ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

Εισαγωγή στα Δίκτυα Υπηρεσιών

Assisting Lecture 5 - Restful Web Services

Μύρων Παπαδάκης
Τμήμα Επιστήμης Υπολογιστών

CS592: Introduction to Service Networks - Spring 2015

Assisting Lecture: Rest Web Services
Myron Papadakis (myrpap@gmail.com)

References

- Several slides copied from course:INFOH 511 WEB SERVICES
  - LECTURE 2: REST & ROA: http://cs.ulb.ac.be/public/_media/teaching/infoh511/2-rest.pdf

Introduction to Rest

- REST describes a set of architectural principles for designing distributed systems.

- REST principles could be applied to any distributed system, but in practice it is used most often for web applications and services, where clients and servers communicate using the HTTP protocol.

Introduction to Rest

- REST = REpresentational State Transfer.
- REST is a lightweight alternative to mechanisms like RPC (Remote Procedure Calls) and Web Services (SOAP, WSDL, et al.)
- An important concept in REST is the existence of **resources** (sources of specific information), each of which is referenced with a global identifier (e.g., a **URI** in HTTP).
- In order to manipulate these resources, components of the network (user agents and origin servers) communicate via a **standardized interface (e.g., HTTP**) and exchange **representations** of these resources (the actual documents conveying the information).

SOAP Vs Rest: A first look

- Querying a *phonebook application* for the details of a given user. All we have is the user's ID.
- Using SOAP and Web Services

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
 <soap:body pb="http://www.acme.com/phonebook">
  <pb:GetUserDetails>
   <pb:UserID>12345</pb:UserID>
  </pb:GetUserDetails>
 </soap:Body>
</soap:Envelope>
```

- Using Rest
  - http://www.acme.com/phonebook/**UserDetails**/12345
  - It is just a URL..

SOAP Vs Rest Usage

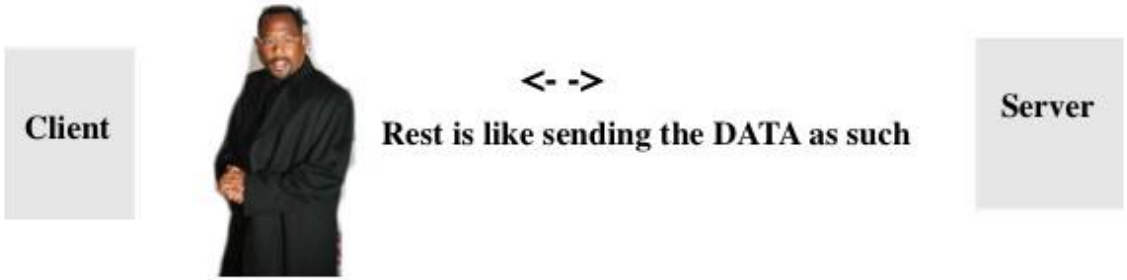
- **Who's using REST?**
  - All of Yahoo's web services use REST, including Flickr, del.icio.us API uses it, pubsub, bloglines, technorati, and both eBay, and Amazon have web services for both REST and SOAP.
- **Who's using SOAP?**
  - Google seams to be consistent in implementing their web services to use SOAP, with the exception of Blogger, which uses XML-RPC. You will find SOAP web services in lots of enterprise software as well.

SOAP Vs Rest

Consider "Martin Lawrence" as your data

SOAP

REST



Source:
http://stackoverflow.com/questions/209905/representational-state-transfer-rest-and-simple-object-access-protocol-soap

Why Rest?

- **Less overhead** (no SOAP envelope to wrap every call in)
- **Less duplication** (HTTP already represents operations like DELETE, PUT, GET, etc. that have to otherwise be represented in a SOAP envelope).
- More standardized - HTTP operations are well understood and operate consistently. Some SOAP implementations can get finicky.
- More human readable and testable (harder to test SOAP with just a browser).
- Don't need to use XML (well, you kind of don't have to for SOAP either but it hardly makes sense since you're already doing parsing of the envelope).
- Libraries have made SOAP (kind of) easy. But you are abstracting away a lot of redundancy underneath as I have noted. Yes, in theory, SOAP cango over other transports so as to avoid riding atop a layer doing similar things, but in reality just about all SOAP work you'll ever do is over HTTP.

SOAP Vs Rest

#	SOAP	REST
1	A XML-based message protocol	An architectural style protocol
2	Uses WSDL for communication between consumer and provider	Uses XML or JSON to send and receive data
3	Invokes services by calling RPC method	Simply calls services via URL path
4	Does not return human readable result	Result is readable which is just plain XML or JSON
5	Transfer is over HTTP. Also uses other protocols such as SMTP, FTP, etc.	Transfer is over HTTP only
6	JavaScript can call SOAP, but it is difficult to implement	Easy to call from JavaScript
7	Performance is not great compared to REST	Performance is much better compared to SOAP - less CPU intensive, leaner code etc.

Rest Vs Soap Design Methodology

- Identify resources to be exposed as services (e.g., book catalog, purchase order)
- Define "nice" URLs to address them
- Distinguish read-only and side-effects free resources (GET) from modifiable resources (POST, PUT, DELETE)
- Relationships (e.g., containment, reference) between resources correspond to hyperlinks that can be followed to get more details
- Implement and deploy on Web server

- List what are the service operations (the "verbs") into the service's WSDL document
- Define a data model for the content of the messages exchanged by the service (XML Schema data types)
- Choose an appropriate transport protocol to bind the operation messages and define the corresponding QoS, security, transactional policies
- Implement and deploy on the Web service container (note the corresponding SOAP engine end-point)

http://webapps.cse.unsw.edu.au/webcms2/course/showfile.php?cid=2366&color=green&addr=Notes/REST_II.pdf

Rest Vs SOAP

- SOAP is verbose: large overhead of metadata and boilerplate text

- For more information see:
  - http://www.petefreitag.com/item/431.cfm

Rest > Key Concepts

- Key Concepts:
  - Resources (things)
  - Resource names (URIs)
  - Resource Representations
  - Links between resources
- And 4 key properties:
  - Addressability
  - Statelessness
  - Connectedness
  - The Uniform Interface

Rest in a nutshell (Key Concepts)

- **Representational State Transfer (REST) is an "architectural style" defined by Roy Fielding.**
  - The concepts of REST are independent of the web, but the web is well-suited to REST

- **Rest Key Concepts. Simplistically, a "RESTful architecture" includes:**
  - Resources (things) with
  - Unique ids (URLS) that can come in many
  - Representations (text, html, json) that you operate on with
  - Verbs (GET, PUT, DELETE, POST)

- **Every object manipulated by the Web Service (or web application) should be identified and exposed as a resource**

- **Here the focus is on interacting with resources rather than messages or operations**

Rest in a nutshell

- **Urls must name resources. They are nouns**
  - http://cnn.com/story/hawaii/fireworks
- **Not verbs:**
  - http://cnn.com/addComment?name=fireworks
- **Use operators (verbs) such as GET, PUT, POST, DELETE to operate on resources (nouns)**
  - **GET http://cnn.com/story/hawaii/fireworks**
  - **PUT http://cnn.com/story/hawaii/explosion**
  - **POST http://cnn.com/story/hawaii/nye/comments**
  - **..**
  - More on these in next slides

Rest > 1. Resources

- Each resource has a unique **name**.

"

> *The key abstraction of information in REST is a resource. Any information that can be named can be a resource: a document or image, a temporal service (e.g., "today's weather in Los Angeles"), a collection of other resources, a nonvirtual object (e.g., a person), a concept and so on.*

Resource examples

- A historical building
- The newspaper "le soir"
- The newspaper "le soir" at a particular date
- The collection of all Belgian newspapers
- The Belgian prime minister
- The preferred newspaper of the prime minister

Rest > 2. URIs (Naming resources)

- In the context of the web (and the HTTP protocol), **each resource is named by a URL**.
- A URI is the name of a resource
- **Guidelines**
  - **Every resource must be assigned at least one URI. This ensure it is addressable**
  - **A URI should never represent more than one resource**
  - **Resources can have multiple URIs, but should have as few URIs as possible**
- **Examples**
  - http://www.lesoir.be
  - http:// www.lesoir.be/edition/20-01-2012
  - http:// www.example.org/newspapers/belgium
  - http:// www.example.org/newspapers?country=belgium

Rest > 2. URIs (Naming resources)

- The URLs specify the server and "path" of the resource.
  - The server part of the URL specifies which server to contact to access the resource.
  - The path part of the URL names the resource to distinguish it from other resources on the same server.

