Collaborative Network for Industry, Manufacturing, Business and Logistics in Europe

D2.3

Design of an Open API for the NIMBLE Platform

Project Acronym	NIMBLE	
Project Title	Collaboration Network for Industry, Manufacturing, Business and Logistics in Europe	
Project Number	723810	
Work Package	WP2 Platform Technology Specification	
Lead Beneficiary	IBM	
Editor	Benny Mandler	IBM
Reviewers	Suat Gonul	SRDC
	Felix Strohmeier	SRFG
Contributors		
Dissemination Level	PU	
Contractual Delivery Date	31/10/2017	
Actual Delivery Date	31/10/2017	
Version	V1.0	

Abstract

The NIMBLE project aims to perform research leading to the development of a cloud and IoT platform specifically targeted to supply chain relationships and logistics. Core capabilities will enable firms to register, publish machine-readable catalogues for products and services, search for suitable supply chain partners, negotiate contracts and supply logistics, and develop private and secure information exchange channels between firms. The intention is to support a federation of such NIMBLE instances, all providing a set of core services, and each potentially specifically tailored to a different aspect (regional, sectorial, topical, etc.).

The main goal of this deliverable is to specify the externally visible interfaces of the NIMBLE platform, focusing on the externally visible Application Programming Interfaces (APIs). The connection between the API and internals of the platform is described.

The intention is to provide a clear and easy-to-use and understand API. The outcome of this deliverable comprises specifications for the external interfaces for accessing the main NIMBLE platform components and capabilities. This deliverable will form the basis for the implementation of the next version of the platform. Nevertheless, it is foreseen that the final API specification will be the result of an iterative process, for which this deliverable comprises the first step. Advances with the design, implementation, and validation of different components of the platform will be incorporated into the definitive version of the API specification.

The deliverable starts from a high-level description of the platform including major functional components. Later, the deliverable delves into more detailed descriptions of the underlying components, and the interactions thereof. In addition, the external API is presented.

Document History

Version	Date	Comments
V0.1	25/09/2017	Initial version and assignments distribution
V0.2	04/10/17	First version for review
V0.3	16/10/17	Review of the first version along with some minor additions - SRDC
V0.4	20/10/2017	Review of the first version along with some minor additions - SRFG
V0.5	29/10/17	Integrate regulations related API from AIDIMME
V0.6	29/10/17	Integrate review comments - SRFG
V1.0	31/10/2017	Consolidate a final version

Table of Contents

List of Figures

List of Tables

Acronyms

Table 1: Acronyms

Acronym	Meaning
ACL	Access Control
API	Application Programming Interface
BPMN	Business Process Model and Notation
B2B	Business to Business
CA	Certification Authority
CRUD	Create, Read, Update and Delete
GUI	Graphical User Interface
IIoT/ Industrie 4.0	Industrial Internet of Things
IoT	Internet of Things
NIMBLE	Collaboration Network for Industry, Manufacturing, Business and Logistics in Europe
PaaS	Platform as a Service
RAMI	Reference Architecture Model Industrie 4.0
UAA	User Account and Authentication
UBL	Universal Business Language
WP	Work Package
WSDL	Web Service Definition Language

Glossary (ALL)

- Angular – develop applications for different deployment platforms
- Camunda - platform for workflow and business process management.
- Cloud Foundry – an open and extensible Platform as a Service offering
- Consumable - Designing an easy to use product
- CouchBase – Scalable document oriented NoSQL data store
- External applications – Independent applications that are developed and hosted outside the platform but internally make use of NIMBLE applications.
- NIMBLE applications – cloud and IoT applications, which make use of NIMBLE core Services, and are deployed within the NIMBLE cloud platform
- Platform-as-a-Service - a cloud computing concept which offers the developer and deployer of cloud based applications the infrastructure, both HW and middleware, needed for creating and deploying successfully such applications on a cloud environment.
- RDF - A standard model for data interchange on the Web. Used in NIMBLE for storing entities description in a searchable manner.
- SPARQL – an RDF query language.
- Swagger – An API creation framework for the OpenAPI Specification
- UBL – A standard data model to represent catalogue and built-in business process messages
- User Account and Authentication – The Cloud Identity Management component.
- YAML - human-readable configuration format

1 Introduction

NIMBLE is aiming at creating a B2B platform specifically geared towards improving the efficiency of supply chain creation and operations. At its core lie several services which enable companies to publish digital versions of their catalogues, containing the range of products they sell and business services (e.g. transportation, packaging and so on) they provide, and this in turn enables other companies to efficiently search and find required counterparts. In addition, once potential partners find each other they are able to initiate a negotiations process through the platform and finally establish a supply chain relationship among them, including the creation of private information exchange channels. This deliverable describes the external APIs that enable accessing the NIMBLE platform capabilities.

The focus is on providing an open API for the core platform components. These components would enable entities to register with the platform, publish their capabilities and services, making them discoverable, and participate in resulting supply chain engagements. These basic capabilities will be enhanced at a second phase with more advanced capabilities such as enabling the selective sharing of data among partners.

Even though there are various efforts for providing platforms for the IIoT, easy to use and federated platforms targeting small and large companies seem to be missing.

NIMBLE aims to provide a technological platform for easily establishing various kinds of business relationships between different players. Therefore, the simplification of using and participating in the platform is a centre-piece of the NIMBLE architecture. NIMBLE envisions an easy-to-use front end, which will serve as a single point of interaction for external companies, while hiding all the complexities of a cloud based platform at the back-end. Thus, users will enjoy the automation without suffering from the complexity needed to achieve it.

NIMBLE will ultimately provide an open platform infrastructure, which can be deployed in multiple instances, with each instance adhering to the published APIs.

Figure 1 depicts a high level schematic view of the main components of the NIMBLE platform. The main interaction point that external entities have with the platform is via the Front-end, which acts as a gateway for accessing internal components and capabilities. In addition, there is the API as a programmatic way for accessing platform capabilities, both for the use of external applications, and for the federation layer of the platform itself.

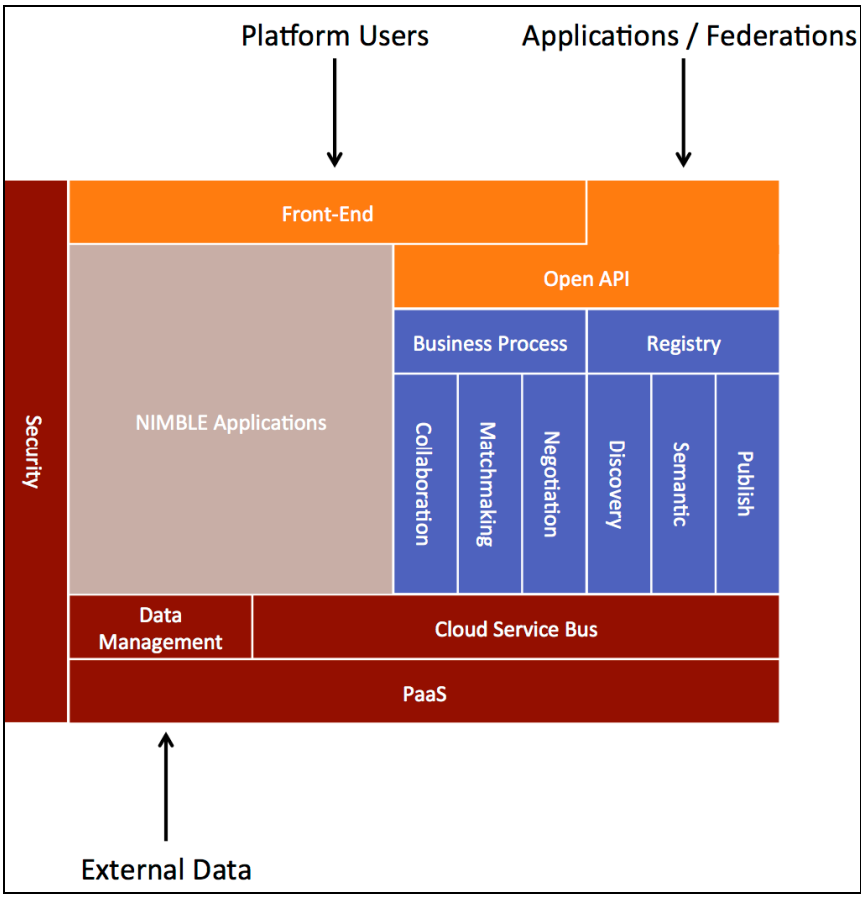

Figure 1: Main Components

The two main components include a *Registry* and a business processes component.

- The registry supports publishing information, semantically indexing it, and searching through it
- The *Business Processes* component enables matchmaking processes among entities, and offers negotiation capabilities which finally lead to a business process execution.

The Front-end and accompanying APIs goal is to ease the on-boarding of new companies into the platform and the use of available capabilities.

The API handles all externally visible interfaces of the NIMBLE platform. It serves as the first entry point for the different roles using the platform, namely, publishers, searchers, and business process deployers and executors. The Application Programming Interface (API) enables developers to register, publish, search, or establish business processes by utilizing NIMBLE platform features. The access to information gathered within the platform is also provided through the API. All externally visible interfaces will be provided as well through a common Web-based GUI, implemented as the front-end service. Thus, NIMBLE has a single-entry point for its users.

The main goal of this deliverable is to identify and specify the main ingredients of the platform API that will satisfy the identified requirements. A conceptual architecture is presented, that consists of interconnected components that are integrated in the NIMBLE platform.

1.1 Major considerations for the API

We envision a one -stop shop to easily provide services required by companies for establishing business relationships between them. Thus, we aim at covering the various stages of a relationship lifecycle, in a horizontal manner, made accessible via the API. We aim for enabling different entities to be able to form different kinds of relationships and applications based on their current needs using the same infrastructure and the API to access the system. We aim for information concerning capabilities and services to be readily available, published and shared such that it can be used in newly forming business relationships and applications. The platform will provide the technological backbone, in a unified manner, accessing published information and integrate it into new applications and processes. The APIs can be used by developers to implement new applications intended for deployment using the NIMBLE platform.

The platform shall be federated and potentially personalized such that different entities can obtain specific capabilities they require, but at the heart there should be a core set of capabilities that are supported throughout the federation. This deliverable covers that core set of capabilities. Internally, the federation should be aware of sibling instances, such that operations that need to flow across instances will be able to do so. The federated interactions shall be performed using the API described herein.

2 Architecture High Level View

At the heart of the NIMBLE vision lies the existence of many different companies that need to create their own business networks. The companies need to be able to publish their capabilities and look for corresponding capabilities they need. In addition, upon establishment of a business relation, data and notifications need to flow among entities.

The architecture high-level view describes the three main components: the end-users, front-end and API, and the interfaces to the NIMBLE platform internals. The front-end and API integrate all management and developer tools for end users and developers. The focus is on the entire life cycle management of business transactions. At its core the API layer serves three main aspects of the platform:

1. Provide easy access for developers to the capabilities provided by the platform, with an initial focus on the core services provided by the platform.
2. Federation services, in which incoming requests may be directed to different and multiple instances of the NIMBLE federation.
3. Platform extension services for 3rd party developers – to bind their applications to the NIMBLE platform and to extend the open API by new (open) API calls.

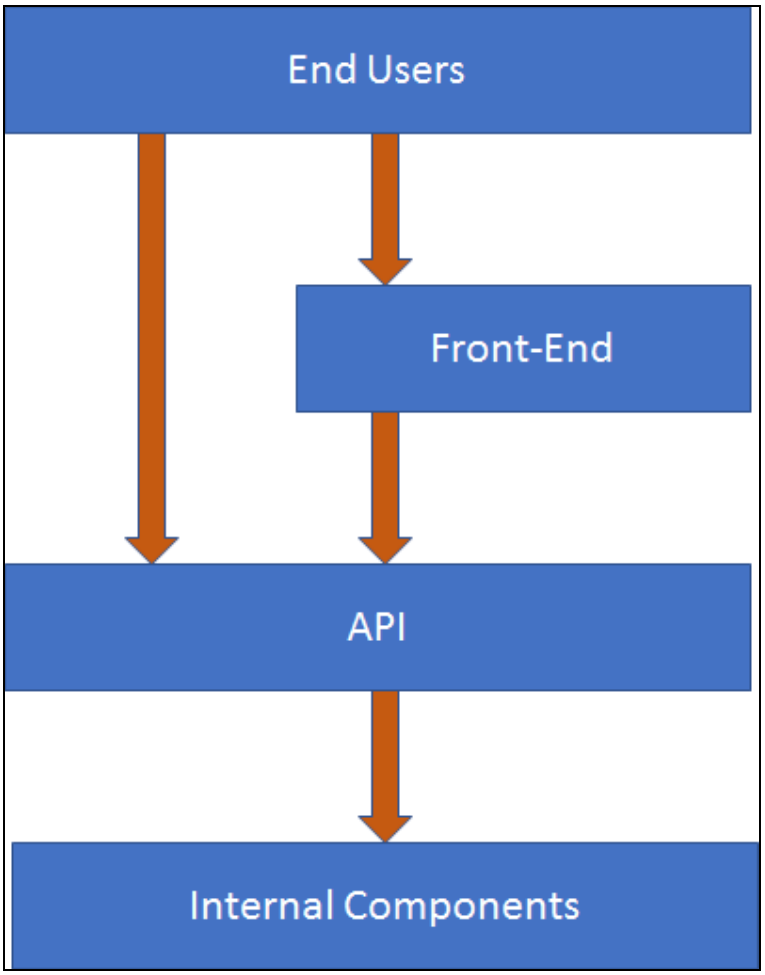

Figure 2: High Level View

The high-level view shows how major components relate to each other and also indicates how some of the design and development tasks of the work plan are related to the architecture. Internal components include core platform capabilities aimed at, for example, enabling the discovery and establishment of business processes among different entities.

The platform capabilities can be used from the outside, via the platform *front-end* and via an *API*. The API is intended to enable the formation of ecosystems where a NIMBLE platform is either a central part or just a provider of certain services, to some other ecosystem.

2.1 External Access

External access to the platform will be available via a front-end and a programmatic API. These components shall serve two kinds of groups, namely platform users and developers who wish to develop applications using NIMBLE services. The two target audiences have slightly different requirements that need to be served by both components.

Specifically, the API will enable a developer to perform all required interactions with the platform. Main capabilities to be provided are for registration of companies and individuals, publishing capabilities, search for capabilities, and business process related activities such as

negotiations and execution of established processes. The graphical interface will serve as a human-accessible, user-friendly front-end to the platform API.

As can be seen in Figure 3, the API resides on top of the NIMBLE platform and brokers between the external developers and the underlying technical implementation of the different components. Under the covers the API communicates with the NIMBLE platform via a RESTful API, that is provided by the front-end components. That in turn serves as the only entry point into the system, and serves as a gateway into the technical deployment of the system components, which in turn resides within a protected managed cluster. The platform API layer may serve as a simple broker translating between external to the proper internal calls. In some other cases, some additional functionality may be carried out by the API layer, for example for authentication, or internal context based filtering.

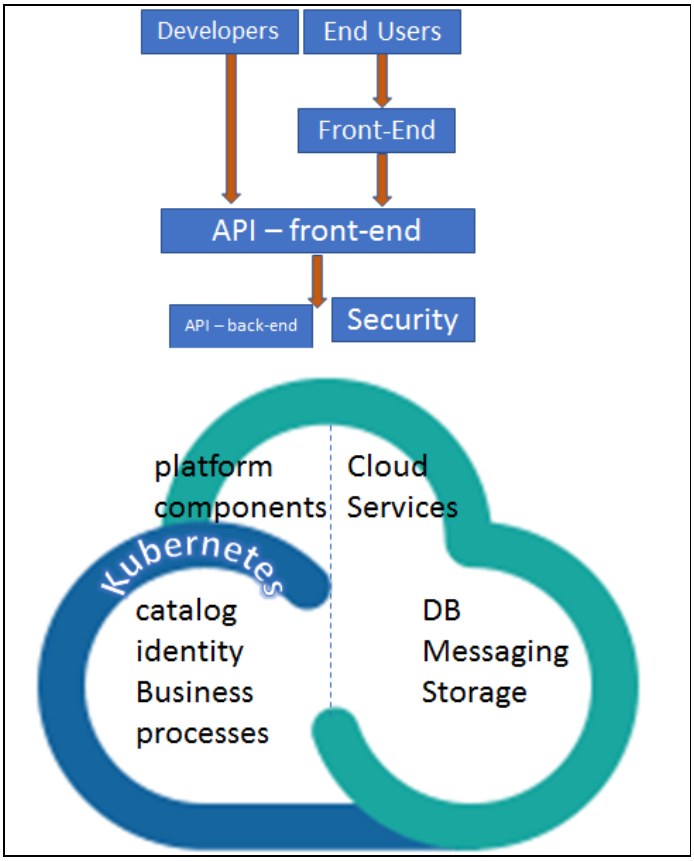

Figure 3: High level components interaction

2.2 Federation

The vision is for a federation of platforms for B2B operations, including a set of core services that is shared by all platforms, and optional added-value business functions that can be provided by third parties and may be available in some platforms but not in others. Thus, prospective NIMBLE providers can take the open source infrastructure and bundle it with sectorial, regional or functional added value services to launch a new platform instance in the federation.

The architecture of the platform is designed in this "federation ready" manner. Namely, a cloud based deployment pattern including a set of mandatory core services with a well-defined API. The API will enable both access to internal platform services, as well as a federation API, which will enable separate instances of the federation to collaborate.

2.3 B2B Collaboration: catalogue and business processes

Setting up a B2B collaboration infrastructure among business entities regardless of their size, type and role in the supply chain is the main motivation behind developing a platform like NIMBLE. The main path to achieve collaboration among registered entities goes through publishing and searching for capabilities, followed by a negotiation process. If successful that shall lead to the design, deployment, and execution of a business process, which entails the exchange of information among entities. All these capabilities need to be made available via the platform API.

Before collaborating on NIMBLE, business entities need to register on the platform by providing metadata about the entity itself such as trading preferences, target sectors, certifications, etc. Being the first step to be visible on NIMBLE, such information makes establishing the initial agreement with new partners easier. The next step is to publish capabilities in the form of products manufactured, materials supplied, and services provided. These are the major resources describing the entity on NIMBLE and make it discoverable for collaboration. Entities might negotiate on the trading conditions as well as data to be shared based on previous agreements or type of collaboration.

B2B collaboration through business processes is realized in three steps: design, configuration and execution. The design and configuration steps are enabled by the Business Process Design Service which is one of the core services of NIMBLE. This is a cloud based service with user interfaces enabling specification of a concrete business process by determining the type of entities to be involved in the process e.g. manufacturer and supplier; designing the information exchange flow among the entities and assigning the entities to data sender and recipient roles. Afterwards, the process is instantiated and executed by Camunda (https://camunda.org/), an open source business process engine.

2.4 Micro-Services Approach

Microservices is an architecture form and methodology for breaking up a large complex system into small units, each fulfilling a small and unique well-defined task, in an independent manner. Each such microservice can be deployed separately and interacts with its peer microservices using standard communication protocols, under the supervision of an application orchestrator. Such an approach eases the task of long-term maintenance and evolution of a large system, when compared to a monolithic system. In addition, internal changes within a microservice do not disturb any other system component as long as the external contract of the microservice is adhered to. Moreover, each component can be implemented in a different programming language, using different middleware.

