
# Designing a good REST service

Pascal Robert
Conatus/MacTI

# Using the correct HTTP verbs and codes

# REST and HTTP

- The principle of REST is to use all of the HTTP spec (verbs, HTTP codes, ...)
- SOAP principle is to use HTTP only for transport

# Good vs bad URLs

- Bad URLs:
  - POST http://api.twitter.com/version/statuses/update.format
  - POST http://dev.twitter.com/doc/post/statuses/destroy/:id
- Good URLs
  - POST http://api.twitter.com/version/statuses.format
  - DELETE  http://dev.twitter.com/doc/post/statuses/:id

# Use the correct HTTP codes

- DON'T use codes in 2xx and 3xx for errors! Use the codes in the 4xx and 5xx range.
- Error codes in the 4xx covers most of the error types, no need to create new codes.
- The 5xx status code range is for server-side errors
- Using the correct codes = automatic handling by browsers and libs!

# 400 (Bad Request)

- If a required attribute is not part of a request's body content, return a 400 (Bad Request) HTTP code instead of 500
- Since June 15, ERRest do it for you!
  - Use ERXRest.strictMode=false if you want to use the old behavior (500)

# 201 Created

- 201 (Created) is the code to use when new resources are created.
- Should have a Location header with a link to the new resource in the response.
  - Why? Client can get a updated representation right away

# 201 Created

```
  public WOActionResults createAction() throws Throwable {
    ERXKeyFilter filter = ERXKeyFilter.filterWithAttributesAndToOneRelationships();
    filter.include(SomeEntity.DESTINATION_ENTITIES.dot(DestinationEntity.SOME_ATTRIBUTE));
    SomeEntity newEntity = create(filter);
    newEntity.editingContext().saveChanges();
    ERXRouteResults response = (ERXRouteResults)response(newEntity, filter);

    String host = this.request()._serverName();
    if (this.request().isSecure())  {
      host = "https://" + host;
    } else {
      host = "http://" + host;
    }
    if (this.request()._serverPort() != null) {
      host = host + ":" + this.request()._serverPort();
    }

    NSDictionary queryDict = new NSDictionary();
    response.setHeaderForKey(host + ERXRouteUrlUtils.actionUrlForRecord(_context, newEntity, "index", this.format
().name(), queryDict, this.request().isSecure(), this.request().isSessionIDInRequest()), "Location");
    return response;
  }
```

# 405 (Not Allowed)

Use this HTTP code when someone try to execute a method (GET/POST/PUT/DELETE) that is not supported.

# 405 (Not Allowed)

If you are using ERXDefaultRouteController...

Instead of doing this:

```
  public WOActionResults destroyAction() throws Throwable {
    return null;
  }
```

You should do this:

```
  public WOActionResults destroyAction() throws Throwable {
    return errorResponse(ERXHttpStatusCodes.METHOD_NOT_ALLOWED);
  }
```

# 405 (Not Allowed)

If you are using ERXRouteController, it's done automatically!

... unless your ERRest framework is older than June 14
... or that ERXRest.isStrictMode = false
... so do that if that's the case:

```
MyBaseController extends ERXRouteController

protected WOActionResults performUnknownAction(String actionName) throws Exception {
  throw new ERXNotAllowedException(actionName);
}

MyBaseMissingController extends ERXMissingRouteController

@Override
public WOActionResults missingAction() {
   boolean isStrictMode = ERXProperties.booleanForKeyWithDefault("ERXRest.strictMode", true);
    if (isStrictMode) {
      return errorResponse(ERXHttpStatusCodes.METHOD_NOT_ALLOWED);
    }
    super.missingAction();
    }
}
```

# Resources that went or moved away

- If a resource was deleted or moved away, handle it with:
  - It's gone for good and you have a trace it's gone? Use HTTP code 410 (Gone).
  - It's gone but can't find out if it was there? Use HTTP code 404 (ERRest do if for you).
  - Resource was moved to a different URL? Use HTTP code 301 and the Location header.

# Long running jobs

- Problem: client send a request, but the job on the server-side will take a long time to do it. You can't really use a WOLongResponsePage in those cases...
- Solution: Use HTTP codes 200, 202 and 303 (See More).
  - Client submit the job, answer with a 202 code.
  - Client ask for status, return 200 while it's still running.
  - Return a 303 code when it's completed.