```
GET /news/ HTTP/1.1
Host: example.org
Accept-Encoding: compress, gzip
User-Agent: Python-httplib2
```

Resource = http://example.org/news/

Rest > 2. URIs (Naming resources)

- One important kind of resource is the kinds of fruit that are available. For example, the business might make the list of the kinds of fruit it sells available at the URL
  - http://ycpfruit.com/fruit
- More information about specific kinds of fruit could be made available through URLs like
  - http://ycpfruit.com/fruit/Apples
  - http://ycpfruit.com/fruit/Oranges
- It is common to **organize resource URLs in a directory-like scheme**: for example, we can think of the
  - path /fruit as naming fruit in general, and more specific paths such as
  - /fruit/Apples as naming specific kinds of fruit.

Introduction to Rest > 3. Representations

- There are *representations* of resources.
- For any resource, a web application might present it in a variety of formats.
  - For example, the same resource could be represented using HTML and CSS for presentation in a web browser, and also by XML or JSON for use by programs.

  - There are ***verbs*** which clients can use to create, access, and modify resources. In the context of the web, the verbs are the HTTP request methods.
  - The most common HTTP methods are **GET**, **PUT**, **POST**, **DELETE**.

Rest in a nutshell > Representations (3)

- **Representations (how resources get manipulated)**
  - **A representation is a description (of some part) of the resource**
  - As an external user, you cannot manipulate resources directly.
  - **Instead you manipulate "representations" of that resource**.
    - Many people can "get" a representation of that single resource
    - The same resource can be represented in many different ways
- Example
  - Resource: person (Todd)
  - Service: contact information (GET)
  - Representation
    - Name, address, phone number
    - JSON or XML format
- Example:
  - http:// www.example.org/newspapers/belgium could support both HTML and JSON

Sidenote: JSON, that other XML

- JSON =JavaScript Object Notation
- A syntactical fragment of JavaScript for describing data
- Due to its increased popularity, libraries for reading & writing JSON exist for virtually every programming language
- Sometimes more compact than XML, but since it does not specify a schema, XML-like data-binding tools (JAXB, …) are not available
- Mainly used in conjunction with AJAX

```
{ "reportData":
  [
    { "year": 2011, "profit": 2000000 },
    { "year": 2010, "deficit": 1000   },
  ],
  "author": "John Doe"
}
```

Example of XML-formatted data

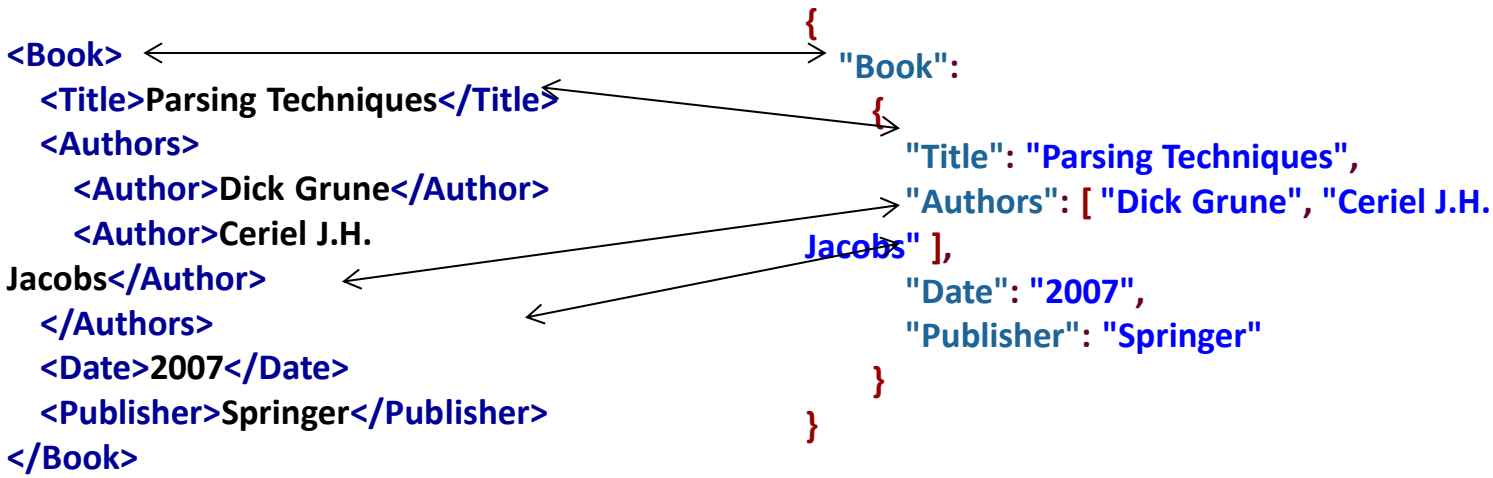
The below XML document contains data about a book: its title, authors, date of publication, and publisher.

