

Quality Patterns

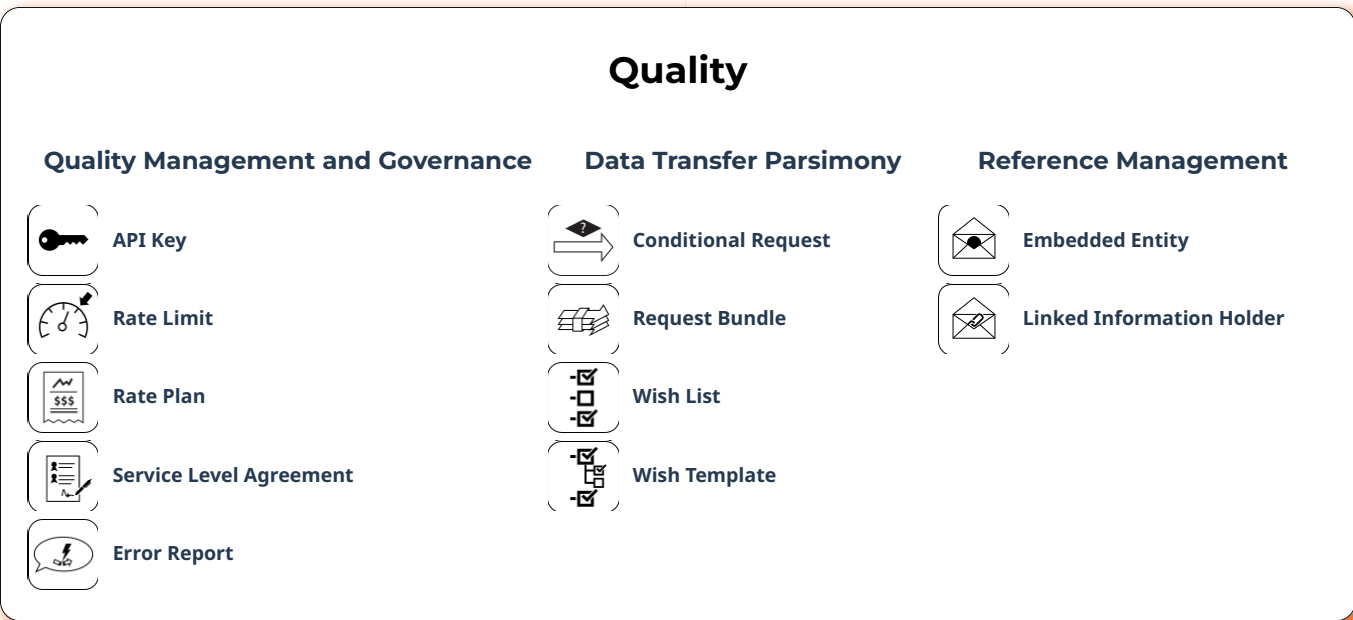
An API provider has to compromise between providing a high-quality service in a cost-effective way. The patterns in this category are all addressing or contributing to the following overarching design issue:

How to achieve a certain level of quality of the offered API, while at the same time utilising the available resources in a cost-effective way?

← Structure Patterns
Evolution Patterns →

Category Overview

The Quality Patterns category comprises the following patterns:





Quality of an API has many dimensions, starting with the functionality described in the API contract, but also including many other qualities such as reliability, performance, security, scalability. Some of those technical qualities are often referred to as *Quality of Service* (QoS) properties. They might be conflicting and almost always need to be balanced with economic forces such as costs, time to market. There are many quality measures related to QoS of services or distributed systems in general, but only few of them directly concern the APIs.


Most decisions in this category have to be made for combinations of API clients and the APIs those clients can access. Many decisions can be made for large groups of those combinations, such as all clients with freemium access to an API or all clients accessing a specific API. One decision (*Avoid Unnecessary Data Transfer*) needs to be made at the level of operations in the API.


The following patterns of the quality category have been published so far:


Quality Management and Governance


	PATTERN: <u>API KEY</u>
Problem	How can an API endpoint identify and authenticate the client making a request?
Solution	As an API provider, assign each client a unique token – the API Key – that the client can present to the API endpoint for

	PATTERN: <u>API KEY</u>
	identification purposes.