# Long running jobs

```
    --> POST /some/long/runningJob HTTP/1.1
--> ...Some data...
<-- HTTP/1.1 202 Accepted
<-- Content-Location: http://www.example.org/some/long/runningJob/<idOfJob>

--> GET /some/long/runningJob/<idOfJob> HTTP/1.1
<-- HTTP/1.1 200 Ok
<-- <status>Still running, sorry about that</status>

--> GET /some/long/runningJob/<idOfJob> HTTP/1.1
<-- HTTP/1.1 303 See Other
<-- Location: http://www.example.org/results/of/long/runningJob

--> GET http://www.example.org/results/of/long/runningJob HTTP/1.1
```

# Query arguments

- Use query arguments (?attr1=value&attr2=value) for non-resources data, or to filter results
- Examples:
  - GET /ra/users/login?username=user&password=btggdfgdf
  - GET /ra/products/index?batchSize=10&sort=name
- Use request().formValueForKey() to get the arguments

# Batching/sorting

- If you use ERXFetchSpecification...
  - ?batchSize=25&batch=1
  - ?sort=someAttr|asc,anotherAttr|desc
  - ?qualifier=someAttr%3D'SomeValue'
- Example: /cgi-bin/WebObjects/App.woa/ra/stuff.json?batchSize=5&sort=endDate|desc&qualifier=name%3D'General Labs'
- If not using ERXFetchSpecification, use request().formValueForKey to get those parameters
- Can also use the Range/Content-Range header for batching, but that's a violation of the HTTP spec

# Caching

# Why caching?

- To reduce bandwith and processing time
- Access to data when disconnected from the Net
- More performance when people are behind proxies

# When caching can't be done

- POST/PUT/DELETE requests are not cached, only GET and HEAD can be cached
- SSL responses are not cached in proxies

# Caching options

- Last-Modified/If-Modified-Since
- Etag/If-None-Match
- Cache-control/Expires

# Last-Modified/If-Modified-Since

- If-Modified-Since: Client check if the requested resource was updated after that value
- Last-Modified: Server return the date of when the resource was last updated
- HTTP code 304 (Not Modified): return this if the document was not updated after the value of If-Modified-Since
- Will reduce bandwith and processing usage
- ...You do need something on your EO that can generate a "last updated" timestamp to set Last-Modified

# Last-Modified/If-Modified-Since

```
First request:

--> GET /some/attribute.json HTTP/1.1
<-- HTTP/1.1 200 Ok
<-- Last-Modified: 25 May 2011 13:02:30 GMT
<-- { someData }

Second request:

--> GET /some/attribute.json HTTP/1.1
--> If-Modified-Since: 25 May 2011 13:02:30 GMT
 if last modified for representation == 25 May 2011 13:02:30 GMT
<-- HTTP/1.1 304 Not Modified
  else
<-- HTTP/1.1 200 OK
<-- { someData }
```

# Last-Modified/If-Modified-Since

```
  public WOActionResults showAction() throws Throwable {
  SomeEntity someEntity = routeObjectForKey("someEntity");

  String ifModifiedSinceHeader = request().headerForKey("If-Modified-Since");
  String strLastModifiedDate = formatter.format(someEntity.lastModified());
  java.util.Calendar later = GregorianCalendar.getInstance();
  later.add(Calendar.HOUR, 1);

  if (ifModifiedSinceHeader != null) {
    NSTimestamp ifModifiedSince = new NSTimestamp(formatter.parse(ifModifiedSinceHeader));
    if ((ifModifiedSince.after(someEntity.lastModified())) || (ifModifiedSince.equals(someEntity.lastModified()))) {
      WOResponse response = new WOResponse();
      response.setStatus(304);
      response.appendHeader(strLastModifiedDate, "Last-Modified");
      response.appendHeader(formatter.format(later), "Expires");
      return response;
    }
  }

  ERXRouteResults results = (ERXRouteResults)response(someEntity, ERXKeyFilter.filterWithAttributesAndToOneRelationships());
  results.setHeaderForKey(strLastModifiedDate, "Last-Modified");
  results.setHeaderForKey(strLastModifiedDate, "Expires");
  results.setHeaderForKey("max-age=600,public", "Cache-Control");
  return results;
}
```

