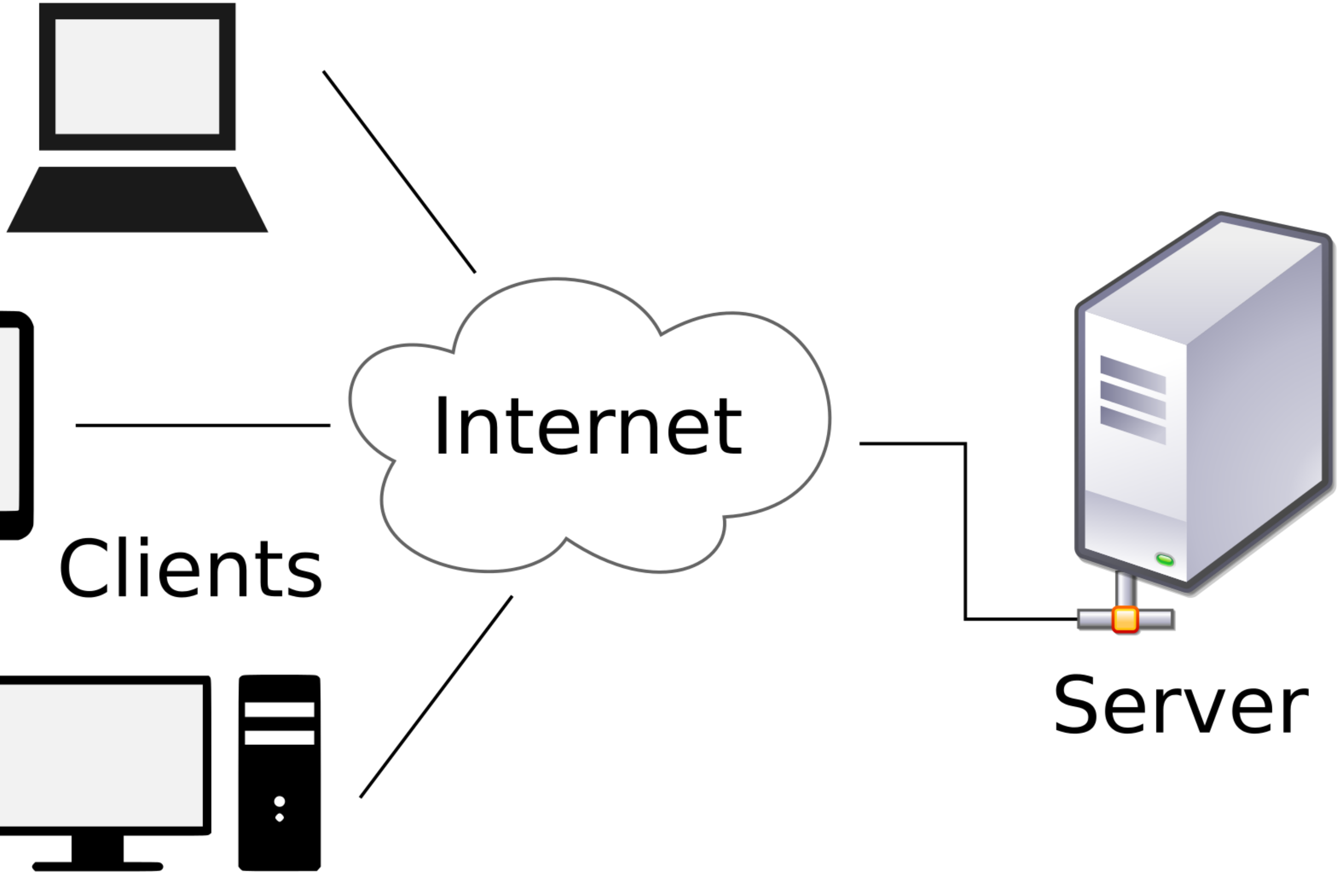
Goal: Distribute data

- Human consumption (H2M)
- Hypertext
- Uniform API and technologies
- Single client (Web browser)

TECHNOLOGIES FOR THE WWW

- Backend: Business logic + data storage (databases)
- Transport protocol: HTTP
- Data serialization languages

Client server model



© David Vignoni LGPL license
https://en.wikipedia.org/wiki/Client%E2%80%93server_model#/media/File:Client-server-model.svg

DATABASES

Definition

- Databases emerged to solve challenges of storing and managing huge amounts of data
- A database:
  - is a data structure
  - stores organized information
  - can be easily accessed, managed and updated
- DBMS (Database Managing System) is the software that allows creating, managing and storing database structures.
  - Responsible for data integrity, recovery and access
  - Provides a way for extract or modify the data
- There are different ways to model the data in the database
  - Lately divided into relational models and non-relational models

ACID properties

- Atomicity
  - Each transaction is atomic.
  - If one part of the transaction fails the whole transaction fails and the database is not modified.
- Consistency
  - Databases moves from one valid state to another valid state in each transaction.
  - A state is valid if meets all the constraints
- Isolation
  - Concurrent access is processed as serial access.
  - Not completed transactions might not be visible to other users
- Durability
  - Once a transaction is committed it remains in the db.

Relational – Non-relational

- Relational:
  - Database model developed by E.F. Codd in 1970
    - Codd, Edgar F (June 1970). "A Relational Model of Data for Large Shared Data Banks"
  - Data is represented in terms of tuples (rows), grouped into relations (tables) that can be linked with each other.
  - Developed almost in parallel with SQL language
- Non-Relational:
  - Sometimes miscalled Non SQL databases
  - Umbrella that gathers different databases that are not relational.
  - Data is not organized in related tables.
    - Some store objects, some store key-value pairs, some store documents
  - More flexible and scalable

RDBMS Concepts

- **CRUD**
  - Databases stores data persistently
  - There are four basic functions to manage persistent data:
    - **C**reate
    - **R**ead
    - **U**pdate
    - **D**elete
- **ORM (Object relational mapping)**
  - To access a relational database from an object oriented language context (PHP, Python, Java…)
    - interface translating relational logic to objects logic is needed.
    - Such interface is called **Object-relational mapping** (**ORM**, **O/RM**, and **O/R mapping**).

Examples - Relational

- Relational databases are still the most commonly used.
- Relational databases are mainly composed by tables.
- A table is formed by zero (empty) or more rows.
- A row consists of one or more fields
  - Each has a certain datatype. (columns)

FirstName	Surname	PersonalId
John	Smith	3321
Jack	Johnson	4352
Mary	Smith	9807

- Some examples are: PostgreSQL, MySQL, SQLite

Examples – Non-relational

- MongoDB
  - Scalable, open source database
  - JSON based data store: BSON
  - Document-oriented database
    - Database formed by Collections of Documents
  - Example of MongoDB document:

```
{
  name: “jim”,
  surname: “smith”,
  grade: 3
}
```

  - Example of MongoDB query:

```
db.students.find({grade:{$gt:3}});
```

TRANSPORT PROTOCOL: HTTP

Examples – Non-relational

- MongoDB
  - Scalable, open source database
  - JSON based data store: BSON
  - Document-oriented database
    - Database formed by Collections of Documents
  - Example of MongoDB document:

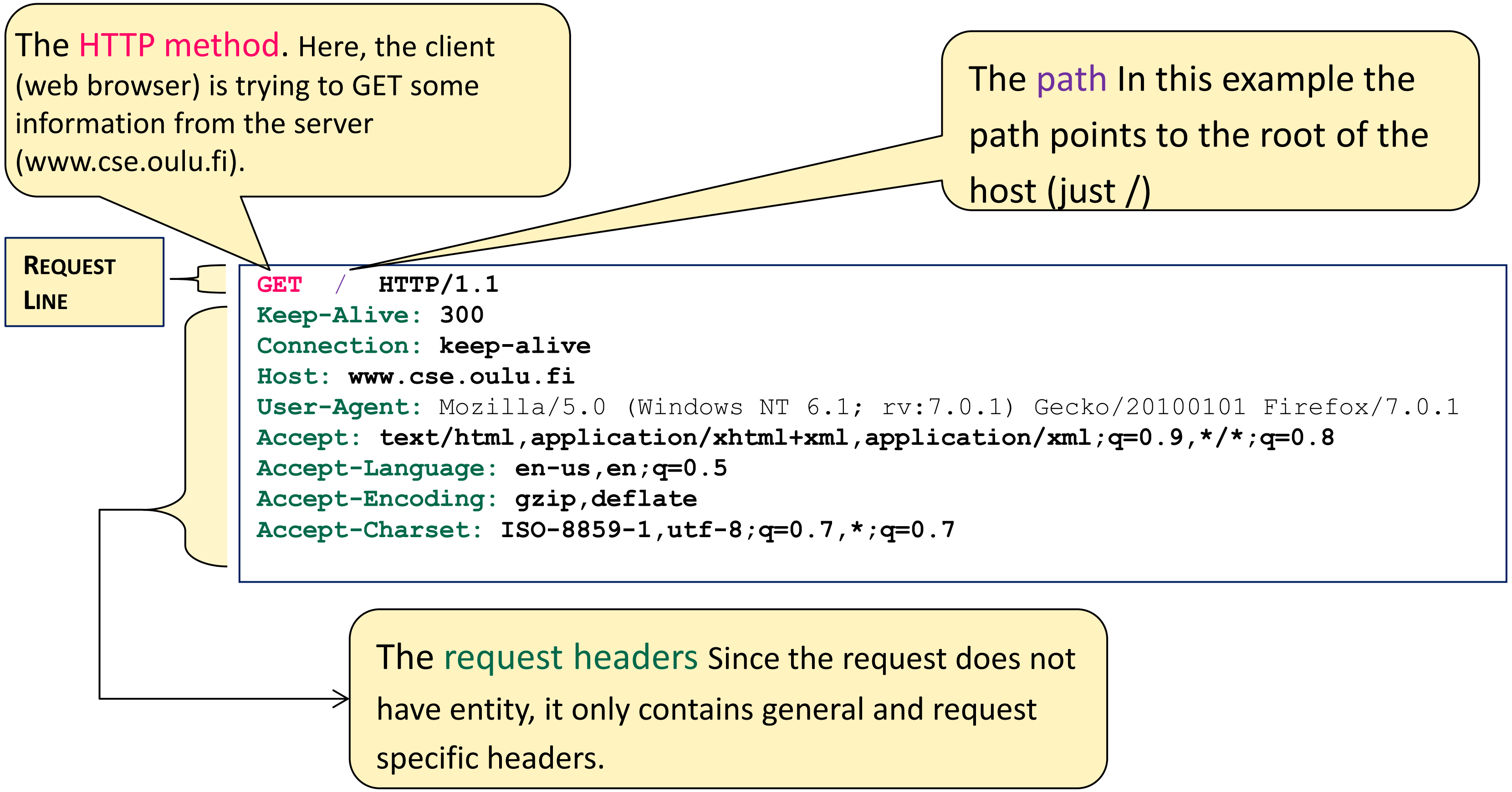
```
{
  name: “jim”,
  surname: “smith”,
  grade: 3
}
```

  - Example of MongoDB query:

```
db.students.find({grade:{$gt:3}});
```

HTTP Request parts

- HTTP request example to http://www.cse.oulu.fi



The HTTP method. Here, the client (web browser) is trying to GET some information from the server (www.cse.oulu.fi).