A gateway to the microservices is usually deployed, which acts as the public entry point to the microservice, providing routing, filtering, and load balancing capabilities. From an API design and deployment perspective this is the place where the main externally visible API will reside, while internally it shall be calling the APIs of the internal components to fulfil the user request.

Finally, security and access control are the responsibility of the User Account and Authentication (UAA) & Identity Management component which administrates identities on the

platform. Cloud Foundry's UAA[1] or Keycloak[2] can be used as authentication server for the OAuth2 and OpenID Connect standards.

3 Platform Core Components

The core services represent the heart of the capabilities that should be served by any NIMBLE platform deployed instance.

A NIMBLE platform deployment resides on a cloud infrastructure, with its first manifestation realized within a Kubernetes cluster. The deployment is then connected and served by generic cloud based services which are required for enabling statefull components and for interaction among components. Additional helper components may be deployed as well, to serve as an entry point gateway and load balancer into the platform; for service discovery, which helps individual components to find other components it needs to interact with. The NIMBLE internal components are deployed within the cloud, and the relative URL of each component is defined at deployment time. The front-end is then deployed, which can access all internal components.

The following are the main components which will be available through the platform API.

3.1 Registry

The registry is the component to store the information that makes companies discoverable within the NIMBLE platform. The registry holds information about different companies and the services they provide on the platform.

Catalogue ingestion

The catalog ingestion component enables companies to insert their data using specific formats or not. Internally the information is normalized such that it can be discovered later, indexed as key-value pairs in a free-text index. The NIMBLE API needs to support two flavours of catalogue ingestion, namely one at a time, or in bulk supplying a filled in pre-defined template.

By default, NIMBLE makes use of the UBL standard for representation of catalogues. In addition, it is possible to make use of standards personalized for a specific industry, such as MODAML, or furniture compliant catalogues. For UBL-based catalogues, during the publishing process, users are able to semantically annotate their products with readily available product categories. Sector-specific categories can be used provided that there is a related adapter associated with the platform. Thus, the platform defines an API that offers basic semantic annotation capabilities referring to different ontologies, taxonomies, and controlled vocabularies, thus providing support for plugging in sector specific standards.

Catalogue Discovery

This component is the counter-part of the ingestion mechanism, which enables users to find information stored and indexed from supplied catalogues within the platform. The API needs to support different kinds of search capabilities, namely keyword based search, faceted search, and semantic search. All these need to be supported in an accessible manner to developers. Under the covers, internally, the platform may perform different kinds of search activities to respond to the incoming query. Internal platform matchmaking capabilities may take place under the

[1] https://github.com/cloudfoundry/uaa
[2] http://www.keycloak.org/

covers as well, to provide more accurate and relevant responses based on additional knowledge of restrictions and requirements.

Once deployed, this component will serve a REST interface for supporting incoming queries. Each search query will include a user input data and potentially a search function. The results will be provided in a JSON format.

3.2 Business process

A business process involves data exchanges between different entities. All participating entities should be registered in the platform, and the entity initiating the process can either target an apriori known counter-part, or use the platform discovery capabilities to choose a partner.

Business process support covers several steps, namely template creation (involved parties and data, sequence of operations), followed by the deployment and gradual execution of the business process.

The execution follows several steps from each of the partners in their own turn. In each such stage, data to be shared with the target entity is prepared. The recipient entity at this stage needs to listen for incoming events. NIMBLE will make use of Business Process Model and Notation (BPMN) 2.0[3] at the core for business process representation.

NIMBLE will provide and make available through the API, built-in templates and business processes for generic supply chain operations. Once all nodes of a business process are configured, the process becomes ready to be executed.

3.3 Data management and analytics

The data management component deals with data ingestion, storage, and potentially some level of processing for shallow data analysis. Selective data sharing should be supported as well. Moreover, data based notifications can be supported by connecting the data pipe to the cloud services bus which serves as the generic messaging pipe.

Ultimately the data management component will be provided as-a-service to the rest of the NIMBLE components internal and external to the platform. The Big Data Toolset will be made available through the PaaS model, enabling developers to deploy, host, and manage applications that can handle data coming from IoT-enabled devices. Thus, any NIMBLE component will be able to invoke data management procedures such as storage and analytics.

Overall these data management services along with their respective API shall be elaborated, further designed and implemented in later stages of the project, and more detailed information will be provided in the relevant deliverables. In particular enclosed are some initial design goals for two of the main corresponding components:

Product data sharing throughout the supply chain

Data can be provided in two ways:

1. Store data over the NIMBLE platform: Saving data and metadata that can be indexed via ElasticSearch, to be easily found and retrieved; the metadata must be provided along with the data.

[3] http://www.omg.org/spec/BPMN/2.0/

2. Register data retrieval services in the platform, with their metadata to find them. These services are run by the registering company and are called every time in which the data is needed.

Handling IoT and M2M data

IoT is becoming an important data source for a variety of market segments, among them business process within and across companies. Within the platform there shall be provisions for ingesting IoT data of different kinds. First and foremost, for the use of the entity owning the data originating device. Moreover, the platform shall enable the definition of processing elements that should be applied to incoming data as its flowing through the system, as well as the possibility to store such data for further future offline processing and analysis.

Established business processes between entities may be able to obtain a glimpse of such data flowing into the platform, in its raw form or as the end result of the processing of the actual raw data. Thus, data or its effects may be linked across companies, both electronically or to the enterprise itself.

3.4 Security

NIMBLE requires registration to the platform to be able to fully participate, and shall keep track of corresponding authentication and authorization. These capabilities shall be reflected in the API. Under the covers standard and open source components shall be used to enforce security.

Standard security mechanisms will be used to accomplish confidentiality, integrity and availability in the microservice architecture. Users will have to be authenticated and authorised on the platform to access sensitive resources. Open standards such as OAuth2 shall be used.

APIs shall be included, among other services, for registration and login, and will interact with the underlying mechanism ensuring security across microservices. Internally, identity management will be handled using open source tools.

Different access policies can be applied to every user interaction with the platform and some functionality can be limited to specific user roles. In general, entity-related data, catalogue data and business process data will be subject to such a role-based authorization. Role definition and association capabilities will be provided via the KeyCloak tool. More detailed information is provided in D6.1 (Security and Privacy Requirements).

3.5 Front-End

The front-end provides a graphical user interface for accessing the NIMBLE capabilities. It serves as a central receiver for user interactions/requests and delegates those requests to the appropriate services, such as search, publish and business process related capabilities.

The back-end of the front-end invokes the APIs of the different internal components. These will serve as the basis for the overall platform API.

4 Flows through the platform

The front-end is the main component which external users encounter when dealing with the NIMBLE platform. The API serves the same purpose for developers of third-party applications.

The main task of both entities is to help different kinds of users accessing internal capabilities of the platform.

The first action to be taken by an end user would be to *register* first as a person and then register his company with the platform. The registration process will involve security and identity management aspects. Once the registration is performed the next step for a user to interact with the platform would be to *login*.

As a logged-in entity our preferred user shall *publish* a catalogue including capabilities and services they can provide. A complementary task would be for a company to *search* for supply chain partners. Both actions would be internally using the registry.

Finally, a *business process* is initiated after potentially going through a negotiation process, and the relevant flow of information and transactions can begin for the entities involved in the business process.

In addition, data flows and exchange of different types of messages can be initiated via the API as well. During such registration, the initiators shall be able to define security policies applied to their data. These policies may define data sharing restrictions.

NIMBLE provides a programmatic API which will enable users to access capabilities programmatically, and this will form the basis for the federation capabilities.

4.1 End-user: Interaction with the platform

As already mentioned, the main component dealing with end-user interaction is the platform front-end. Nonetheless, some functionality shall also be directly accessible via API endpoints in order to facilitate integration with existing platforms and legacy systems.

Generally, one can break the interaction capabilities down into the following main features:

- Registration of users and companies with different roles and affiliations (UI only)
- Product and service catalogue publishing (UI and API)
- Establishing a business process; includes searching for supply chain partners, matchmaking and negotiation (UI and API)
- Managing data flows and processing capabilities (UI and API)

5 External interfaces and technologies to access NIMBLE

As discussed in the previous sections, NIMBLE will provide a set of services (core services and value-added services, cross-instance federation), which should be accessible in a programmatic way via an API and a human-interactive way via a GUI. Target users for the APIs are application developers, while target users of the GUI are usually domain experts from specific industrial sectors. Also, the application developers may have domain knowledge of a specific sector. All components should be accessible via the platform front-end with a unified User Interface (UI) and User Experience (UX) concept. The front-end shall focus on Web technologies and development frameworks providing a good UX experience on various platforms.

In most cases, the GUIs will simply use the APIs for accessing the functionalities of the NIMBLE services. As a general principle, platform services should provide a RESTful API based on HTTP for its public services. Data should be exchanged in JSON-Objects, following a documented structure. A short documentation of each API shall be provided to allow other developers to make use of it without major difficulties. The Swagger Framework[4] shall be used for the API documentation. Using this YAML-based specification format would allow to create documentation as well as an implementation of client libraries in more than 30 different programming languages. In addition, it can also be used to generate server stubs in different formats. This enables the platform to provide documentation as well as downloadable example client implementations without any additional effort.

Some industry sectors / application domains may require SOAP/WSDL services to allow seamless integration with legacy applications. Using SOAP is therefore an option for the implementation of value-added services. Still, all core services must at least provide a RESTful API Endpoint.

In both cases, HTTP(S) will be the primary choice for data transport. Session management and security features from HTTP can be used as needed by the services. For the Create, Read, Update and Delete (CRUD) data manipulation operations, the HTTP methods for GET/POST/PUT/DELETE shall be used. The API shall be based on REST and HTTP protocol. This choice eases the task of external developers, since these are very well known and used technologies. Moreover, a wide variety of programming language and tools and supporting technologies is available. The RESTful API is based on the operations for CRUD, which are mapped to the corresponding HTTP methods implemented by the different components. The payload of the HTTP requests and responses will be JSON, being a lightweight format and supported by a variety of programming languages. The API shall enable developers to interact directly with NIMBLE provided components and capabilities. The API handling shall be made scalable such that a large amount of interactions with the platform can be supported simultaneously. For services that provide large result data sets (e.g. item lists), paging and offset mechanisms must be implemented to limit the amount of transferred data per request.

All NIMBLE services with an Open API should be published via a central access point, called "Gateway Proxy". Although the loads can then be distributed to multiple instances in case of high performance requirements, the address of the gateway proxy – and therefore the exposed API address – should be stable.

Finally, we propose the use of the following URL naming pattern for all NIMBLE services:

https://<nimble-platform-instance-address>/<service-id>/[<version>/<subservice/.../>/]

with the following components:

- **<nimble-platform-instance-address>**: this is the DNS-name and port number to access the platform instance from the public internet, e.g.: https://nimble-platform.uk-south.containers.mybluemix.net/job/Nimble-Platform/
- **<service-id>:** instance-unique id or name of the service. This part is managed by the Gateway Proxy, which will forward requests to the actual service in the microservice architecture.
- **<version>**: Optional version number of the service. If no version number is used by the request, the service should automatically provide its latest API.
- **<subservice>:** this parameter directs the call inside the service to several sub-services or simple html-pages. The subservice name "**api**" is reserved and should point to the

[4] https://swagger.io/

documentation of the service. This could be the swagger specification (api.yml) or the auto-generated HTML-documentation.

- The definition of service parameters (e.g. GET or POST parameters) as well as response types is up to the service itself, and will be defined in the API documentation of each service.

The API is the layer that establishes the interaction between the NIMBLE platform components and the external end-users. The NIMBLE API is designed as a Web API implemented using HTTP and REST principles. Thus, the API is built on top of HTTP REST operations. As it is based on REST principles, the API has a view of resources, triggering the functionality depending on the operations performed on these resources, mainly CRUD operations. The following NIMBLE components expose an external API:

- User management
- Registry (ingestion and query)
- Business processes

The following sections describe in more details the external components and interactions.

The API Web Services may be developed using a Java programming language API, such as **JAX-RS** (Java API for RESTful Web Services). JAX-RS provides some annotations to aid in mapping a resource class as a web resource. In addition, it provides further annotations to method parameters to pull information out of the request.

Jersey is the REST framework providing the JAX-RS Reference Implementation and more. Jersey provides its own APIs that extend the JAX-RS toolkit with additional features and utilities to further simplify RESTful service and client development

As a JSON processor a Java library will be used, such as **Jackson** (http://jackson.codehaus.org/). Jackson is a high-performance suite of data-processing tools for Java, including the flagship JSON parsing and generation library, as well as additional modules.

The Jackson Project also has handlers to add data format support for JAX-RS implementations like Jersey.

As an HTTP Web Server, a Java Servlet container we will use a technology such as **Jetty** or **Tomcat**. Jetty is a pure Java-based HTTP server and Java Servlet container. Jetty is often used for machine-to-machine communications, usually within larger software frameworks. Jetty is developed as a free and open source project. Tomcat is a widely used open source servlet container hosting web applications.

The API layer may use for back-end services if it requires to store and index data. It mostly is realized as a web application.

5.1 General API requirements

Enclosed is a list of guidelines that the different API components must adhere to:

- Support the HTTP family as the main mode of interaction (https will be the preferred embodiment)
- Support UTF8 for encoding
- Support CRUD REST verbs
- Support JSON representation

- Support Security Requirements
- API calls must include a security token for authorized access. Technologies to be used may include: OAuth 2.0, and Open ID Connect
- Support return codes from the families: 200, 400, 500

Enclosed is a list of guidelines that the different API components may adhere to:

- Support PUT for updates
- Support additional HTTP status codes, especially for errors.
- Support additional representation mechanisms such as XML.

5.2 Detailed API descriptions

At this stage we report the current state and already planned API, but a complete and final API documentation at this time would be premature. The different components maintain their information up to date as a part of the component's repository. For example, the online version of the identity service API can be found in: http://nimble-platform.salzburgresearch.at:443/identity/swagger-ui.html#/

5.2.1 User Management API

The User Management Interface of the API consists of lifecycle operations for a certain company and user, such as creating, updating, and deleting, in addition to the definition of roles. The user Management component is closely related to the internal identity management procedure, which is responsible for authentication and authorization, and roles management.

5.2.2 Catalogue Management API

The registry Management API provides an entry point to registry related capabilities, mostly in the form of adding information to the registry, incorporating capabilities and services provided, and querying and searching through the catalogue in various manners. This API enables access to the internal ingestion and query related components.

Registry operations are exposed through a RESTful API that uses HTTP as a transport and that acts as the registry front-end. This implies that catalogue information can be identified unambiguously using unique URIs. The API provides operations through the four main HTTP operations: GET (retrieve), POST (create), PUT (update) and DELETE. Table 1 contains a summary of the operations implemented in the REST API. Catalogues are created by POSTing a JSON document to the API. The document is a basic description of the main properties of the catalogue about to be created.

Table 2: Summary of registry API methods

Operation	URI	Action
Retrieve	GET /catalogue	Retrieve a catalogue
Create	POST /catalogue	Add a catalogue

Update	PUT /catalogue	Update a catalogue
Delete	DELETE /catalogue	Delete a catalogue
Retrieve	GET /catalogue/template	Retrieve a template
Create	POST /catalogue/template	Add a catalogue based template
Retrieve	GET /catalogue/catalogueLine	Retrieve a catalogue line
Create	POST /catalogue/catalogueLine	Add a catalogue line
Update	PUT /catalogue/catalogueLine	Update a catalogue line
Retrieve	GET /catalogue/category	Get categories

Catalogue ingestion API

The Catalogue ingestion API provides the standard CRUD (Create, Read, Update, Delete) operations over capabilities and services descriptions, along with corresponding ontologies. The information is then stored in an RDF store, and indexed via free-text and key-value indices. The API is intended mainly for storing newly created service descriptions or remove already existing services.

As already mentioned above, NIMBLE has adopted the UBL standard for representing catalogues by default. The UBL data model is not documented via the Swagger tool, but the version NIMBLE uses can be accessed via [5].

[5] https://github.com/nimble-platform/common/tree/master/data-model/ubl-data-model/src/main/schema/NIMBLE-UBL-2.1-Catalog-Subset

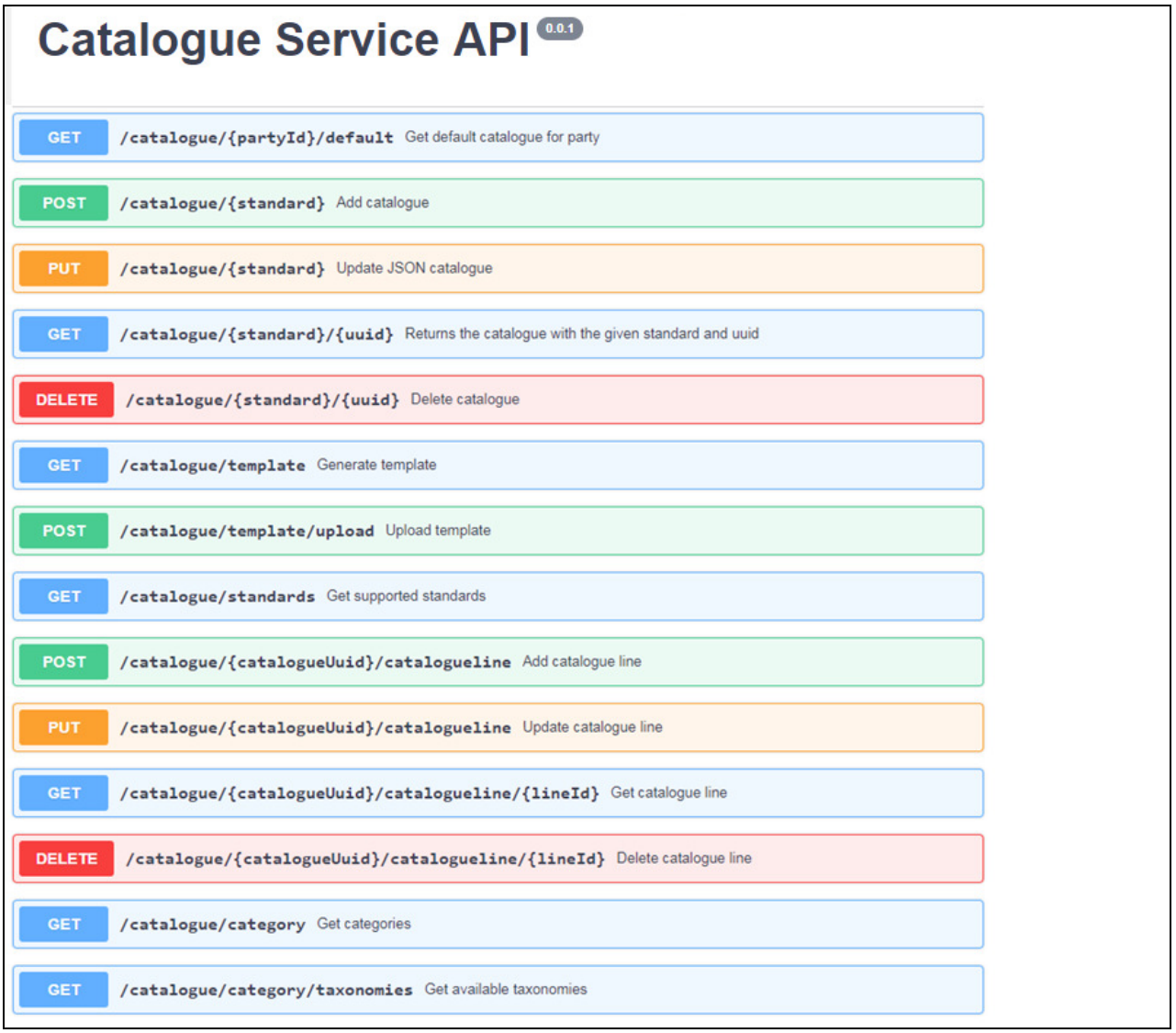

Figure 4: Catalogue Service API

Catalogue Discovery API

The Catalogue Discovery API provides access to the registry component. This API enables searching through and querying the underlying registry using various modalities. This API can be used by a company for locating required capabilities or services offered by another company. The underlying registry uses capabilities descriptions and metadata to provide matching results.

