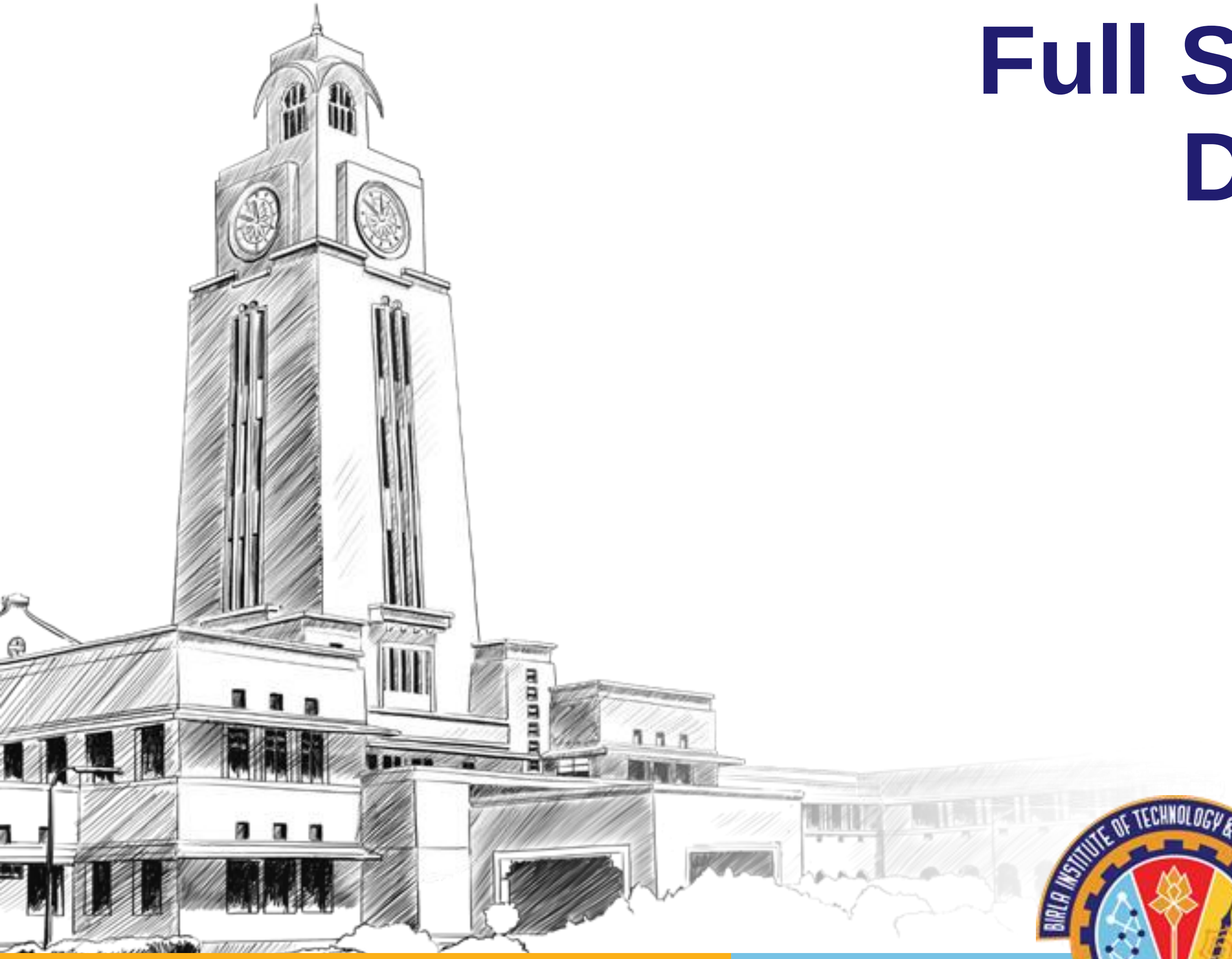


Full Stack Application Development

REST

Akshaya Ganesan

REST - Concepts

- REST stands for REpresentational State Transfer
- REST is an architectural style.

- A RESTful web service
  - ✓ exposes information about itself in the form of information about its resources
  - ✓ enables the client to take actions or perform CRUD operations on those resources.

