REST API Guidelines

Table of Contents

- Overview
- Taxonomy
    - Conceptual View
    - API Type Details
    - API Types Detailed
- Guidelines
    - Documentation
    - Naming Conventions
        - Overview
        - Baseline naming standard - Microsoft
        - String Convention
        - Version numbering
        - Versioning strategy
        - Collections
        - Working with multiple resources
        - Designing for re-use
        - HATEOAS (Hypermedia As The Engine Of Application State) alignment
    - Responses
        - Response Codes
        - Error Handling
    - MetaData
        - Traceability
        - Header Information
- Todo

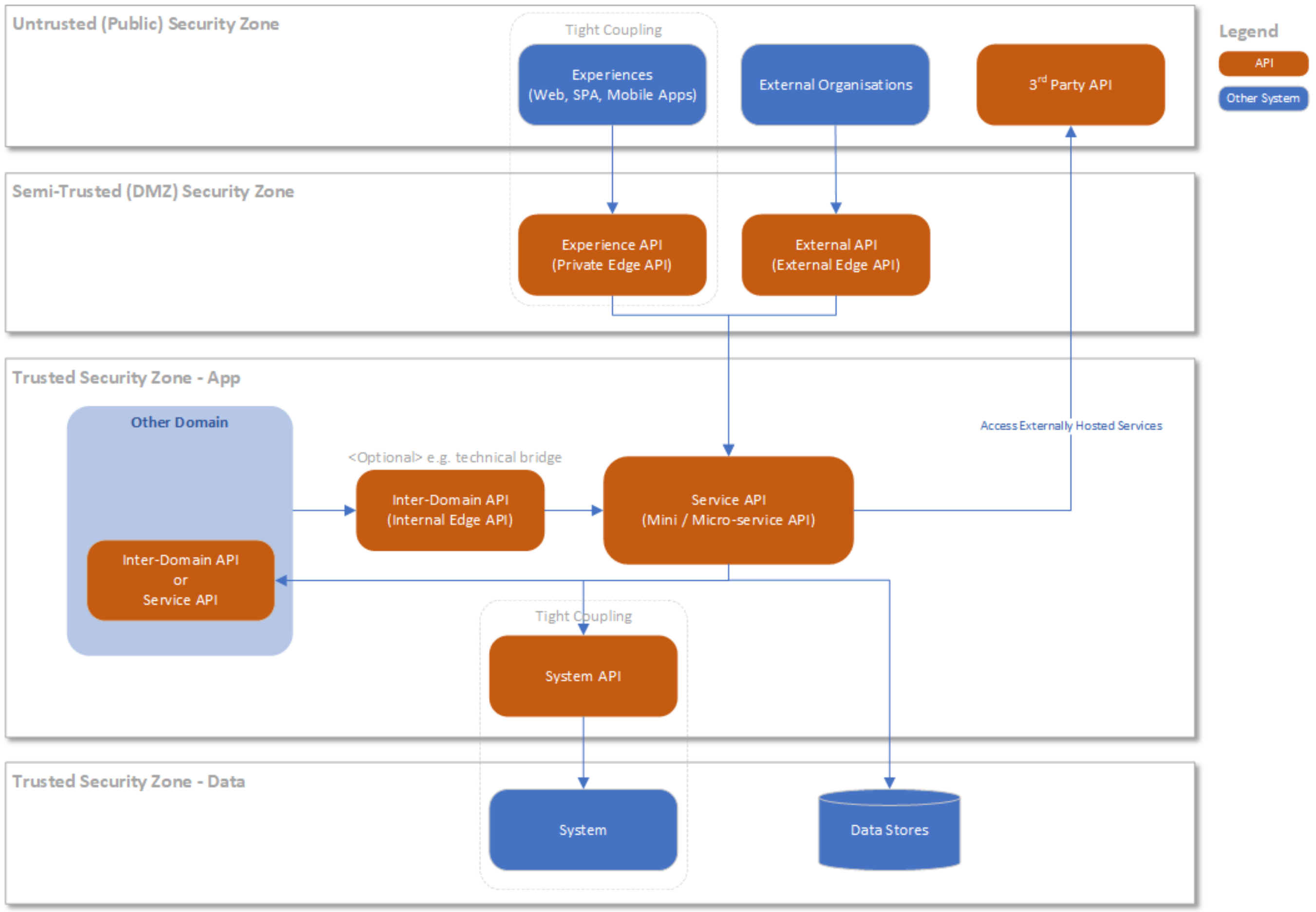
Overview

This page looks to establish the standards and guidelines for the ASX when building APIs at the ASX. This page will continue to be updated as new issues are encountered, or approaches need to be re-considered.

Taxonomy

Conceptual View

The following describes the different types of APIs at a conceptual level and the layers they sit within.



Untrusted (Public) Security Zone
Tight Coupling
Experiences (Web, SPA, Mobile Apps)
External Organisations
3rd Party API
Legend
API
Other System
Semi-Trusted (DMZ) Security Zone
Experience API (Private Edge API)
External API (External Edge API)
Trusted Security Zone - App
Other Domain
Access Externally Hosted Services
<Optional> e.g. technical bridge
Inter-Domain API (Internal Edge API)
Service API (Mini / Micro-service API)
Inter-Domain API or Service API
Tight Coupling
System API
Trusted Security Zone - Data
System
Data Stores

API Type Details