API specification

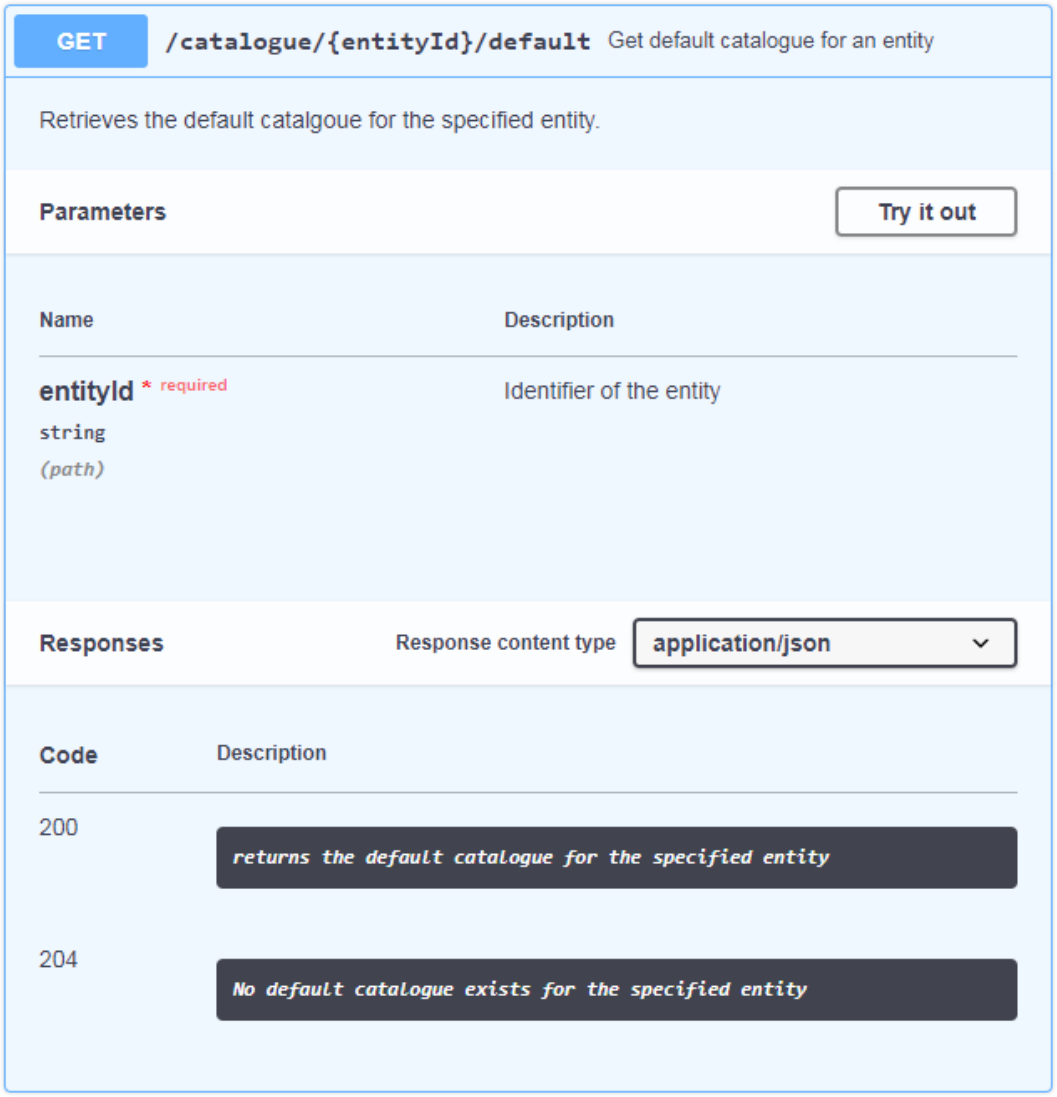

Figure 5: Get default catalogue API

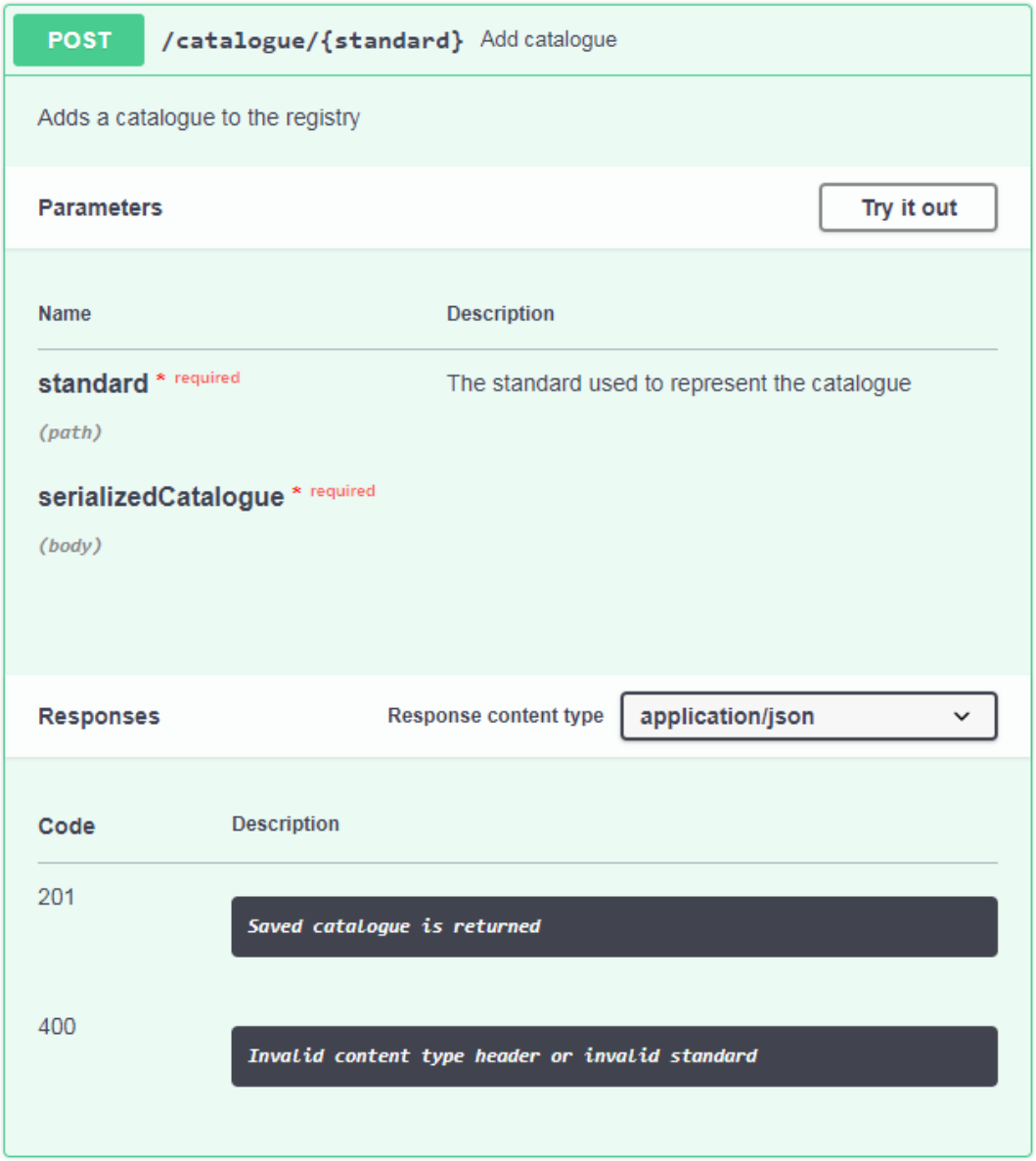

Figure 6: Add a catalogue API

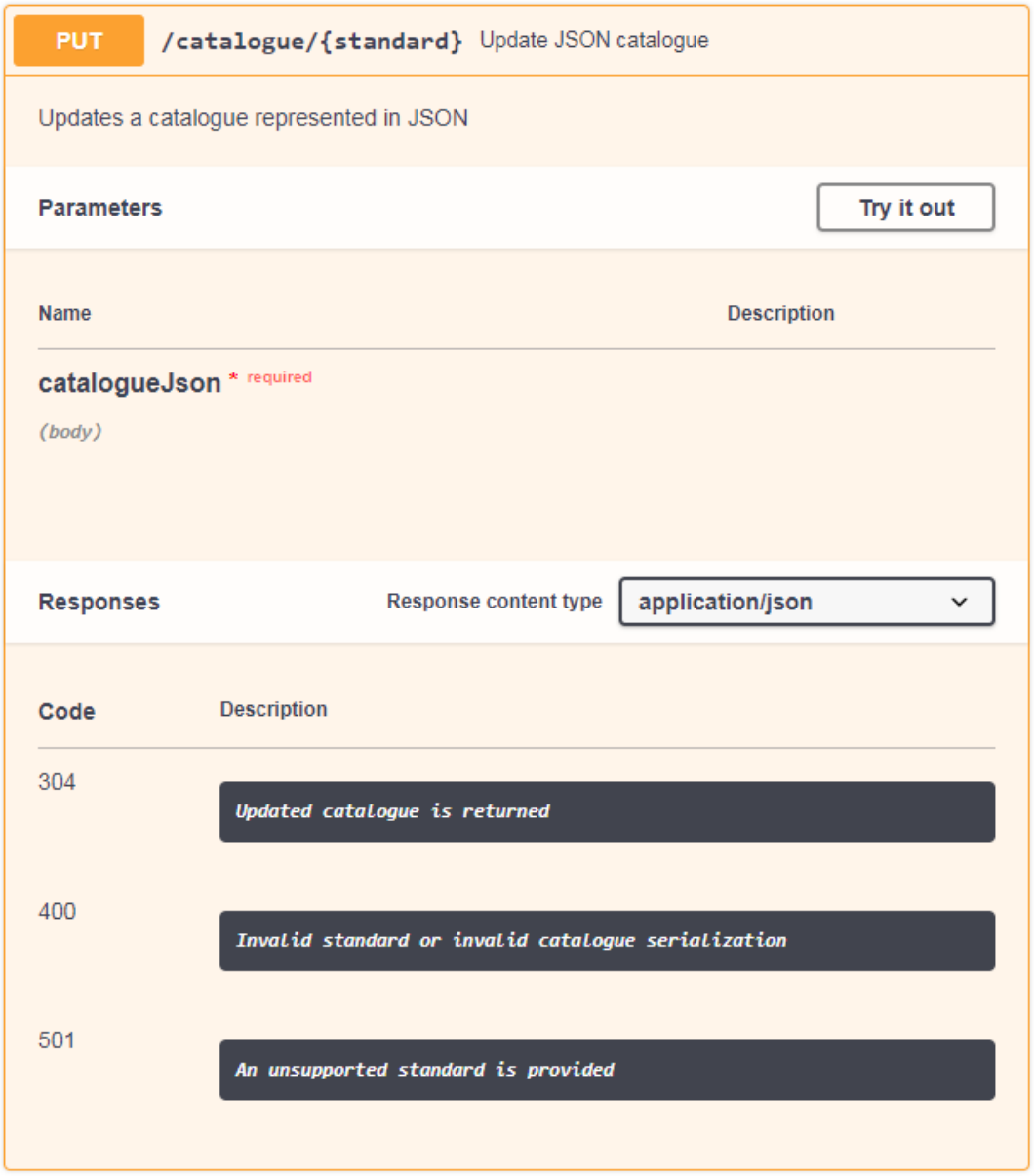

Figure 7: Update catalogue API

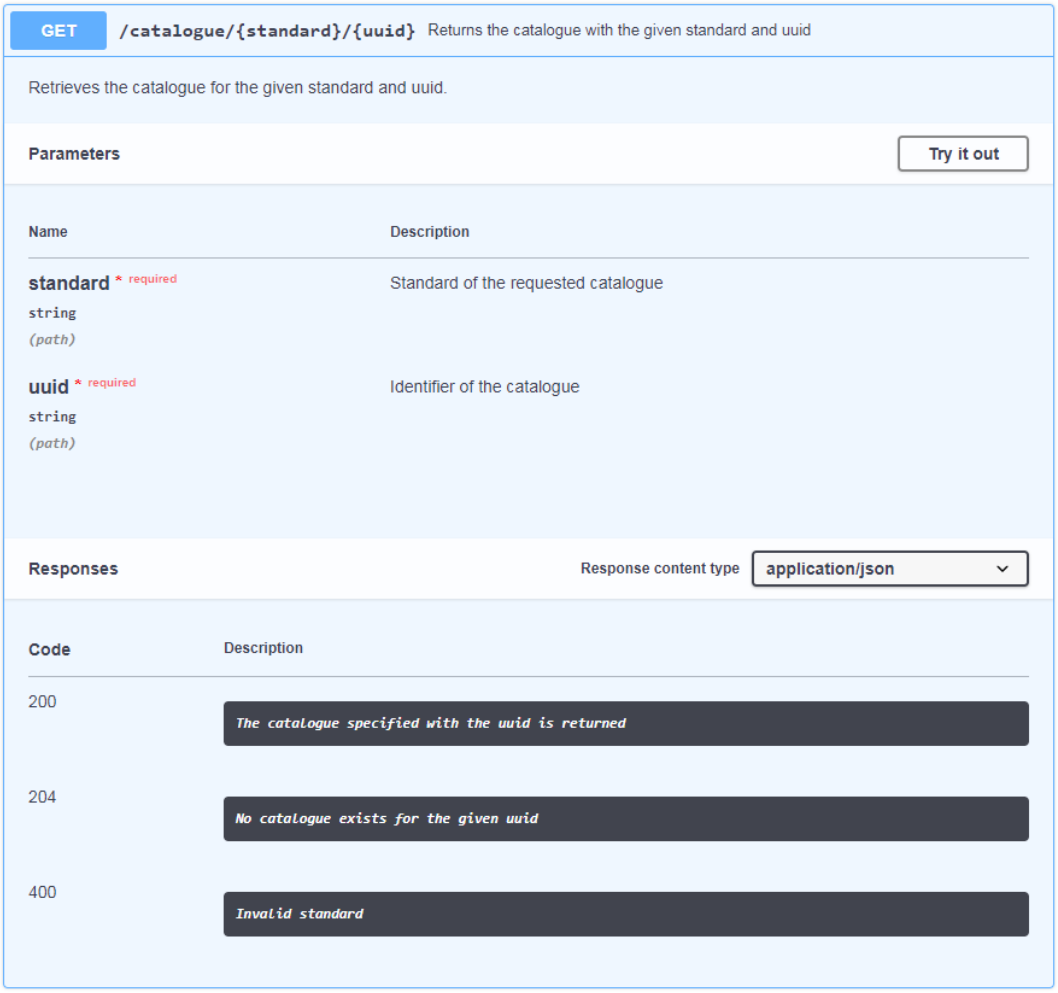

Figure 8: Retrieve a catalogue API

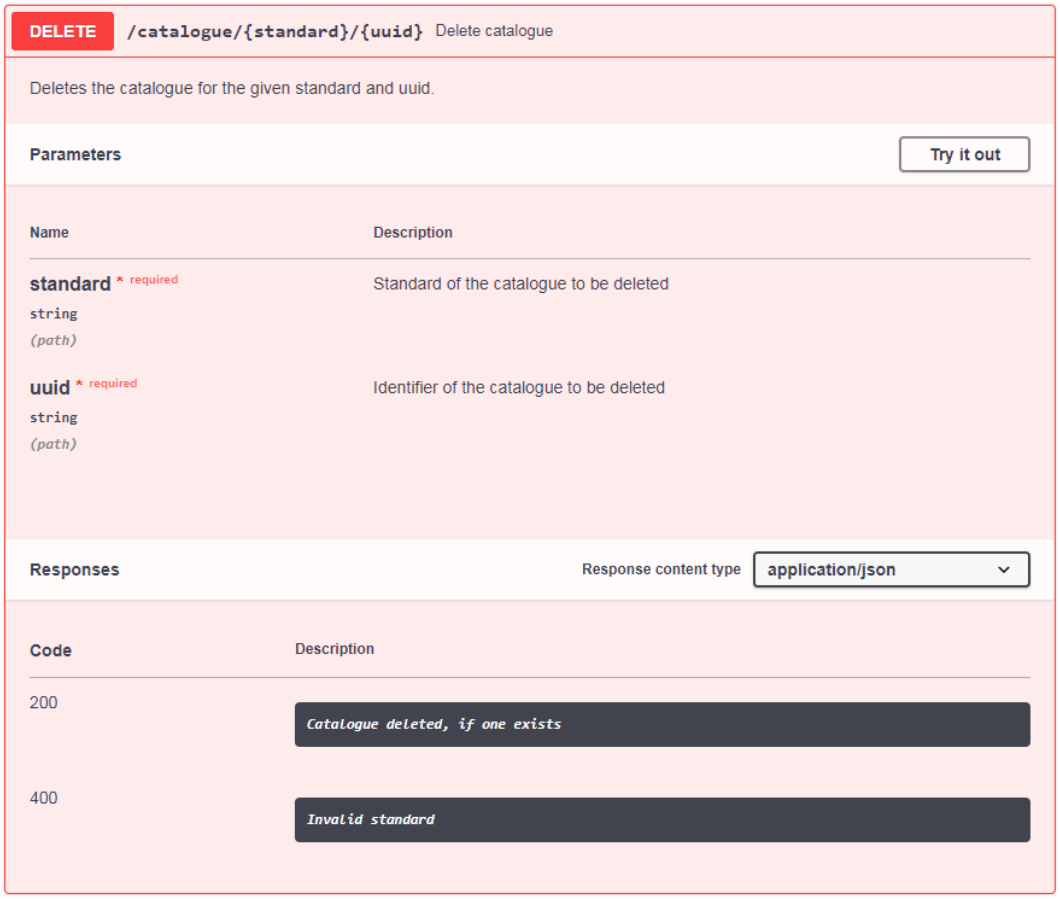

Figure 9: Delete catalogue API

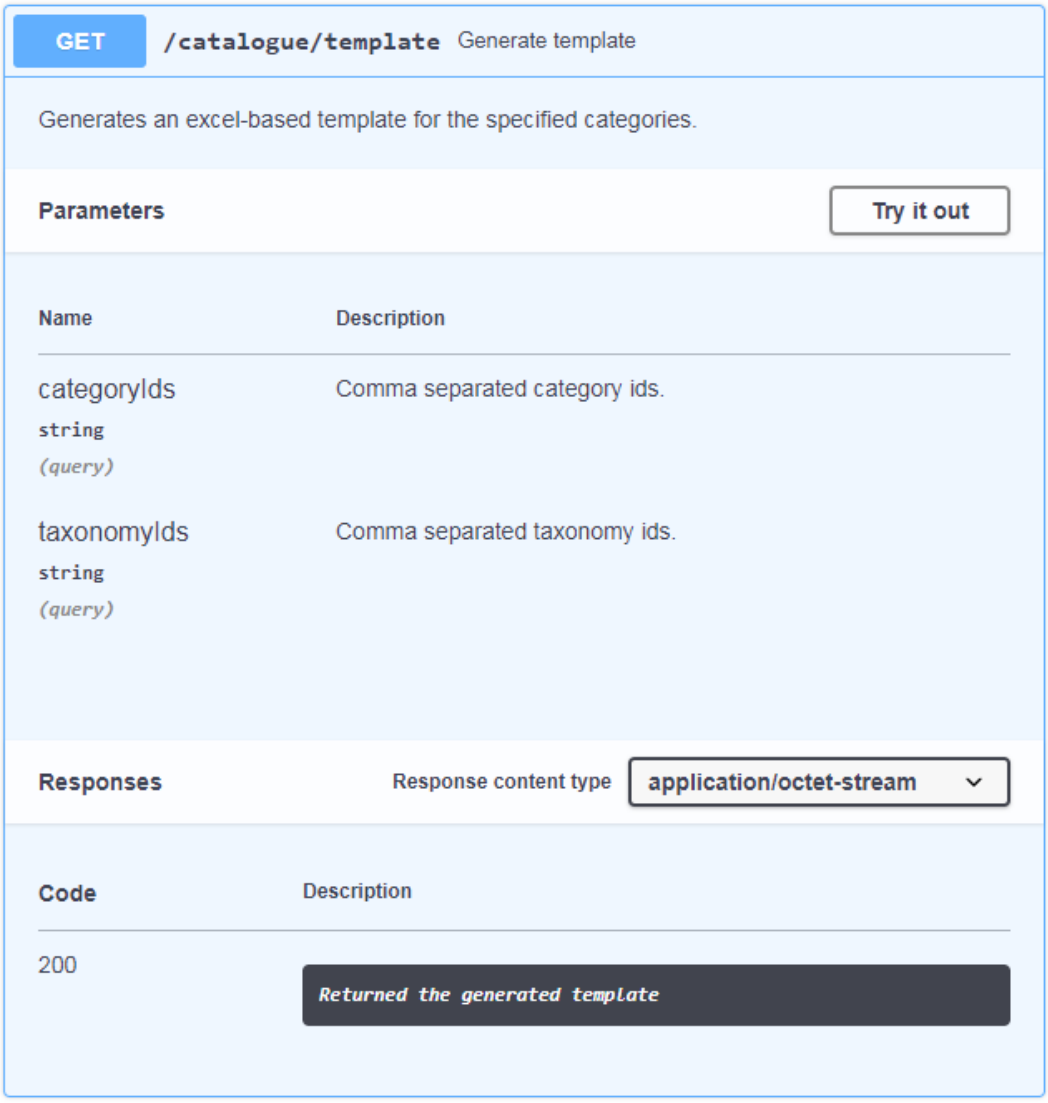

Figure 10: Generate a catalogue template API

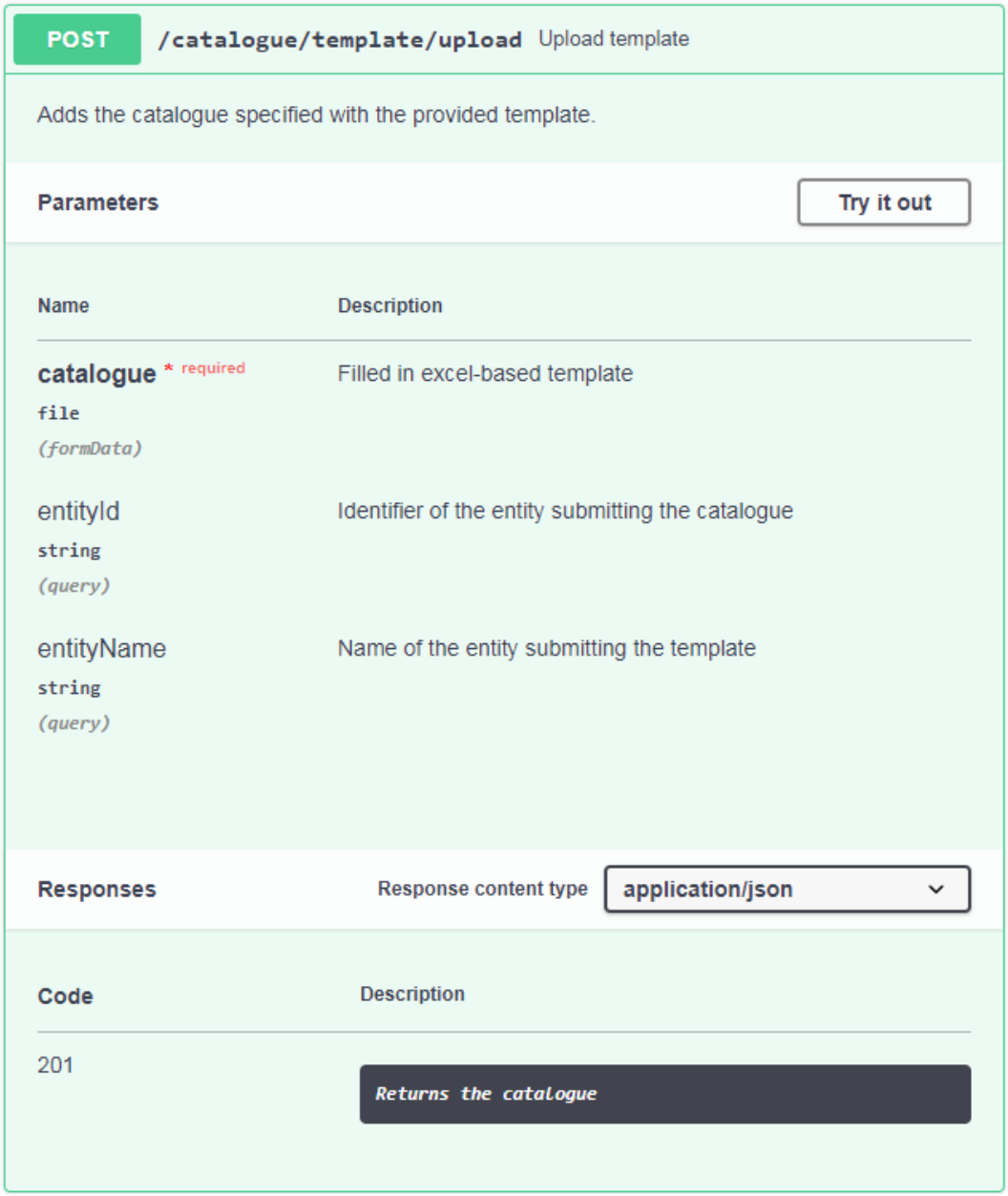

Figure 11: Upload a template based catalogue API

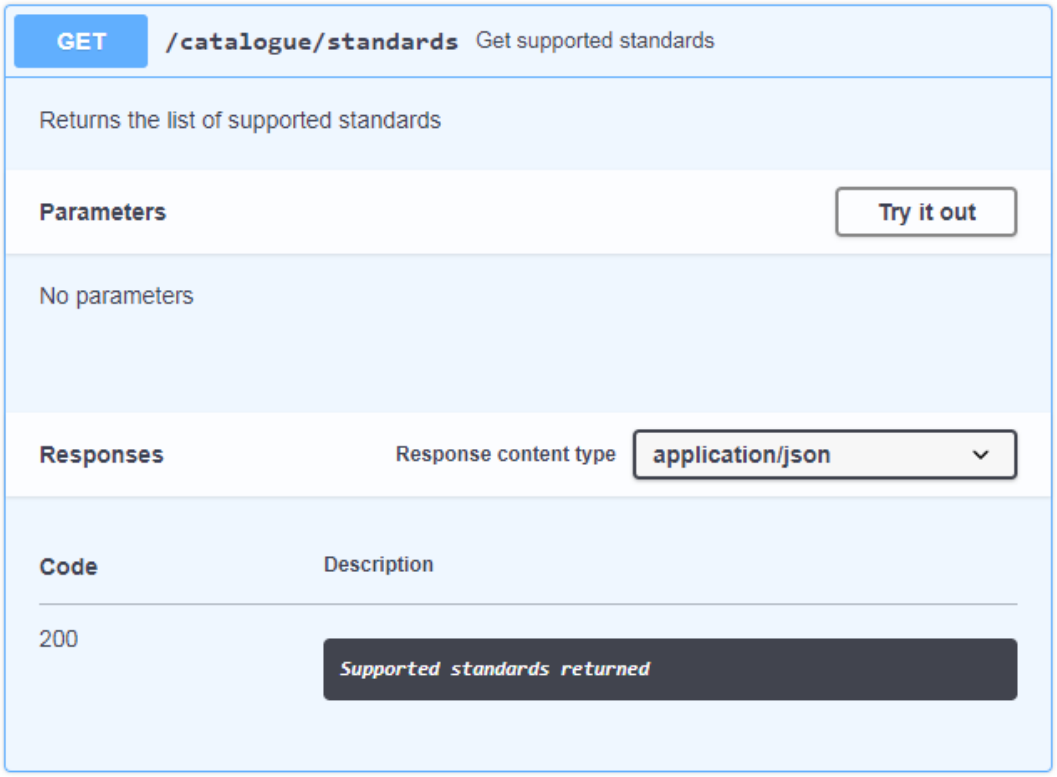

Figure 12: list supported standards API

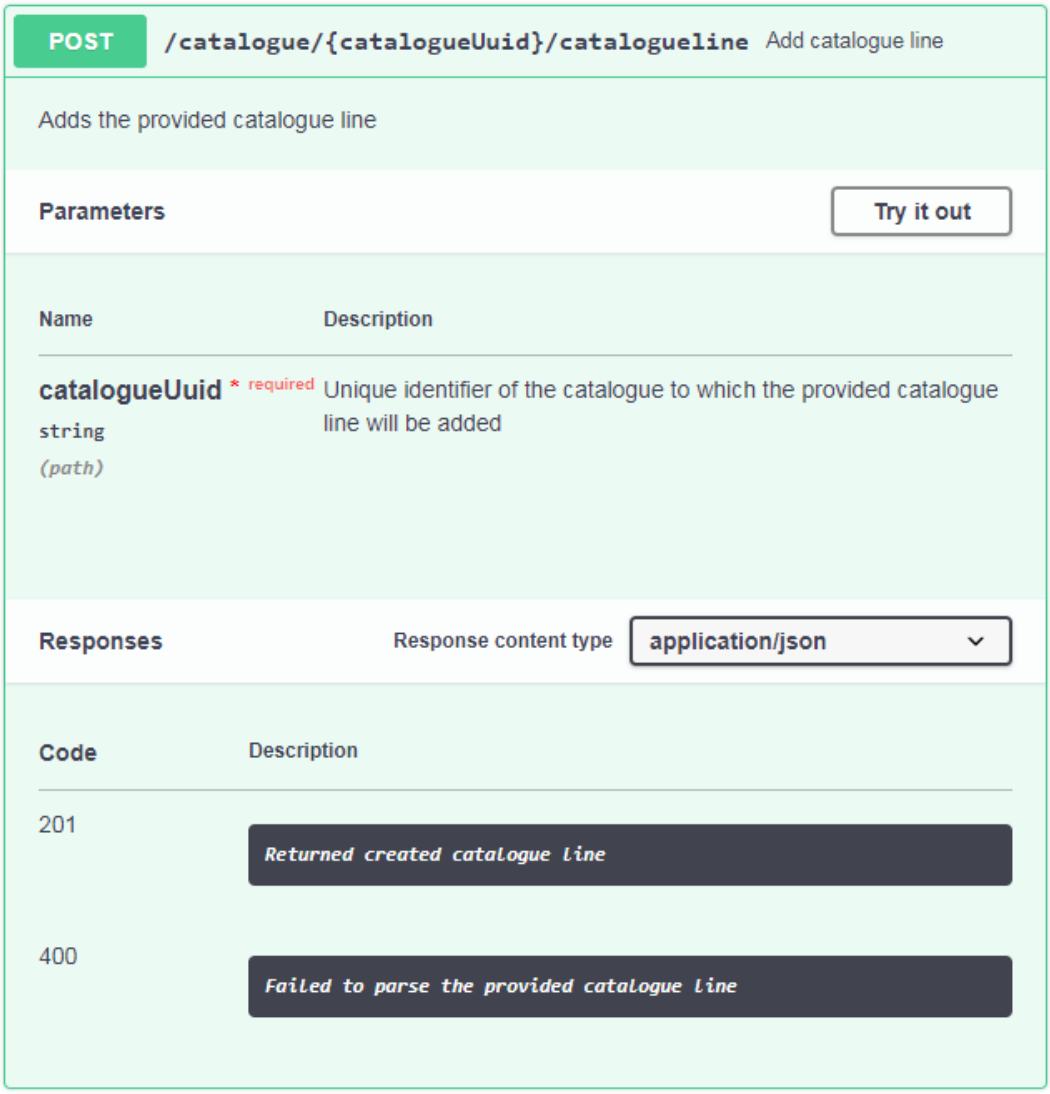

Figure 13: Add a catalogue line API

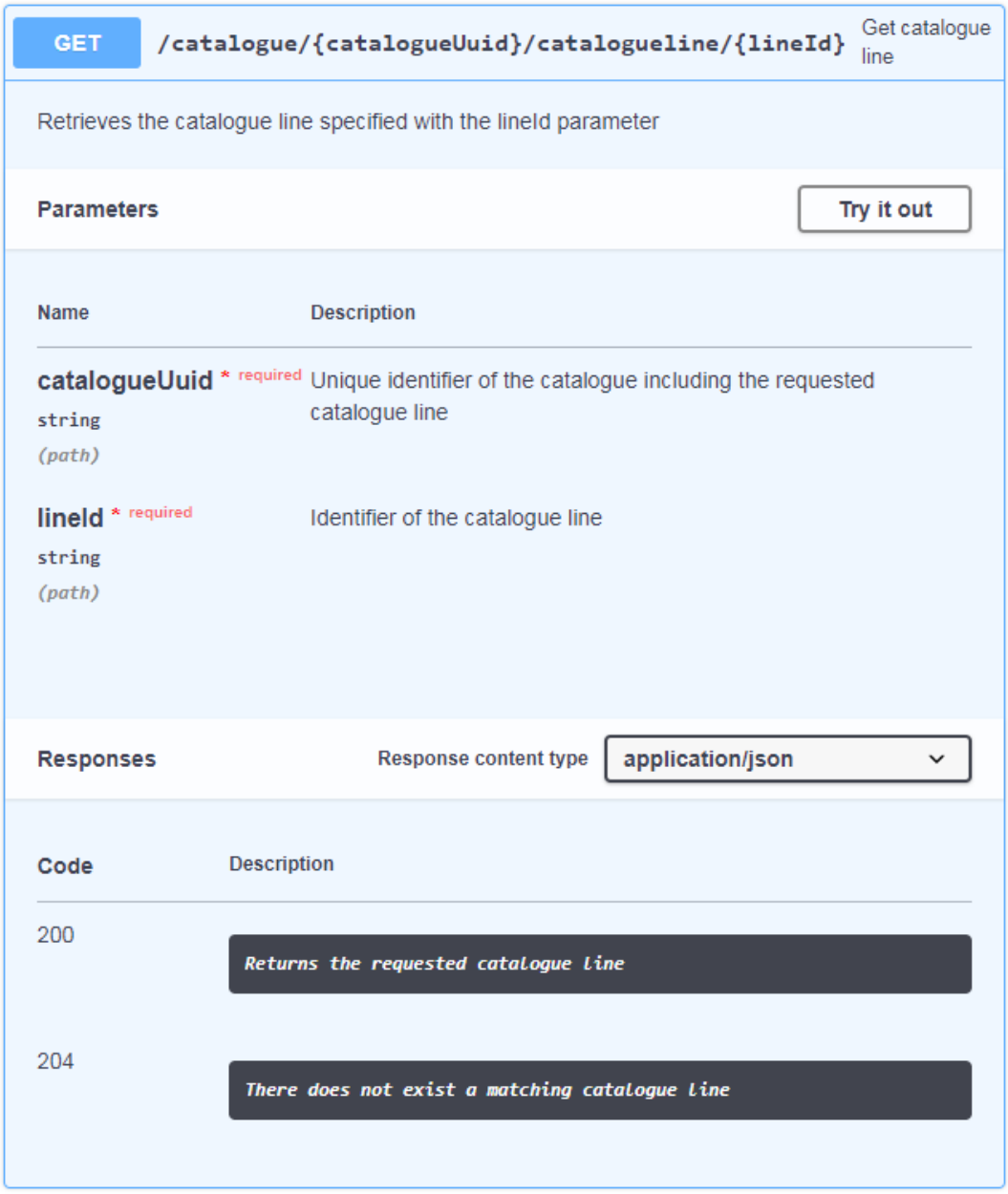

Figure 14: Retrieve a catalogue line API

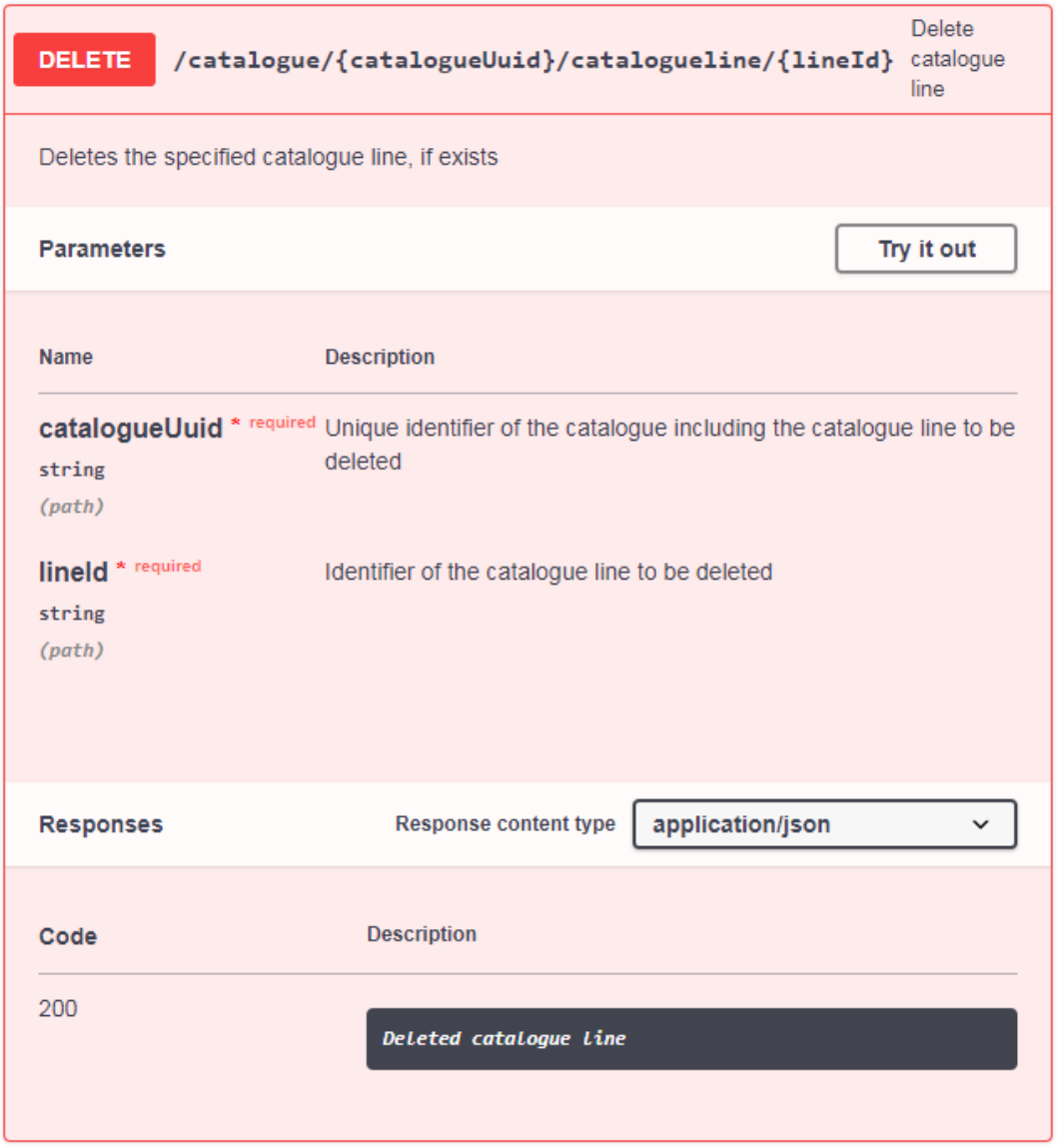

Figure 15: Delete catalogue line API

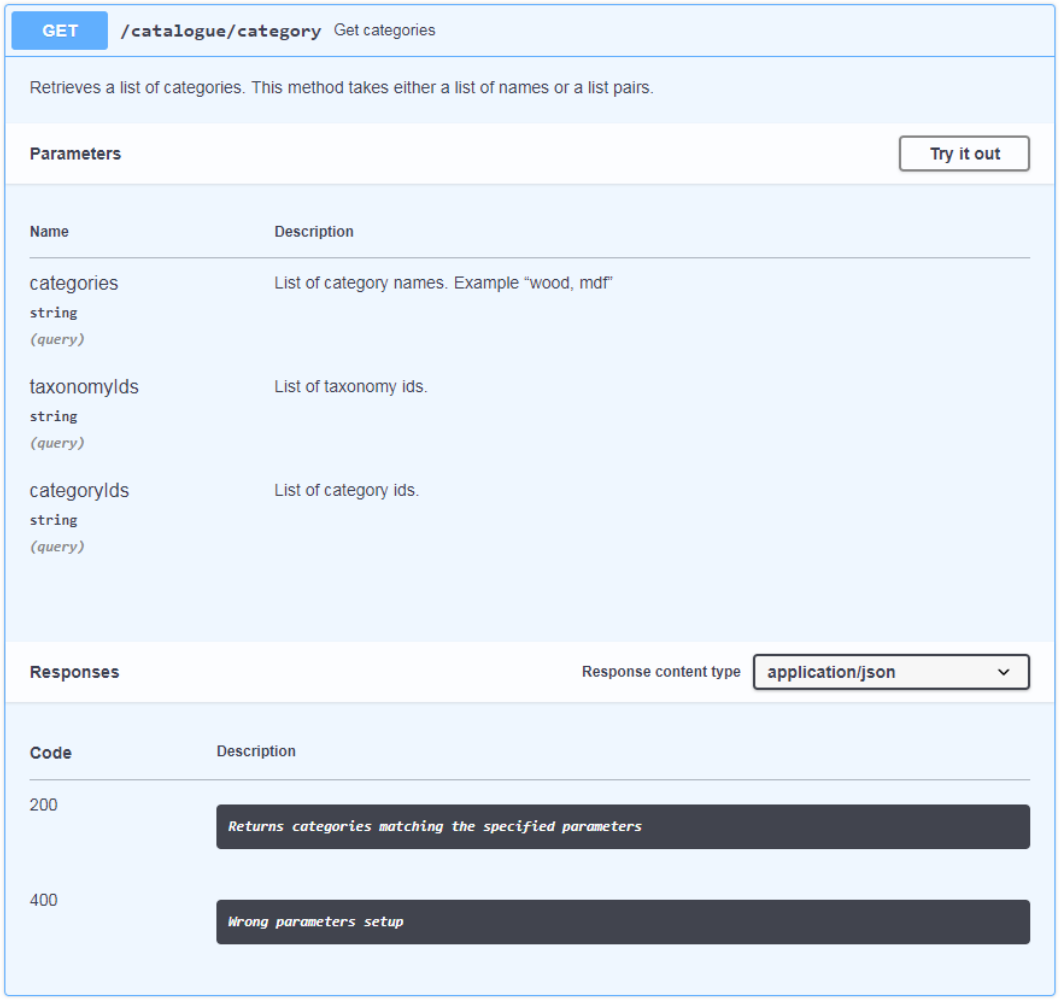

Figure 16: Retrieve a list of categories API

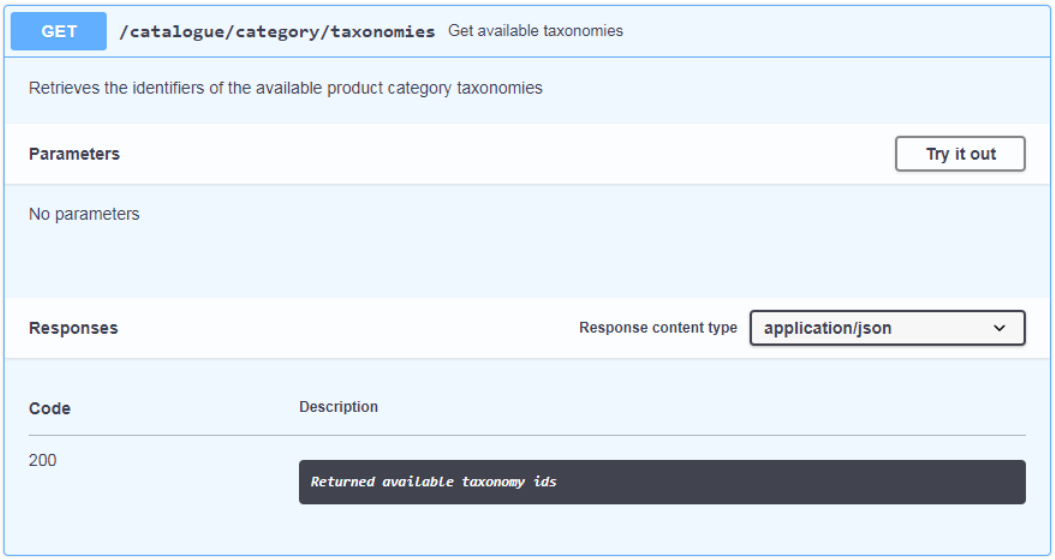

Figure 17: get taxonomies API

Service provided by 3rd party tools

The services described above are provided by the NIMBLE catalogue service itself. In addition to those, there exist REST services provided by the open source tools that NIMBLE uses for catalogue ingestion. There are two such tools namely Apache Marmotta and Apache Solr. Although these tools provide both update and retrieval services, users are only supposed to use retrieval services within the NIMBLE context.

Marmota is a triple store maintaining the data in RDF triples. Therefore, it provides execution of SPARQL queries over the stored data.

Solr is a free-text index engine maintaining the indexed data as key-value pairs. Based on this capability, it enables performing faceted search over the indexed data[6]. Furthermore, users can make use of other advanced search capabilities of Solr based on their needs[7].