- Resource
  - ✓ Can be anything(docs, data, media, images) the web service can provide information about
  - ✓ Each resource has a unique identifier - can be a name or a number

When a RESTful API is called, the server will transfer to the client a representation of the state of the requested resource!

REST - example

- When a developer calls Instagram API to fetch a specific user (the resource)
  - ✓ the API will return the state of that user, including
    - their name
    - the number of posts that user posted on Instagram so far
    - how many followers they have
    - and more

- The representation of the state can be in a JSON format
  - ✓ probably for most APIs this is indeed the case
  - ✓ but can also be in XML or HTML format

Resources

- The main building blocks of a REST architecture are the resources
- **Representations:** It can be any way of representing data (binary, JSON, XML, etc.).
- A single resource can have multiple representations.
- **Identifier:** A URL that retrieves only one specific resource at any given time.
- **Metadata:** Content type, last-modified time, and so forth.
- **Control data:** Is-modifiable-since, cache-control.

REST and HTTP

- The fundamental principle of REST is to use the **HTTP** protocol for data communication.
- RESTful web service makes use of HTTP for determining the action to be carried out on the particular resources
- REST gets its motivation from HTTP. Therefore, it can be said as a structural pillar of the REST
- Commonly Used ones
  - ✓ GET - to read (or retrieve) a representation of a resource
  - ✓ POST - utilized to create new resources
  - ✓ PUT - to update a resource
  - ✓ PATCH - to modify a resource - only needs to contain the changes to the resource, not the complete resource
  - ✓ DELETE - to delete a resource identified by a URI.
- Rarely Used ones
  - ✓ Head – requests the headers.
  - ✓ Options - requests permitted communication options for a given URL or server

REST

- Server responds based on the identifier and the operation
  - ✓ An identifier for the resource you are interested in
    - URL for the resource, also known as the endpoint
    - URL stands for Uniform Resource Locator
    - A REST service exposes a URI for every piece of data the client might want to operate on.(Scoping Information)
  - ✓ The operation you want the server to perform on that resource
    - in the form of an HTTP method, or verb
    - common HTTP methods are GET, POST, PUT, and DELETE
    - One way to convey method information in a web service is to put it in the HTTP method(Method Information)

Method Information

- In a SOAP, REST, RPC
- How the client can convey its intentions to the server?

REST Get Request

An HTTP GET request for http://www.oreilly.com/index.html

```
GET /index.html HTTP/1.1
Host: www.oreilly.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.12)...
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,...
Accept-Language: us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-15,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

The response to an HTTP GET request for http://www.oreilly.com/index.html

```
HTTP/1.1 200 OK
Date: Fri, 17 Nov 2006 15:36:32 GMT
Server: Apache
Last-Modified: Fri, 17 Nov 2006 09:05:32 GMT
Etag: "7359b7-a7fa-455d8264
Accept-Ranges: bytes
Content-Length: 43302
Content-Type: text/html
X-Cache: MISS from www.oreilly.com
Keep-Alive: timeout=15, max=1000
Connection: Keep-Alive

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
...
<title>oreilly.com -- Welcome to O'Reilly Media, Inc.</title>
-----
```

Image Ref: Book T2: Restful Web services *Leonard Richardson and Sam Ruby*

SOAP Request

A sample SOAP RPC call

```
POST search/beta2 HTTP/1.1
Host: api.google.com
Content-Type: application/soap+xml
SOAPAction: urn:GoogleSearchAction
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<gs:doGoogleSearch xmlns:gs="urn:GoogleSearch">
<q>REST</q>
...
</gs:doGoogleSearch>
</soap:Body>
</soap:Envelope>
```

Part of the WSDL description for Google's search service

```
<operation name="doGoogleSearch">
<input message="typens:doGoogleSearch"/>
<output message="typens:doGoogleSearchResponse"/>
</operation>
```

XML-RPC Example

```
POST /rpc HTTP/1.1
Host: www.upcdatabase.com
User-Agent: XMLRPC::Client (Ruby 1.8.4)
Content-Type: text/xml; charset=utf-8
Content-Length: 158
Connection: keep-alive
<?xml version="1.0" ?>
<methodCall>
<methodName>lookupUPC</methodName>
...
</methodCall>
```

Example 1-12. An XML document describing an XML-RPC request