The path In this example the path points to the root of the host (just /)

REQUEST LINE

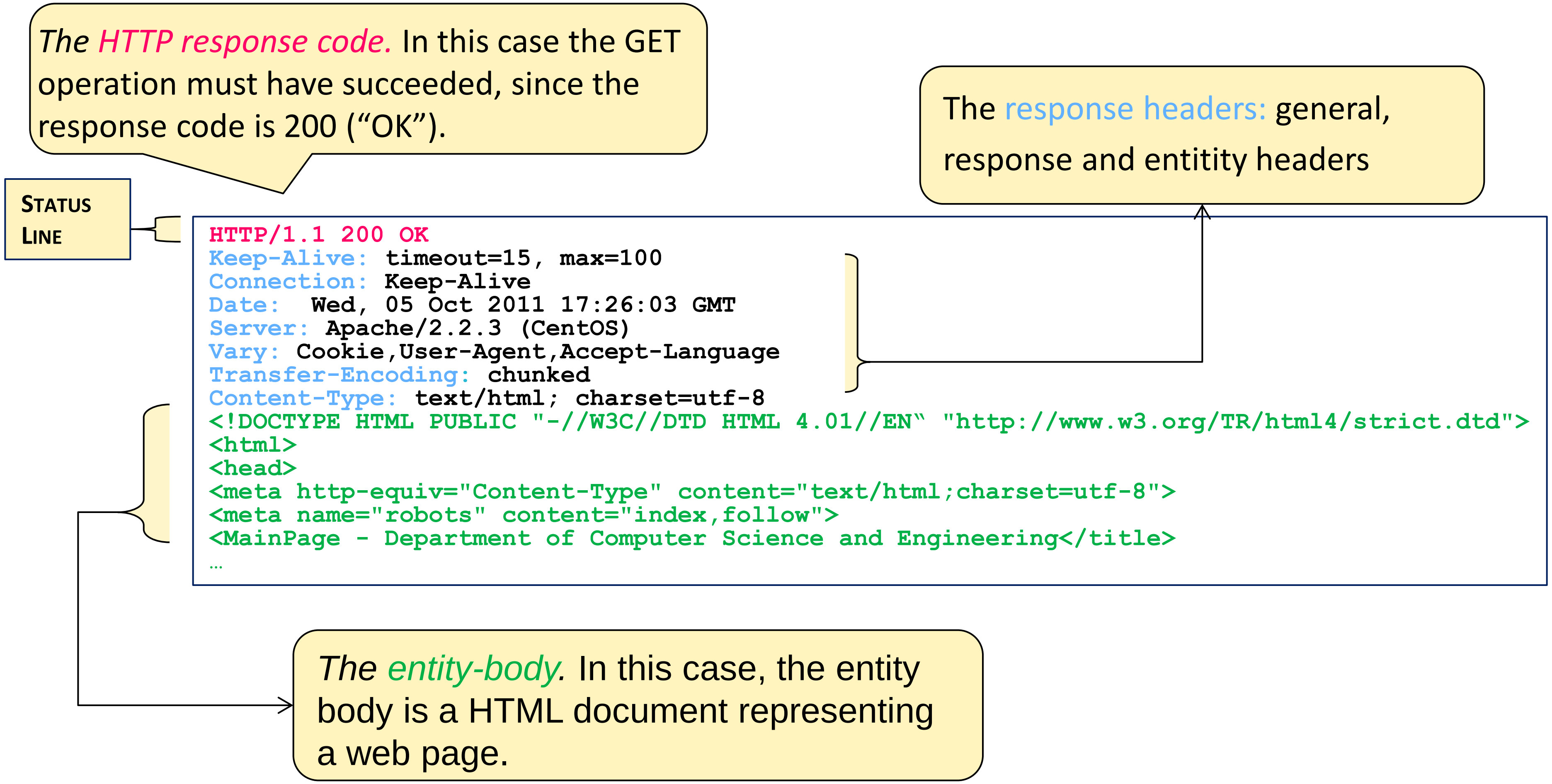
```
GET  /  HTTP/1.1
Keep-Alive: 300
Connection: keep-alive
Host: www.cse.oulu.fi
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:7.0.1) Gecko/20100101 Firefox/7.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

The request headers Since the request does not have entity, it only contains general and request specific headers.

The entity-body This particular request has no entity body, which means the envelope is empty! This is typical for a GET request, where all the information needed to complete the request is in the path and the headers.

HTTP Response parts

- Response Example: http://www.cse.oulu.fi



The HTTP response code. In this case the GET operation must have succeeded, since the response code is 200 (“OK”).

STATUS LINE

The response headers: general, response and entitity headers

```
HTTP/1.1 200 OK
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Date:  Wed, 05 Oct 2011 17:26:03 GMT
Server: Apache/2.2.3 (CentOS)
Vary: Cookie,User-Agent,Accept-Language
Transfer-Encoding: chunked
Content-Type: text/html; charset=utf-8
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN“ "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;charset=utf-8">
<meta name="robots" content="index,follow">
<MainPage - Department of Computer Science and Engineering</title>
…
```

The entity-body. In this case, the entity body is a HTML document representing a web page.

HTTP Methods

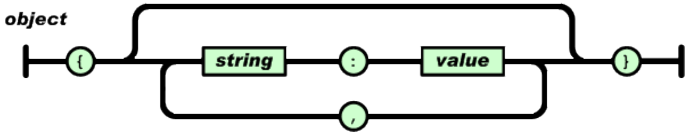
Defined in RFC2616

Method	Description
GET	Returns the resource representation
HEAD	Identical to GET except that the server returns only headers information in the response
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

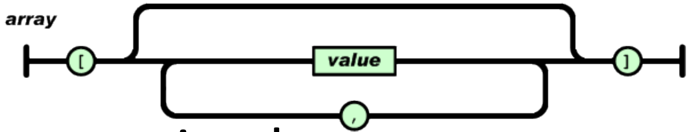
DATA SERIALIZATION LANGUAGES

JSON

- JavaScript Object Notation
- Based on a subset of the JavaScript Language
- Built on two structures:
  - A collection of name/value pairs



  - An ordered list of values



- These structures can be mapped to structures in almost any programming language
- Example

