RESTful API Design Specification

URI Construction

RESTful APIs use Uniform Resource Identifiers (URIs) to address resources, ONAP projects should comply with a unified URI standard to implement the m icroservices.

- URI structure (Mandatory)
  URI structure http://[host]:[port]/api/{service name}]/v{version number}/{resource}
  The URI is comprised of a fixed base uri /api/{service name}]/v{version number} and the resource path. Only major version number is used in the URI. An example: http://127.0.0.1:8080/api/petstore/v1/pets/dogs
- CRUD function names should not be used in URIs. Instead, CRUD actions should be represented by HTTP methods. Below is the proposed methodology to implement CRUD operations in a RESTful API. (Recommended)

Resource	POST	GET	PUT	DELETE
/api/petstore/v1/pets/dogs	Create a new dog	List dogs	Replace dogs with new dogs(Bulk update)	Delete all dogs
/api/petstore/v1/pets/dogs/bailey	Error	Show dog	If exist update dog else ERROR	Delete dog

Do not use:

/api/petstore/v1/pets/getalldogs

/api/petstore/v1/pets/createdog

/api/petstore/v1/pets/deletedog

- A trailing forward slash (/) should not be included in URIs (Mandatory)

A forward slash (/) adds no semantic value and may cause confusion. RESTful API's should not expect a trailing slash and should not include them in the links that they provide to clients.

- Forward slash separator (/) must be used to indicate a hierarchical relationship (Mandatory)
- Use Hyphens (-) instead of Underscores (_) if separation of words is needed in the URI (May have conflict with SOL, TBD)

Underscores (_) can be hidden by the underline of URIs in browsers.

- Lowercase letters should be preferred in URI paths (Recommended)

When convenient, lowercase letters are preferred in URI paths since capital letters can sometimes cause problems. RFC 3986 defines URIs as case-sensitive except for the scheme and host components.

- File extensions should not be included in URIs (Recommended)

A REST API should not include artificial file extensions in URIs to indicate the format of a message's entity body. Instead, they should rely on the media type, as communicated through the Content-Type header, to determine how to process the body's content.

- A plural noun should be used for collection names (Mandatory)
  For example, the URI for a collection of dog documents uses the plural noun form of its contained resources: /api/petstore/v1/pets/dogs
- A singular noun should be used for document names (Mandatory)
  For example, the URI for a single dog document would have the singular form: /api/petstore/v1/pets/dogs/bailey

Versioning

API is a public contract between a Server and a Consumer, so it's crucial to make sure the changes are backwards compatible. We need to introduce new versions of the API while still allowing old versions to be accessible when it's necessary. So a versioning mechanism should be considered.

- Semantic Versioning http://semver.org/ (Mandatory)

- The version in the swagger file and URI (Mandatory)
  The main idea is that there are two kinds of versions:
    - Major version: The version used in the URI and denotes breaking changes to the API. Internally, a new major version implies creating a new API and the version number is used to route to the correct implementation.
    - Minor and Patch versions: These are transparent to the client and used internally for backward-compatible updates. They should be reflected in the swagger files to inform clients about a new functionality or a bug fix.

  For example, if the version of a RESTful API of an ONAP component is 1.3.2, the Info Object of swagger file should look like

```
title: Swagger Sample App
description: This is a sample server Petstore server.
termsOfService: http://swagger.io/terms/
contact:
  name: API Support
  url: http://www.swagger.io/support
  email: support@swagger.io
license:
  name: Apache 2.0
  url: http://www.apache.org/licenses/LICENSE-2.0.html
version: 1.3.2
```

  The URI should look like: http://{host}/api/v1/pets

- Version discovery (Optional)

  Need to be discussed: Should ONAP adopt a version discovery mechanism like OpenStack? :https://wiki.openstack.org/wiki/VersionDiscovery

HTTP response status codes

The server should always return the right HTTP status code to the client. (Recommended)

- Standard HTTP status codes

  200 – OK – Everything is working
  201 – OK – New resource has been created
  204 – OK – The resource was successfully deleted

  304 – Not Modified – The client can use cached data

  400 – Bad Request – The request was invalid or cannot be served. The exact error should be explained in the error payload. E.g. „The JSON is not valid“
  401 – Unauthorized – The request requires a user authentication
  403 – Forbidden – The server understood the request but is refusing it or the access is not allowed.
  404 – Not found – There is no resource behind the URI.
  422 – Unprocessable Entity – Should be used if the server cannot process the entity, e.g. if an image cannot be formatted or mandatory fields are missing in the payload.

  500 – Internal Server Error – API developers should avoid this error. If an error occurs in the global catch blog, the stracktrace should be logged and not returned as in the response.

- Use error payloads

  All exceptions should be mapped in an error payload. Here is an example how a JSON payload should look like.

```
{
  "message": "Sorry, the requested resource does not exist",
  "code": 34
}
```

  Need to be discussed: Should ONAP define a unified error response format across projects?

Security

- Token Based Authentication (Recommended)
  Ideally, microservices should be stateless so the service instances can be scaled out easily and the client requests can be routed to multiple independent service providers. A token based authentication mechanism should be used instead of session based authentication
- API-Token Authentication (Recommended)
  ONAP is supposed to be accessed by third-party apps such as OSS/BSS, in the authentication process a user is not involved. API-Token can be used in such cases.

- Centralized Authentication/Authorization (Recommended)
  The MSB API Gateway can serve as the entry point to authenticate client requests and forwards them to the backend services, which might in turn invoke other services.
  MSB doesn't do the authentication itself, instead, MSB can work with a security provider to provide centralized Authentication or ONAP with its pluggable architecture.
- Use https for external communication for security reason
  The MSB External API Gateway can translate the https request from the external systems to light weight http communication inside ONAP system. The individual projects don't have to handle the trivial details such as certification configuration and avoid the overhead of https inside ONAP system.

Multiple tenants support

TBD

API Documentation(Swagger)

- All the RESTful API must follow Swagger Specification for API documentation: http://swagger.io/specification (Recommended)
  The ONAP-Components must provide swagger files for its RESTful API definitions, there are two possible approaches.
  - The developers write the Swagger file, then use CI system to generate the stub codes for implementation.
  - The developers write the Swagger annotation in the source codeI, then use CI system to generate the Swagger file.
- The Swagger file should be placed under the base url of the service so the API definition could be discovered by clients or management tools at runtime. (Recommended)
  For example, if the base url of peststore service is /api/petstore/v1/, the URL of swagger file should be /api/petstore/v1/Swagger.json

Appendix

Swagger Specification http://swagger.io/specification

Swagger UI https://swagger.io/swagger-ui/

Swagger code generator https://swagger.io/swagger-codegen/

Maven plugin to generate Swagger documents https://github.com/WASdev/tool.swagger.docgen

MSB Centralized Authentication & Authorization solution https://wiki.onap.org/download/attachments/3246982/Capture7.PNG?version=1&modificationDate=1495079131000&api=v2

Swagger SDK https://wiki.open-o.org/display/CLIEN/Swagger+SDK+for+Open-O