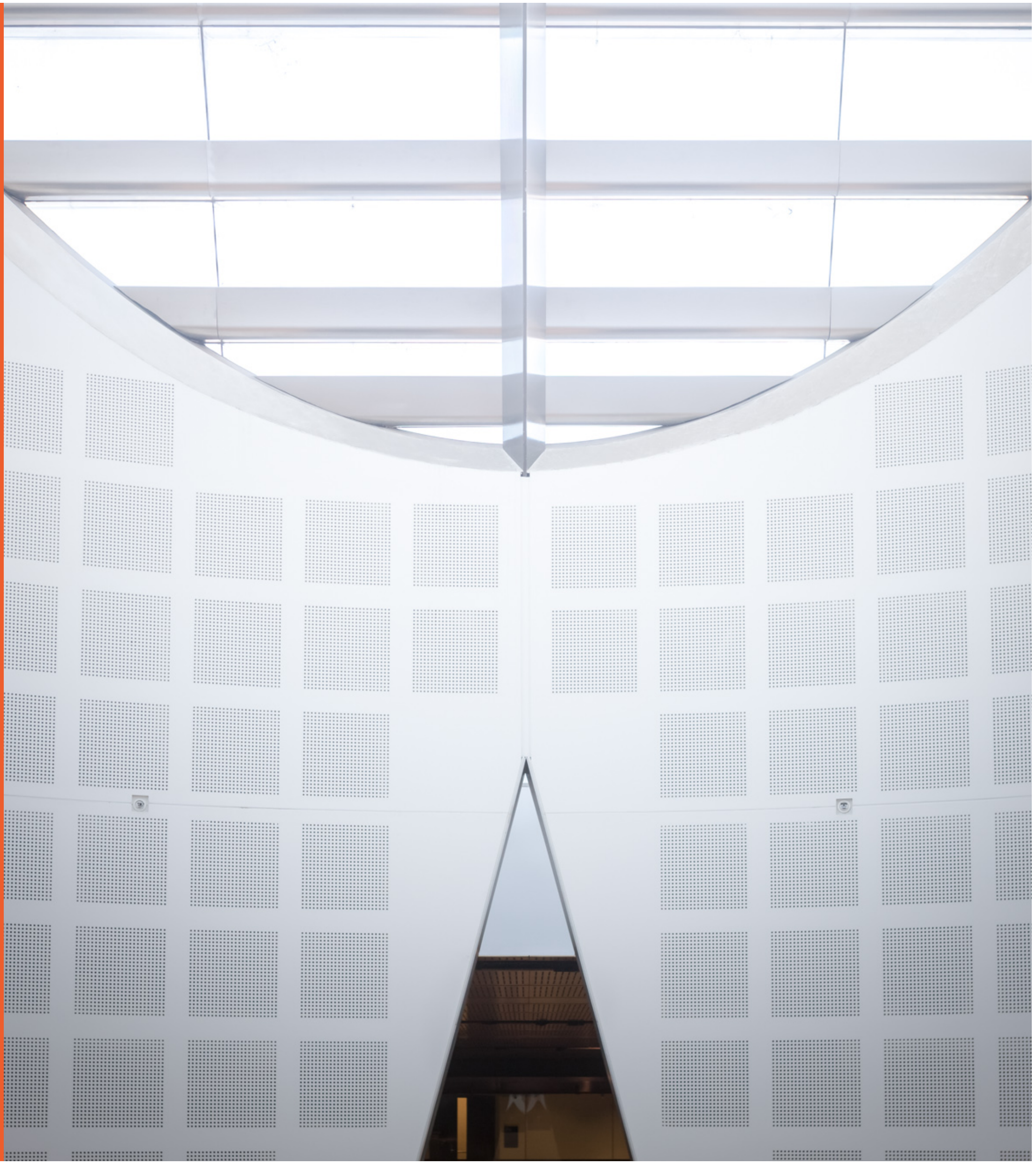
Software Design and Construction 2
SOFT3202 / COMP9202
Representation State Transfer (REST)

Prof Bernhard Scholz

School of Computer Science

THE UNIVERSITY OF SYDNEY



Representation State Transfer (REST)

- Defined by Roy Fielding in his 2000 thesis
  - REpresentational State Transfer (REST, “RESTful”)

- Web and agile developer community has been working on ways to make a ‘Web’ for programs rather than people

  - Adopt ideas that make WWW successful, but suited to consumption by programs rather than human readers

