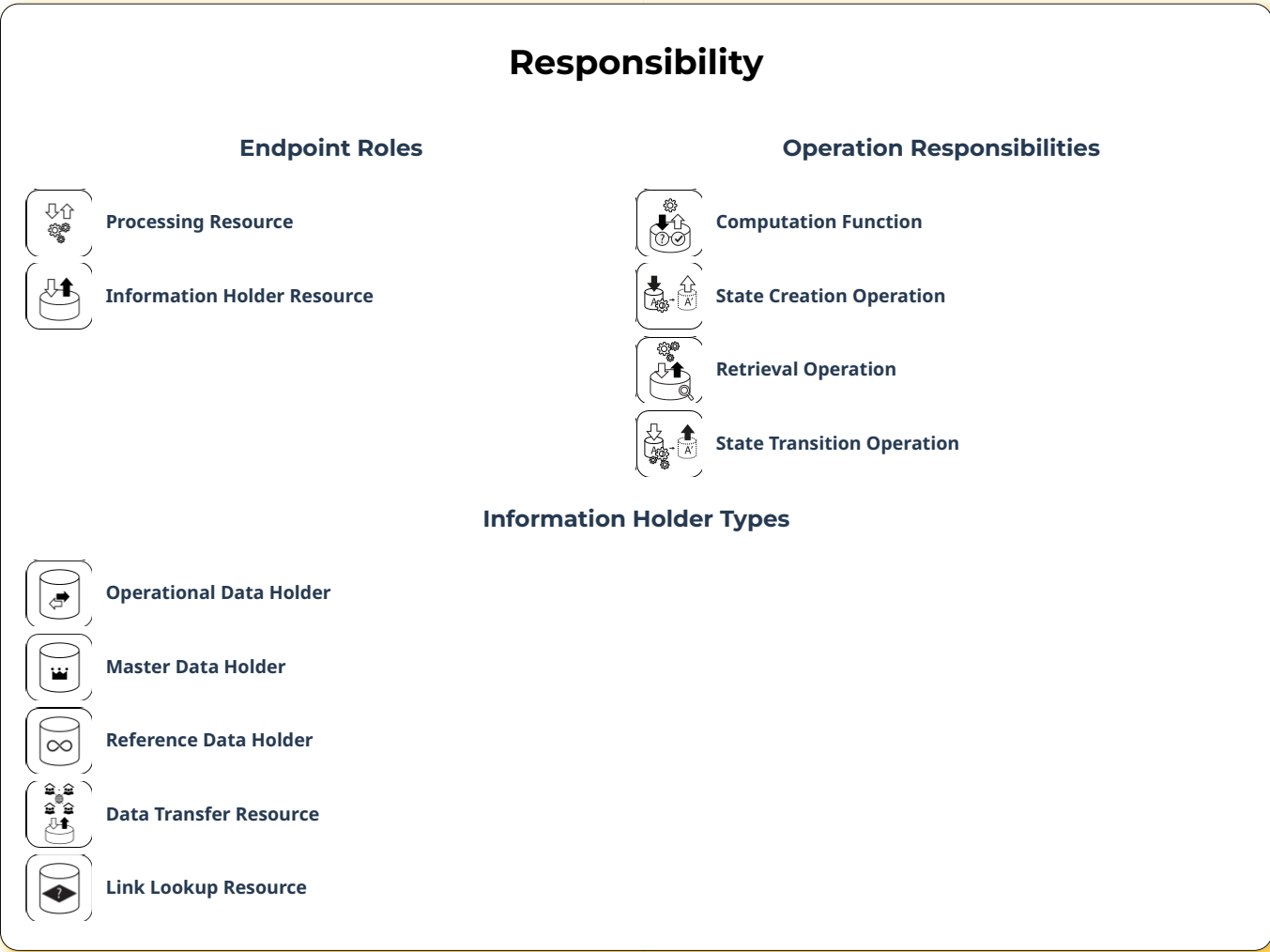


Responsibility Patterns


The patterns in the responsibility category clarify the architectural roles of API endpoints and the responsibilities of their operations.


Category Overview

The Responsibility Patterns category comprises the following patterns:





Endpoint Roles


	PATTERN: <u>INFORMATION HOLDER RESOURCE</u>
Problem	How can domain data be exposed in an API, but its implementation still be hidden?
Solution	Add an Information Holder Resource endpoint to the API, representing a data-oriented entity. Expose Create, Read, Update, Delete (CRUD), and search operations in this endpoint to access and manipulate this entity. Define and manage reference links to other endpoints.