```
<?xml version="1.0" ?>
<methodCall>
<methodName>lookupUPC</methodName>
<params>
<param><value><string>001441000055</string></value
></param>
</params>
</methodCall>
```

Scoping Information

- *http://flickr.com/photos/tags/penguin*
- *http://api.flickr.com/services/rest/?method=flickr.photos.search&tags=penguin*
- http://www.upcdatabase.com/upc/00598491'

- One obvious place to put it is in the URI path.

- A RESTful, resource-oriented service exposes a URI for every piece of data the client might want to operate on.

how the client tells the server which part of the dataset to operate on? In REST, SOAP, RPC

Method and Scoping information

- In RESTful web service,
  - ✓ the method information goes into the HTTP method.
  - ✓ The scoping information goes into the URI.
- Given the first line of an HTTP request to a RESTful web service you should understand basically what the client wants to do.
- “GET /reports/docs HTTP/1.1”
- If the HTTP method doesn’t match the method information, the service isn’t RESTful.
- The service is not resource-oriented if the scoping information isn’t in the URI.

Key principles

- Everything is a resource
- Each resource is identifiable by a unique identifier (URI)
- Use the standard HTTP methods
- Resources can have multiple representations
- Communicate statelessly

REST Constraints

- For a web service to be RESTful, it has to adhere to 6 constraints:
  - ✓ Client - Server separation
  - ✓ Stateless
  - ✓ Cacheable
  - ✓ Uniform interface
  - ✓ Layered system
  - ✓ Code-on-demand (Optional)

Client - Server separation

- The client and the server act independently, each on its own
  - ✓ Interaction between them is only in the form of
    - ▪ requests initiated by the client only
    - ▪ responses, which the server send to the client only as a reaction to a request

- The server sits there waiting for requests from the client to come
  - ✓ doesn’t start sending away information about the state of some resources on its own
  - ✓ Responds only when a request comes in

Stateless

- Stateless means the server does not remember anything about the user who uses the service
  - ✓ doesn’t remember if the user already sent a GET request for the same resource in the past
  - ✓ doesn’t remember which resources the user of the API requested before
  - ✓ and so on...