```
{"widget": {
          "debug": "on",
          "window": {
                    "title": "Sample Konfabulator Widget",
                    "name": "main_window",
                    "width": 500,
                    "height": 500 }
}}
```

http://www.json.org

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.

“*Hypertext […]* the *simultaneous presentation of information and controls such that the information becomes the affordance through which the user obtains choices and selects actions. Machines can follow links when they understand the data format and the relations type*”

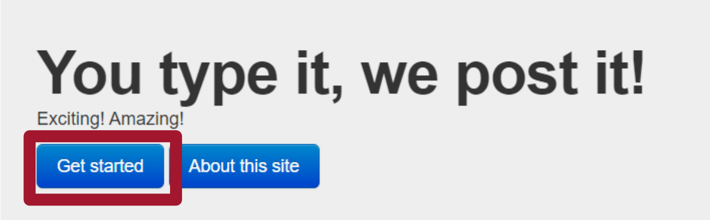
Roy Fielding, “A little REST and Relaxation*”

- Hypermedia
  - **Data**
  - **Hypermedia controls**. Indicates what actions could I do next, what are the target resource to perform the action (link) and how can I perform those actions (http method / response).

* http://www.slideshare.net/royfielding/a-little-rest-and-relaxation

Hypermedia (HTML)

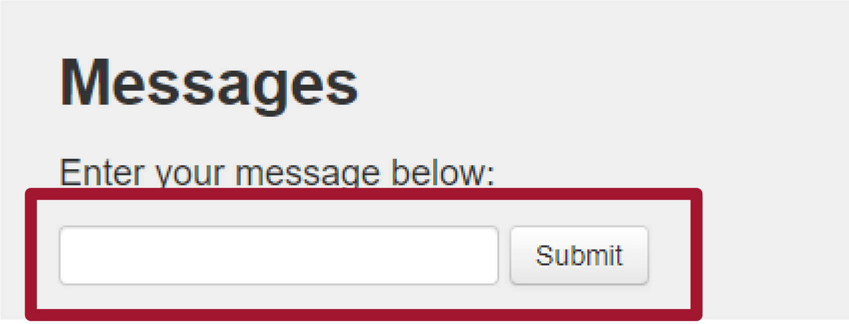
```
<a href="http://www.youtypeitwepostit.com/messages/">
        Get started
</a>
```



You type it, we post it!

Exciting! Amazing!

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

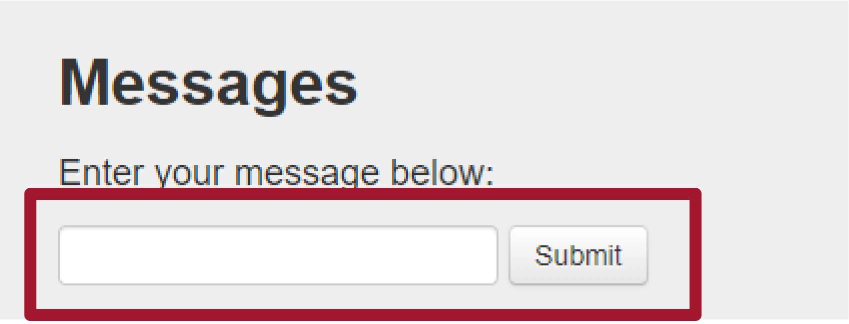


Messages

Enter your message below:

Hypermedia (HTML)

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

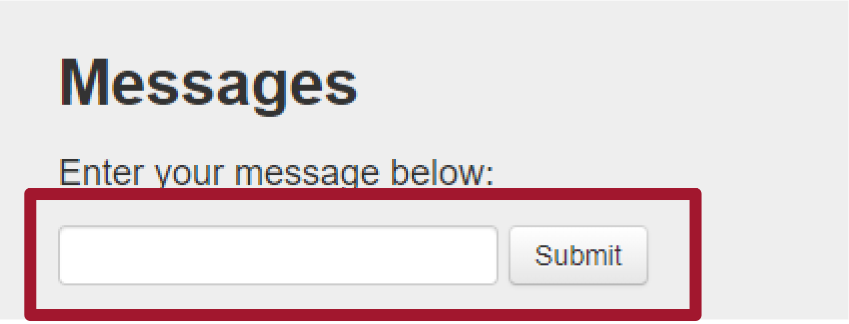


Messages

Enter your message below:

Hypermedia (HTML)

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



Messages

Enter your message below:

Hypermedia (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 FORMATS IN APPENDIX 3: Hypermedia formats

CLIENTS

Types of clients

- Human driven clients
  - Decisions made by humans. IMPORTANT: how to represent information to humans
- Crawlers
  - It starts following all links iteratively from certain web, executing an algorithm for each link followed
  - E.g. Google
- Monitors
  - Checks the state of a resource periodically
  - E.g. RSS aggregator
- Scripts
  - Simulate an human repeating a determined set of actions (eg. Accessing sequentially a list of links).
- Agents
  - Try to emulate humans who are actively engaged with a problem. Looks to representation and take autonomous decisions based on states.

Web browser. An Human Driven client.

- A web browser is the client for ALL websites and web applications.

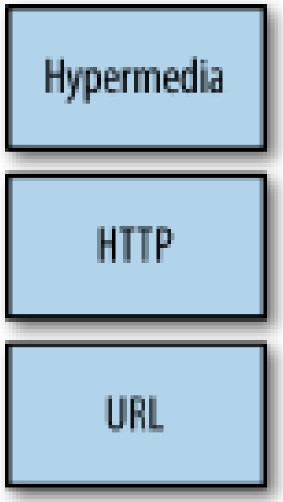
- **TECHNOLOGIES:**
  - **HTML**-> Markup language which defines the content to be rendered by the browser
  - **CSS**-> Style sheet language used for describing the look and formatting of a document
  - **JAVASCRIPT->** Scripting language that listen for events triggered by the users, the network or the host system and execute predefined actions.
  - **AJAX**-> A set of techniques based on Javascript which enable asynchronous interaction between a web browser and a server
  - **WebSocket**-> Computer communication protocol over TCP that provides full-duplex communication. It enables for instance, pub/sub.

PROGRAMMABLE WEB

What about current Web APIs (RPC or CRUD)?

- Need excessive documentation
  - Exhaustive description of required protocol: HTTP methods, URLs …
- Integrating a new API inevitably requires writing custom software
  - Similar applications required totally different clients
- 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

Programmable Web



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

How can I communicate with the resource?

Where is the resource? What is its id?

Web:

- Targeted to humans
- One client

Programmable Web:

- Targeted to machines
- Heterogeneous clients
- Multi language

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**

Programmable Web Project
Part 3: RESTful Web APIS

Spring 2025

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

INTRODUCTION TO ROA

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

ROA Introduction

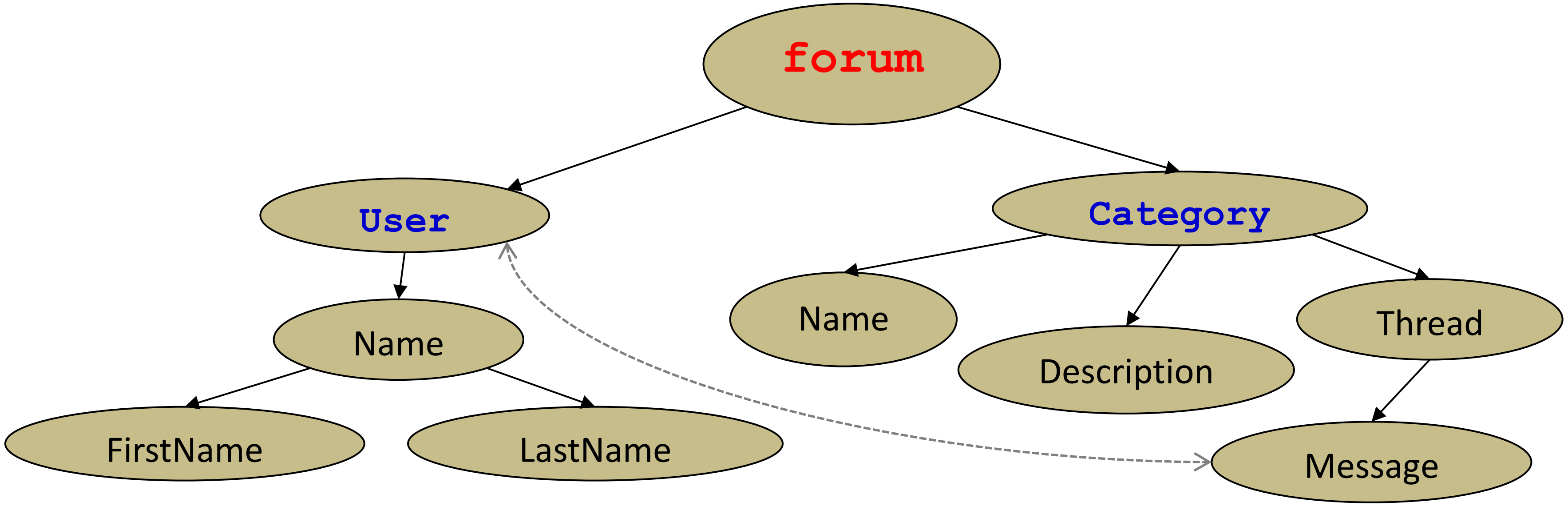
- **Resource Oriented Architecture (ROA)**
  - Architecture for creating Web APIs
  - It conforms the REST design principles
  - **Base technologies: URLs, HTTP and Hypermedia**
- **Resource :**
  - Anything important enough to be referenced as a thing itself
    - For example: List of the libraries 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 identifier
- We operate with resources **representations** by means of **HTTP Requests**
  - Retrieve or manipulate the state of the resource

ROA pillars

Four properties:

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

Forum Resource hierarchy

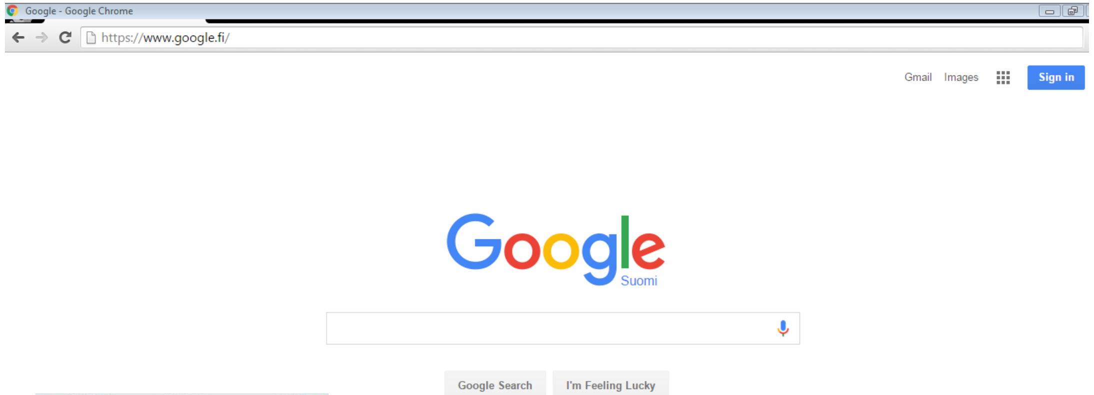


Addressability

- 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

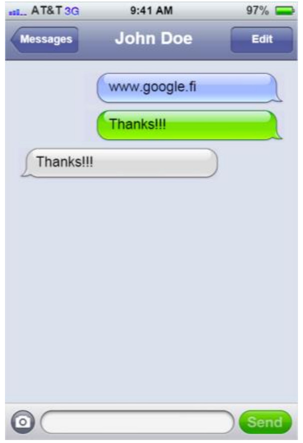
Addressability in WWW

- The WWW is addressable



https://www.google.fi/

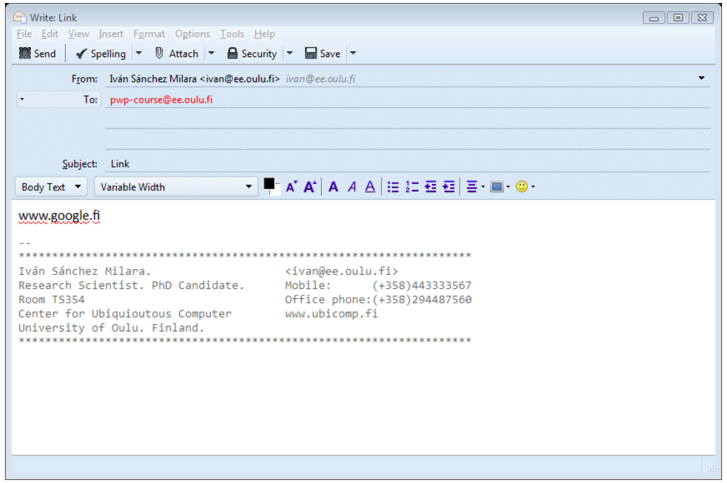
Google.fi offered in: suomi svenska



www.google.fi

Thanks!!!

Thanks!!!



From: Iván Sánchez Milara <ivan@ee.oulu.fi> ivan@ee.oulu.fi

To: pwp-course@ee.oulu.fi

Subject: Link

www.google.fi

```
--
********************************************************************
Iván Sánchez Milara.                <ivan@ee.oulu.fi>
Research Scientist. PhD Candidate.  Mobile:      (+358)443333567
Room TS354                          Office phone:(+358)294487560
Center for Ubiquioutous Computer    www.ubicomp.fi
University of Oulu. Finland.
********************************************************************
```

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

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 }]`

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

Uniform interface in WWW

- Only GET and POST supported in HTML

- Rest of HTTP methods supported through Javascript

Statelessness (I). State concept.

- **Resource state**:
  - A resource representation that is exchanged between server and client
  - Same for all the clients making simultaneous requests
  - Lives <u>in the server</u>

- **Application state**:
  - Snapshot of the entire system at a particular instant, including past actions and possible future state transitions
  - Future possible application states are informed in the resource representation sent by the server.
  - Lives <u>in the client</u>

STATLESSNESS => REFERS TO APPLICATION STATE

Statelessness (II)

- **Every HTTP request happens in complete isolation (STATELESS) -> (application state)**
  - Server never operates based on information from previous requests, **SERVER 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. However, it provides information on which are the future states.
  - **Client handles** the application **workflow**

Statelessness in WWW

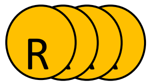
- Originally the WWW is statless
  - GET an URL always should return same website

- Multiple applications needs state information (login, last accessed, visited pages)
  - Cookies
  - Session id in URL

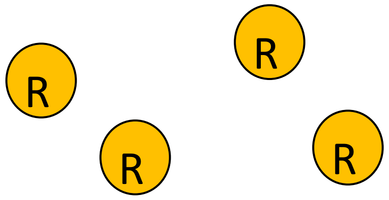
Connectedness (I)

- Resource representation MUST contain links to other resources
- Links must include
  - The relation among resources
  - Optionally, information on how to access linked resources

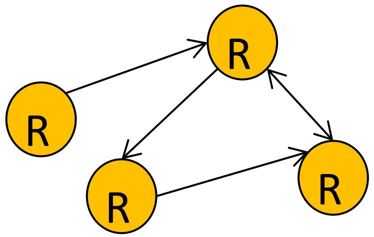
R=Resource



Service exposes everything under single URI
not addressable, not connected



Service is addressable, but not connected



Service is addressable and connected

Connectedness (II)