  - “Web-friendly”: use web technologies in ways that match what the Web expects

REST – Architectural Style

- REST is an architectural style rather than a strict protocol

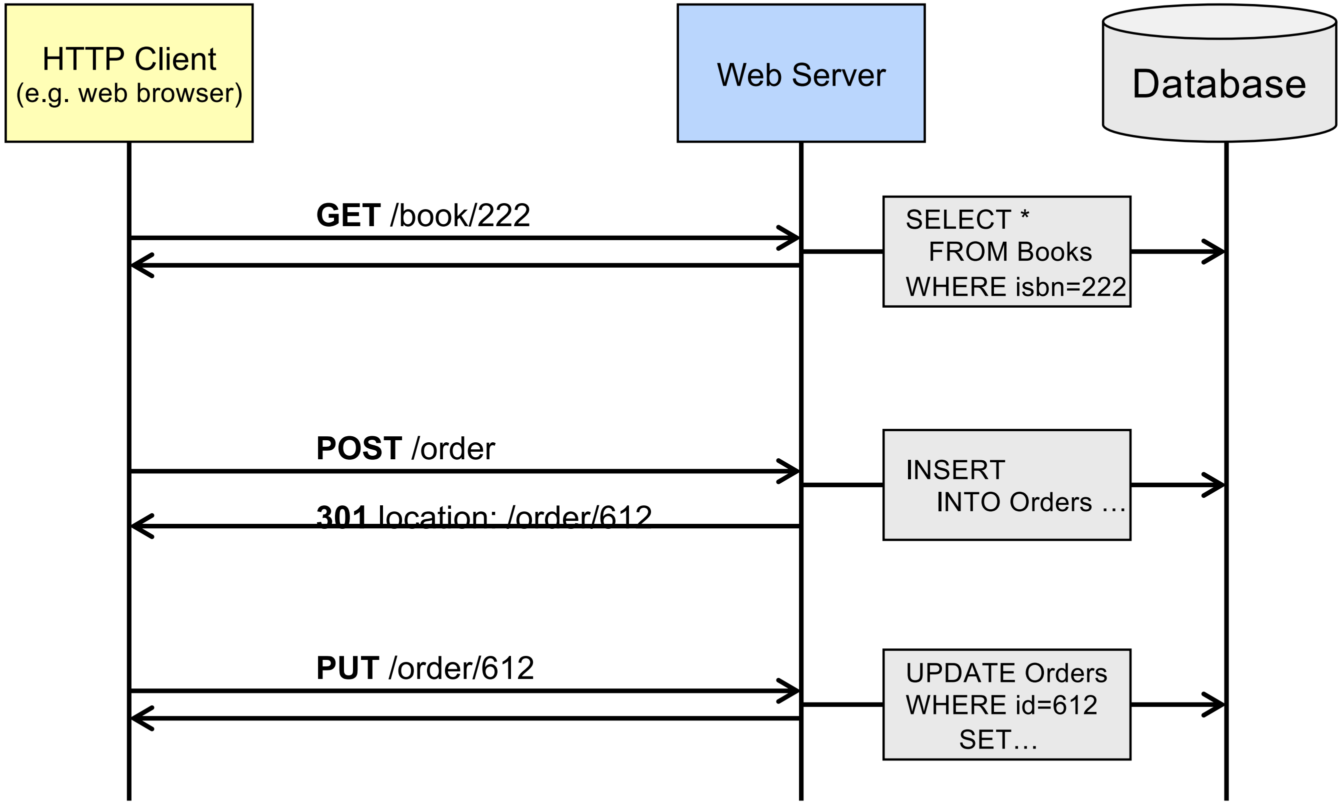
- REST-style architectures consist of clients and servers
  - Requests and responses are built around the transfer of representations of resources

- A resource can be essentially any coherent and meaningful *concept* that may be addressed

- A representation of a resource is typically a document that captures the current or intended *state* of a resource