- Each individual request contains all the information the server needs to perform the request and return a response, regardless of other requests made by the same user.
- The client is responsible for sending any state information to the server whenever it's needed.
- No session stickiness or session affinity on the server for the calling request

The statelessness constraint helps services to be more scalable and reliable.

Cacheable

- Data the server sends contain information about whether or not the data is cacheable

- If the data is cacheable, it might contain some version number
  - ✓ version number is what makes caching possible
- The client knows which version of the data it already has (from a previous response)
  - ✓ the client can avoid requesting the same data again and again
- client should also know if the current version of the data is expired,
  - ✓ will know it should send another request to the server to get the most updated data about the state of a resource

Explicit labeling of resource as cacheable or non-cacheable

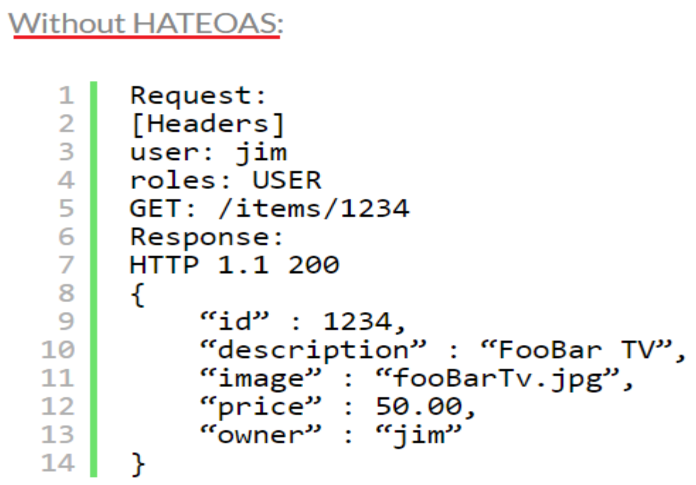
Uniform interface

- There are four guiding principles suggested by Fielding that constitute the necessary constraints to satisfy the uniform interface
  - Identification of resources
  - Manipulation of resources
  - Self-descriptive messages
  - Hypermedia as the engine of application state
- The request to the server has to include a resource identifier
- Each request to the web service contains all the information the server needs to perform the request
  - ✓ Each response the server returns contain all the information the client needs to understand the response
- The response the server returns include enough information so the client can manipulate the resource
- Hypermedia as the engine of application state
  - ✓ Application mean the web application that the server is running
  - ✓ Hypermedia mean the hyperlinks, or simply links, that the server can include in the response
  - ✓ means that the server can inform the client , in a response, of the ways to change the state of the web application

The intention of a uniform interface is to retain some common vocabulary across the internet

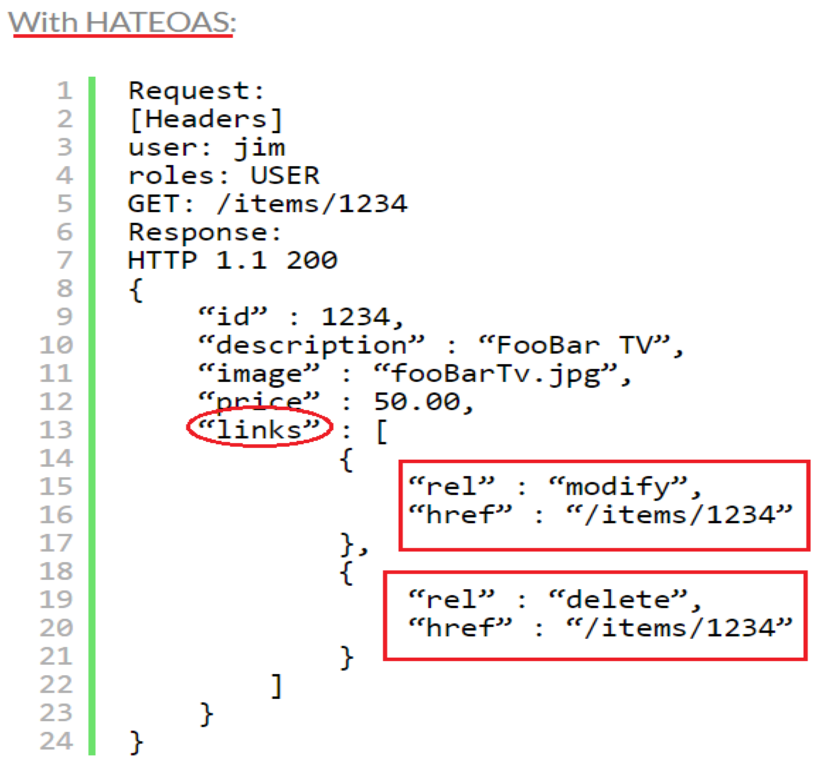
HATEOS

- Hypermedia as the Engine of Application State



Without HATEOAS:

