**API Layer:**

This layer’s role within a Software Architecture, is to provide a set of capabilities to generate a layer of abstraction, security, deployment and publication for Web APIs. Also, this layer delivers the elements to engage third parties with APIs providers, via an API Portal, Pages, Communities, etc.

Depending on the purpose degree of the API, OMESA divides the API layer into two sub-layers:

* **Single Purpose - API** – APIs’ capabilities in this category are those whose functional spectrum are targeted to cover a specific use case. For example: providing a subset of employee data for presentation in a mobile application. This scenario the functionality of the API would use business (multi purpose) APIs to gather the relevant data and then trim the information specifically for how that mobile application needs the data. In adopting this model we can address efficiency and user experience considerations.
* **Multi Purpose - API** – These types of APIs’ capabilities are those whose functional spectrum are targeted to cover different usages. The nature of these capabilities is to provide multiple usages, for example: consolidated views of key entities within the organization like Customer or product. Such an API can therefore be used across multiple systems, for example a Product API may support retail channels, supply chain or distribution processes, These kinds of APIs are more aligned with the goals of SOA.

Within OMESA, Core Capabilities are meant to simplify design, as well as to justify and explain the presence of any layer in a software architecture. Once the desired capabilities have been identified, they can be usually realized by applying or extending one or more design patterns.

**Core Capability Definitions:**

**AuthN/AuthZ –** authentication and authorization as mechanisms to control access, where authentication is the lighter check confirming a user is who they claim they are (username and password check against some for of identity management either via federation management or directly). Authorization considers the credentials further not only determining whether the credentials provided are valid, but also has the user been attributed with a suitable role for the operation required.

**API Key Validation –** when integrating applications as an alternative or supplement to AuthN/AuthZ is the use of an allocated key or token which is represented as typically a pseudorandom value in a string format. Prior to the application calling an API a developer will request through a developer portal a key for their application. During the API call then the key is verified. This has the added advantage the API consumer can also revoke access to the API in addition to the provider.

**Tailored Contracts** – API contracts should be a perfect fit to what they represent. Contracts cannot be misleading, cannot contain less or more to what they are intended to be. Consumers are just aware about the contract, if the contact is not tailored for them, then the probability of not using it may increase. That is why the contract needs to be perfect fit to what they intend to be. An example around this are tailored contracts for different channels: mobile, web, etc.

**Threat Protection –** Provides the elements to identify and protect the APIs for damage actions caused from the consumer usage or for non-consumers that want to compromise them.

**API Monitoring and Analytics -** Focuses on usage statistics and health for an API. Not only for requests and responses, but to correlate the consumers with their APIs. Correlate applications with APIs. Deliver health status of the APIs. Identity abnormal activity and report it. It can be both historic data or real-time as we can see in the next capability.

Enable a real time analysis on the usage of the APIs. This capability will support to have in real time, information of the usage, engagement, errors, faults, threats of the APIs.

**Consumer SDKs –** Allow the service consumers to have an SDK to use and engage with the APIs. This will give to the consumers all the elements to develop their own applications using the exposed APIs through this SDK. The SDK include: blueprints, samples and usage scenarios.

**API Gateways -** Allows APIs to be deployed to the outside world. This is a piece of software that will allow the APIs to be deployed either they live on premise or in the cloud. This is a composed capability that use/offer other capabilities such as Threat Protection, Authentication, Authorization, Security ,etc.

**API Portal & API Pages –** These capabilities are related to the exposure of the web channels to enable the engagement with third parties. This is in the form of a Web Portal where third parties can subscribe, search, register, read the documentation of the APIs

**Community Management -** .the development and publication of APIs will often involve multiple individuals (covering consumers and providers) with different access rights to different APIs. This therefore requires the means to manage such groups of communities.

**API Discovery –** Enable the third parties to discover the APIs. This can be done through the API Portal. It is a very important capability that will allow the APIs to have the metadata, tags, documentation in order for third parties to discover them and interpret them in the right way. This is very relevant and is related with the level of abstraction that the APIs need to have

**HTTP Routing –** whilst a unified set of APIs needs to be offered and providing a single managed point of entry into an environment the implementation of APIs maybe spread across a wider network, therefore invocations of APIs need to routed into the correct part of the network for execution or even potentially further routing when networks have further subdivisions to isolate different data groups and classifications along with the differentiation of internal and external services.

**API Resiliency –** APIs implementation needs to be fault tolerant. Consumers rely on the usage of the APIs, therefore the API should behave to fulfill the consumer expectations. Those expectations are high and include that the APIs should be available as much time as they can.

We refer to the API implementation to include fault tolerance, using some of the Service Layer characteristics, such as: Timeouts, Bulkheads, Circuit breakers, Redundancy. Applying those technics/characteristics will increase the fault tolerance of our APIs.

**API Load Balancer –** an API Load Balancer is a specialized or enhanced gateway that has the means to not only control access and route traffic to the realization of an API but also has the means to apply load balancing algorithms when multiple instances an API realization exist. Typically, this requires the API Load balancer to contain a registry of implementations of an API. The registry is then used to record the arrival and disappearance of the implementations of APIs. In doing this we have the means to establish fault tolerant behavior and other factors needed when using scaling that is both elastic and dynamic in nature. <add link to OTN article once published>

Traffic Management – an API Gateway needs to manage the traffic, either is to limit the API calls for an specific consumer, or limiting the API calls during a window of time. Throttling the calls is also a key capability to enable transaction and traffic control of the APIs.