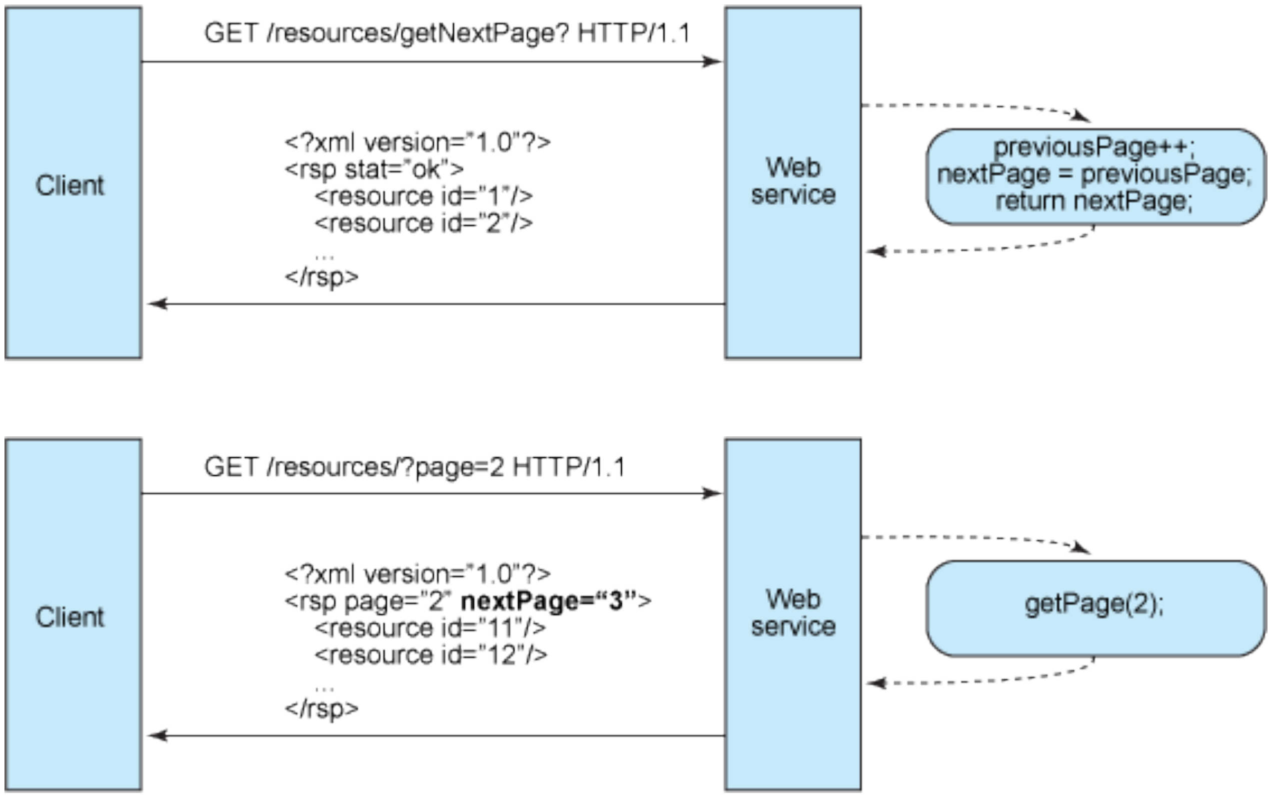
Based on Roy Fielding's doctoral dissertation, rephrased by Wikipedia http://en.wikipedia.org/wiki/Representational_State_Transfer

REST – The Basic Idea



[adapted from Cesare Pautasso, http://www.pautasso.info, 2008]

REST – Stateless vs. Stateful



```
GET /resources/getNextPage? HTTP/1.1
```

```
<?xml version="1.0"?>
<rsp stat="ok">
    <resource id="1"/>
    <resource id="2"/>
    ...
</rsp>
```

```
previousPage++;
nextPage = previousPage;
return nextPage;
```

```
GET /resources/?page=2 HTTP/1.1
```

```
<?xml version="1.0"?>
<rsp page="2" nextPage="3">
    <resource id="11"/>
    <resource id="12"/>
    ...
</rsp>
```

```
getPage(2);
```

http://www.ibm.com/developerworks/webservices/library/ws-restful/

REST – Design Principles

1. Resource Identification through URI

2. Uniform Interface for all resources(HTTP verbs)
   - GET (Query the state)
   - PUT (Modify (or create)
   - POST (Create a resource, with system choosing the identifier)
   - DELETE (Delete a resource)

3. “Self-Descriptive” Messages through Meta-Data

4. Hyperlinks to define the application state
   - Address the resources explicitly in the request message

REST – Tenets

- Resource-based rather than service-oriented
- Addressability: interesting things (resources) should have names
- Statelessness: no stateful conversations with a resource
- Representations: a resource has state representation
- Links: resources can also contain links to other resources
- Uniform Interface: HTTP methods that do the same thing

REST – Resources

- A resource is something "interesting" in your system
  - Anything like Blog posting, printer, a transaction, others