The API types are described below. Each API implements their own security controls according to their position in the hierarchy and security zone, following a **defense-in-depth** model. Each layer offers limited trust of the application or layer above it to eliminate opportunities of data breaches or leakage. For example, the Services API doesn't trust that the edge API has verified the end-user has permission to the data requested, and will perform its own authorisation check before executing the request.

Column Legend:

- API Type - the name of the API type being described in the row
- Description - Long description of the API
- API Gateway - What type of Gateway would normally be employed to implement / manage some of the security controls
- Re-usable - Whether the API would be re-used across consumers
- Security Zone - The physical layer the API would normally be sat in - defined as Untrusted (public), Semi-Trusted (DMZ), Trusted - App, Trusted - Data.
- Security Controls - What type of security controls would be met by the API implementation. The controls are typically:
    - System Authentication - Verifies system credentials (certs, tokens, secrets, keys etc)
    - System Authorisation - Verifies the system credentials allow access to the API
    - End-user Authorisation - Verifies the end-user credentials (passed in the header) allow the user to use the API for that purpose
    - (always required) Encryption - Ensures encryption of data via TLS, mutual SSL, PKI etc as appropriate

API Type	Description	API Gateway	Re-usable	Security Zone	Security Controls
3rd Party API	3rd Party APIs that the ASX consumes, such as SaaS products like Salesforce	• **Proxy Server** for outbound routing	Yes	Untrusted (Public)	Defined by the owner of the 3rd Party API
Edge APIs	APIs at the edge of an ASX domain and service inbound requests				