# ETag/If-None-Match

- ETag is like a hash code of the data representation
- Better than Last-Modified, no need for a stored timestamp
  - ... but you do need to generate a hash code that will stay the same if the EO didn't change
  - ... and that's not easy to do with EOs
  - Best option is to create a MD5 digest with some values coming from attributes

# ETag/If-None-Match

```
First request:

--> GET /some/attribute.json HTTP/1.1
<-- HTTP/1.1 200 Ok
<-- ETag: a135790
<-- { someData }

Second request:

--> GET /some/attribute.json HTTP/1.1
--> If-None-Match: a135790
 if generated ETag for representation == a135790
<-- HTTP/1.1 304 Not Modified
  else
<-- HTTP/1.1 200 OK
<-- { someData }
```

# ETag/If-None-Match

```
  public WOActionResults showAction() throws Throwable {
  SomeEntity someEntity = routeObjectForKey("someEntity");

  String ifNoneMatchHeader = request().headerForKey("If-None-Match");

  ERXRouteResults results = (ERXRouteResults)response(someEntity, ERXKeyFilter.filterWithAttributesAndToOneRelationships());
  String etagHash = ERXCrypto.sha256Encode(results.responseNode().toString(results.format(), this.restContext()));

  if (ifNoneMatchHeader != null) {
    if (ifNoneMatchHeader.equals(etagHash)) {
      WOResponse response = new WOResponse();
      response.setStatus(ERXHttpStatusCodes.NOT_MODIFIED);
      response.appendHeader(etagHash, "ETag");
      return response;
    }
  }

  results.setHeaderForKey(etagHash, "ETag");
  return results;
}
```

# Cache-control/Expires

- Both let you add a "age" on the request
- Cache-control is for HTTP 1.1 clients
- Expires is for HTTP 1.0 clients
  - It's just a date value

# Cache-control options

- Visibility:
  - **public**: default, both private and shared (eg, proxies) will cache it
  - **private**: client will cache it, but shared caches (eg, proxies) will not cache it
  - **no-cache** and **no-store**: don't store a cache
- Age:
  - **max-age**: freshness lifetime in seconds
  - **s-maxage**: same as max-age, but for shared caches
- Revalidation:
  - **must-revalidate**: cache(s) have to check the origin server before serving a cached presentation
  - **proxy-revalidate**: same as must-revalidate, but for shared caches only

# Using Cache-control

```
First request:

--> GET /some/attribute.json HTTP/1.1
<-- Date: 25 May 2011 14:00:00 GMT
<-- HTTP/1.1 200 Ok
<-- Last-Modified: 25 May 2011 13:02:30 GMT
<-- Cache-Control: max-age=3600,must-revalidate
<-- Expires: 25 May 2011 15:00:00 GMT
<-- { someData }

Second request made 10 minutes later, response will come from the cache:

--> GET /some/attribute.json HTTP/1.1
<-- HTTP/1.1 200 OK
<-- Cache-Control: max-age=3600,must-revalidate
<-- Expires: 25 May 2011 15:00:00 GMT
<-- Age: 600
```

# Our friend : optimistic locking

# Optimistic locking

- If you have two REST clients doing a PUT on the same resource, client B will override the changes made by client A... No optimistic locking for you!
- ... Unless you use the ETag or Last-Modified headers

# Optimistic concurrency control

- Last-Modified/If-**Un**modified-Since
- ETag/If-Match
- HTTP code 412 (Precondition failed)

# Optimistic concurrency control (Last-Modified method)

```
Both Client A and Client B ask for the same data:

--> GET /some/data.json HTTP/1.1
<-- Date: 25 May 2011 14:00:00 GMT
<-- HTTP/1.1 200 Ok
<-- Last-Modified: 25 May 2011 13:02:30 GMT
<-- Expires: 25 May 2011 15:00:00 GMT
<-- { someData }

Client A send an update:

--> PUT /some/data.json HTTP/1.1
--> If-Unmodified-Since: 25 May 2011 13:02:30 GMT
{ someAttribute: 'blabla from client A', id: 1, type: 'SomeEntity' }
-----> Last-Modified on the representation before the update is 25 May 2011 13:02:30 GMT, so we are good to go <-----
<-- HTTP/1.1 200 OK
<-- Last-Modified: 25 May 2011 14:05:30 GMT
<-- { someData }

Client B send an update:

--> PUT /some/data.json HTTP/1.1
--> If-Unmodified-Since: 25 May 2011 13:02:30 GMT
{ someAttribute: 'blabla from client B', id: 1, type: 'SomeEntity' }
-----> BUSTED! Last-Modified on the representation is now 25 May 2011 14:05:30 because of Client A update! <-----
<-- HTTP/1.1 412 Precondition Failed
<-- { error: 'Ya man, someone before you updated the same object' }
```