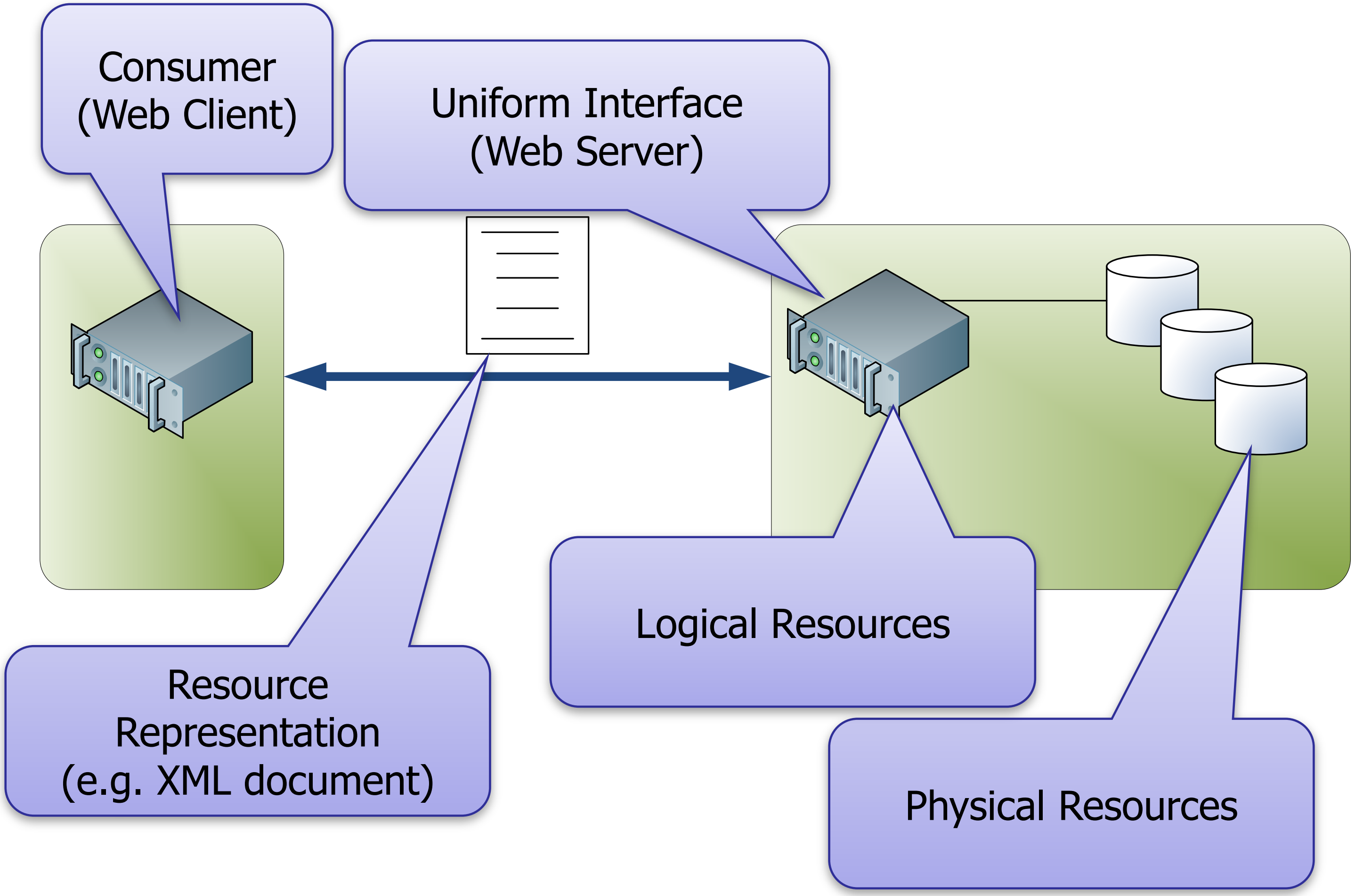
- Requests are sent to URIs ("nouns")
  - Choosing the resources and their URIs is the central design decision for making a RESTful service

- The operations ("verbs") are always the same simple HTTP ones ("get", "post", "put") and they act as expected

REST – Resource Representations

- We deal with representations of resources
  - Not the resources themselves
    - “Pass-by-value” semantics
  - Representation can be in any format
    - Representations like JSON or XML are good for Web-based services
- Each resource has one or more representations
- Each resource implements the uniform HTTP interface
- Resources URIs

REST – Resource Architecture



Consumer (Web Client)

Uniform Interface (Web Server)