▪ Experience API (Private)	APIs that are provide services to ASX owned user experiences such as a mobile or web site apps / components. These are tightly coupled to a specific app or component and not re-used elsewhere (for agility of change reasons).	• **Lightweight Gateway** (ForgeRock IG)	No, per UI app	Semi Trusted (DMZ)	REQUIRED ENCRYPTION REQUIRED SYSTEM AUTHENTICATION - e.g. Salesforce system has presented a certificate that is recognised and is active REQUIRED SYSTEM AUTHORISATION - e.g. Salesforce system may access the "Update Customer" external API OPTIONAL END USER AUTHORISATION - not typically performed at this layer, usually the responsibility of the "actioning" system, for example the service or Service API
▪ External API (External)	APIs that provide services to external parties apps, systems or platforms. Where the consumer is trusted the API gateway used would be a basic gateway such as Forgerock IG. Where the consumer is semi-trusted (customers, partners) or untrusted (public) a full suite API gateway such as an Apigee or Mashery is employed, allowing API management functions such as authentication services, throttling and monetisation etc.	• **Lightweight Gateway** (ForgeRock IG) for semi-trusted parties such as Salesforce Lightning DXP • **Full Gateway** (Tibco Mashery) for 3rd parties	Yes	Semi Trusted (DMZ)	
▪ Inter-Domain API (Internal)	Internal-use only APIs that allow the domain to expose services to other domains or systems within the ASX to consume. For example the digital domain might expose inter-domain APIs allowing notifications to be sent to end users. This API is optional. The (value-add) reason for creating one is often as a technical bridge such as exposing a SOAP endpoint rather than REST, or providing a different security implementation to the underlying Service API.	• Not required, however could be optionally employed environment for consistency	Yes	Trusted - App	REQUIRED ENCRYPTION REQUIRED SYSTEM AUTHENTICATION - e.g. Trade24 system has presented a valid certificate REQUIRED SYSTEM AUTHORISATION - e.g. The Trade24 is authorised to access the "Send Notification" inter-domain API NOT REQUIRED END USER AUTHORISATION - not typically performed in this context
Service APIs	Internal-use only APIs providing services in a re-usable form, regardless of the consumer which can be an end-user app or an internal system. Edge APIs often consume one or more of these services to fulfill their function. These APIs can be designed to accept end user information for API authorisation checks, or in the case of ASX systems consuming the API, accept system credentials (no authorisation)	• Not required	Yes	Trusted - App	REQUIRED ENCRYPTION REQUIRED SYSTEM AUTHENTICATION - e.g. The user mgmt experience API has presented valid system credentials REQUIRED SYSTEM AUTHORISATION - e.g. The user mgmt experience API is authorised to access this "Create User" Service API OPTIONAL END USER AUTHORISATION - This control requirement is determined by the nature of the API and calling system: 1. (required) the user "bob jane" wants to create user "rodney jane" for organisation "bob jane t-mart" 2. (not required) The trade24 system wants to call the notification API to send a "trade complete" notification to asx user "ben furryface"
▪ Mini-service API	Internal-use only, **coarse** grained, re-useable APIs that often performs a higher-level function to those provided by micro-services APIs. Also referred to as process / orchestration APIs.	• Not required	Yes	Trusted - App	
▪ Micro-service API	Internal-use only, **fine** grained, re-useable APIs that align with a specific business service.	• Not required	Yes	Trusted - App	

System APIs	An API provided by a system or technology such as a COTs product. These often expose the internal data as a system level API rather than as a business service, and so are usually wrapped in a service API to provide a level of abstraction.	• Not required	Yes	Trusted - App	REQUIRED ENCRYPTION REQUIRED SYSTEM AUTHENTICATION - e.g. The user mgmt service API has presented valid account credentials OPTIONAL SYSTEM AUTHORISATION - e.g. The user mgmt service API is authorised to access the Forgerock Identity API NOT REQUIRED END USER AUTHORISATION - End-user authorisation - is not typically performed in this context

API Types Detailed

A more detailed view with the API gateways included, and salesforce as an example