5.2.3 Business Processes API

Lifecycle API

The Lifecycle API enables developer to manage the lifecycle of defined business processes on the NIMBLE platform. It provides a set of capabilities for each stage of the lifecycle: editing, deploying, or removing. The interface accepts the necessary data to describe a business process that can be deployed by the platform.

The Deployment API enables the developer to deploy previously drafted business process template instances to NIMBLE. This action represents a run-time entity created out of a process template.

Table 3: Summary of Business process API methods

Operation	URI	Action

[6] https://lucene.apache.org/solr/guide/6_6/faceting.html
[7] https://lucene.apache.org/solr/guide/6_6/searching.html

Retrieve	GET /content/ processID	Retrieve a business process definition
Create	POST /content	Add a business process
Update	PUT /content	Update a business process definition
Delete	Delete /content/ processID	Delete a business process
Create	POST /preference/partnerID	Add a new partner business process preference
Update	PUT /preference/partnerID	Update a business process preference
Retrieve	GET /preference/partnerID	Retrieve a business process preference
Delete	DELETE /preference/partnerID	Delete a business process preference
Create	POST /application	Add a new partner business process applications preference
Update	PUT /application	Update a business process application preference
Retrieve	GET /application	Retrieve a business process application preference
Delete	DELETE /application	Delete a business process application preference
Initiate	POST / start	Start an instance of a business process
Resume	POST / continue	Resume an instance of a business process
Create	POST /document	Add a business process document
Update	PUT /document	Update a business process document
Delete	DELETE /document/documentID	Delete a business process document