```
<msg:Thread>
   <msg:Message messageID="msg-7">
      <atom:link rel="self" href="http://forum/messages/msg-7"></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>
```

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*»

Connectedness in WWW

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

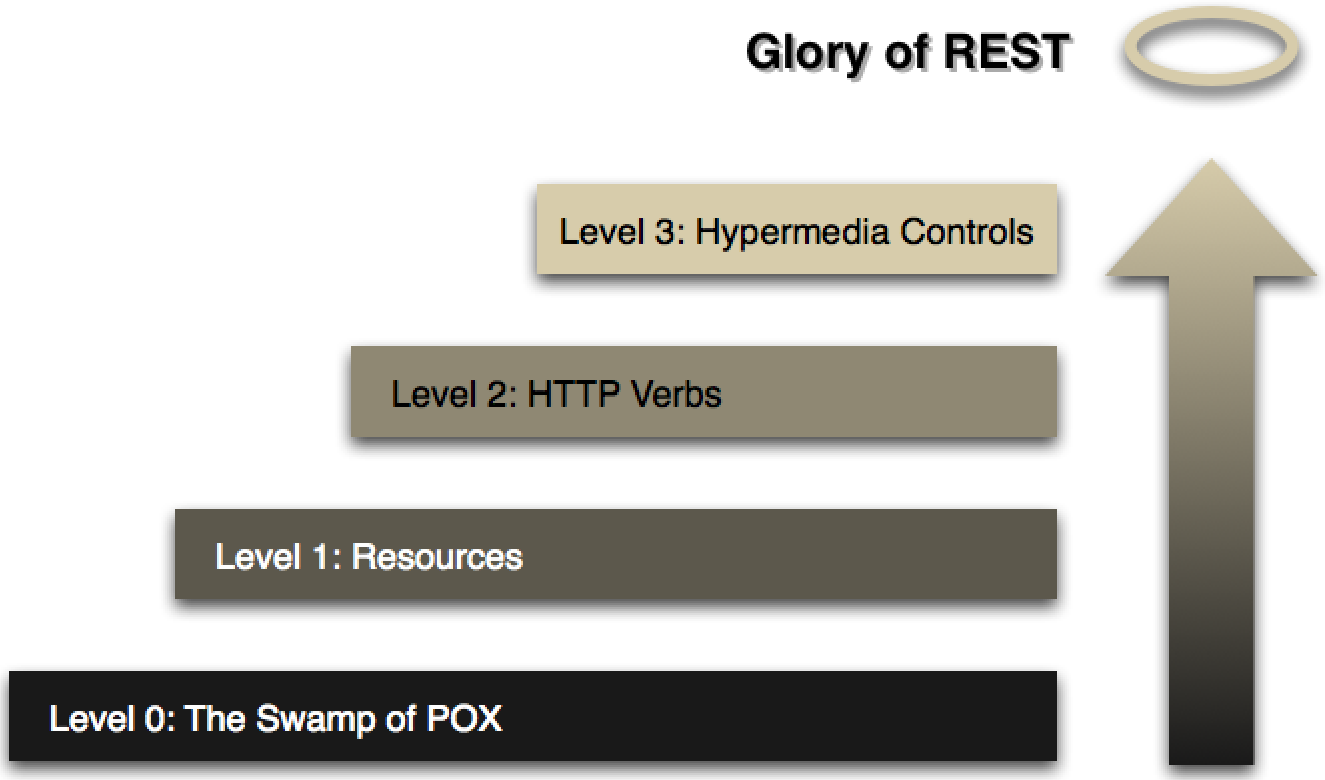
```
<a href="http://www.youtypeitwepostit.com/messages/">
          See the latest messages
</a>
```

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

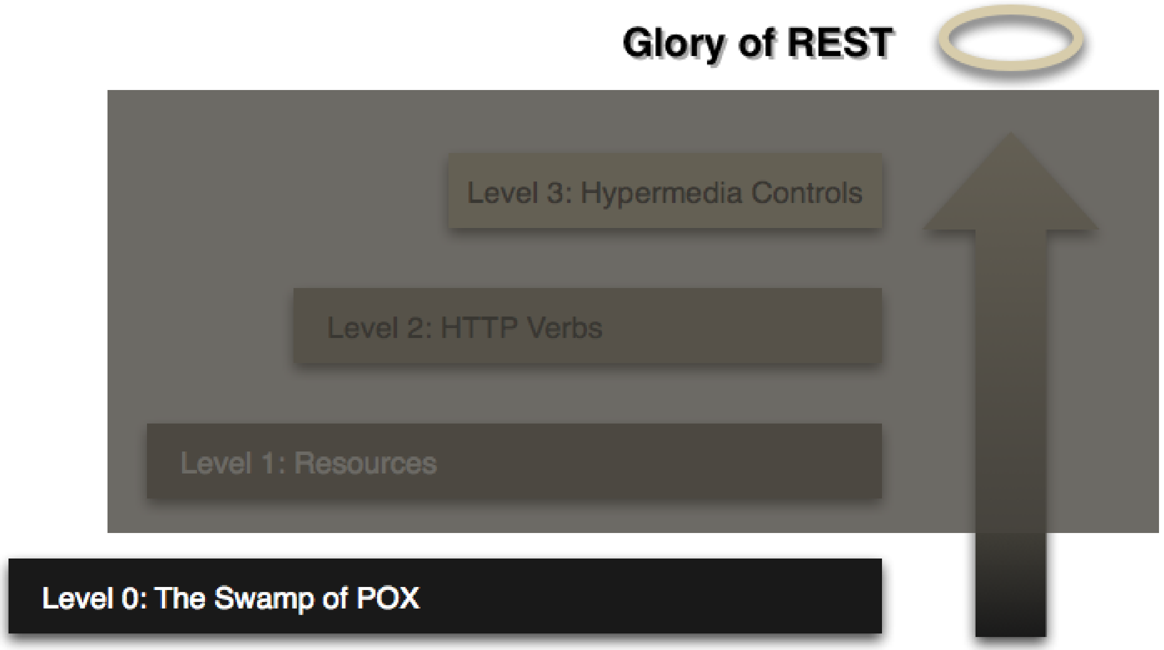
```
<form action="http://www.youtypeitwepostit.com/messages"
method="post">
          <input type="text" name="message" value=""
required="true" />
          <input type="submit" value="Post" />
</form>
```

RESTFUL WEB APIS. HYPERMEDIA.

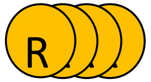
Richardson Maturity Model



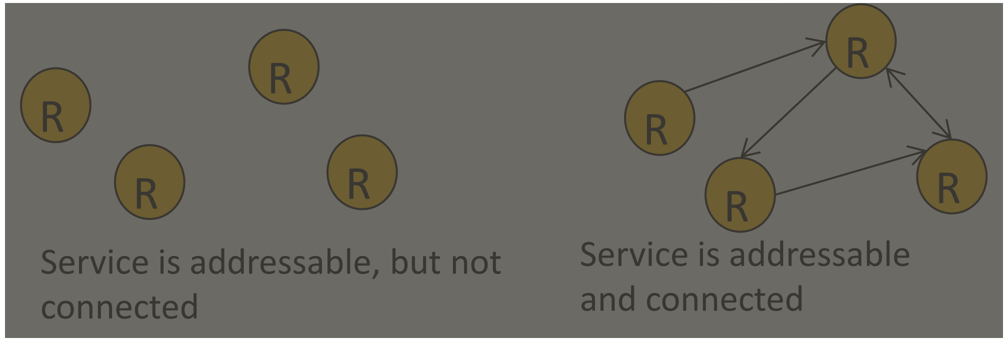