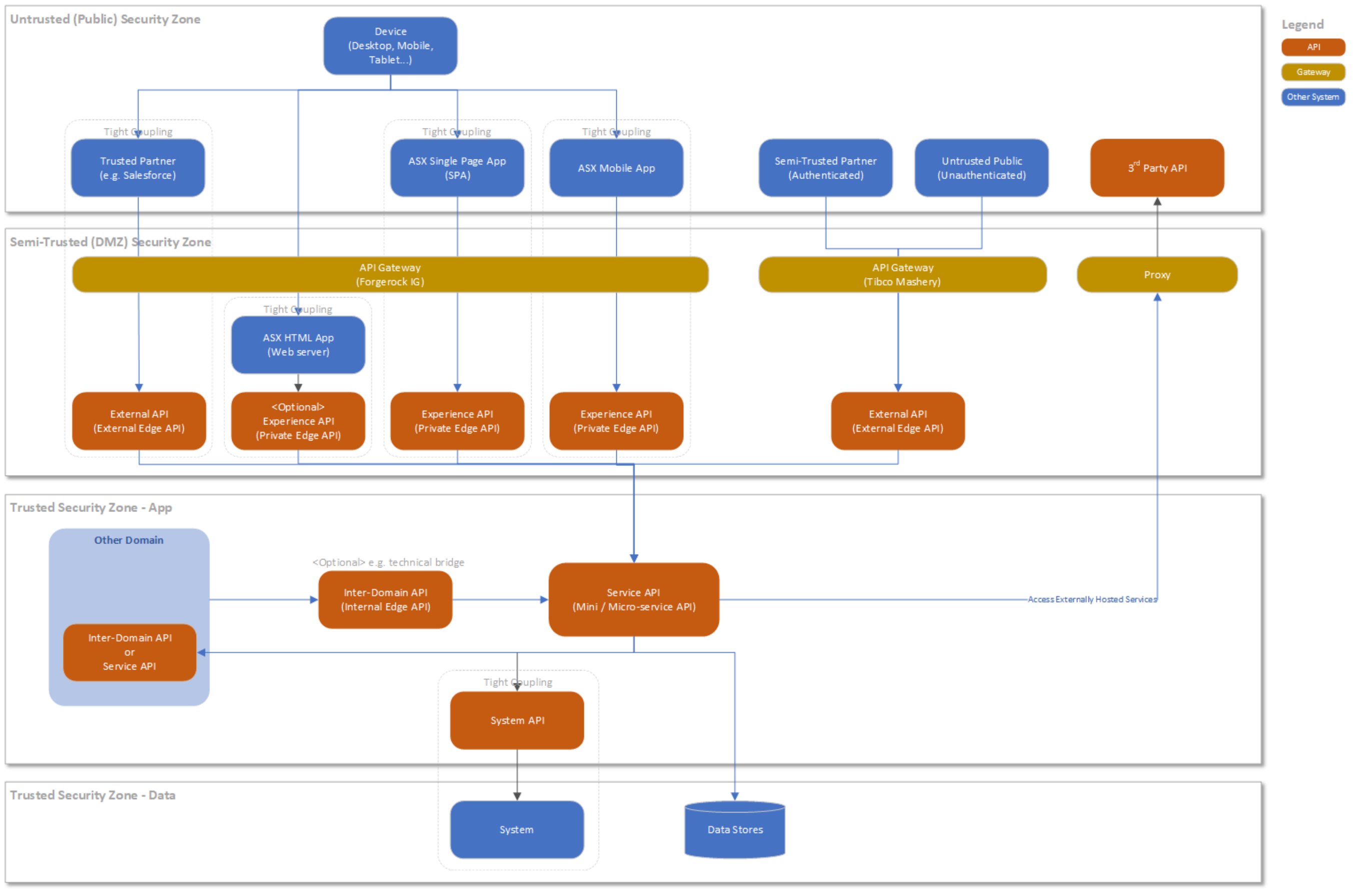

Untrusted (Public) Security Zone
Device (Desktop, Mobile, Tablet...)
Tight Coupling
Trusted Partner (e.g. Salesforce)
ASX Single Page App (SPA)
ASX Mobile App
Semi-Trusted Partner (Authenticated)
Untrusted Public (Unauthenticated)
3rd Party API
Semi-Trusted (DMZ) Security Zone
API Gateway (Forgerock IG)
API Gateway (Tibco Mashery)
Proxy
ASX HTML App (Web server)
External API (External Edge API)
<Optional> Experience API (Private Edge API)
Experience API (Private Edge API)
Experience API (Private Edge API)
External API (External Edge API)
Trusted Security Zone - App
Other Domain
Inter-Domain API or Service API
<Optional> e.g. technical bridge
Inter-Domain API (Internal Edge API)
Service API (Mini / Micro-service API)
Access Externally Hosted Services
System API
Trusted Security Zone - Data
System
Data Stores
Legend
API
Gateway
Other System