Figure 18: Business processes API

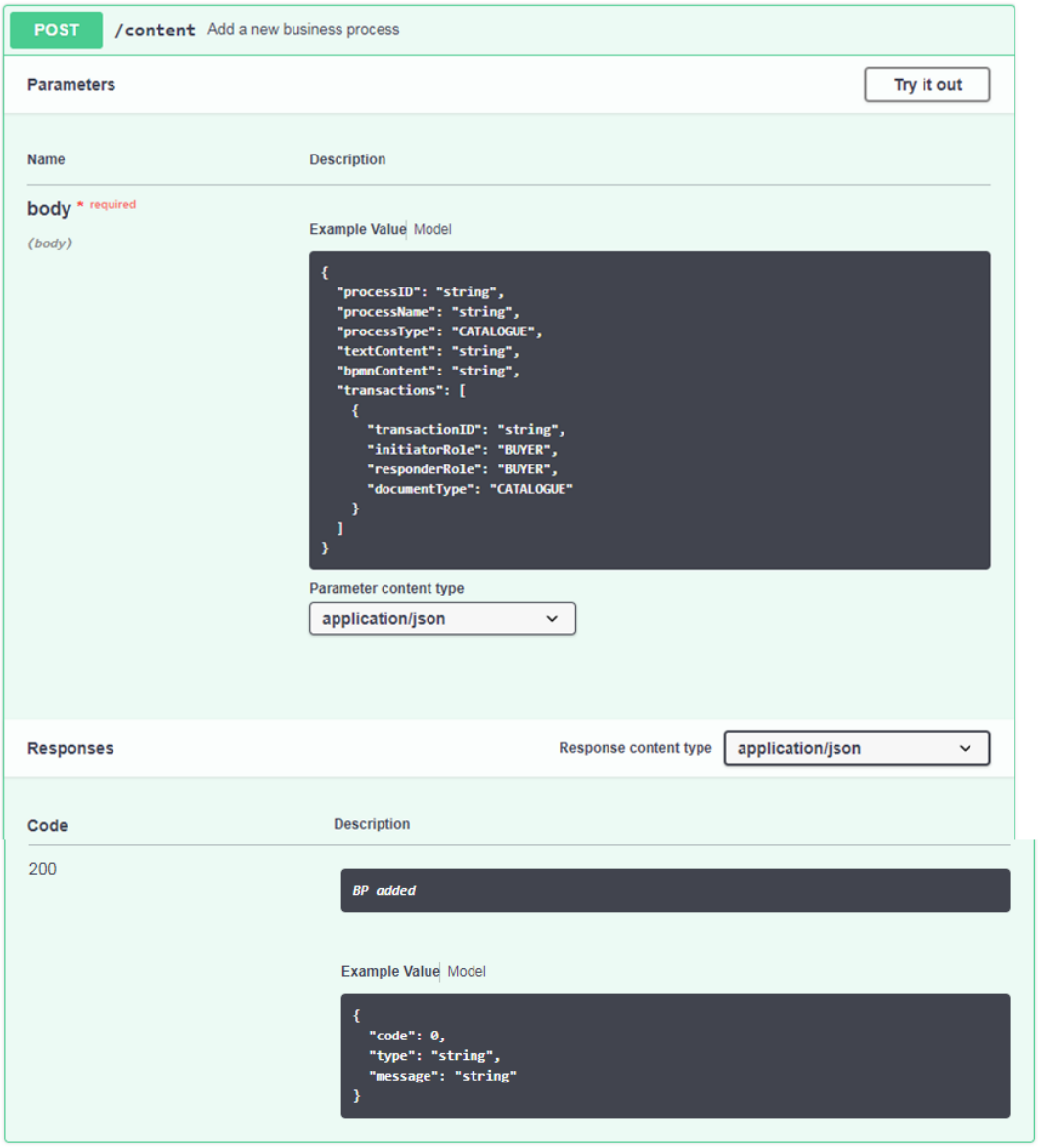

Figure 19: Add business process API

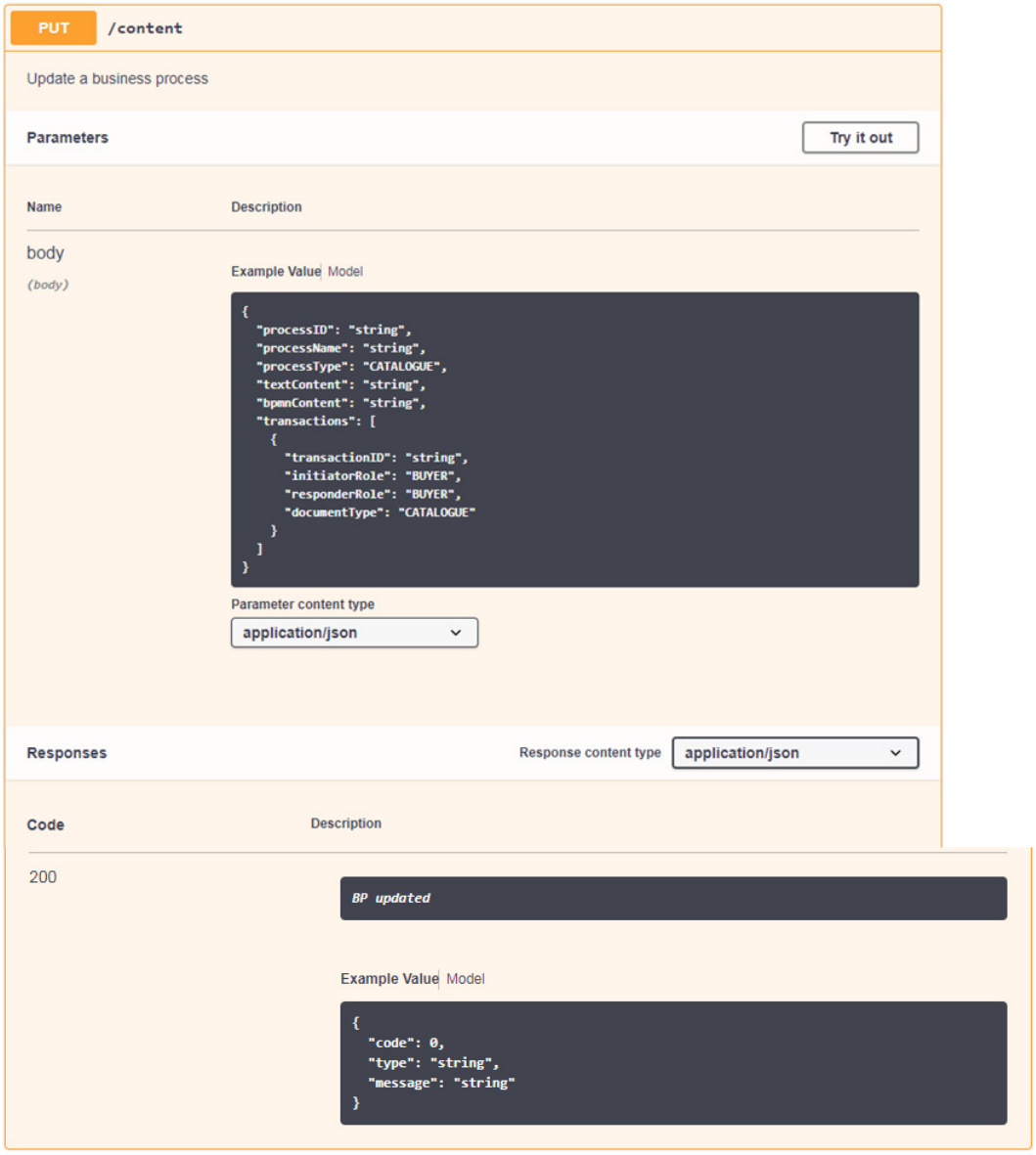

Figure 20: Update a business process API

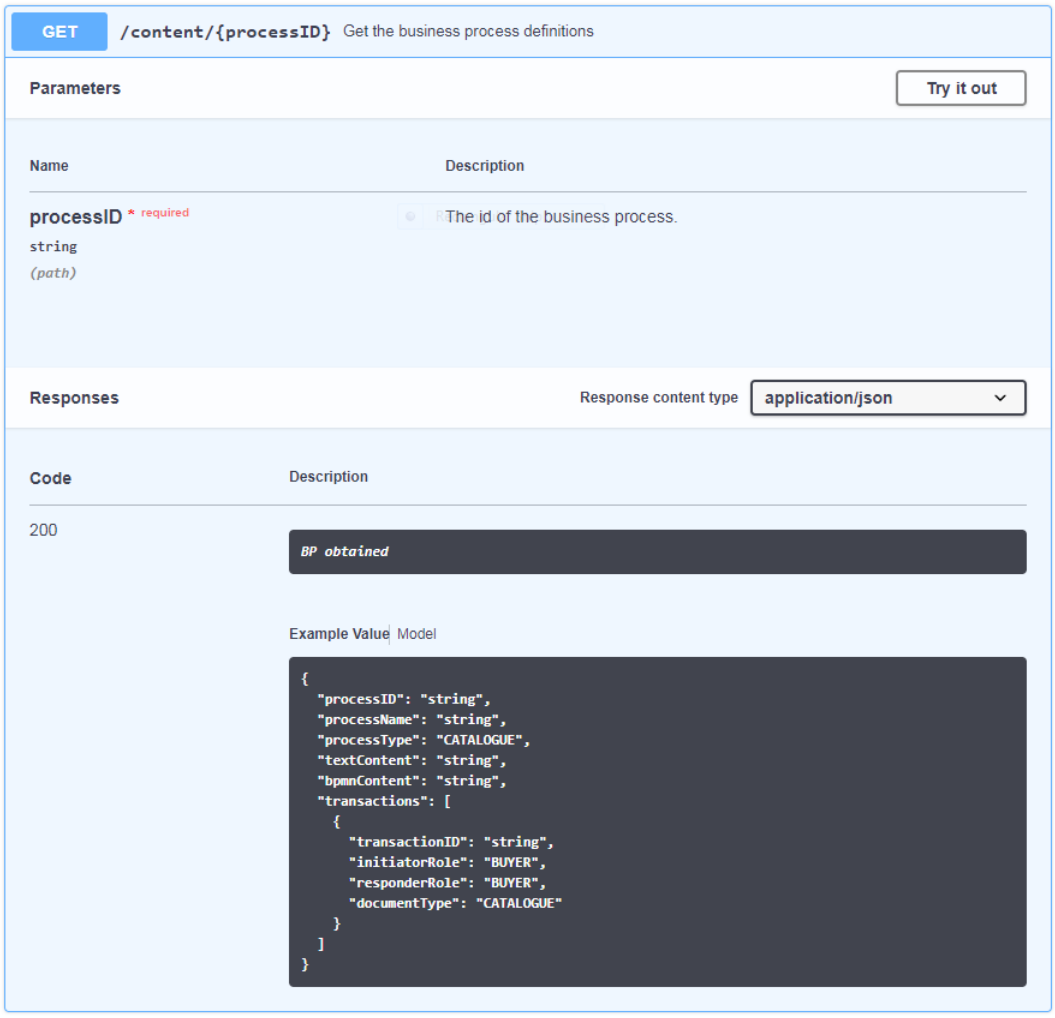

Figure 21: Get business process definitions API

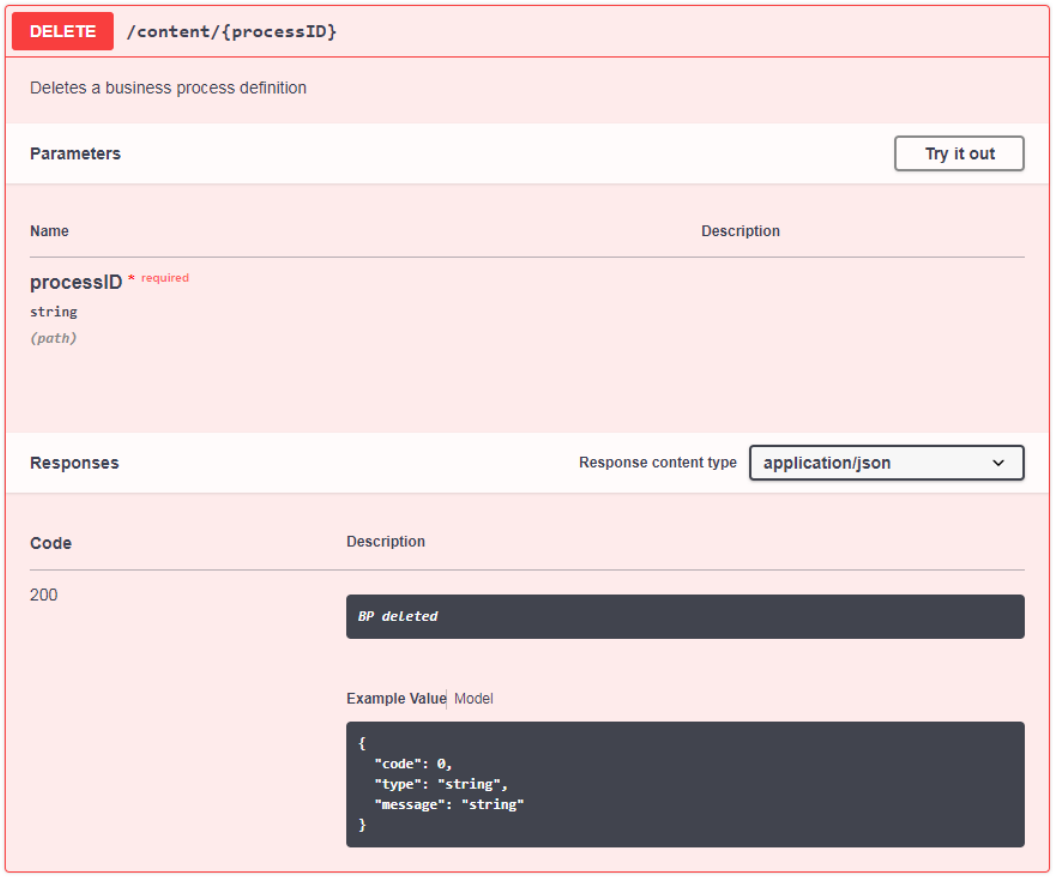

Figure 22: Delete a business process API