Logical Resources

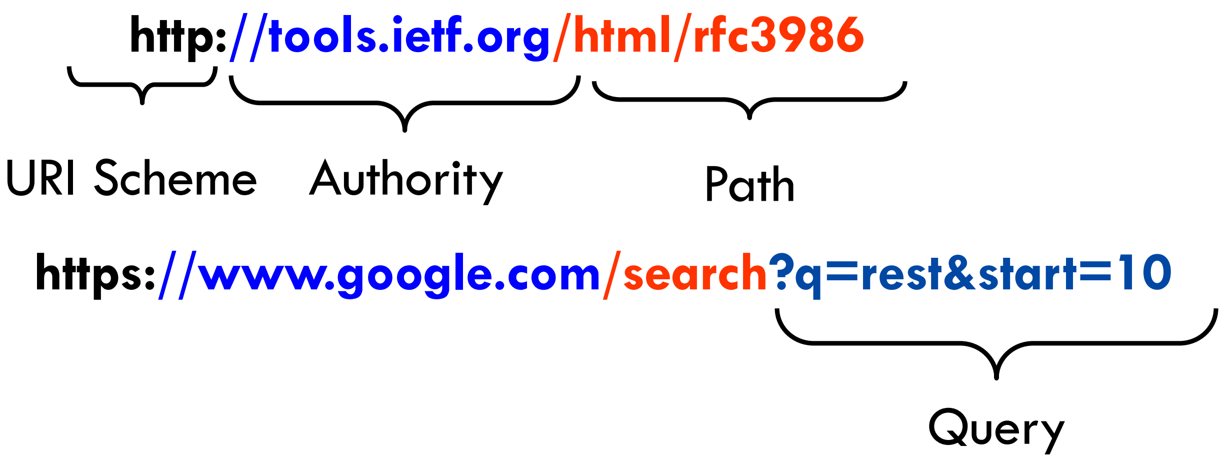
Resource Representation (e.g. XML document)

Physical Resources

REST – Uniform Resource Identifier (URI)

- Internet Standard for resource naming and identification

- Examples:



http://tools.ietf.org/html/rfc3986

URI Scheme Authority Path

https://www.google.com/search?q=rest&start=10

Query

- REST advocates the use of “nice” URIs…
- In most HTTP stacks URIs cannot have arbitrary length (4Kb)

REST – Resource URIs

- URIs should be descriptive?
  - Convey some ideas about how the underlying resources are arranged
    - http://spreadsheet/cells/a2,a9
    - http://jim.webber.name/2007/06.aspx

- URIs should be opaque?
  - Convey no semantics, can't infer anything from them

REST – Links

- Connectedness is good in Web-based systems

- Resource representations can contain other URIs
  - Resources contain links (or URI templates) to other resources

- Links act as state transitions
  - Think of resources as states in a state machine
  - And links as state transitions

- Application (conversation) state is captured in terms of these states
  - Server state is captured in the resources themselves, and their underlying data stores

REST – The HTTP Verbs

- Retrieve a representation of a resource: **GET**
- Get metadata about an existing resource: **HEAD**
- Create a new resource: **PUT** to a new URI,
  or **POST** and the resource will get a new system-chosen URI
- Modify an existing resource: **PUT** to an existing URI
- Delete an existing resource: **DELETE**
- See which of the verbs the resource understands: **OPTIONS**

Resource Types

- Most of the time we can differentiate between *collection type of resources* and *individual resource*
  - Revisions and revision
  - Articles and article

- This can be nested and developers/architect often decide the nesting direction
  - /movies/ForrestGump/actors/TomHanks
  - /directors/AngLee/movies/LifeOfPi

REST – Request URLs and Methods

Action	URL path	Parameters	Example
Create new revision	/revisions		http://localhost:3000/revisions
Get all revisions	/revisions		http://localhost:3000/revisions
Get a revision	/revisions	revision_id	http://localhost:3000/revisions/123
Update a revision	/revisions	revision_id	http://localhost:3000/revisions/123
Delete a revision	/revisions	revision_id	http://localhost:3000/revisions/123

Request Method	Use case	Response
POST	Add new data in a collection	New data created
GET	Read data from data source	Data objects
PUT	Update existing data	Updated object
DELETE	Delete an object	NULL

Uniform Interface Principle (CRUD Example)

CRUD	REST	
CREATE	PUT (user chooses the URI) or POST (system chooses the URI)	Initialize the state of a new resource
READ	GET	Retrieve the current state of a resource
UPDATE	PUT	Modify the state of a resource
DELETE	DELETE	Clear a resource; afterwards the URI is no longer valid

GET Semantics

- GET retrieves the representation of a resource
- Should not affect the resource state (idempotent)
  - Shared understanding of GET semantics
  - Don’t violate that understanding!