```
<Book>
  <Title>Parsing Techniques</Title>
  <Authors>
    <Author>Dick Grune</Author>
    <Author>Ceriel J.H. Jacobs</Author>
  </Authors>
  <Date>2007</Date>
  <Publisher>Springer</Publisher>
</Book>
```

Same data, JSON-formatted

```
{
   "Book":
      {
         "Title": "Parsing Techniques",
         "Authors": [ "Dick Grune", "Ceriel J.H. Jacobs" ],
         "Date": "2007",
         "Publisher": "Springer"
      }
}
```

XML and JSON, side-by-side



More on JSON in next lectures…

Verbs (1/2)

- **In a RESTful web service, the HTTP actions (a.k.a. methods) are used as verbs to create, access, and modify resources.**
- The **GET** verb accesses a resource: in other words, it requests that the web service **return a representation of the named resource.**
- The **POST** verb creates a resource.
  - For example, a **POST** request to the path /fruit, where the body of the request contains a representation of a fruit resource (perhaps in XML or JSON format), would create a new fruit resource (which could then be accessed by subsequent requests).

Verbs (2/2)

- The **PUT** verb creates a new resource or replaces an existing resource. Use PUT to create a new resource when you know the URI for the new resource
  - For example, a **PUT** request to the path /fruit/Apples would replace the Apples resource with the one contained in the request body (assuming that a resource named Apples existed previously).
- The **DELETE** verb deletes the resource named by a resource. For example, a **DELETE** request specifying the path /fruit/Kumquats would delete Kumquats as a fruit, and no further requests to access that resource would succeed.

Rest in a nutshell > The Web and REST

- The web's primary protocol (HTTP) is tailor-made for RESTful architectures.
  - Unique IDs for resources (URIs)
  - Verbs (HTTP operators)
  - Multiple representations (Media types)

- **The web has a uniform and constrained interface.**
  - HTTP, for example, has a small number of **methods. Use these to manipulate resourses**.

- But many (most) web applications do not implement REST!
  - Use of GET for everything
  - URLS that indicate actions, not things.
  - Many other subtle ways to violate REST

Rest Example > The World Wide Web

- **The largest known implementation of a system conforming to the REST architectural style is the World Wide Web**

- A web page is a representation of a resource …
  - The representation is not the resource (we can have several representations)

Rest Requests > How Simple is Rest?

- Querying a phonebook application for the details of a given user. All we have is the user's ID.
- Using SOAP and Web Services

Exposing operations vs exposing resources

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
 <soap:body pb="http://www.acme.com/phonebook">
  <pb:GetUserDetails>
   <pb:UserID>12345</pb:UserID>
  </pb:GetUserDetails>
 </soap:Body>
</soap:Envelope>
```

- Using Rest
  - http://www.acme.com/phonebook/**UserDetails**/12345
  - It is just a URL..
- Despite being simple, REST is fully-featured; there's basically nothing you can do in Web Services that can't be done with a RESTful architecture.

More Complex Rest Requests

- First Example: one parameter
- http://www.acme.com/phonebook/UserDetails?firstName=John&lastName=Doe
- If you need to pass long parameters, or binary ones, you'd normally use HTTP POST requests, and include the parameters in the POST body.
- As a rule, **GET requests should be for read-only queries**;
  - they should not change the state of the server and its data.
- For creation, updating, and deleting data, use POST requests.

SOAP and Rest: The letter analogy

- **A nice analogy for REST vs. SOAP is mailing a letter**:
  - with SOAP, you're using an **envelope**;
  - with REST, it's a **postcard**.
    - Postcards are easier to handle (by the receiver), waste less paper (i.e., consume less bandwidth), and have a short content. (Of course, REST requests aren't really limited in length, esp. if they use POST rather than GET.)

- But don't carry the analogy too far:
  - unlike letters-vs.-postcards, **REST is every bit as secure as SOAP**. In particular,
    - REST can be carried over secure sockets (using the HTTPS protocol), and content can be encrypted using any mechanism you see fit.
    - Without encryption, REST and SOAP are both insecure; with proper encryption in place, both are equally secure.

Rest Constraints (in a nutshell) (1/2)

- The formal REST constraints are:
- Client-Server
  - Assume a disconnected system
  - Uniform interface is the link between the two..
- Stateless
  - Each message (request) is self-descriptive, the message has enough info for the server to process the msg
  - The uri should contain all the state for the given state (use a framework)
- Cache
  - server responses/what comes back are/must be cacheable

Rest Constraints (in a nutshell) (2/2)

- Interface / Uniform Contract
  - Defines the interface between client and server
  - We use the http spec with uris being our resource names
  - WE use the http verbs (post, delete, put, post, some others too..) as the actions we are going to take on these resources
- Layered System
  - Software, hardware intermediaries between client and server
  - Client can't assume direct connection to the server
- Code-On-Demand (optional constraint)
  - Transfer logic to client
  - Client executes code (e.g. javascript)

- ***Violating any constraint other than Code-on-Demand means that the service is not strictly restful***

The Uniform Interface

- **All access to resources happens through HTTP uniform interface (GET, POST, PUT, DELETE, HEAD, OPTIONS).**

- All information necessary to understand the request must be contained in the request message.