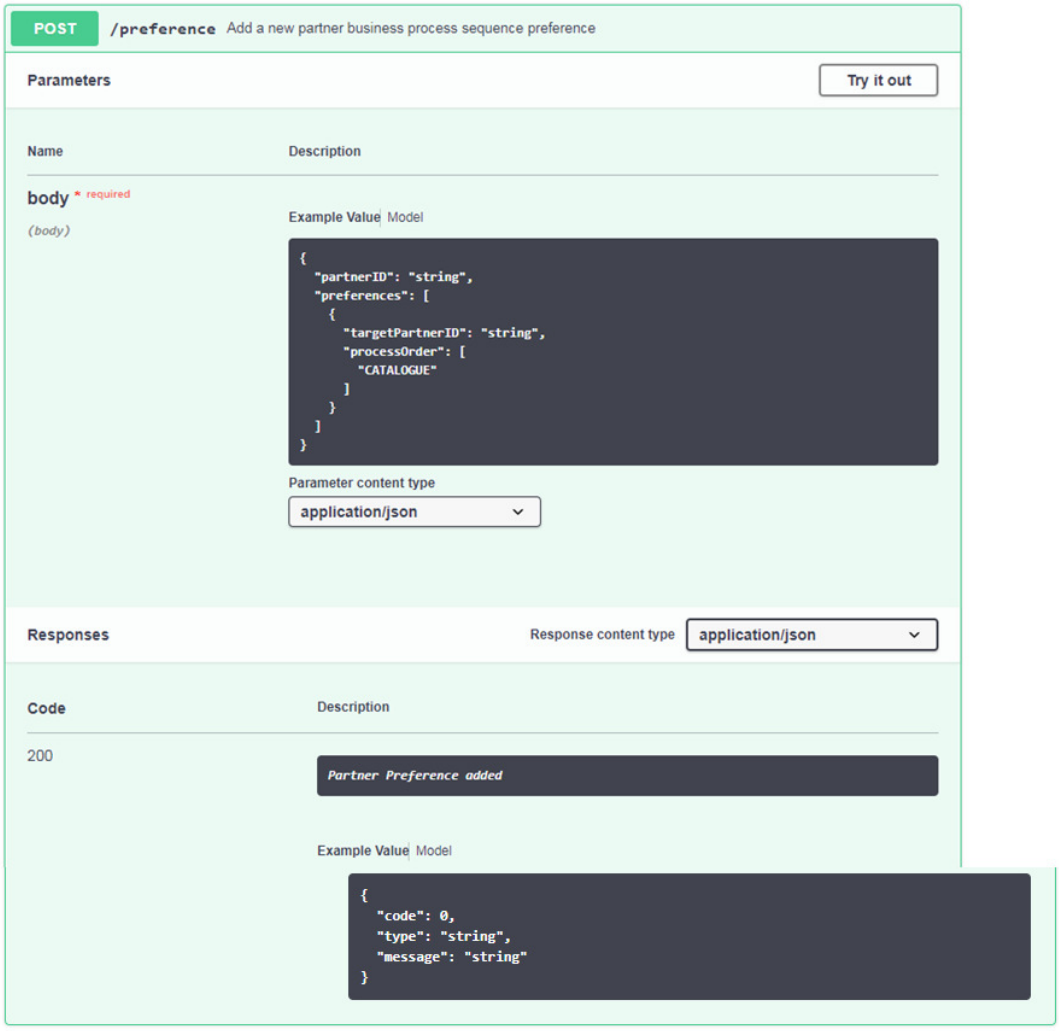

Figure 23: Add preference API

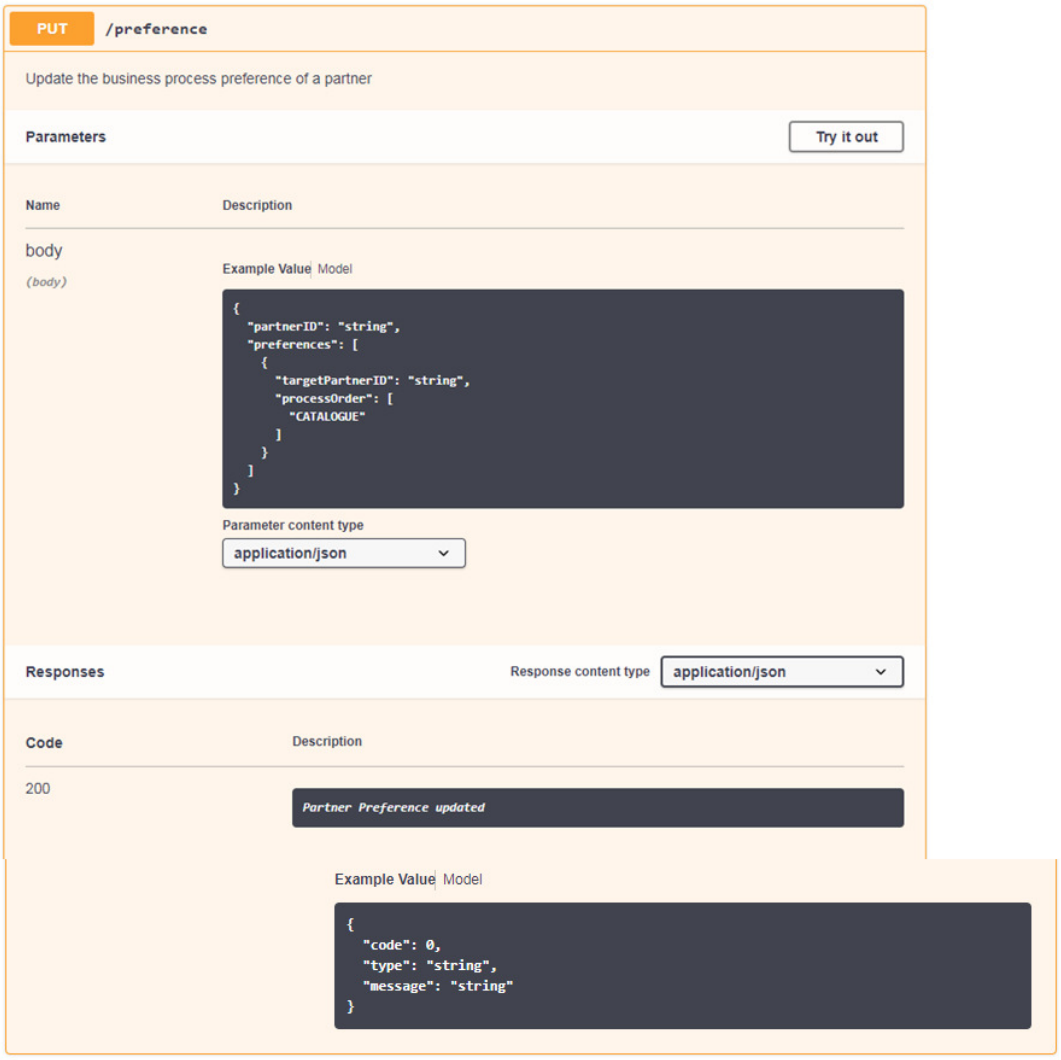

Figure 24: Update preference API

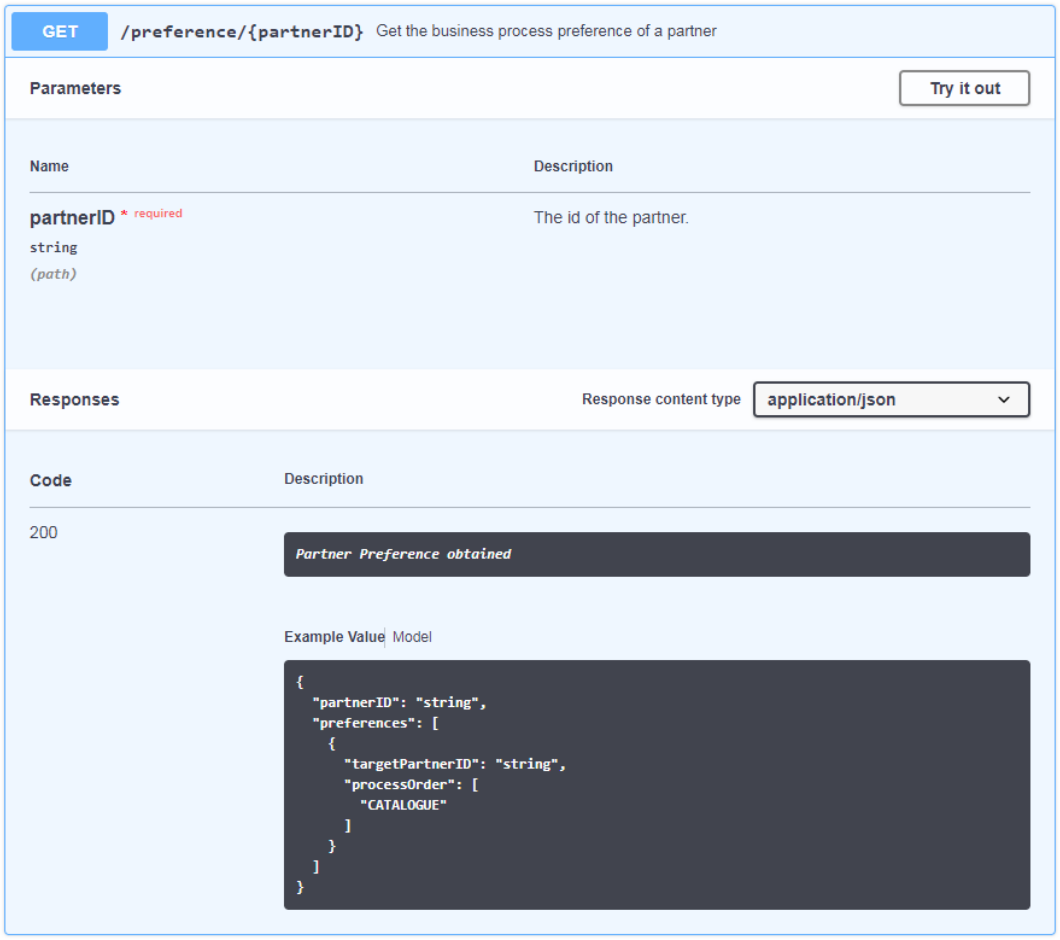

Figure 25: Get preference API

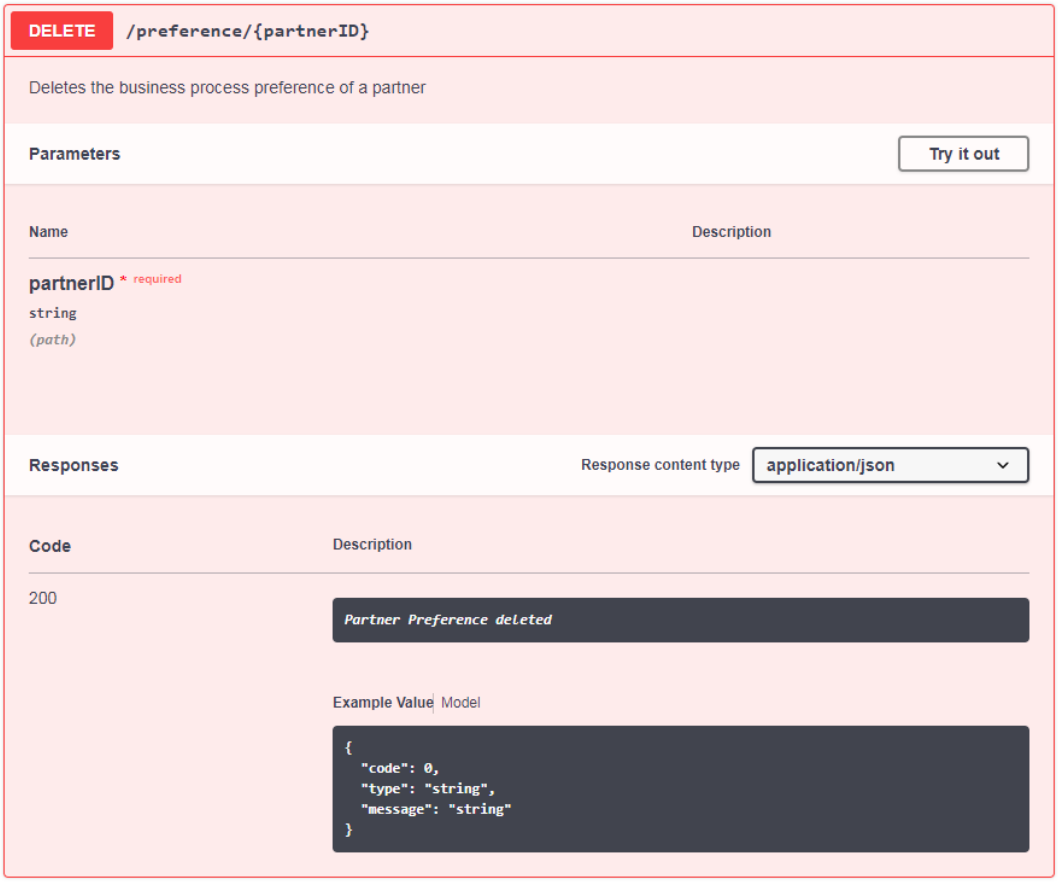

Figure 26: Delete preference API

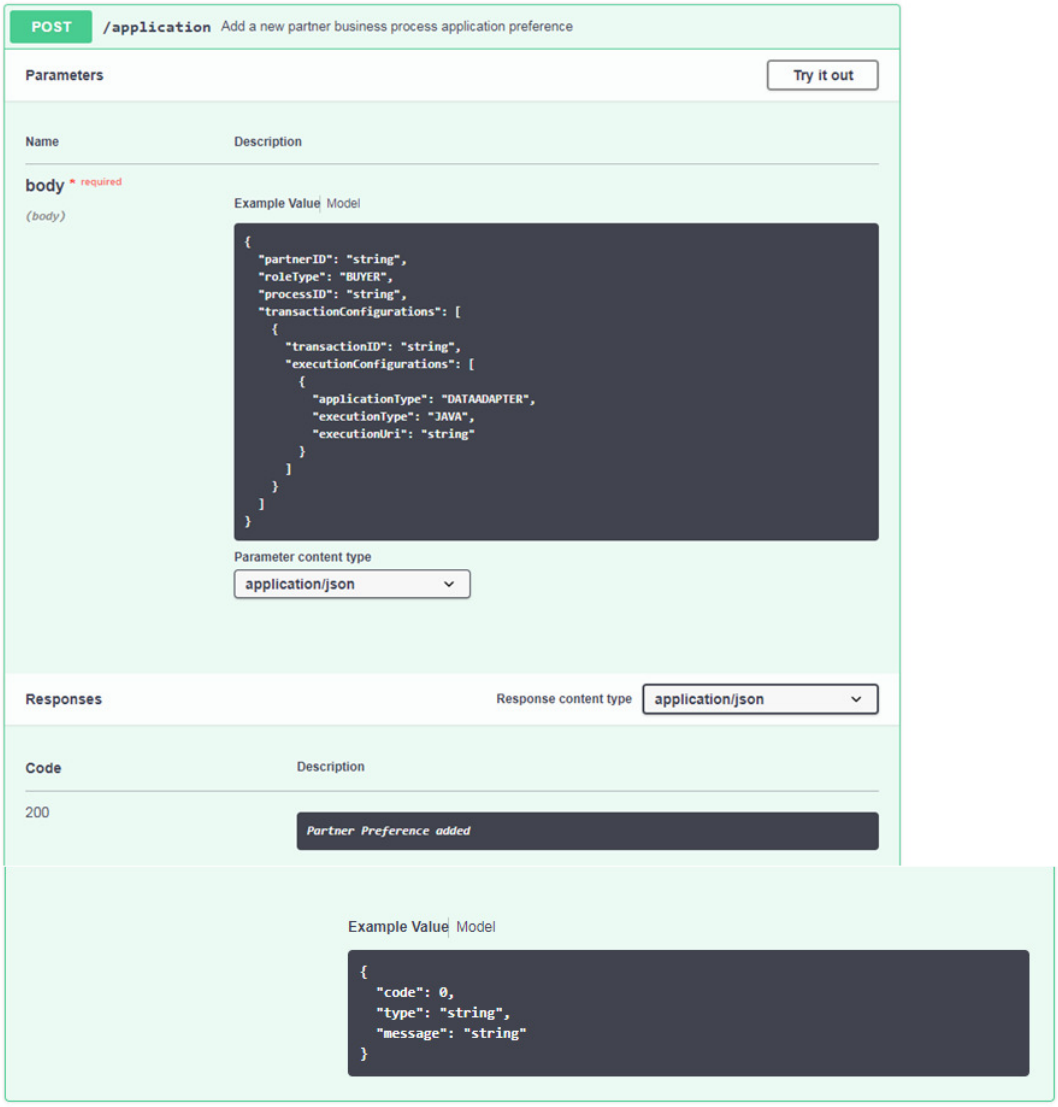

Figure 27: Add application preference API