```
Request:
[Headers]
user: jim
roles: USER
GET: /items/1234
Response:
HTTP 1.1 200
{
    “id” : 1234,
    “description” : “FooBar TV”,
    “image” : “fooBarTv.jpg”,
    “price” : 50.00,
    “owner” : “jim”
}
```



With HATEOAS:

```
Request:
[Headers]
user: jim
roles: USER
GET: /items/1234
Response:
HTTP 1.1 200
{
    “id” : 1234,
    “description” : “FooBar TV”,
    “image” : “fooBarTv.jpg”,
    “price” : 50.00,
    “links” : [
            {
                “rel” : “modify”,
                “href” : “/items/1234”
            },
            {
                “rel” : “delete”,
                “href” : “/items/1234”
            }
        ]
    }
}
```

Layered system

- Between the client who requests a representation of a resource's state, and the server who sends the response back
  - ✓ there might be a number of servers in the middle
  - ✓ servers might provide a security layer, a caching layer, a load-balancing layer etc.,
  - ✓ layers should not affect the request or the response

- The client is agnostic as to how many layers, if any, there are between the client and the actual server responding to the request.

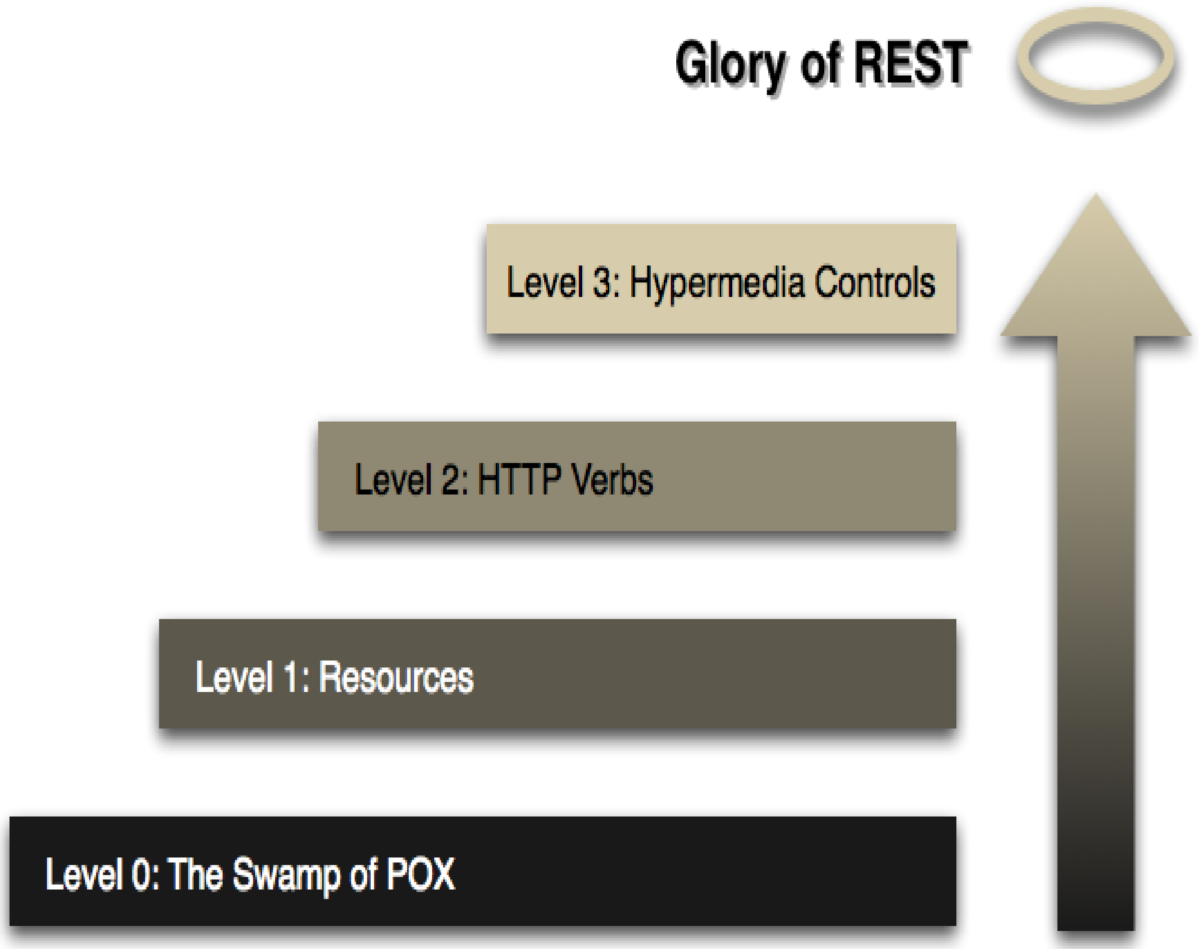
Code-on-demand (Optional)

- Is optional — a web service can be RESTful even without providing code on demand

- The client can request code from the server
  - ✓ response from the server will contain some code
  - ✓ when the response is in HTML format, usually in the form of a script
- The client then can execute that code

REST Maturity Model

- Richardson used three main factors to decide the maturity of service. These factors are
- URI,
- HTTP Methods,
- HATEOAS (Hypermedia)



Summary

REST Constraints

REST Principles

Thank You!