- Guideline:
  - Specify, for every URI (and hence, resource): the HTTP methods supported (e.g., GET and POST, but not DELETE, PUT)
  - Allow querying of these operations by supporting OPTIONS

Why the Uniform interface matters

- Consider a GET of
  - http://api.del.icio.us/posts/delete

- This misuses GET and does not adhere to the uniform interface

- But software programs don't know this. Programs that follow a link by GETTing it may hence (inadvertly) delete data.[e.g.,Google Web Accelerator]

- **Guideline: Use the HTTP methods correctly when designing web services.**

Connecting Resources

- Server response representations should include links to other relevant resources
- This makes a web service self-documenting (the representation can be parsed to see what other resources can be accessed).

The Uniform Interface > Safety and indempotency

- All access to the resources happens through HTTP uniform interface (GET, POST, PUT, DELETE, HEAD, OPTIONS)

CRUD	REST	
CREATE	POST	Create a (sub)resource
RETRIEVE	GET	Retrieve a representation of a resource
UPDATE	PUT	Modify a resource/create a new resource
DELETE	DELETE	delete a resource
	OPTIONS	Discover what HTTP methods are supported by the resource
	HEAD	requests headers only (similar to GET but omits representation)

The Uniform Interface > Safety and indempotency

- GET, HEAD, OPTIONS are read-only operations but PUT, POST, DELETE are read-write operations with side effects.
- An operation f is called idempotent if
  - f(f(x)) = f(x)
- PUT and DELETE are idempotent.
- Idempotent and read-only operations can safely be re-executed multiple times (e.g., network timeouts) without risking errors
- POST is not idempotent nor read-only, and is not safe to re-execute

The Uniform Interface

- Idempotent: can be made several times

HTTP	Method	CRUD	Desc.
POST	CREATE	Create	-
GET	RETRIEVE	Retrieve	Safe,Idempotent,Cacheable
PUT	UPDATE	Update	Idempotent
DELETE	DELETE	Delete	Idempotent

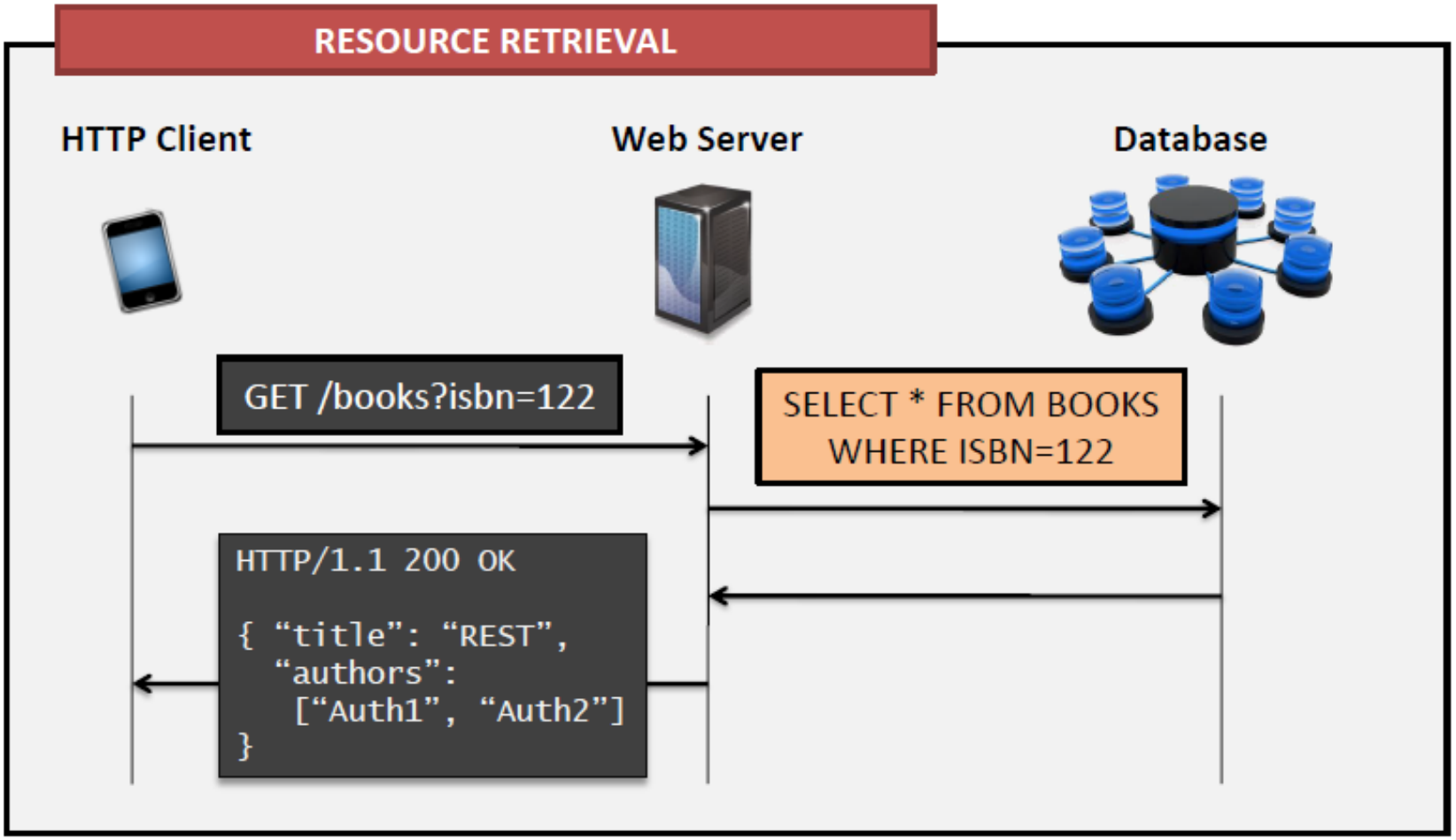
On Post and PUT

- The key difference between PUT and POST is that **PUT is idempotent while POST is not.**
  - No matter how many times you send a PUT request, the results will be same.
- POST is not an idempotent method.
- Making a POST multiple times may result in multiple resources getting created on the server.