Diagram Source
API Taxonomy.vsdx

Guidelines

Documentation

For documenting APIs please use the following:

- OpenAPI (formerly Swagger), latest version
- It should be employed in the design phase for contract definition and agreement between the concerned parties (producer and consumers)
- It is also used for external developer portal documentation for 3rd parties integrating with digital

Naming Conventions

Overview

An ASX naming standard will drive consistency of approach.

Previous guidelines worth reviewing are here:

1. 9.1 - REST API Naming Guidlines

Baseline naming standard - Microsoft

We are using the Microsoft API Guidelines as the baseline standard for the ASX.

Other resources considered were:

- Google was also considered which is more accessible but less comprehensive
- For a repository of API guidelines see here

The below items are the ASX approach where either the Microsoft standard required clarification, offered choices (may, could etc), or the agreed ASX preference varies to the standard.

String Convention

- kebab-case - all lower case, with a dash (-) separating words. kebab-case will be used for URL.
- for a discussion on string convention options see here
- Low camel case will be used for object and field name in the payload

Version numbering

- external standard - https://github.com/Microsoft/api-guidelines/blob/master/Guidelines.md#12-versioning
- use the major version v1 convention within the URL for versions, with version as the first value
  - 👍 GET /**v1**/shares/{share-code}
  - ⛔ GET /**1**/shares/{share-code}
  - ⛔ GET /**v1.0**/shares/{share-code}
  - ⛔ GET /**V1**/shares/{share-code}
  - ⛔ GET /shares/{share-code}/**v1**
  - ⛔ GET /shares/{share-code}**?v=1**

Versioning strategy

- it is more common to extend an api in a non-breaking way, for example adding new fields or collections into an api
- api consumers should be built with the expectation that an api will be extended, with new information added over time
  changes requiring a new version are rare:
  - 👍 a true breaking change, for example:
    - an existing field is removed
    - a new non-optional input parameter that cannot have a default (must be explicitly passed)
    - a new error response (e.g. http 4xx, 5xx) that previously didn't exist and impacts requires consumer applications to change (this should be rare)
  - 👍 over time it has become useable or readable and for the sake of clarity a rewrite is desirable
  - ⛔ aesthetic reasons or driving for perfection in every interation (batch these up for the next rewrite)
  - ⛔ renaming a field to something a bit better (batch these up for the next rewrite)

Collections

- external standard - https://github.com/Microsoft/api-guidelines/blob/master/Guidelines.md#9-collections
- Collections are plural (section 9.3 in MS standard), for example:
  - 👍 GET /v1/share**s**/{share-code}
  - ⛔ GET /v1/share/{share-code}
  - 👍 GET /v1/compan**ies**/{company-id}/logo
  - ⛔ GET /v1/company/{company-id}/logo

Working with multiple resources

- Use **views** convention (similar to SQL) to return multiple resource types through a single api call, for example
  - ⛔ GET /v1/**company-and-roles**/{company-id}
  - 👍 GET /v1/companies/{company-id} – Returns a company resource and nothing else
  - 👍**POST** /v1/companies/{company-id} – Inserts a new company into the companies resource collection
  - 👍 GET /v1/companies/{company-id}/**roles** – Returns only the roles associated with a company, not the main company resource
  - 👍 GET /v1/companies/{company-id}/**roles-view** – Returns both the company resource AND the roles associated with a company
  - 👍 GET /v1/companies/{company-id}/**full-view** – Returns a full graph of all the information held on a company, e.g. roles, addresses, individuals, etc
  - ⛔ **POST** /v1/companies/{company-id}/role-view – Like SQL, you shouldn't write through views
  - 👍 PUT /v1/companies/{company-id}/roles/{role-id} – Inserts a new role against a particular company

