

TECHNICAL APPROACH DESIGN AND DEVELOPMENT OF RESTFUL WEB SERVICES

Army Training Information Architecture-Migrated (ATIA-M) System Sustainment Services

Contract: W911S0-12-C-0036

10 DECEMBER 2012



Prepared for:

U.S. Army Training Support Center (ATSC)
Attn: TCM-ATIS
Building 3308
Fort Eustis, VA 23604-5166

Submitted by:



ZCSolutions, LLC
1600 Tysons Boulevard, Suite 1150
McLean, VA 22102



REVISION RECORD

Version	Description	Date	Submitter
1.0	Initial draft	10 Dec 2012	B. Craig

DRAFT

Table of Contents

1	Purpose	4
2	Goal	4
3	Representational State Transfer (REST)	4
3.1	HTTP Request Methods	4
3.1.1	GET	4
3.1.2	PUT	4
3.1.3	POST	5
3.1.4	DELETE	5
3.1.5	HEAD	5
3.1.6	OPTIONS	5
3.2	Safety and Idempotency	5
3.3	REST Principles	5
4	RESTful Resource-Oriented Architecture	5
4.1	The Resource	6
4.2	The Resource URI	6
4.3	The Resource Representations	6
4.4	The Links between	6
5	Service Design	6
6	Client	6
7	Rules	7
7.1	URI Format	7
7.2	URI Path Design	7
7.3	URI Query Design	7
7.4	Request Methods	7
7.5	Response Methods	7
7.6	HTTP Headers	8
7.7	Representation Design	8

1 Purpose

This document serves as the TCM-ATIS technical approach and guide for developers and its participating clients in the design, development, and implementation of RESTful web services following a Resource Oriented Architecture (ROA) set of principles.

2 Goal

The goal of this document is to help developers understand how to design and implement RESTful web services following the TCM-ATIS approach. By following this guide, developers should gain sufficient knowledge and understanding on how to produce RESTful web services and corresponding clients quickly and consistently in varying programming languages.

3 Representational State Transfer (REST)

Representational State Transfer (REST) describes an architectural style commonly used in the HTTP context however it is not limited to the HTTP protocol. With regard to its use in the HTTP context it uses a standard vocabulary complemented by as set of constraints and guided by a hand full of principles. There are differing implementations of “RESTful” web services that are technically “RESTful” however are merely hybrid combinations of XML-RPC or misuse of the HTTP methods as defined by their intended purpose. This document intends to clarify some of this confusion regarding what it actually means to be considered a web services as “RESTful” within the HTTP context.

RESTful web services are implemented using HTTP and the principles of REST. A RESTful web service a collection of resources defined by:

1. The base URI
2. A set of supported operations defined by HTTP Methods
3. The internet media type consumed and produced by the web service
4. The API must be hypertext driven.

3.1 HTTP Request Methods

REST uses the well defined HTTP request methods; GET, PUT, POST, DELETE, HEAD, and OPTIONS to retrieve, create, and manipulate resources. Use of these HTTP methods indicates to the server how the state of a resource is to be handled.

3.1.1 GET

The HTTP GET method is used to retrieve a representation of a resource.

3.1.2 PUT

To create a new resource or modify an existing resource the HTTP PUT method is used.

NOTE:

To determine the appropriate usage between PUT and POST, the client uses PUT when it is in charge of deciding what the new URI resource should be when it is created and the client uses POST when the server is in charge of assigning a new URI to the resource being created.

3.1.3 POST

The POST method is used to create a new resource. Typically the client only needs to know the parent URI resource in which the server creates the resource under the parent. The Server response is usually HTTP Status 201 (“Created”) with the location header containing the URI of the newly created resource.

3.1.4 DELETE

HTTP DELETE is used to delete an existing resource.

3.1.5 HEAD

The HTTP HEAD method is used to retrieve a metadata representation of the resource to include the HTTP headers. It is identical to the GET method without a message body in the response or the representation itself.

3.1.6 OPTIONS

The HTTP OPTIONS method provides information about the request and response options a client is allowed to do to a resource and the resource representations that are available.

3.2 Safety and Idempotency

The HTTP GET and HTTP HEAD methods are considered “*Safe*” when used correctly. Since the request does not change any resource state. The request can be issued multiple times causing no harm.

HTTP PUT and HTTP DELETE methods are idempotent. The idea behind idempotence is that there are no side effects issuing the same request multiple times. DELETE a resource and it is gone. Issue another DELETE and it is still gone the resource state does not change. The same with PUT, resending the PUT does not change the resource state. HTTP GET and HEAD also share this property.

The HTTP POST method is neither safe nor idempotent. This method is the most problematic method for REST. Usually POST is used to create or modify resources and should be carefully implemented to prevent duplicates since it is not idempotent.