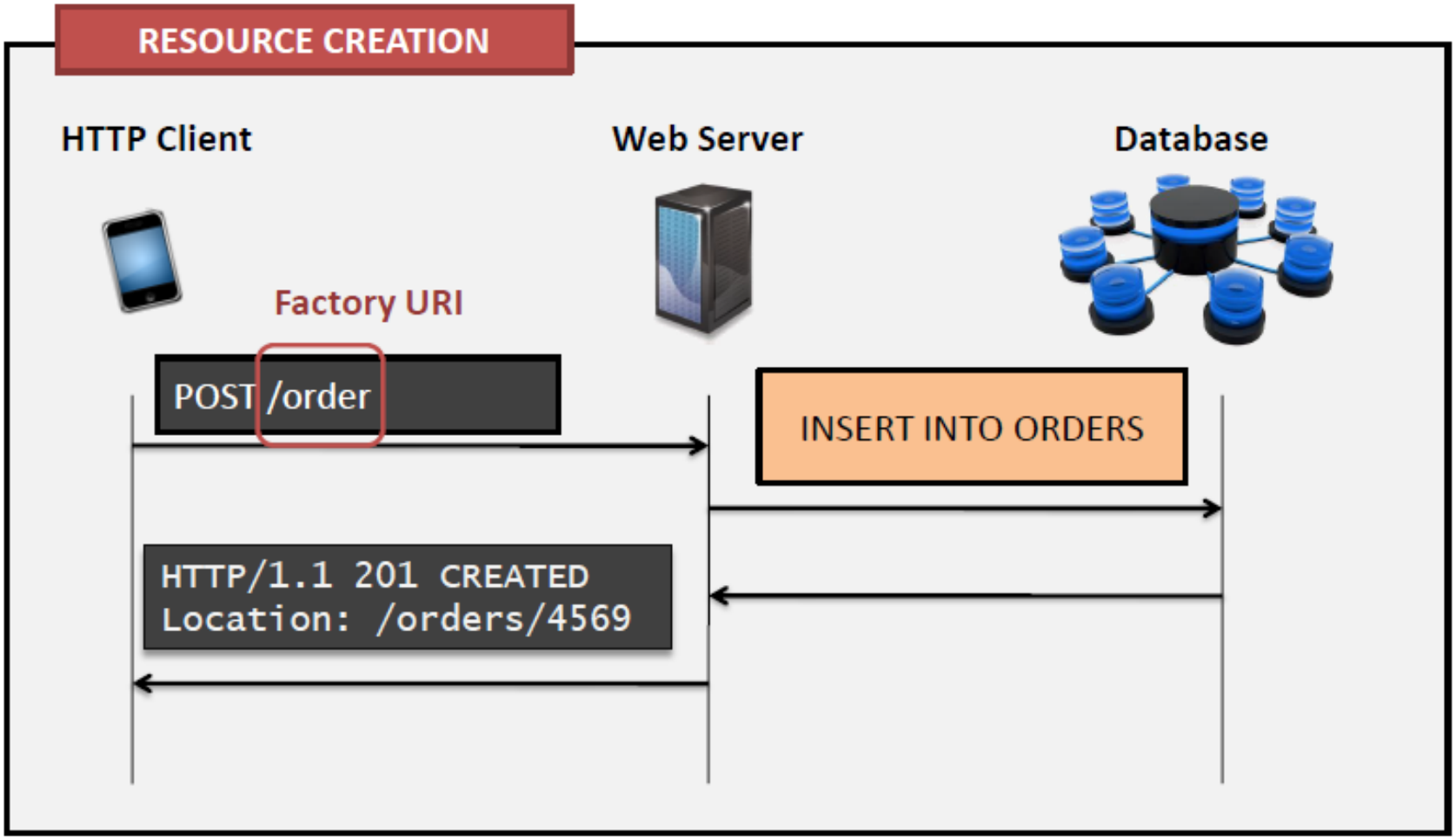
On Post and PUT

- Another difference is that, **with PUT, you must always specify the complete URI of the resource**.
- This implies that the client should be able to construct the URI of a resource even if it does not yet exist on the server.
- This is possible when it is the client's job to choose a unique name or ID for the resource, just like creating a user on the server requires the client to choose a user ID.
- If a client is not able to guess the complete URI of the resource, then you have no option but to use POST.