Richardson Maturity Model



R=Resource



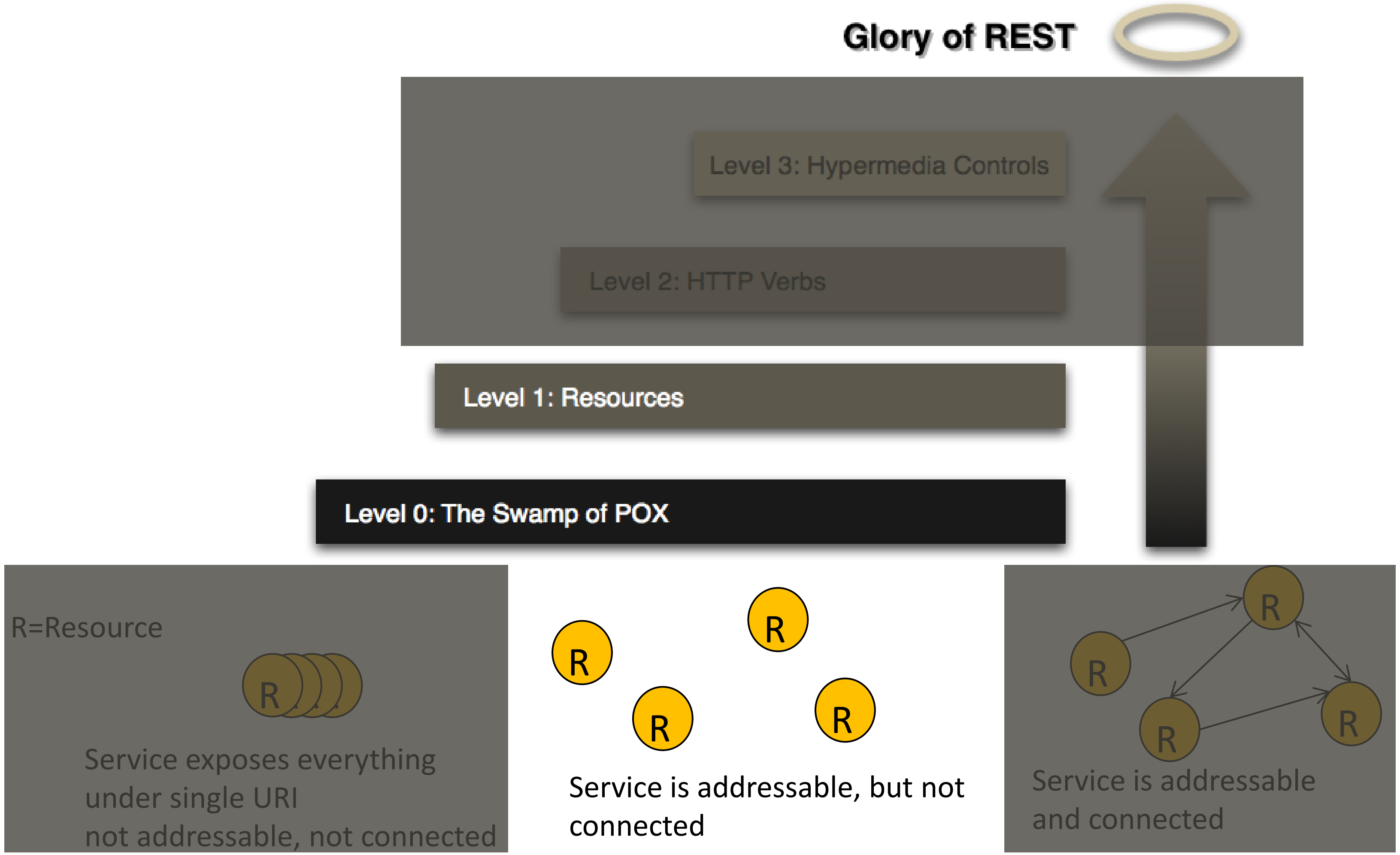
Service exposes everything
under single URI
not addressable, not connected



Service is addressable, but not connected

Service is addressable and connected

Richardson Maturity Model



Glory of REST

Level 3: Hypermedia Controls

Level 2: HTTP Verbs

Level 1: Resources

Level 0: The Swamp of POX

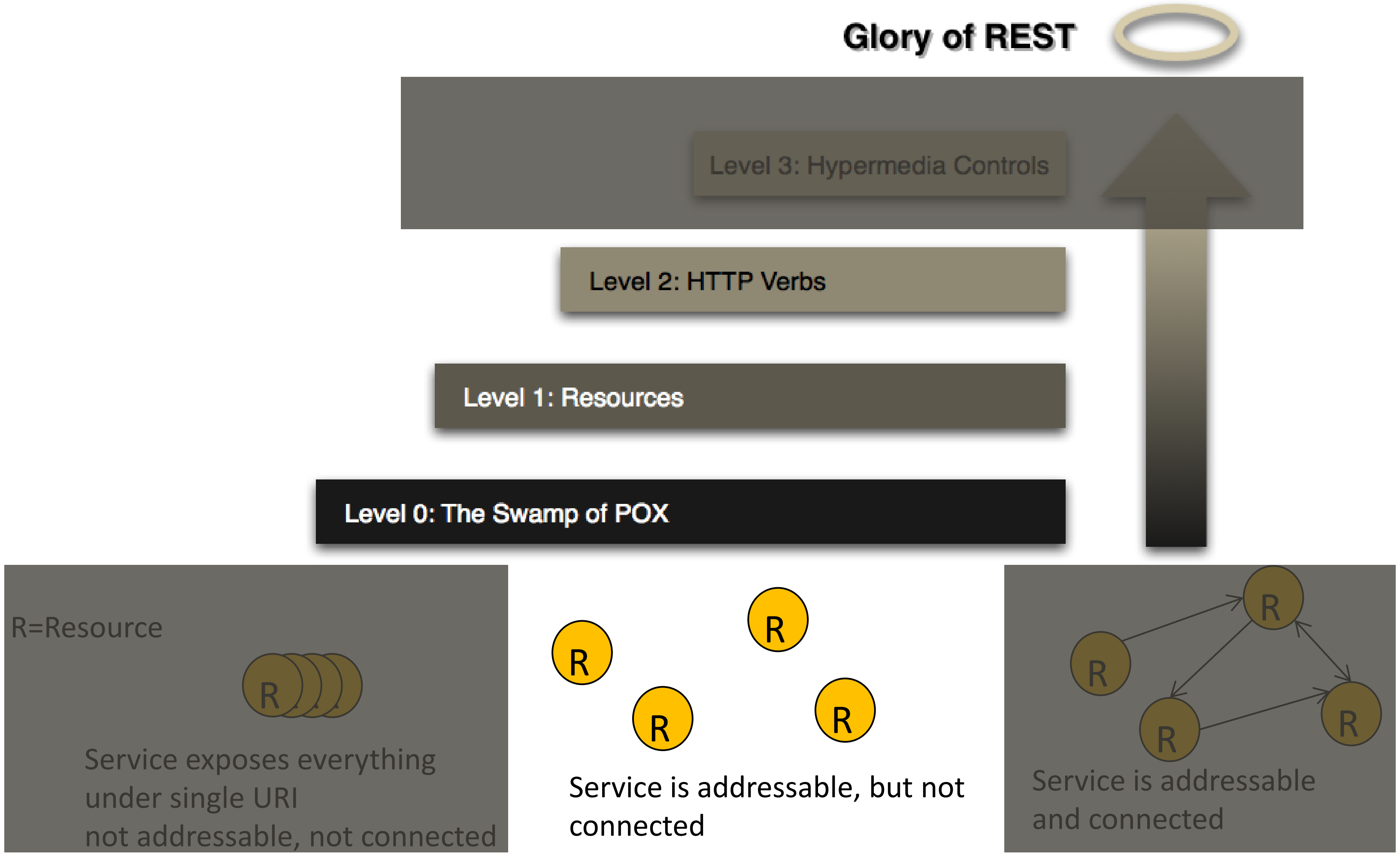
R=Resource

Service exposes everything under single URI not addressable, not connected

Service is addressable, but not connected

Service is addressable and connected

Richardson Maturity Model



Glory of REST

Level 3: Hypermedia Controls

Level 2: HTTP Verbs

Level 1: Resources

Level 0: The Swamp of POX

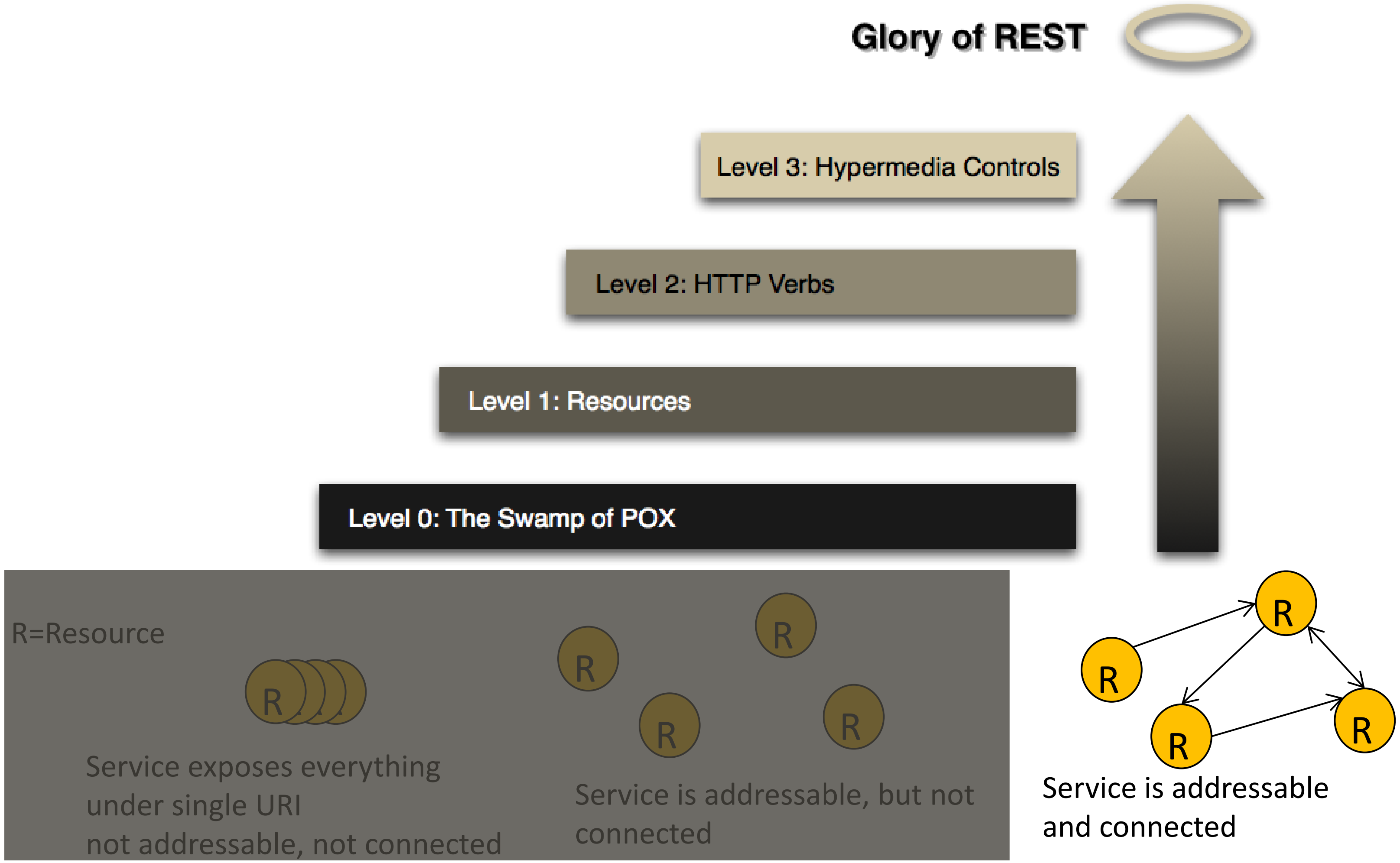
R=Resource

Service exposes everything under single URI
not addressable, not connected

Service is addressable, but not connected