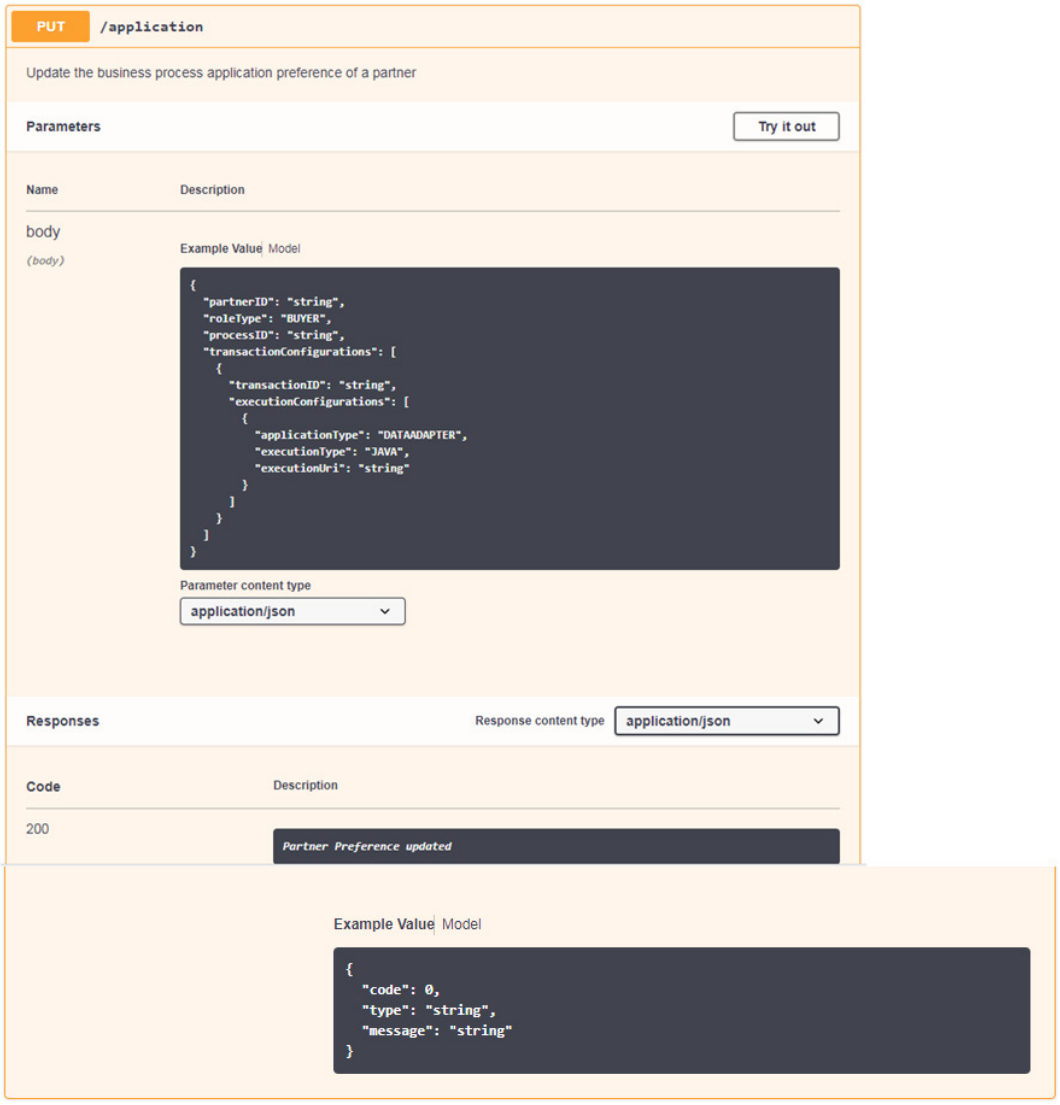

Figure 28: update application preference

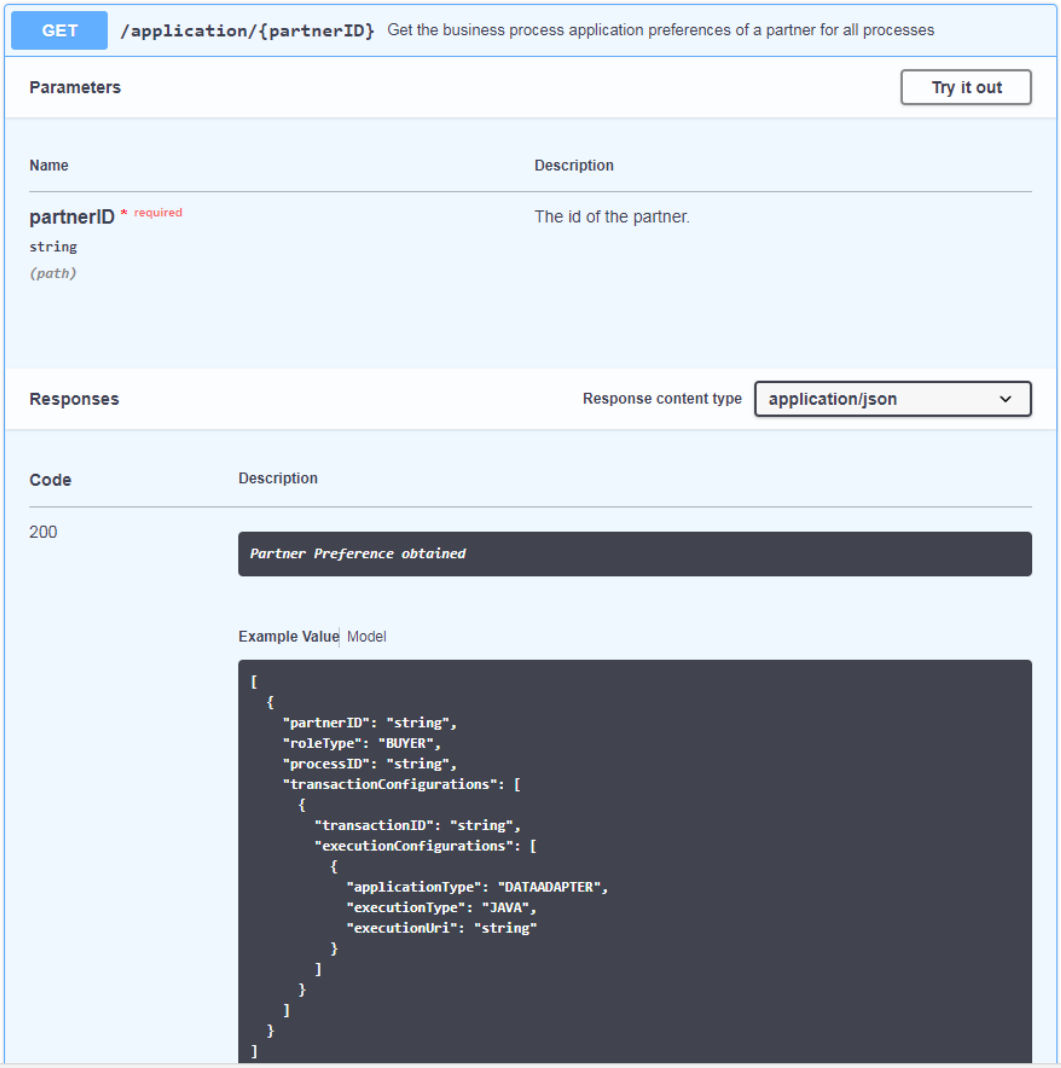

Figure 29: get application preference

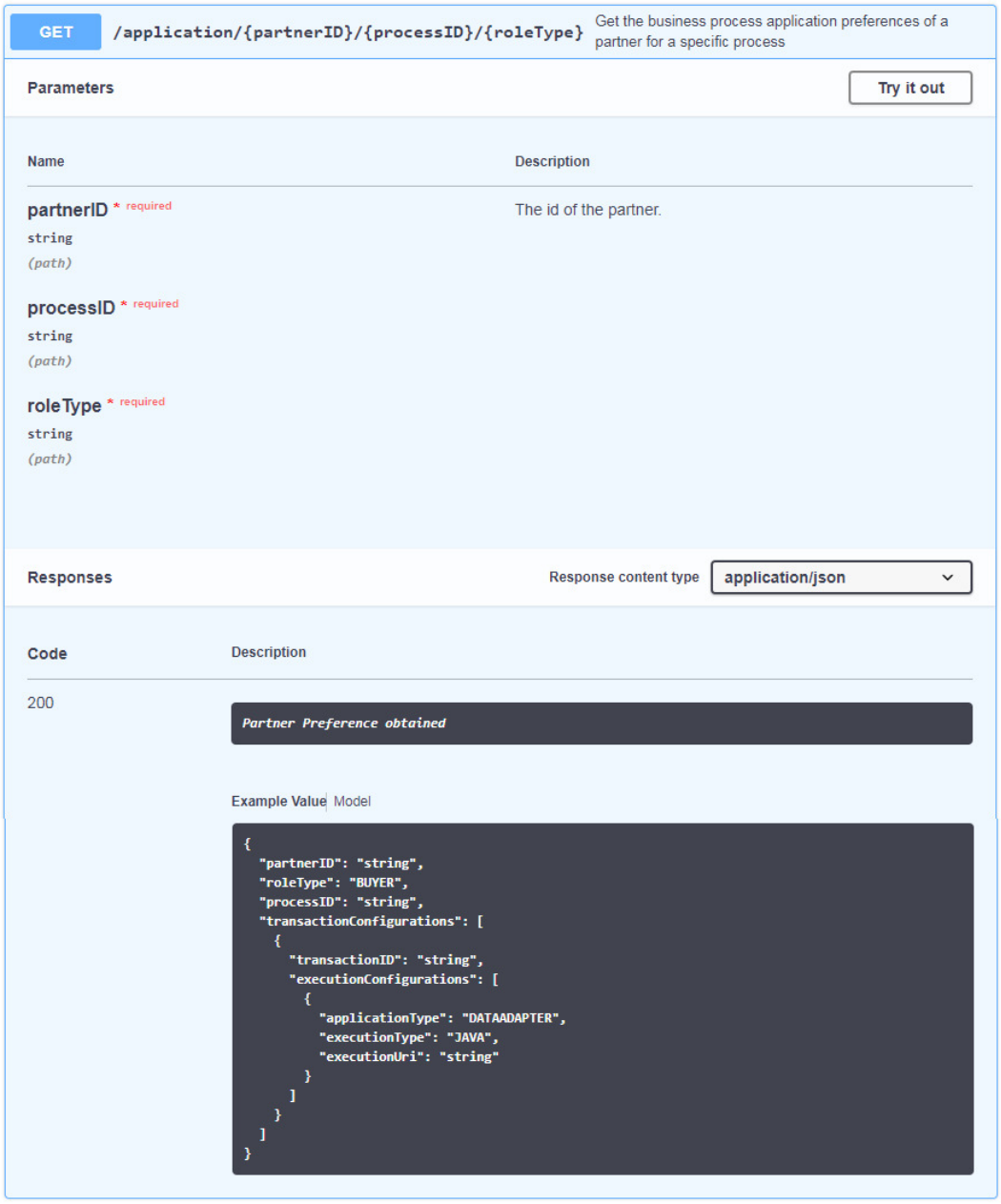

Figure 30: Get preference per process API

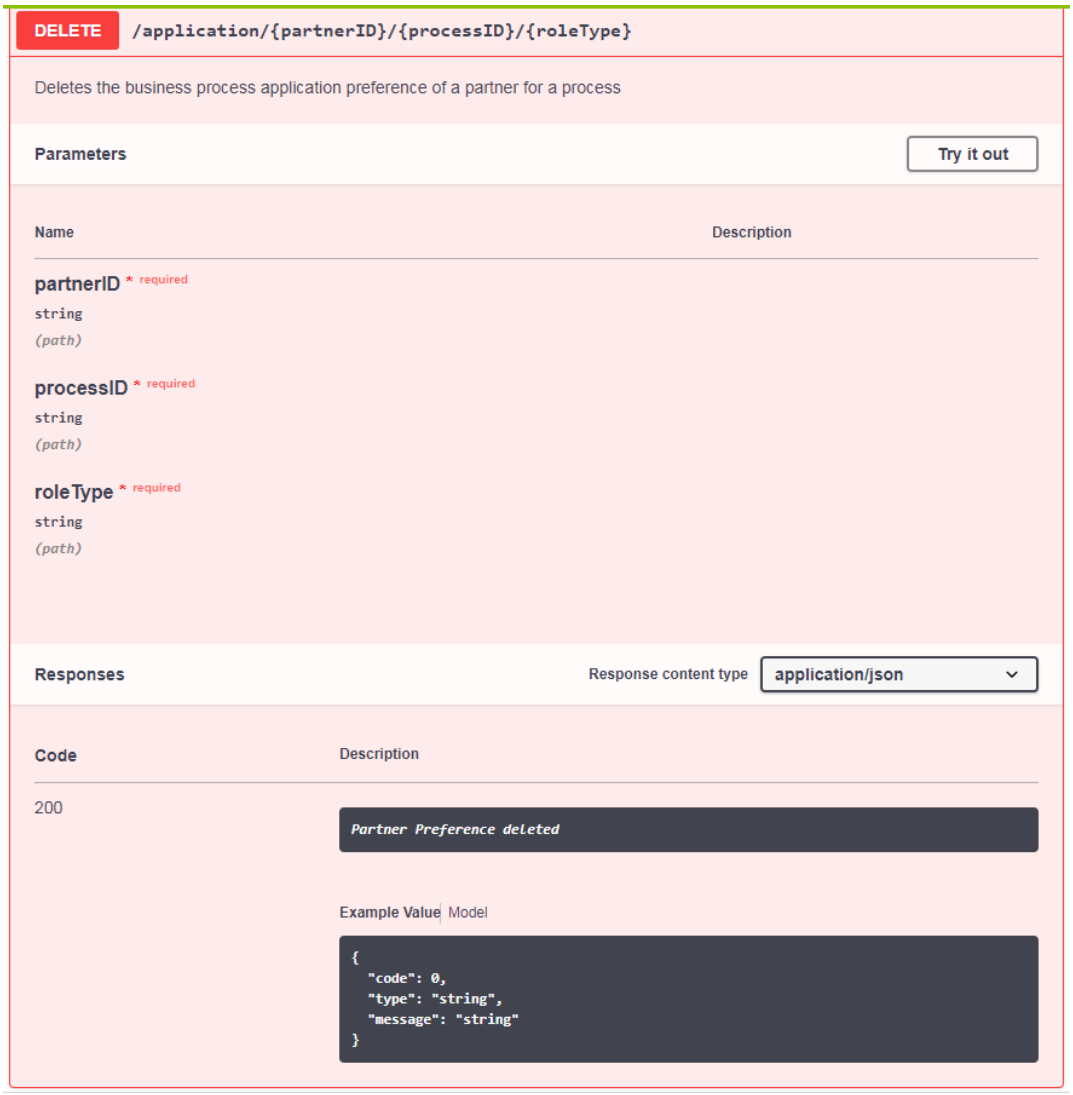

Figure 31: delete preference per process API

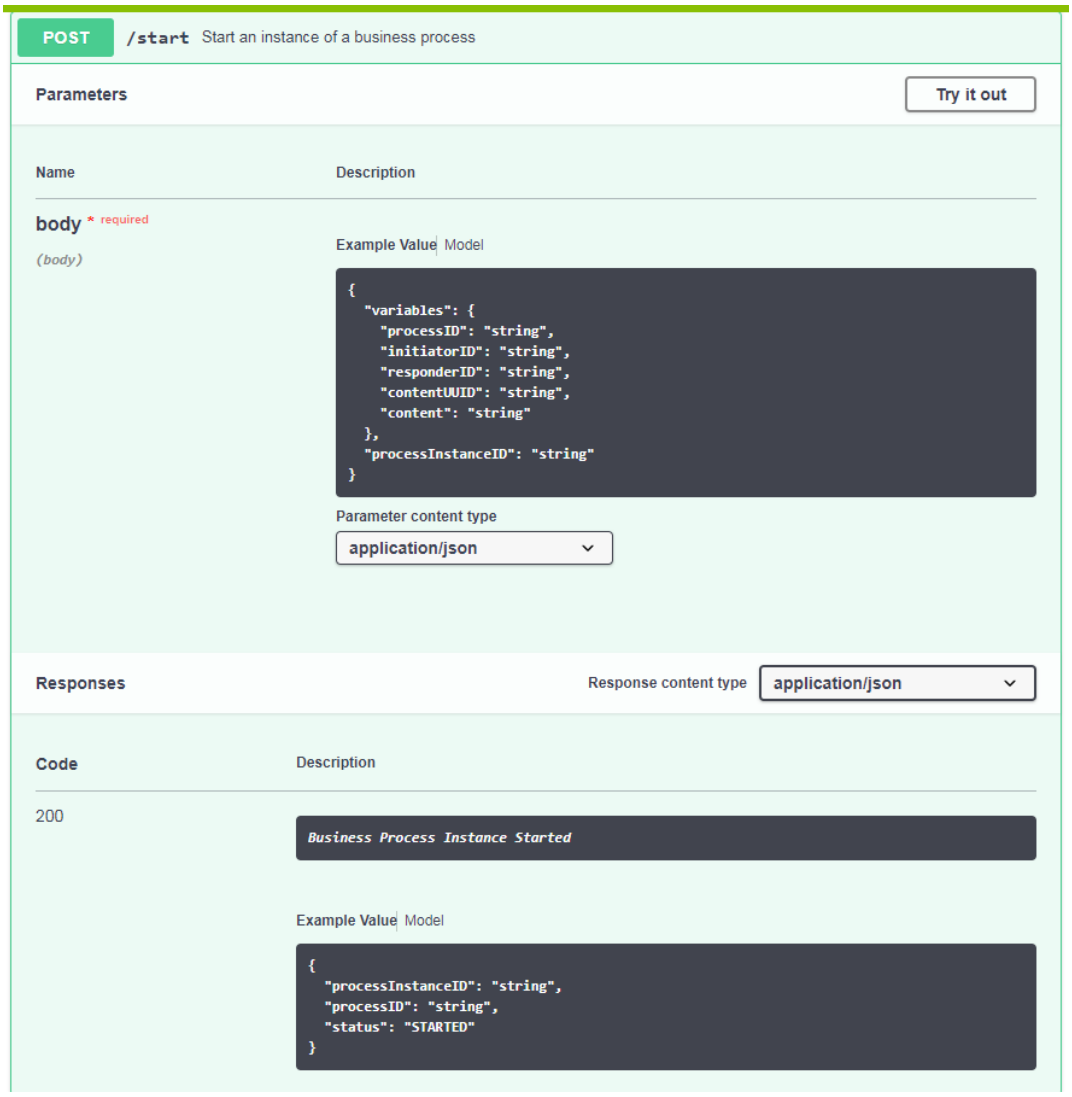