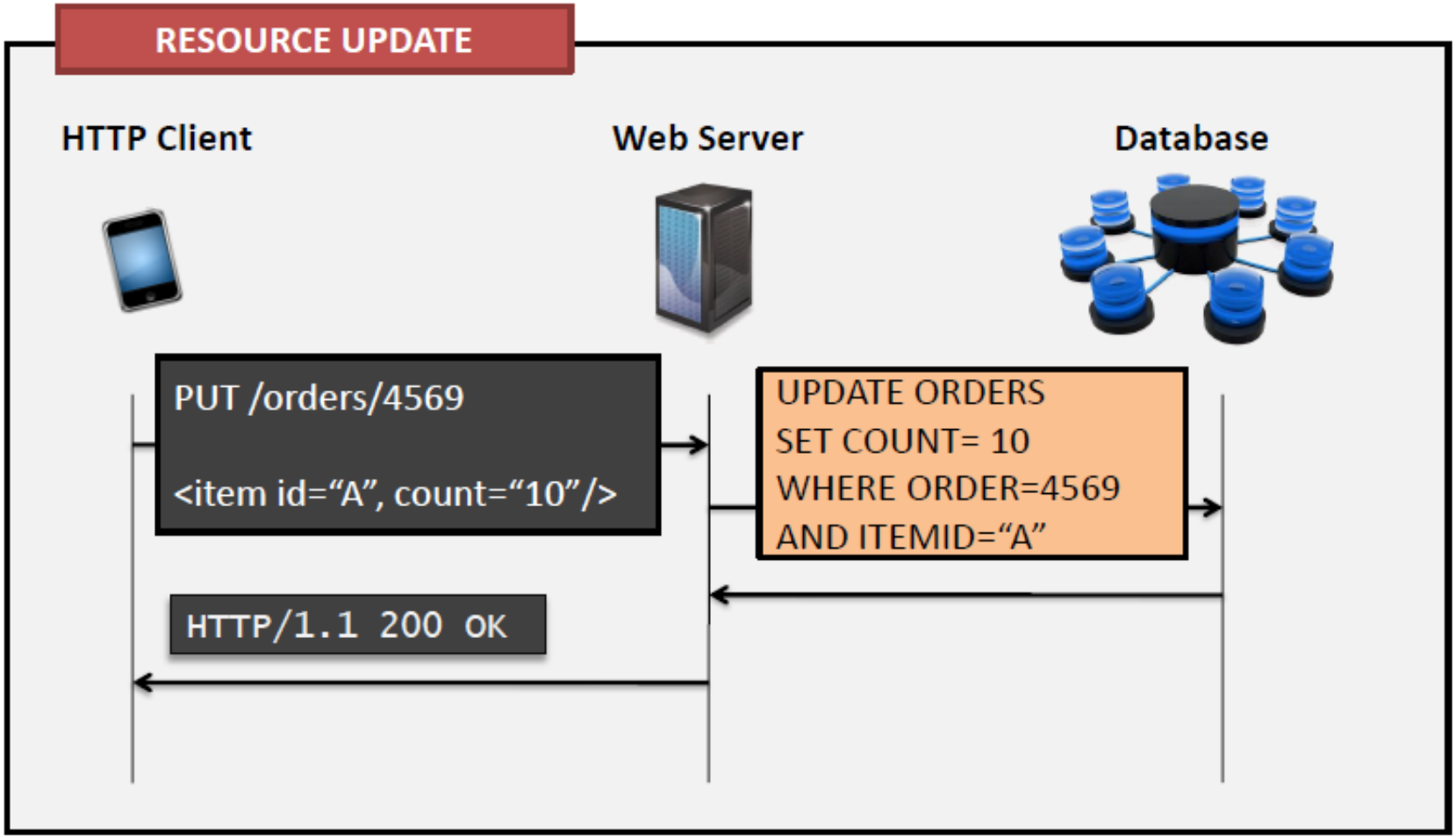
The Uniform Interface



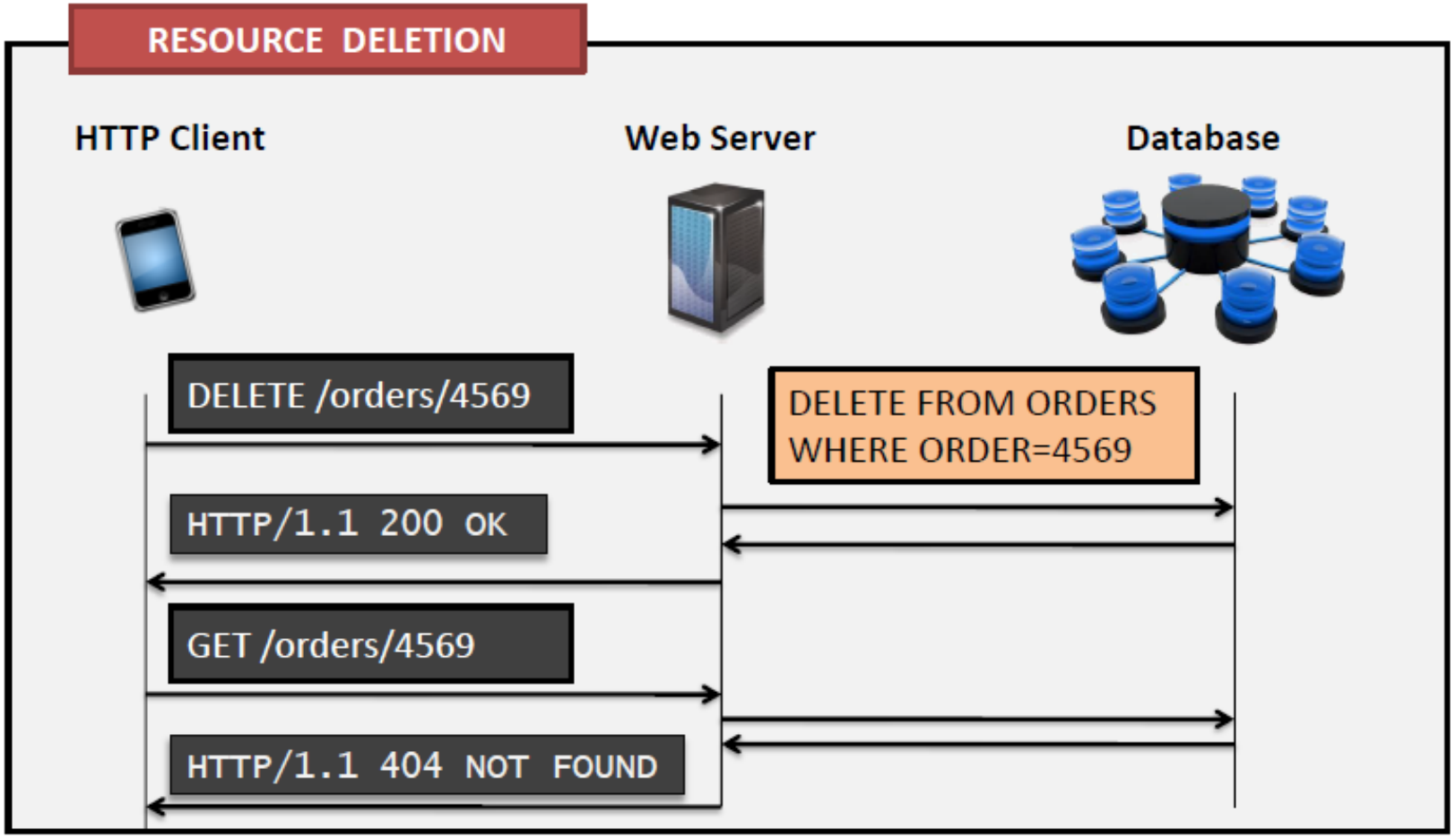
The Uniform Interface



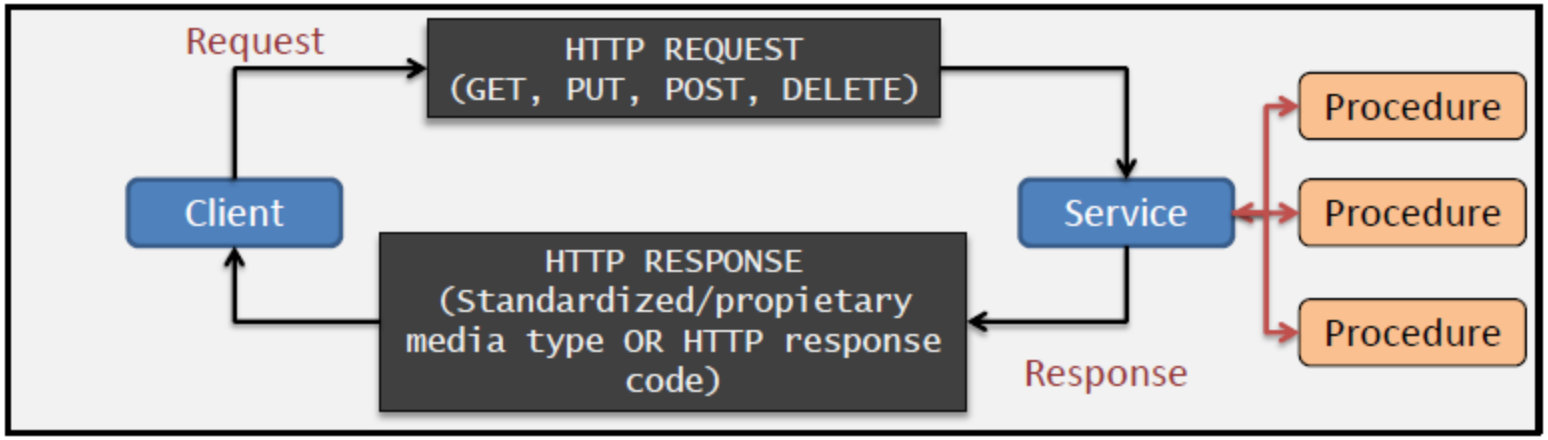
The Uniform Interface



The Uniform Interface



Summary: REST = Resource-based API



- In a resource based API all procedures instances of domain data and files are given a URI.
- HTTP is used as a complete application protocol to define standard service behavior.
- Information is exchanged based on standardized media types (JSON,XML,…) and HTTP response codes where possible
- Clients manipulate the state of resources through representations (e.g., a database table row may be represented as XHTML, XML, or JSON).