Service is addressable and connected

Richardson Maturity Model



Glory of REST

Level 3: Hypermedia Controls

Level 2: HTTP Verbs

Level 1: Resources

Level 0: The Swamp of POX

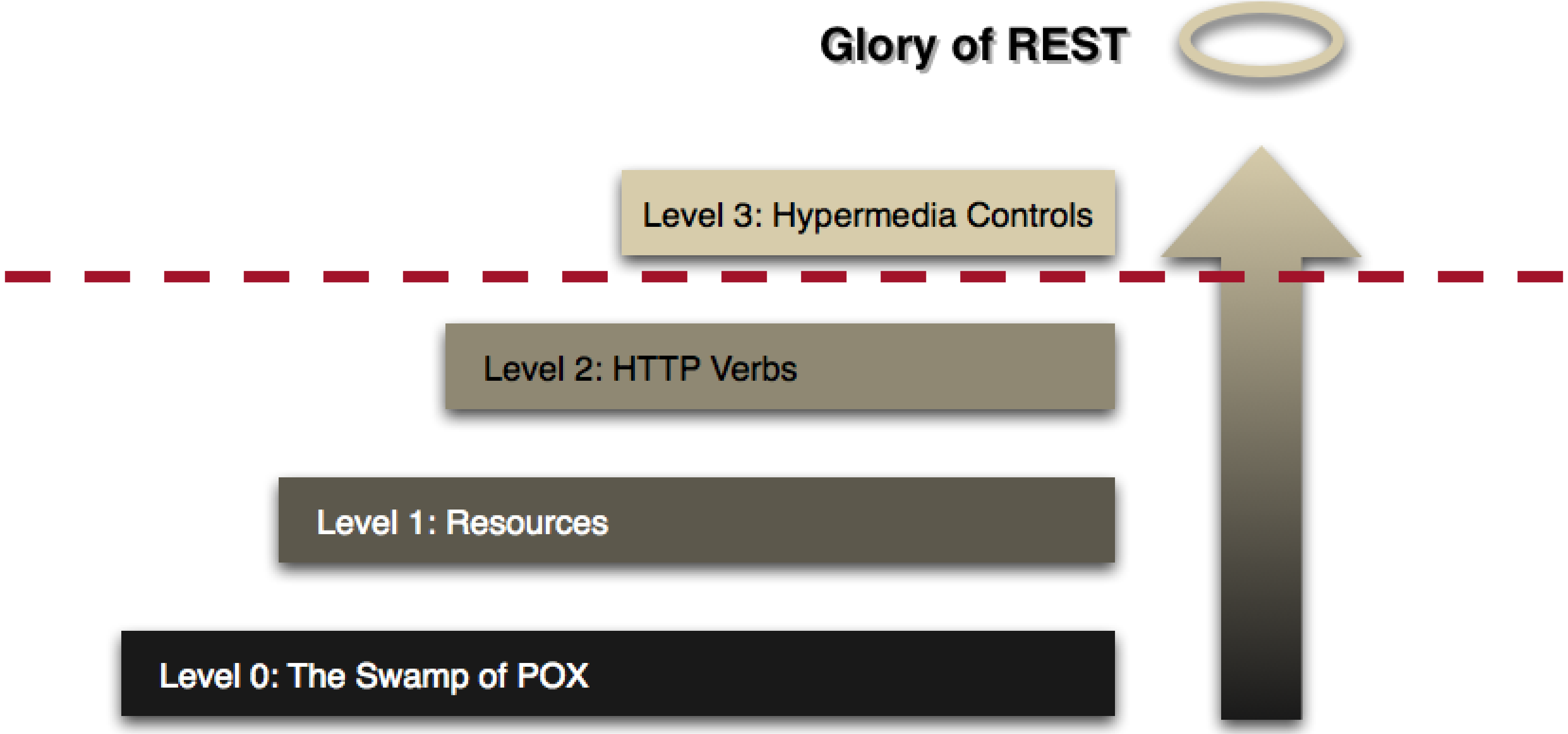
R=Resource

Service exposes everything under single URI not addressable, not connected

Service is addressable, but not connected

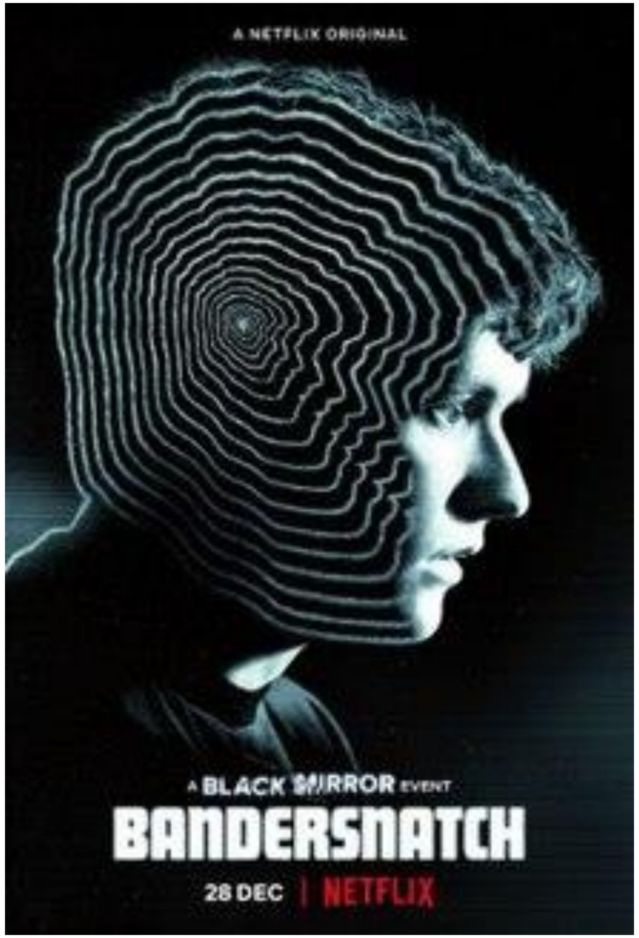
Service is addressable and connected

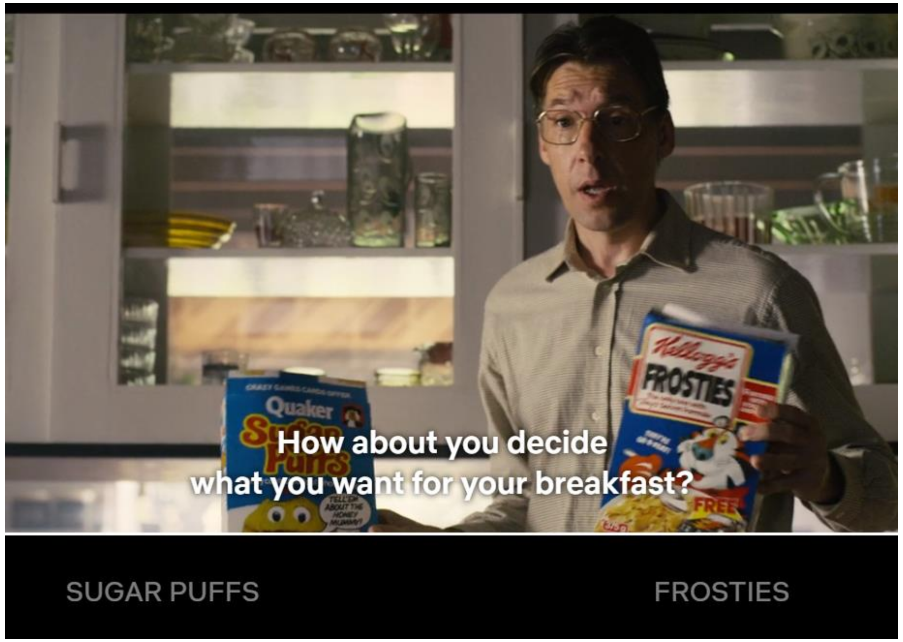
Richardson Maturity Model



HATEOAS

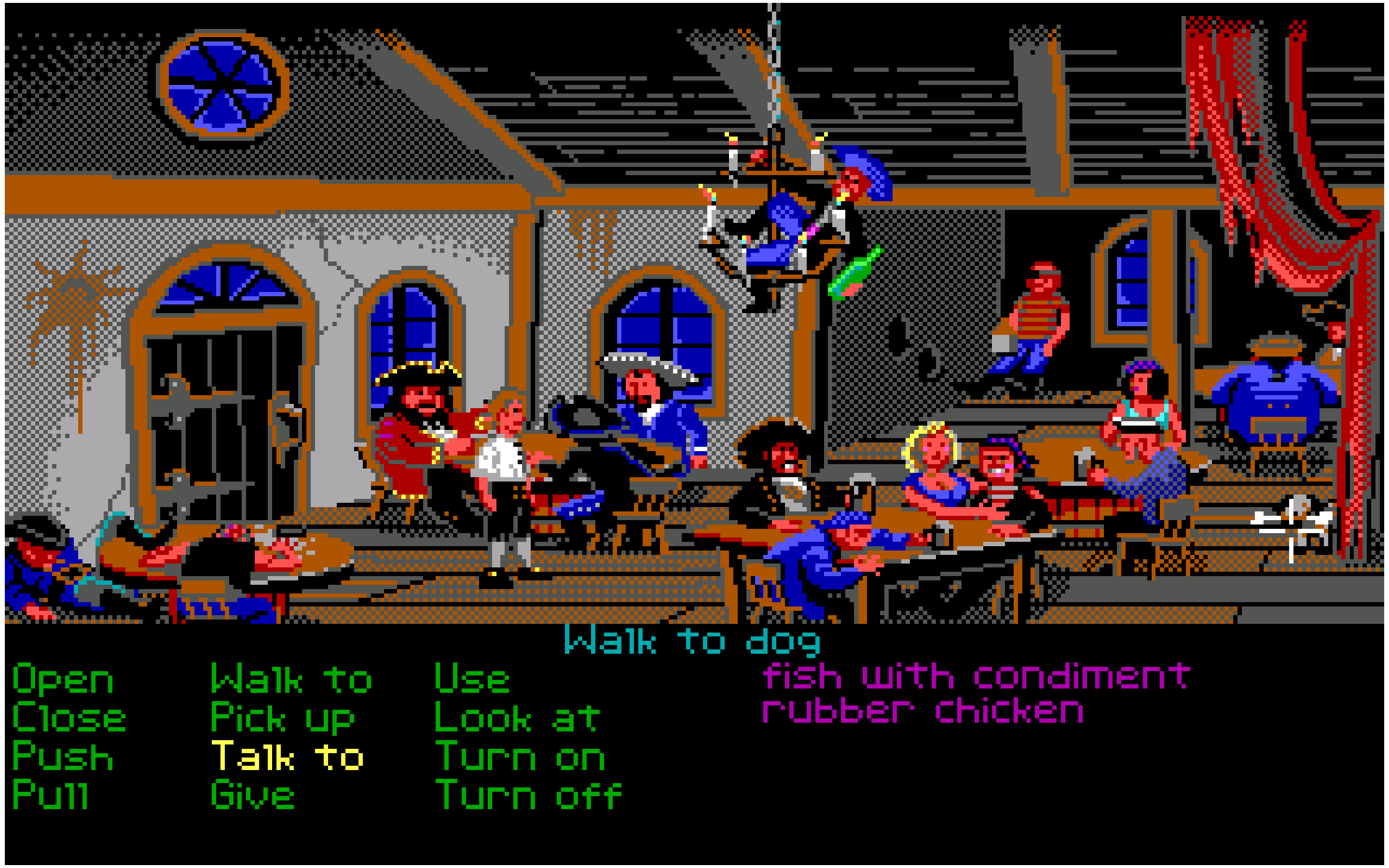
Hypermedia As The Engine Of Application State





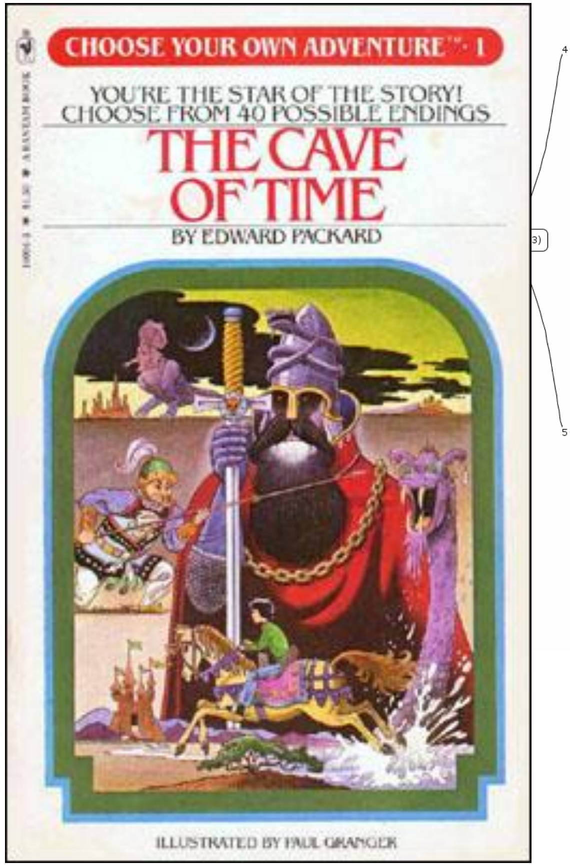
HATEOAS

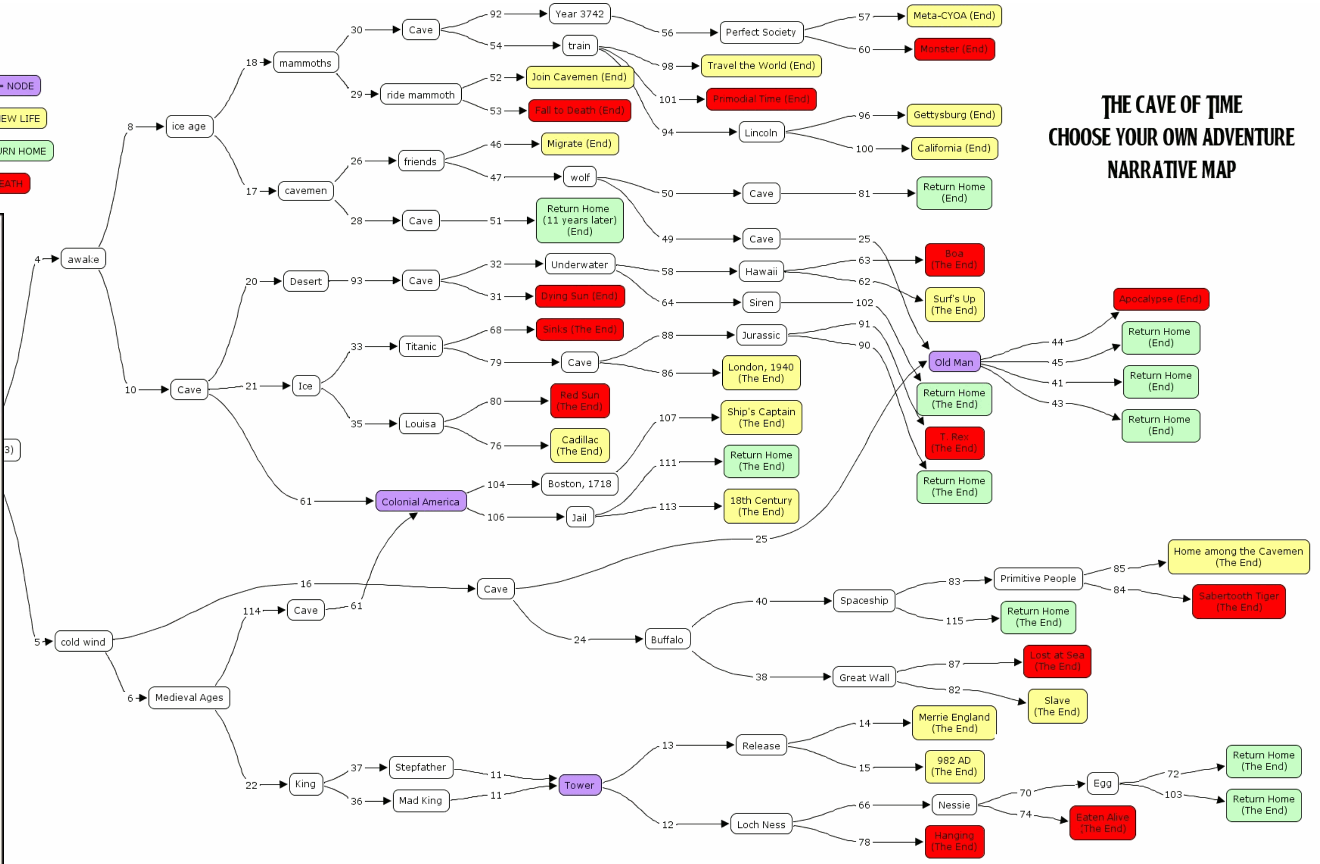
Hypermedia As The Engine Of Application State



HATEOAS

Hypermedia As The Engine Of Application State


CHOOSE YOUR OWN ADVENTURE™ 1
A BANTAM BOOK
YOU'RE THE STAR OF THE STORY!
CHOOSE FROM 40 POSSIBLE ENDINGS
THE CAVE OF TIME
