POL00337645

CTO

CTO Principles, Guidelines and Patterns

API Naming

Version:0.2

Dated: 28-05-2019

CTO

POL00337645

Document Control

Abstract	Principles, Guidelines and Patterns for Microservices based Systems
Owner	CTO
Current Status	Draft
CTO Document Type	Principles

Change History

Version	Date	Author	Status	Comment
0.1	28/05/2019	Adrian Eales	Draft	Initial Draft
0.2	05/06/2019	Adrian Eales	Draft	Updated following initial comment from Baljeet Nijjhar

Reviewers & Approvers

Name & Organisation	Role	Date
Enterprise Architecture Group	A	
Michael Austin	R	
CTO Team	R	

POL00337645

Table of contents

POL00337645

Preface

Purpose of This Document

This document describes the principles and associated guidance that should be followed when naming APIs.

In particular this document focuses upon externally facing APIs.

Whilst other API standards may be used, the assumption is that external facing APIs will be RESTful in nature.

Intended Audience

This document is intended to be read by Architects, and Lead Developers.

References

#	Artefact	URL	Description
1	Post Office Architecture Principles	https://poluk.sharepoint.com/sites/Extranet/Strategy/AR/SitePages/Architecture-Principles.aspx?web=1	Post Office's top level Architecture Principles
2	Open API 3.0	https://swagger.io/blog/api-design/	Open API 3.0 Design Guideline
3	Zalando API design Guidelines	https://opensource.zalando.com/restful-api-guidelines/	Industry viewpoint
4	POL Domain Naming		POL CTO Principles, Guidelines and Patterns for Domain Naming

POL00337645

1. Scope of Document

1.1 General

The principles, standards and approach set out in the following document will replace all other previous API Naming documents at the Post Office.

Principles and standards are a fundamental part of our Enterprise Architecture. They are used to guide project investment and technology decisions and drive the Post Office forward to its desired future state.
The principles in this document are not technical standards they are overarching and can be applied to any technology based project.

The Governance Stack

- Principles: Provide direction and guide the decision making process
- Policies: Provide firm rules and frameworks to be followed
- Standards: Control specific technical implementations
- Patterns: Provide guidance on preferred methods of constructing solutions

This document is subordinate to the overarching Architectural Principles (Ref #1). In particular this document builds upon the following principles:

1.0 Architect the Big Picture
2.0 Plan for Service Life
3.0 Agility through Standardisation
4.0 Re-use, Buy, Build
6.0 Value for Money

This document sets out the Principles, Standards, Guidelines and Patterns that will be used in the creation of API based systems which have been developed by, or on behalf of, the Post Office.

This document is intended to be read by Architects, and Lead Developers involved in the Architecture and Design of API based systems.

There are varying approaches to API Naming in the IT industry. This document attempts to distill input from a number of sources, e.g. [2] and [3] amongst others, an give a standardized POL view.

1.2 Exclusions

Areas not currently covered in this document:

- Fine grained authentication and role based access/usage of APIs

POL00337645

2. Terminology

Note, where the following terms occur in this document, they are *italicized*.

Term	Description

POL00337645

3. Domain Naming for APIs

The following is partially reproduced from [4] for completeness, for full details please refer to this document for POL Domain Naming elements.

3.1 Root Domain name

The Root Domain Name to be used is:

postoffice.co.uk

3.2 Live Service Domain Names

For live service the root domain name will be prefixed with "api" for API access - ie

api.postoffice.co.uk

3.3 Developer resources

For access to developer resources, e.g. the developer portal and documentation, the root domain will be prefixed with "developer" - ie

developer.postoffice.co.uk

3.4 Non-Live Service Domain Names

For other platform types, e.g. test, prefix the Live Service Domain Name with the system type, ie

{system-type}.api.postoffice.co.uk

e.g.

test.api.postoffice.co.uk

3.5 Principle 1 – API Domain Names

3.5.1 Statement

The live service API Domain name structure shall be:
api.postoffice.co.uk

3.5.2 Rationale

APIs are a POL Enterprise resource and are associated with POL as an orgnisation.

Supports Architecture Principles:
1.0 Architect the Big Picture
3.0 Agility through Standardisation

POL00337645

3.5.3 Implications

- Other domain names/patterns shall not be used for POL issued APIs.

POL00337645

4. API Naming

4.1 Basics

To create an API URI, the Domain Name is suffixed with the API function, e.g.

api.postoffice.co.uk/{api-function}

Where the {api-function) describes either the object being operated upon, or the operation that the API performs.

In a RESTful context, preference is given to (api-function} relating to an object, in which case {api-function} should be a plural noun which describes the object being handled, e.g.

api.postoffice.co.uk/customer-records
api.postoffice.co.uk/postal-orders

Where the API represents a service, {api-function} should describe the operation, using appropriate verbs, e.g.

api.postoffice.co.uk/authenticate-session

In all cases {api-function} should be:
1. Unique across POL
2. Be self-descriptive, such that the name adequately describes the object or service being represented and operated upon.