Rest in the real world

- Twitter
  - http://developer.twitter.com/doc
- Facebook
  - http://developers.facebook.com/docs/api
- Amazon (http://aws.amazon.com/)
  - Amazon has both SOAP and REST interfaces to their web services, and 85% of their usage is of the REST interface (2003)
  - http://oreilly.com/pub/wlg/3005
- Paypal
  - https://developer.paypal.com/docs/api/
- Others (Google, Yahoo, eBay, Rally, etc)

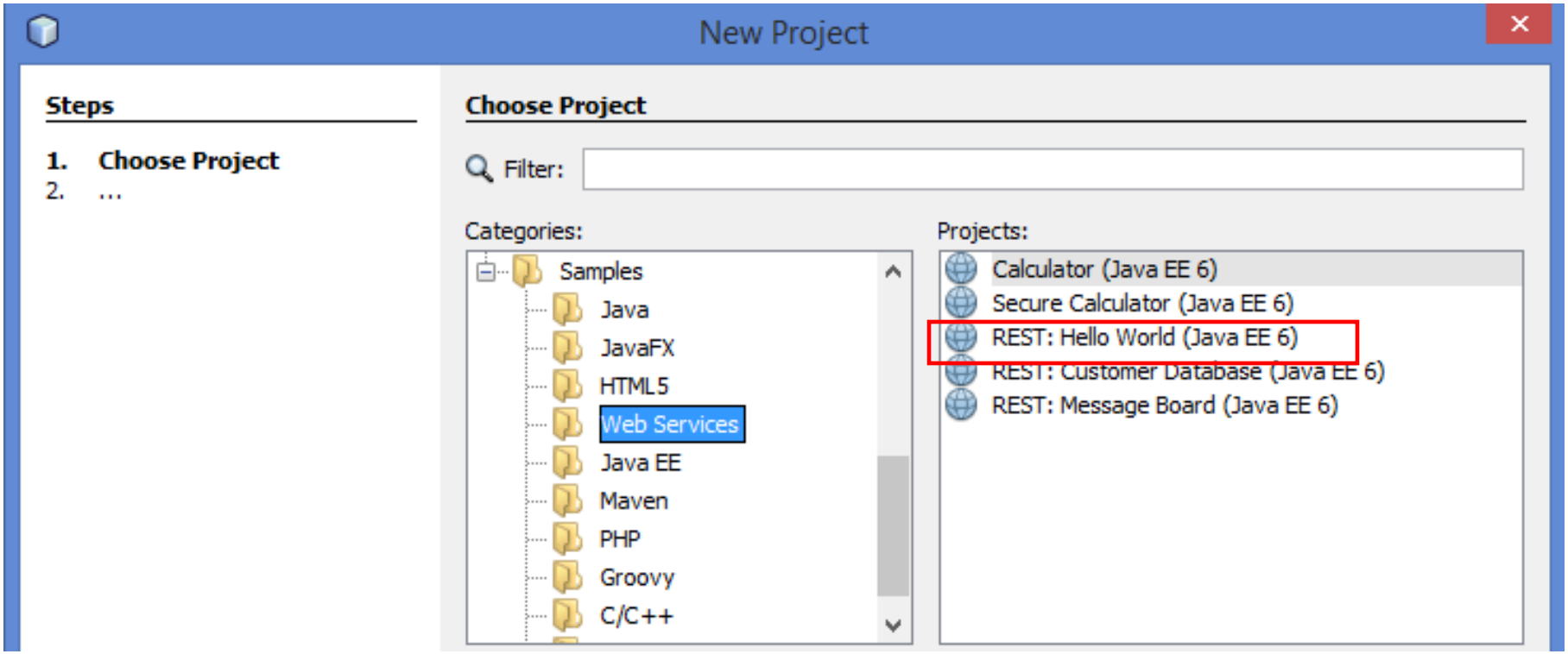
Java API for RESTful Web Services

JAX-RS

Netbeans Restful Example

Import a Sample Project (Also uploaded to moodle for those who have not downloaded the Sample Projects)