BY EDWARD PACKARD
ILLUSTRATED BY PAUL GRANGER



HATEOAS

Hypermedia As The Engine Of Application State

Hypermedia format contain:

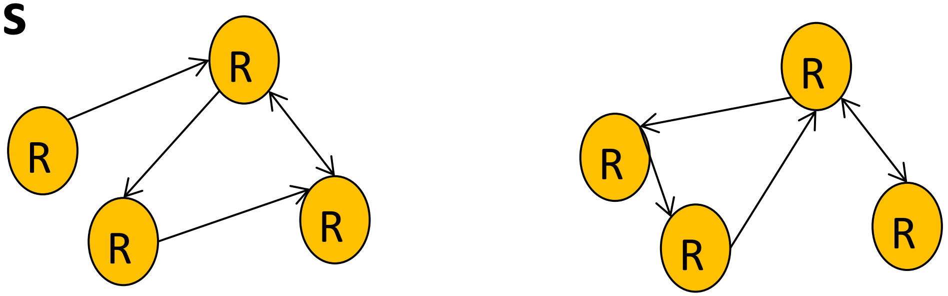
- Data (state of a representation)
- **Hypermedia controls**
  - The URI of the associated resource (link)
  - The relation between both resources
  - Usually, protocol information

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

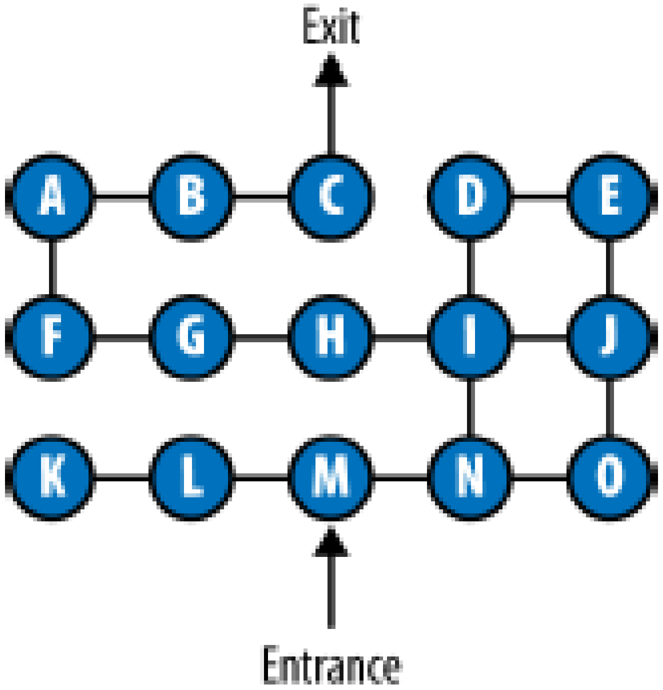
HATEOAS

Hypermedia As The Engine Of Application State

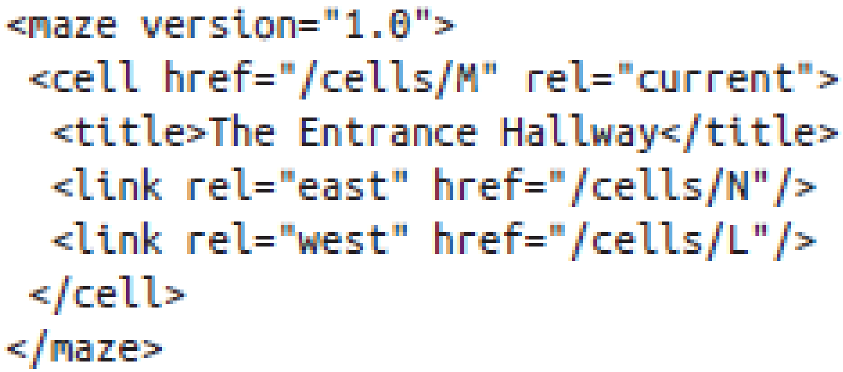
- Ideally, client just need the entry point to a service
  - The rest of the URIs (resources) are discovered through the **hypermedia controls**
  - **RESOURCES AS STATE IN A MACHINE DIAGRAM**
- Well-designed RESTful APIs permit **modifying the server architecture (e.g. URL structure) and data model 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>
```