4.2 Principle 2 – API Naming

4.2.1 Statement

The form of an API Name will be:

{api-domain}/{api-function}

Notes:
{api-domain} is as described in § 3 and Principle 1

{api-function} must be:

1. Uniquely named across POL
2. Self-descriptive, such that the name adequately describes the object or service being represented and operated upon.

4.2.2 Rationale

APIs are Enterprise resources and need to be considered as such.

Supports Architecture Principles:
1.0 Architect the Big Picture
2.0 Plan for Service Life

POL00337645

3.0 Agility through Standardisation
4.0 Re-use, Buy, Build
6.0 Value for Money

4.2.3 Implications

- POL system shall reuse existing, available, APIs before considering creating new.
- Creation of APIs offering competing functionality shall be avoided.
- API users should be able to understand the basic function of an API from the api-function naming.

POL00337645

5. API Versioning

5.1 Basics

There are conflicting views in the industry as to how versioning of APIs should be represented. The 4 main approaches are:

1. API Versioning in URI
2. Custom Request Header
3. Accept Header
4. Backwards compatibility

The approach to be taken within POL is approach (4). No explicit API versioning will be used. This implies that future revisions of an API should be backwards compatible with previous usage, and changes should be additive only.

Where changes are such that backwards compatibility is no possible, then a new API should be created. Note – this route should be used as an exceptional approach only.

5.2 Principle 3 – API Versioning

5.2.1 Statement

APIs will not be versioned as part of their URI, or the *{api-function}* naming.

5.2.2 Rationale

Explicit versioning creates additional load upon clients/development teams.

Supports Architecture Principles:
1.0 Architect the Big Picture
2.0 Plan for Service Life
3.0 Agility through Standardisation
4.0 Re-use, Buy, Build
6.0 Value for Money

5.2.3 Implications

- Revisions to APIs shall be backwards compatible, changes shall be additive.
- Where backwards compatibility is not feasible, exceptionally a new API shall be created, however the new API must be functionally different from the original API.

POL00337645

6. Application Naming

6.1 Basics

The use of a prefix, postfix or suffix naming of an API is not considered a strategic pattern e.g.

{application-name}.api.postoffice.co.uk/{api-function}
api.postoffice.co.uk/{application-name}/{api-function}
However it is recognized that, until a single logical integration layer is in place, routing of APIs to individual cloud platforms will be required. Therefore, as a tactical mechanism, the following is permitted until a POL Integration Layer is in place.

{application-name}.api.postoffice.co.uk/{api-function}

e.g.

hih.api.postoffice.co.uk/ProductCatalogue

Note, the expectation is that the DNS entry for the above will initially point to the HIH platform API Gateway, but will subsequently be pointed to POL Integration Layer API gateway (for legacy purposes only) – thus preventing the need for client systems to require changes. At this point new clients will be required to use a URI/URL without any application naming, ie:

api.postoffice.co.uk/ProductCatalogue

6.2 Principle 4 – Application Naming

6.2.1 Statement

APIs should not use Application Naming as part of their Domain or URI.

Where this is required, tactically, for DNS routing purposes, this shall follow the pattern of:

{application-name}.api.postoffice.co.uk/{api-function}

6.2.2 Rationale

POL are moving to a single enterprise integration layer. Underlying applications, or platforms should be hidden from the client applications.

Supports Architecture Principles:
1.0 Architect the Big Picture
2.0 Plan for Service Life
3.0 Agility through Standardisation
4.0 Re-use, Buy, Build
6.0 Value for Money

6.2.3 Implications

- Application naming shall only be used as a tactical measure to allow routing to API groups hosted in separate cloud instances

POL00337645

- Strategically, once a single API Gateway is in place, all URIs will be terminated upon that gateway.

POL00337645

7. Experiential APIs

7.1 Basics

For experiential responses from an API (e.g. to receive a tailored response based upon a particular channel or device) then the following approach are to be used:

For instances where channel specific content is to be returned, this should be implemented by the use of a query parameter, i.e.

api.postoffice.co.uk/{api-function}?{query-parameter}

e.g.

api.postoffice.co.uk/UserLoginPage?DeviceType=ssk

Where specific authentication or routing at the API Gateway is required, then the use of a custom http Accept header can be utilized, as an alternative approach (note, this is not the preferred approach).

7.2 Principle 5 – Experiential API Naming

7.2.1 Statement

APIs should not use Experiential Naming as part of their Domain or URI.

Where this is required, this shall use query parameters or a custom http header, as per previous section.

7.2.2 Rationale

Where possible APIs should be channel agnostic.
Creation of separate logic for different channels is contrary to an omni-channel operation.

Supports Architecture Principles:
1.0 Architect the Big Picture
2.0 Plan for Service Life
3.0 Agility through Standardisation
4.0 Re-use, Buy, Build
6.0 Value for Money

7.2.3 Implications

- Application naming shall only be used as a tactical measure to allow routing to API groups hosted in separate cloud instances

POL00337645

8. API Keys

8.1 Basics

API Keys are used as coarse grained access management, and to determine the source of a request.

Fine grained access management should be achieved using a scheme such as OAUTH.

Where API Key are required, these shall be implemented through the use of a custom http header:

POL_API_KEY = {issued key}