3.3 REST Principles

The REST principles are

1. Addressability
2. Uniform Interface
3. Statelessness
4. Representations
5. Hypermedia as the Engine of Application State (HATEOAS)

4 RESTful Resource-Oriented Architecture

Resource Oriented Architecture is a specific set of rules for the design and implementation of RESTful web services. Resource-oriented architecture places the importance on the resource itself and is strictly web oriented. The RESTful architecture is not restricted to resources over the HTTP protocol. By combining the method information restricted to the HTTP method, as used in RESTful architectures,

with the scoping information in the URI described by the Resource-Oriented Architecture provides a powerful technique consisting of four concepts.

4.1 The Resource

A resource is any item, object, idea, or thing that is important enough to be referenced in of it-self.

4.2 The Resource URI

A resource has to have at least one Uniform Resource Identifier (URI). The URI is the name and the address of the resource.

4.3 The Resource Representations

4.4 The Links between

5 Service Design

1. Determine the data set
2. Separate the data set into resources
 - a. For each kind of resource:
 - i. Name the resource with a URI
 - ii. Make the resource available - expose it via HTTP method(s)
 - iii. Determine/design the representation format(s) accepted by the client
 - iv. Determine/design the representation format(s) served to the client
 - v. Integrate this resource into existing resources using hypermedia links
 - vi. Consider what's supposed to happen - send appropriate response code
 - vii. Consider what might go wrong – send appropriate response code
3. URI Design
4. URI templates
5. Nouns
6. Plural Noun for collections
7. Singular Noun for a single resource
8. Query parameters
9. Examples
 - a. JAVA
 - b. Ruby
 - c. .NET

6 Client

HTTP Headers
Authentication
HTTP Methods
Verbs
Resources

Representations

HTTP Response Codes

Links between

 Pagination

 HATEOS – Hypertext Engine of

7 Rules

The following rules are listed in the *“REST API Design Rulebook by Mark Masse’ (O’Reilly), Copyright 2012 Mark Masse’, 978-1-449-31050-9.”* They are not all inclusive, however those TCM-ATIS deems important to establish as best practices.

7.1 URI Format

Rule: A forward slash (/) must be used to indicate a hierarchical relationship

Rule: A trailing slash (/) should not be included in a URI

Rule: Hyphens (-) should be used to improve the readability of a URI

Rule: Underscores (_) should not be used in a URI

Rule: Lowercase letters should be preferred in URI paths

7.2 URI Path Design

Rule: A singular noun should be used for resource names

Rule: A plural noun should be used for collections

Rule: CRUD function names should not be used in a URI

7.3 URI Query Design

Rule: The query component of a URI may be used to filter collections

Rule: The query component of a URL should be used to paginate a collection

7.4 Request Methods

Rule: GET and POST must not be used to tunnel other request methods

Rule: GET must be used to retrieve a representation of a resource

Rule: HEAD should be used to retrieve response headers

Rule: PUT must be used to both insert and update a stored resource

Rule: PUT must be used to update mutable resources

Rule: POST must be used to create a new resource in a collection

Rule: DELETE must be used to remove a resource from its parent

7.5 Response Methods

Rule: 200 (“OK”) should be used to indicate nonspecific success

Rule: 200 (“OK”) should not be used to indicate errors in the response body

Rule: 201 (“Created”) must be used to indicate successful resource creation

Rule: 202 (“Accepted”) must be used to indicate successful start of an asynchronous action

Rule: 204 (“No Content”) should be used when the response body is intentionally empty

Rule: 301 (“Moved Permanently”) should be used to relocate resources

Rule: 302 (“Found”) should not be used

Rule: 303 (“See Other”) should be used to refer the client to a different URI

- Rule: 304 (“Not Modified”) should be used to preserve bandwidth
- Rule: 307 (“Temporary Redirect”) should be used to tell clients to resubmit the request to another URI
- Rule: 400 (“Bad Request”) may be used to indicate nonspecific failure
- Rule: 401 (“Unauthorized”) must be used when there is a problem with the clients credentials
- Rule: 403 (“Forbidden”) should be used to forbid access regardless of authorization state
- Rule: 404 (“Not Found”) must be used when a client’s URI cannot be mapped to a resource
- Rule: 405 (“Method Not Allowed”) must be used when the HTTP method is not supported
- Rule: 406 (“Not Acceptable”) must be used when the requested media type cannot be served
- Rule: 409 (“Conflict”) should be used to indicate a violation of resource state
- Rule: 412 (“Precondition Failed”) should be used to support conditional operations
- Rule: 415 (“Unsupported Media Type”) must be used when the media type of a request’s payload cannot be processed
- Rule: 500 (“Internal Server Error”) should be used to indicate API malfunction

7.6 HTTP Headers

- Rule: Content-Type must be used
- Rule: Content-Length should be used
- Rule: Location must be used to specify the URI of a newly created resource

7.7 Representation Design

- Rule: A consistent form should be used to represent links
- Rule: A consistent form should be used to represent media type formats
- Rule: A consistent form should be used to represent errors