Which are the hypermedia controls?

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

Semantic challenge (I)

- In WWW 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 its semantics**.

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

Semantic challenge (III)

Two ways of communicating semantics to the client

Media Types

Profiles

Media types

- Defines the format of the message
  - Sometimes include protocol and application semantics

- There are some general-purpose media types with hypermedia support:
  - Allows defining the protocol semantics and application semantics in the API
  - **HAL, HTML, SIREN, MASON**

⚠ **PLAIN JSON OR PLAIN XML DOES NOT SUPPORT HYPERMEDIA**

LIST OF HYPERMEDIA FORMATS IN APPENDIX A: Hypermedia formats

Media types

PLAIN JSON OR PLAIN XML DOES NOT SUPPORT HYPERMEDIA

```
<users>
   <user>
           <nickname>Axel</nickname>
   </user>
   <user>
           <nickname>Bob</nickname>
   </user>
</users>
```

```
{users:[
           user:{nickname:”Axel},
           user:{nickname:”Bob”}
]}
```

```
<ul>
           <li><a href=”http://myapp/users/axel” rel=”self”>”Axel”</a></li>
           <li><a href=”http://myapp/users/bob” rel=”self”>”Bob”</a></li>
</ul>
```

LIST OF HYPERMEDIA FORMATS IN APPENDIX A: Hypermedia formats

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 FORMATS IN APPENDIX A: Hypermedia formats

Mason

- Mime-type: application/vnd.mason+json
- Link: https://github.com/JornWildt/Mason

```
{”name”: ”eeyore”,
 ”color”: ”grey”
     "@controls": {
         "self": {
           "href": "http://api.example.org/donkey/eeyore"
         },
         "dk:mood": {
           "title": "Change mood",
           "href": "http://api.example.org/donkey/eeyore/mood",
           "method": "PUT",
           "encoding": "json",
           "schema": {
             "type": "object",
             "properties": {
               "Mood": {"type": "string"},
               "Reason": {"type": "string"}
           }
         }
       }
     }
```

LIST OF HYPERMEDIA FORMATS IN APPENDIX A: Hypermedia formats

Profile

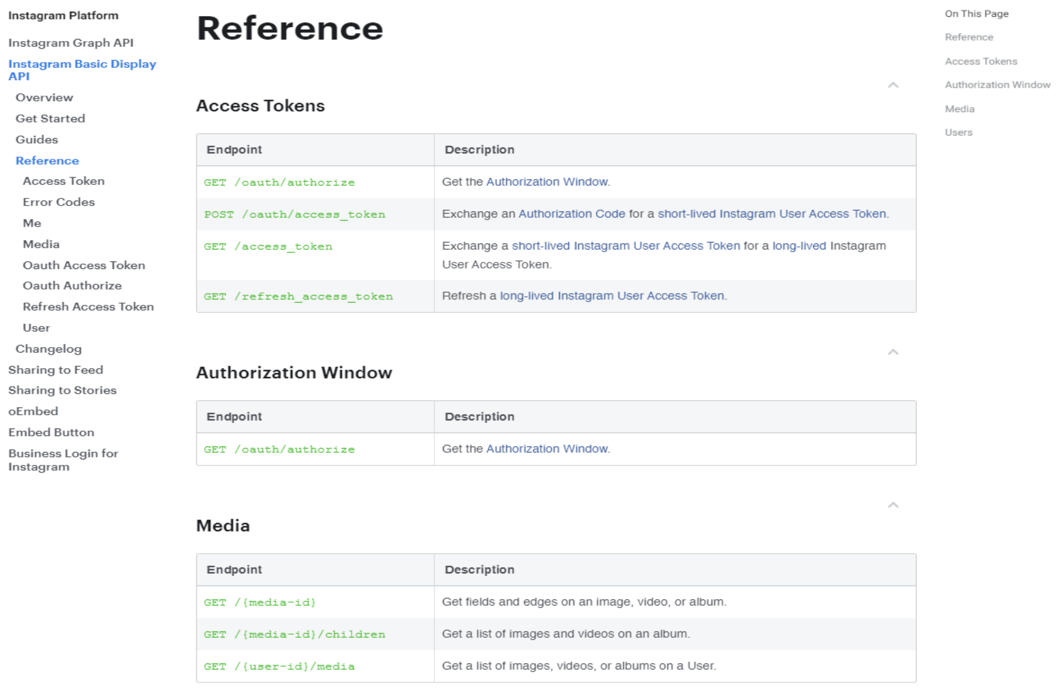
- Explains the document semantics that are not covered by its media type.
  - A profile describes the exact meaning of each **semantic descriptor**

```
<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]

- It is provided to the client either defined **in a text document** or using a specific description language: ALPS, JSON-LD, RDF-Schema, XMDP

Instagram API



Instagram Platform

Instagram Graph API

Instagram Basic Display API

Overview

Get Started

Guides

Reference

Access Token

Error Codes

Me

Media

Oauth Access Token

Oauth Authorize

Refresh Access Token

User

Changelog

Sharing to Feed

Sharing to Stories

oEmbed

Embed Button

Business Login for Instagram

Reference

Access Tokens

Endpoint	Description
`GET /oauth/authorize`	Get the Authorization Window.
`POST /oauth/access_token`	Exchange an Authorization Code for a short-lived Instagram User Access Token.
`GET /access_token`	Exchange a short-lived Instagram User Access Token for a long-lived Instagram User Access Token.
`GET /refresh_access_token`	Refresh a long-lived Instagram User Access Token.

Authorization Window

Endpoint	Description
`GET /oauth/authorize`	Get the Authorization Window.

Media

Endpoint	Description
`GET /{media-id}`	Get fields and edges on an image, video, or album.
`GET /{media-id}/children`	Get a list of images and videos on an album.
`GET /{user-id}/media`	Get a list of images, videos, or albums on a User.

On This Page

Reference

Access Tokens

Authorization Window

Media

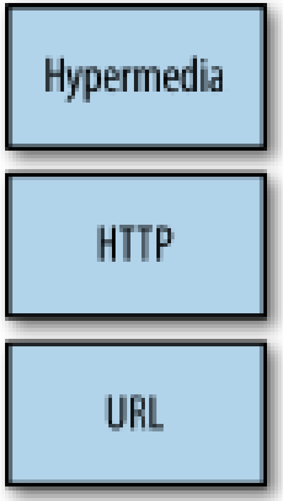
Users

https://developers.facebook.com/docs/instagram-api/

e.g. Comment Moderation: https://developers.facebook.com/docs/instagram-api/guides/comment-moderation

SUMMARY

Programmable Web



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

How can I communicate with the resource?

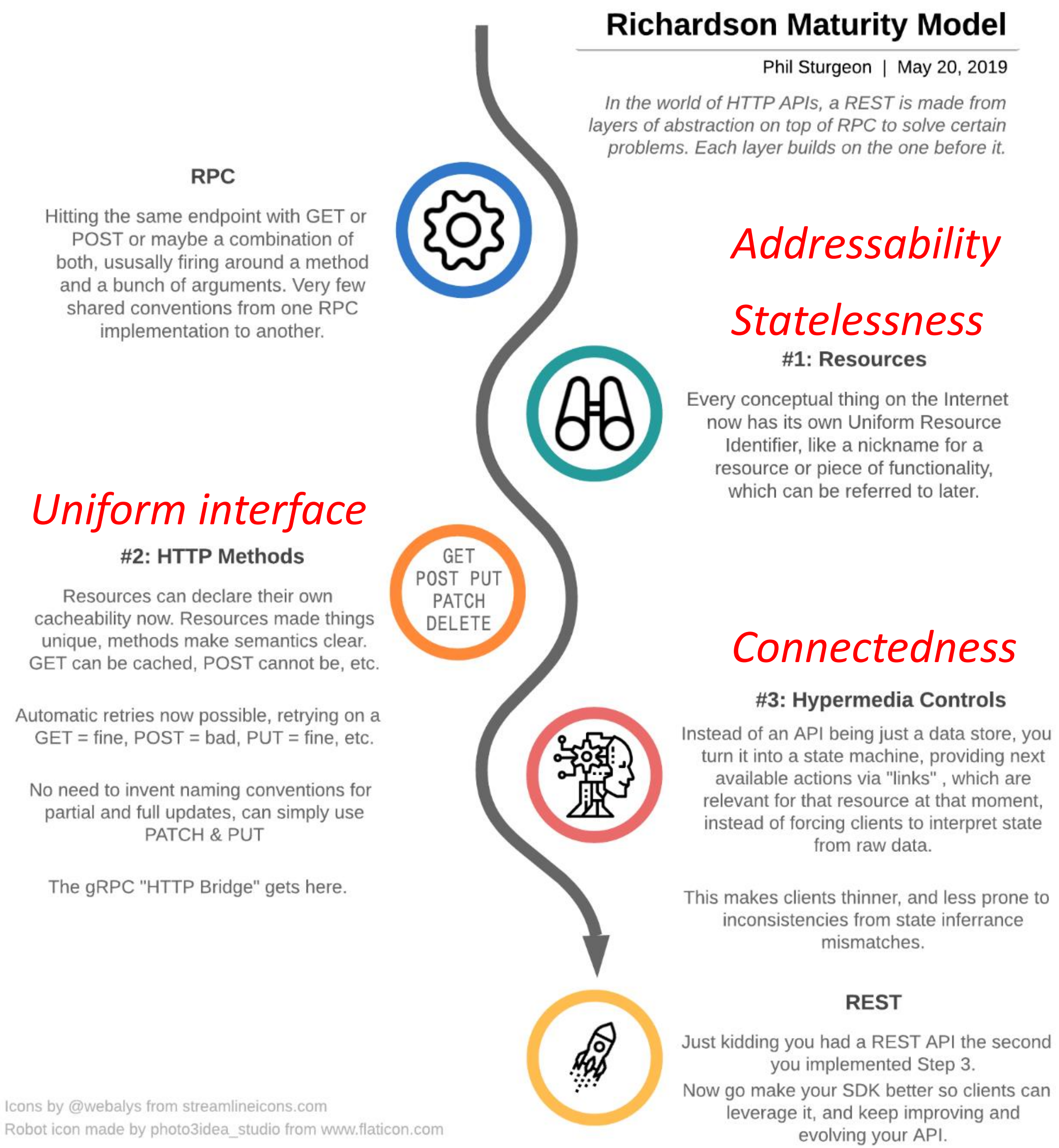
Where is the resource? What is its id?

Web:

- Targeted to humans
- One client

Programmable Web:

- Targeted to machines
- Heterogeneous clients
- Multi language



Richardson Maturity Model

Phil Sturgeon | May 20, 2019

In the world of HTTP APIs, a REST is made from layers of abstraction on top of RPC to solve certain problems. Each layer builds on the one before it.

RPC

Hitting the same endpoint with GET or POST or maybe a combination of both, ususally firing around a method and a bunch of arguments. Very few shared conventions from one RPC implementation to another.

Addressability

Statelessness

#1: Resources

Every conceptual thing on the Internet now has its own Uniform Resource Identifier, like a nickname for a resource or piece of functionality, which can be referred to later.

Uniform interface

#2: HTTP Methods

Resources can declare their own cacheability now. Resources made things unique, methods make semantics clear. GET can be cached, POST cannot be, etc.

Automatic retries now possible, retrying on a GET = fine, POST = bad, PUT = fine, etc.

No need to invent naming conventions for partial and full updates, can simply use PATCH & PUT

The gRPC "HTTP Bridge" gets here.

Connectedness

#3: Hypermedia Controls

Instead of an API being just a data store, you turn it into a state machine, providing next available actions via "links" , which are relevant for that resource at that moment, instead of forcing clients to interpret state from raw data.

This makes clients thinner, and less prone to inconsistencies from state inferrance mismatches.

REST

Just kidding you had a REST API the second you implemented Step 3.

Now go make your SDK better so clients can leverage it, and keep improving and evolving your API.

Icons by @webalys from streamlineicons.com
Robot icon made by photo3idea_studio from www.flaticon.com

https://apisyouwonthate.com/blog/rest-and-hypermedia-in-2019

DESIGN OF RESTFUL WEB APIS USING ROA

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?

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