Designing for re-use

- Designing for re-use is particularly necessary for micro-service or back-end APIs where re-use is a desired outcome
- Avoid using project or vendor names or similar elements that constrain re-use of an API

  - ⛔ GET /v1/**MarkIt**/companies/{company-id}
  - ⛔ GET /v1/**BETA**/companies/{company-id}
  - 👍 GET /v1/companies/{company-id}

HATEOAS (Hypermedia As The Engine Of Application State) alignment

- We will not be adopting the HATEOAS component of REST
- The rationale for not adopting is the additional effort delivers limited return due to the low adoption rates in client applications (see here)

Responses

Response Codes

The following are common HTTP Response codes used in particular for error handling (for a full list of all error codes see this page):

Category	HTTP Status code	Description
2xx: Success	200 OK	Response to a successful GET, PUT, PATCH or DELETE. Can also be used for a POST that doesn't result in a creation.
	201 Created	Response to a POST that results in a creation. Should be combined with a Location header pointing to the location of the new resource
	202 Accepted	The request has been accepted for processing. There's nothing said about the actual processing, and the result of that, which might happen on a separate server, or batched.
	204 No Content	Response to a successful request that won't be returning a body (like a DELETE request)
4xx: Client Error	400 Bad Request	The request is malformed, such as if the body does not parse. Error message should indicate which one and why. 400 is the generic client-side error status, used when no other 4xx error code is appropriate.
	401 Unauthorized	When no or invalid authentication details are provided. Also useful to trigger an auth popup if the API is used from a browser
	403 Forbidden	When authentication succeeded but authenticated user doesn't have access to the resource
	404 Not Found	When a non-existent resource is requested
	405 Method Not Allowed	When an HTTP method is being requested that isn't allowed for the authenticated user. For instance, a read-only resource could support only GET and HEAD, while a controller resource might allow GET and POST, but not PUT or DELETE. A 405 response must include the Allow header, which lists the HTTP methods that the resource supports. For example: `Allow: GET, POST`
	410 Gone	Indicates that the resource at this end point is no longer available. Useful as a blanket response for old API versions
	415 Unsupported Media Type	If incorrect content type was provided as part of the request

	422 Unprocessable Entity	Used for validation errors
	429 Too Many Requests	When a request is rejected due to rate limiting
5xx: Server Error	500 Internal Server Error	Servers are not working as expected. The request is probably valid but needs to be requested again later.

Error Handling

Use HTTP status codes! but don't overuse them. Use HTTP status codes and try to map them cleanly to relevant standard-based codes. In addition to HTTP response code for the errors, it is good to send meaningful error message. An error response must give context and actionable information. The below is the sample error message we can use:

```
{
  "code": "string",
  "errors": [
    {
      "location": "string",
      "message": "string",
      "reason": "string"
    }
  ],
  "message": "string",
  "status": "string",
  "timestamp": "2019-09-17T05:05:54.637Z"
}
```

- Header vs Body
  - It is usually a good idea to use the headers for metadata and the body for the data that is used by the business logic.
  - HTTP Header example
    - Data access by router/firewall
    - Security credentials
  - Body - If the field is used in business logic, then it should go in body.

MetaData

Traceability

- Using a correlation ID to trace single requests through multiple hops of a process can be useful for troubleshooting and security.

Header Information

- What header information should be included as part of each API request e.g. authorisation info?

Todo

- Alternate techniques to authorisation - IAPs
- Design principles - When to build in edge api vs when to use micro-service
- What is an API vs a resource, relationship
- API standards - Common header definitions, authentication information, correlation identification, environment information etc.
- Security standards details - how do we secure internal APIs as opposed to external. What protocol should we use to secure the APIs? CORS etc? Service accounts, tokens, etc. What is the preference. Work with security.
- Add logging and monitoring