Example 2: Import Sample Project



Deploying and Testing

- Right click the project and deploy the application
- Right click and select “Test Restful Services”

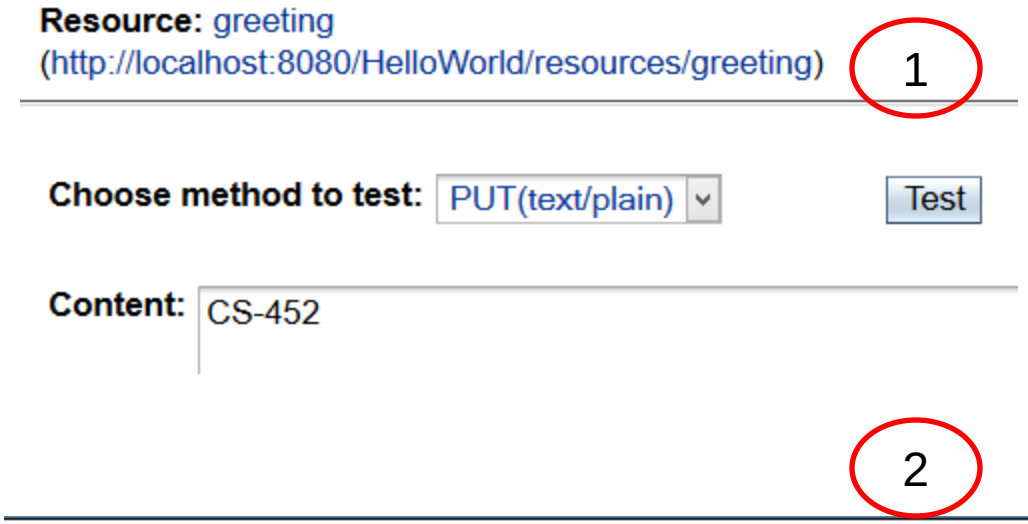
```java
@Stateless
@Path("/greeting")
public class HelloWorldResource {

    @EJB
    private NameStorageBean nameStorage;
    /**
     * Retrieves representation of an instance of helloworld.HelloWorldResource
     * @return an instance of java.lang.String
     */
    @GET
    @Produces("text/html")
    public String getGreeting() {
        return "<html><body><h1>Hello "+nameStorage.getName()+"!</h1></body></html>";
    }

    /**
     * PUT method for updating an instance of Hello
     * @param content representation for the resour
     * @return an HTTP response with content of the
     */
    @PUT
    @Consumes("text/plain")
    public void setName(String content) {
        nameStorage.setName(content);
    }
```

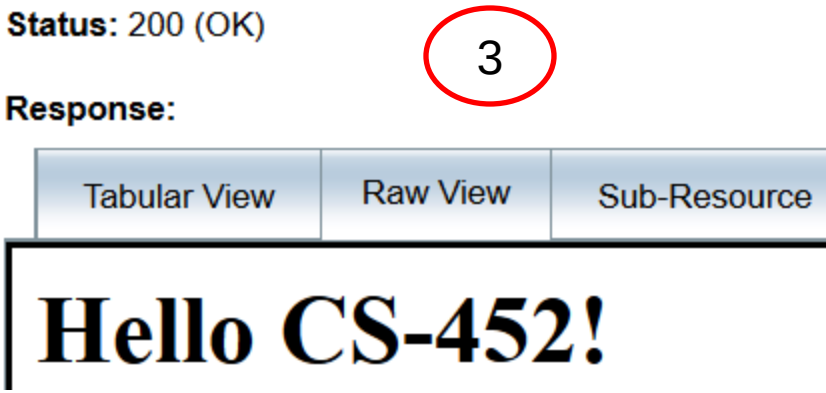
```java
@Singleton
public class NameStorageBean {

    // name field
    private String name = "World";

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Test the Sample Project


```
@Singleton
public class NameStorageBean {

    // name field
    private String name = "World";

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
```

Test the Sample Project > Put a value and get back the response





References

- http://webapps.cse.unsw.edu.au/webcms2/course/showfile.php?cid=2366&color=green&addr=Notes/REST_II.pdf
- http://cs.ulb.ac.be/public/__media/teaching/infoh511/2-rest.pdf

Τέλος Ενότητας

Χρηματοδότηση

•Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.

•Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Κρήτης**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.

•Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Σημειώματα

Σημείωμα αδειοδότησης

- Το παρόν υλικό διατίθεται με τους όρους της άδειας χρήσης Creative Commons Αναφορά Δημιουργού - Μη Εμπορική Χρήση - Παρόμοια Διανομή 4.0 [1] ή μεταγενέστερη, Διεθνής Έκδοση. Εξαιρούνται τα αυτοτελή έργα τρίτων π.χ. φωτογραφίες, διαγράμματα κ.λ.π., τα οποία εμπεριέχονται σε αυτό και τα οποία αναφέρονται μαζί με τους όρους χρήσης τους στο «Σημείωμα Χρήσης Έργων Τρίτων».

[1] http://creativecommons.org/licenses/by-nc-nd/4.0/

•Ως **Μη Εμπορική** ορίζεται η χρήση:

–που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο

–που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο

–που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

•Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

.

Σημείωμα Αναφοράς