	PATTERN: <u>ERROR REPORT</u>
Problem	How can the provider inform the client about communication and processing faults without being restricted by technology specifics such as protocol headers and predefined status codes?
Solution	Reply with an error code (either an integer or a string constant) that indicates the fault in a machine-readable way. In addition, add a textual description of the error for the API client.


	PATTERN: <u>RATE LIMIT</u>
Problem	How can the API provider prevent API clients from excessive API usage?
Solution	Introduce and enforce a Rate Limit to safeguard against API clients that overuse the API.


	PATTERN: <u>RATE PLAN</u>
Problem	How can the API provider meter API service consumption and charge for it?
Solution	Assign a Rate Plan for the API usage that is used to bill API clients, advertisers, or other stakeholders accordingly. Define and monitor metrics for measuring API usage, such as monitoring API usage on a per-operation level.


	PATTERN: <u>SERVICE LEVEL AGREEMENT</u>
Problem	How can an API client learn about the specific quality-of-service characteristics of an API and its operations? How can these characteristics and the consequences of not meeting them be defined in a measurable way?
Solution	As an API provider, define a structured Service Level Agreement (SLA) that is written in an assertive, unambiguous way: the SLA must identify the API operation that it pertains to and must contain at least one measurable Service Level Objective (SLO). An SLO must specify a measurable aspect of the API, such as performance, scalability, or availability.

Data Transfer Parsimony


	PATTERN: <u>CONDITIONAL REQUEST</u>
Problem	How can unnecessary server-side processing and bandwidth usage be avoided when frequently invoking API operations that return rarely changing data?
Solution	Support conditional requests by allowing client to add one or more metadata elements to the request message representations (or protocol headers) that requests are processed by the provider only if a condition is met. If the condition is not met, the provider does not reply with a full response, but returns a special status code instead.


	PATTERN: <u>REQUEST BUNDLE</u>
Problem	How can the number of requests and responses be reduced to save bandwidth and network capacity and to avoid unnecessary message processing?
Solution	Define a Request Bundle as a container that assembles multiple individual requests in a single request message. Add metadata such as number and identifiers of individual requests.

	PATTERN: <u>WISH LIST</u>
Problem	How can an API client inform the API provider at runtime about the data it is interested in?
Solution	As an API client, provide a Wish List in the request that enumerates all desired data elements of the requested resource. As the API provider, deliver only those data elements in the response message that are enumerated in the Wish List.

	PATTERN: <u>WISH TEMPLATE</u>
Problem	How can an API client inform the API provider at runtime about the data it is interested in? How can such wishes be expressed for complex parameters?
Solution	Define one or more additional parameters in the request message that have the same structure as the desired parameter(s) in the response message.

Reference Management

	PATTERN: <u>EMBEDDED ENTITY</u>
Problem	How can you avoid sending multiple messages when receivers require insights from multiple related information elements?
Solution	For any relationship that the client has to follow, embed an Entity Element in the message that contains the data of the target entity (instead of linking to the target entity). For instance, if a purchase order has a relation to product master data, let the purchase order message hold a copy of all relevant information stored in the product master data.

	PATTERN: <u>LINKED INFORMATION HOLDER</u>
Problem	When exposing structured, possibly deeply nested information elements in an API, how can you avoid sending large messages containing lots of data that is not always useful for the message receiver in its entirety?
Solution	Add a Link Element to the message that references an API endpoint. Introduce an API endpoint that represents the linked entity, for instance, an Information Holder Resource. This endpoint represents the referenced information element, for instance an entity from the domain model that is exposed by the API.

Earlier versions of five of these patterns are featured in our EuroPLoP 2018 paper [Interface Quality Patterns: Communicating and Improving the Quality of Microservices APIs](#). A free PDF version of this paper can be downloaded [here](#).

Microservice API Patterns

Copyright © 2019-2021.

All rights reserved. See [terms and conditions of use](#).