# How-to with Last-Modified

```
   public WOActionResults updateAction() throws Throwable {
    SomeEntity someEntity = routeObjectForKey("someEntity");

    String ifUnmodifiedSinceHeader = request().headerForKey("If-Unmodified-Since");

    if (ifUnmodifiedSinceHeader != null) {
      NSTimestamp ifUnmodifiedSince = new NSTimestamp(formatter.parse(ifUnmodifiedSinceHeader));
      if (!(ifUnmodifiedSince.equals(someEntity.lastModified()))) {
        return errorResponse("Ya man, someone before you updated the same object",
ERXHttpStatusCodes.PRECONDITION_FAILED);
      }
    }

    update(someEntity, ERXKeyFilter.filterWithAll());
    someEntity.setLastModified(new NSTimestamp());
    someEntity.editingContext().saveChanges();

    String strLastModifiedDate = formatter.format(new java.util.Date(someEntity.lastModified().getTime()));
    ERXRouteResults response = (ERXRouteResults)response(someEntity, ERXKeyFilter.filterWithAll());
    response.setHeaderForKey(strLastModifiedDate, "Last-Modified");
    response.setHeaderForKey(strLastModifiedDate, "Expires");
    return response;
  }
```

# Optimistic concurrency control

The "last modified" date should be a value that clients can't update

# Optimistic concurrency control (ETag method)

```
Both Client A and Client B ask for the same data:

--> GET /some/data.json HTTP/1.1
<-- Date: 25 May 2011 14:00:00 GMT
<-- HTTP/1.1 200 Ok
<-- ETag: a2468013579
<-- Expires: 25 May 2011 15:00:00 GMT
<-- { someData }

Client A send an update:

--> PUT /some/data.json HTTP/1.1
--> If-Match: a2468013579
{ someAttribute: 'blabla from client A', id: 1, type: 'SomeEntity' }
-----> ETag on the representation before the update was a2468013579, so we are good to go <-----
<-- HTTP/1.1 200 OK
<-- ETag: b3579024680
<-- { someData }

Client B send an update:

--> PUT /some/data.json HTTP/1.1
--> If-Match: a2468013579
{ someAttribute: 'blabla from client B', id: 1, type: 'SomeEntity' }
-----> BUSTED! ETag on the representation is now b3579024680 because of Client A update! <-----
<-- HTTP/1.1 412 Precondition Failed
<-- { error: 'Ya man, someone before you updated the same object!' }
```

# How-to with ETag

```
  public WOActionResults updateAction() throws Throwable {
  SomeEntity someEntity = routeObjectForKey("someEntity");

  ERXRouteResults results = (ERXRouteResults)response(someEntity, ERXKeyFilter.filterWithAttributesAndToOneRelationships());
  String etagHash = ERXCrypto.sha256Encode(results.responseNode().toString(results.format(), this.restContext()));

  String ifMatchHeader = request().headerForKey("If-Match");

  if (ifMatchHeader != null) {
    if (!(ifMatchHeader.equals(etagHash))) {
      return errorResponse("Ya man, someone before you updated the same object", ERXHttpStatusCode.PRECONDITION_FAILED);
    }
  }

  update(someEntity, ERXKeyFilter.filterWithAll());
  someEntity.editingContext().saveChanges();

  results.setHeaderForKey(etagHash, "ETag");

  return results;
}
```

# Future topics (WebEx)

- Localization
- ERRest and Dojo
- ERRest and SproutCore
- HTML5 Storage
- File uploads/downloads
- Consuming REST services
- ERRest and Titanium
- HATEOAS/Atom

# Resources

- RESTful Web Services Cookbook (O'Reilly)
- REST in Practice (O'Reilly)


# Q&A

Pascal Robert