POST Semantics

- POST creates a new resource
- But the server decides on that resource's URI
- Common human Web example: posting to a blog
  - Server decides URI of posting and any comments made on that post
- Programmatic Web example: creating a new employee record
  - And subsequently adding to it

POST Request and Response

- Request

```
POST / HTTP/1.1
Content-Type: application/xml
Host: localhost:8888
Content-Length: ....

<buy>
  <symbol>ABCD</symbol>
  <price>27.39</price>
</buy>
```

Verb, path, and HTTP version

Content type (XML)

Content (again XML)

- Response

```
201 CREATED
Location: /orders/jwebber/ABCD/2007-07-08-13-50-53
```

PUT Semantics

- PUT creates a new resource but the client decides on the URI
  - Providing the server logic allows it

- Also used to update existing resources by overwriting them in-place

- Don’t use POST here
  - Because PUT is idempotent!

PUT Request

```
PUT /orders/jwebber/ABCD/2007-07-08-13-50-53 HTTP/1.1
Content-Type: application/xml
Host: localhost:8888
Content-Length: ....

<buy>
  <symbol>ABCD</symbol>
  <price>27.44</price>
</buy>
```

Verb, path and HTTP version

Updated content (higher buy price)

PUT Response

Minimalist response might contain only status and location

```
200 OK
Location: /orders/jwebber/ABCD/2007-07-080-13-50-53
Content-Type: application/xml

<nyse:priceUpdated .../>
```

DELETE Semantics

- Stop the resource from being accessible
  - Logical delete, not necessarily physical
- Request

```
DELETE /user/jwebber HTTP/1.1
Host: example.org
```

- Response

```
200 OK
Content-Type: application/xml
<admin:userDeleted>
  jwebber
</admin:userDeleted>
```

This is important for decoupling implementation details from resources

HEAD Semantics

- HEAD is like GET, except it only retrieves metadata
- Request

```
HEAD /user/jwebber HTTP/1.1
Host: example.org
```

- Response

```
200 OK
Content-Type: application/xml
Last-Modified: 2007-07-08T15:00:34Z
ETag: aabd653b-65d0-74da-bc63-4bca-ba3ef3f50432
```

Useful for caching, performance

OPTIONS Semantics

- Asks which methods are supported by a resource
  - Easy to spot read-only resources for example

- Request

```
OPTIONS /user/jwebber HTTP/1.1
Host: example.org
```

- Response

```
200 OK
Allowed: GET,HEAD,POST
```

You can only read and add to this resource

HTTP Status Codes

- The HTTP status codes provide metadata about the state of resources
- They are part of what makes the Web a rich platform for building distributed systems
- They cover five broad categories
  - 1xx - Metadata
  - 2xx – Everything's fine
  - 3xx – Redirection
  - 4xx – Client did something wrong
  - 5xx – Server did a bad thing
- There are a handful of these codes that we need to know in more detail

REST Strengths

- Simplicity
  - Uniform interface is immutable (harder to break clients)
- HTTP/XML is ubiquitous (goes through firewalls)
- Stateless/synchronous interaction
  - More fault-tolerant
- Proven scalability
  - “after all the Web works”, caching, clustered server farms for QoS
- Perceived ease of adoption (light infrastructure)
  - just need a browser to get started -no need to buy WS-* middleware
  - easy to use with any popular rapid-dev languages/frameworks

REST Weaknesses

- Mapping REST-style synchronous semantics on top of back end systems creates design mismatches (when they are based on asynchronous messaging or event driven interaction)
- Cannot yet deliver all enterprise-style “-ilities” that WS-* can
  - SOAP services/WS-* provides extensive WS framework and extensions
  - E.g., Security and transactions are well-supported in WS-*
- Challenging to identify and locate resources appropriately in all applications
- Semantics/Syntax description very informal (user/human oriented)
- Lack of tool support for rapid construction of clients

References

- “REST in Practice”, by Jim Webber, Savas Parastatidis and Ian Robinson; O’Reilly 2010
- “Architectural Styles and the Design of Network-based Software Architectures” by R. Fielding (PhD thesis, 2000). http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
- Enterprise-Scale Software Architecture (COMP5348) slides
- Web Application Development (COMP5347) slides