	PATTERN: <u>PROCESSING RESOURCE</u>
Problem	How can an API provider allow its remote clients to trigger actions in it?
Solution	Add a Processing Resource endpoint to the API that bundles and wraps application-level activities or commands as its operations (a.k.a. "actions required").

Information Holder Types

	PATTERN: <u>DATA TRANSFER RESOURCE</u>
Problem	How can two or more communication participants exchange data without knowing each other, without being available at the same time, and even if the data has already been sent before its recipients became known?
Solution	Introduce a special type of Information Holder Resource, a Data Transfer Resource endpoint with a globally unique, network-accessible address that two or more clients can use as a shared data exchange blackboard. Add at least one State Creation Operation and one Retrieval Operation to it.

	PATTERN: <u>LINK LOOKUP RESOURCE</u>
Problem	How can message representations refer to other, possibly many and frequently changing, API endpoints and operations without binding the message recipient to the actual addresses of these endpoints?
Solution	Introduce a special type of Information Holder Resource, a dedicated Link Lookup Resource endpoint that exposes special Retrieval Operation operations that return single instances or collections of Link Elements that represent the current addresses of the referenced API endpoints.

	PATTERN: <u>MASTER DATA HOLDER</u>
Problem	How can I create, read, update, and (possibly) delete data that lives long, does not change frequently, and is referenced often by other data directly or indirectly?
Solution	Mark an Information Holder Resource to be a dedicated Master Data Holder endpoint that bundles master data access and manipulation operations in such a way that the data consistency is preserved and references are managed adequately. Treat delete operations as special forms of updates (that must meet compliance requirements).

	PATTERN: <u>OPERATIONAL DATA HOLDER</u>
Problem	How can an API support clients that want to create, read, update, and/or delete instances of domain entities that represent operational data: data that is rather short-lived, changes often during daily business operations and has many outgoing relations?
Solution	Tag an Information Holder Resource as Operational Data Holder and add API operations to it that allow API clients to Create, Read, Update, and Delete (CRUD) its data often and fast.

	PATTERN: <u>REFERENCE DATA HOLDER</u>
---	--

**PATTERN: REFERENCE DATA HOLDER**

Problem

How should data that is referenced in many places, lives long, and is immutable for clients be treated in API contracts?

Solution

Provide a special type of Information Holder Resource endpoint, a Reference Data Holder, as a single point of reference for the static, immutable data. Provide read operations, but no create, update, or delete operations in this endpoint. Update the reference data elsewhere if needed (backend, separate management API).

Operation Responsibilities

**PATTERN: COMPUTATION FUNCTION**

Problem

How can a client invoke side-effect-free remote processing on the provider side to have a result calculated from its input?

Solution

Introduce an API operation `f` with `f: in -> out` to an API endpoint (for instance, a Processing Resource) that validates the received request message structure, performs the desired function `f`, and returns its result. This Computation Function neither accesses nor changes the server-side application state.

**PATTERN: RETRIEVAL OPERATION**

Problem

How can information owned or controlled by a remote party (a service provider) be retrieved (to satisfy an information need of an end user or to allow further client-side processing)?

Solution

Add a read-only operation `f: (in,S) -> out` to an API endpoint to request a report that contains a machine-readable representation of the requested information (this API endpoint may be a Processing Resource or an Information Holder Resource). Add search, filter, and formatting capabilities to the operation signature.

**PATTERN: STATE CREATION OPERATION**

Problem

How can an API provider allow a client to report that something new has happened that is worth capturing for later processing?

Solution

Add a State Creation Operation `f: in -> (out,S)` to an API endpoint (e.g., a Processing Resource or an Information Holder Resource) that is in essence write-only. Here 'in essence' means that such operations might have to read some state, e.g., to check for duplicate keys in existing data before creation, but their main purpose should be state creation.

**PATTERN: STATE TRANSITION OPERATION**

Problem

How can a client initiate a processing action that causes the server-side application state to advance? How can API clients and API providers share the responsibilities required to execute and control business processes and their activities?

Solution

Introduce an operation in an API endpoint (typically a Processing Resource, or an Information Holder Resource) that combines client input and current state to trigger a provider-side state change `f: (in,S) -> (out,S')` (a.k.a. "update action required").

[← BACK TO HOMEPAGE](#)

Microservice API Patterns

Copyright © 2019-2021.

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