Figure 32: Start a business process API

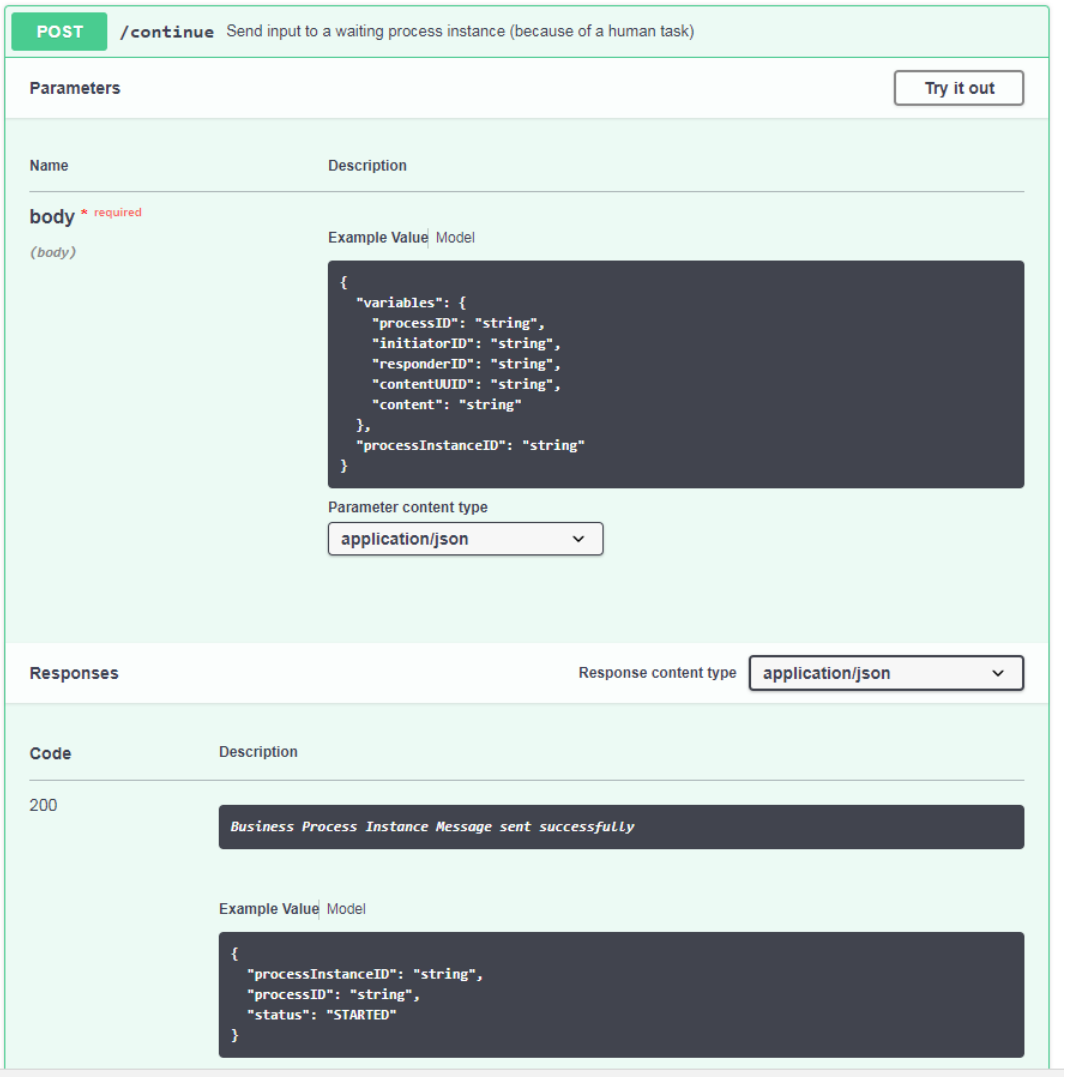

Figure 33: Advance a stage API

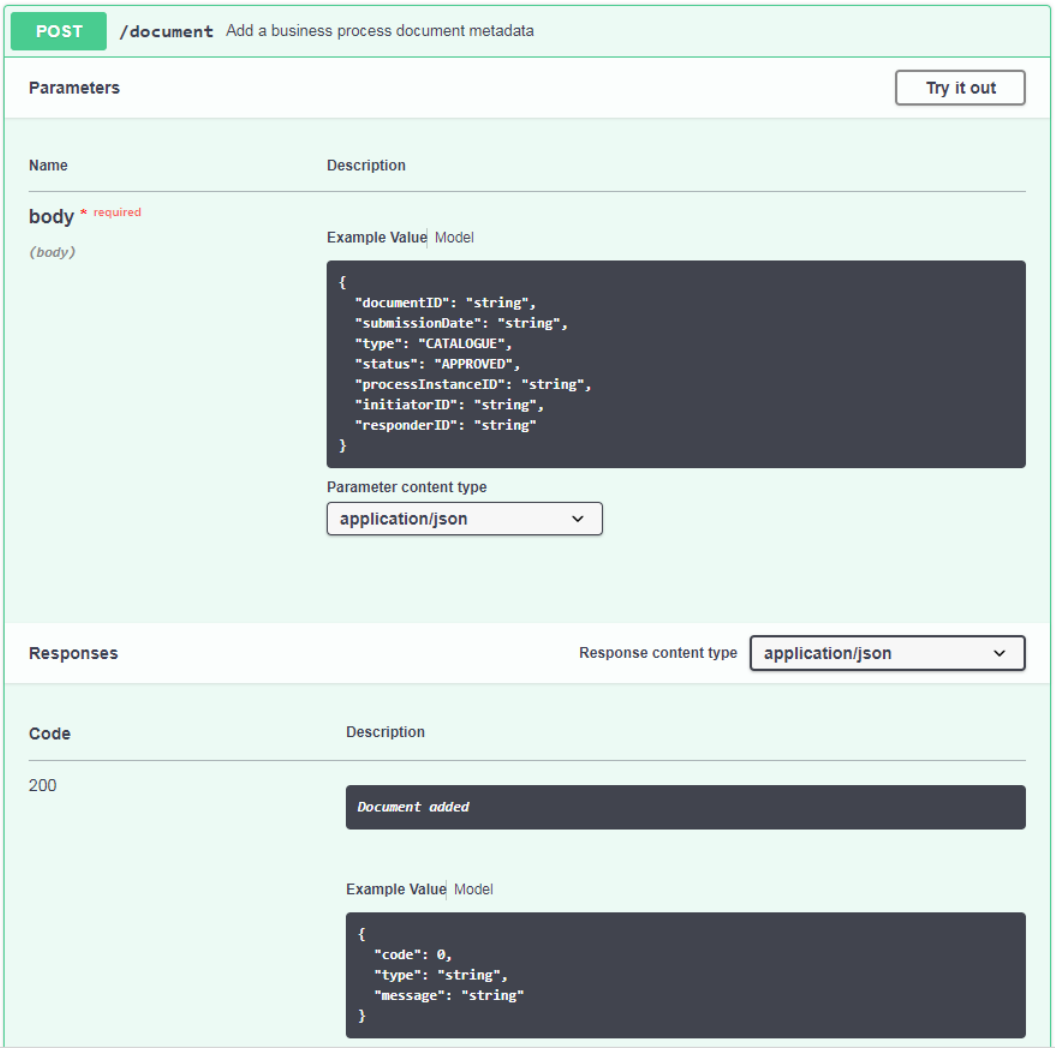

Figure 34: add metadata API

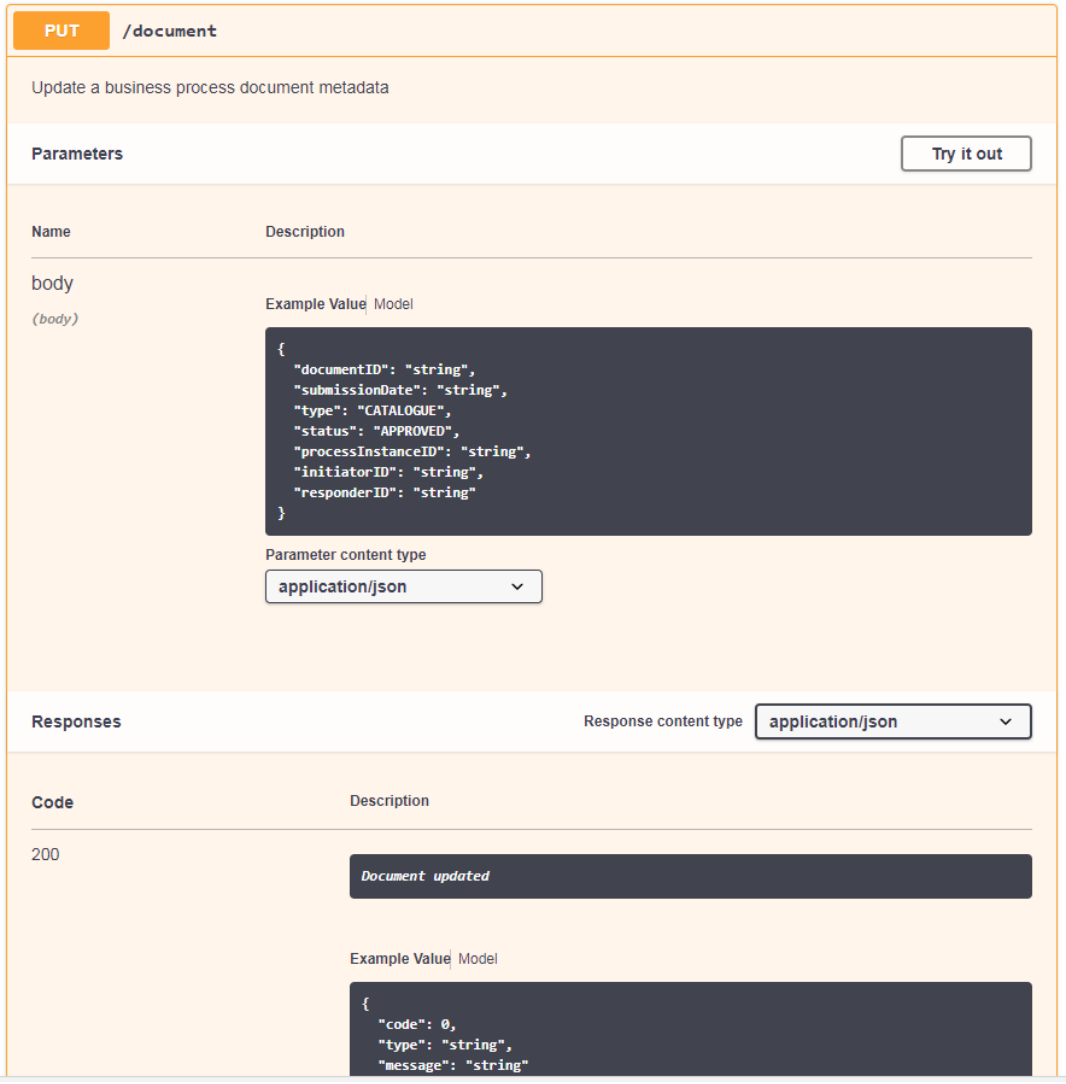

Figure 35: update metadata API

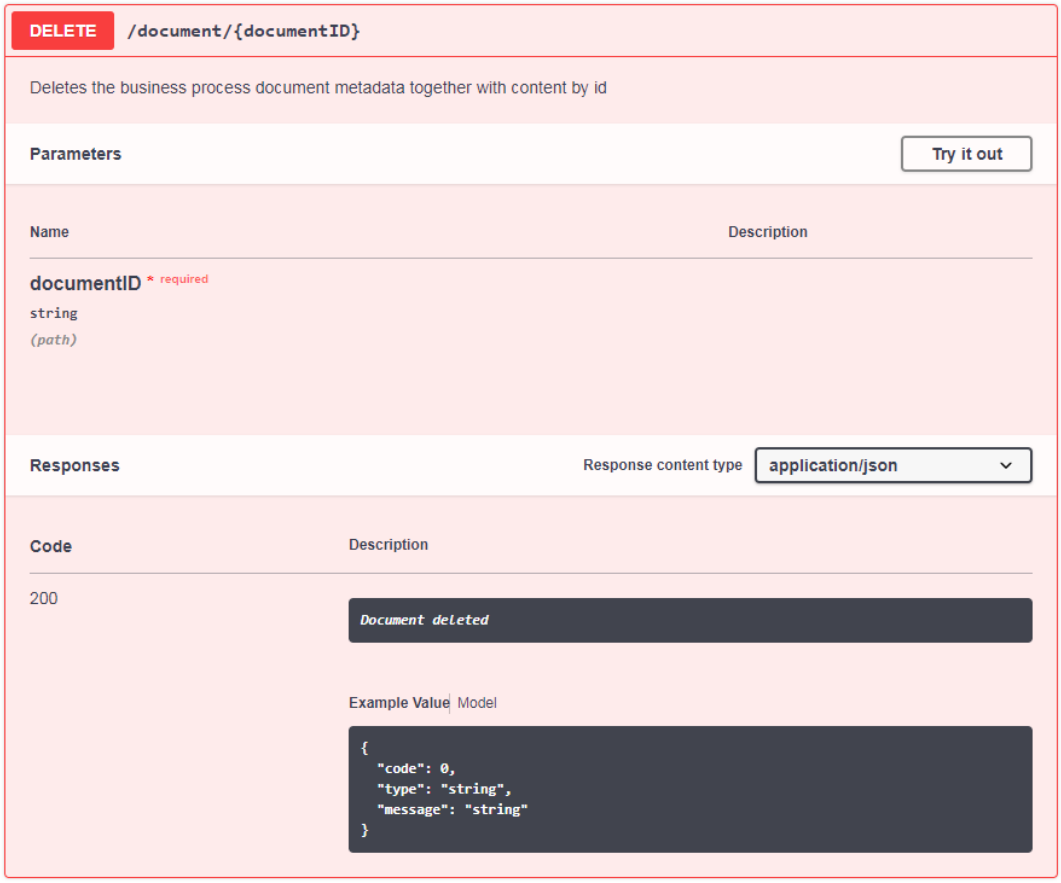

Figure 36: delete metadata API

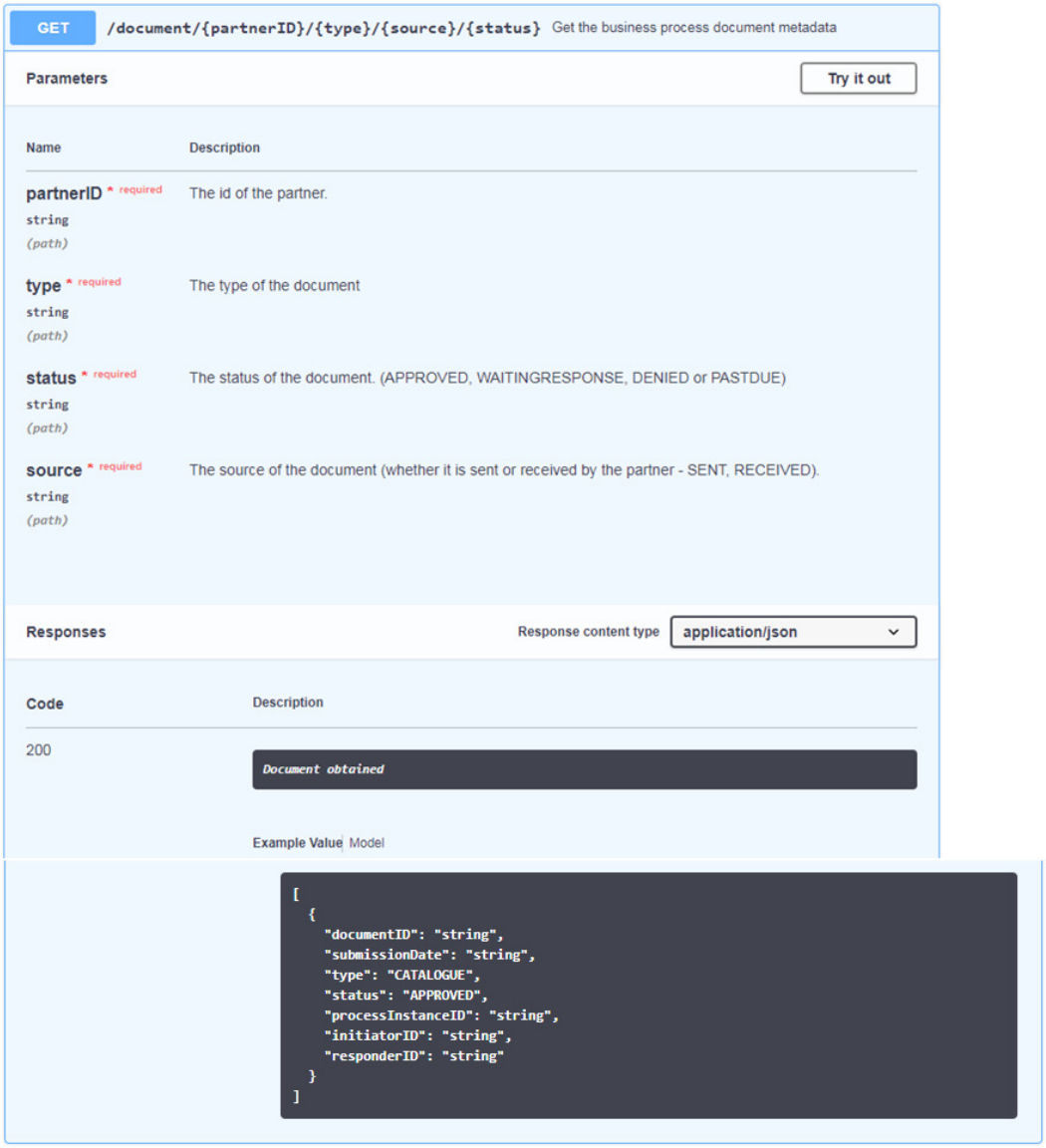

Figure 37: Get metadata API

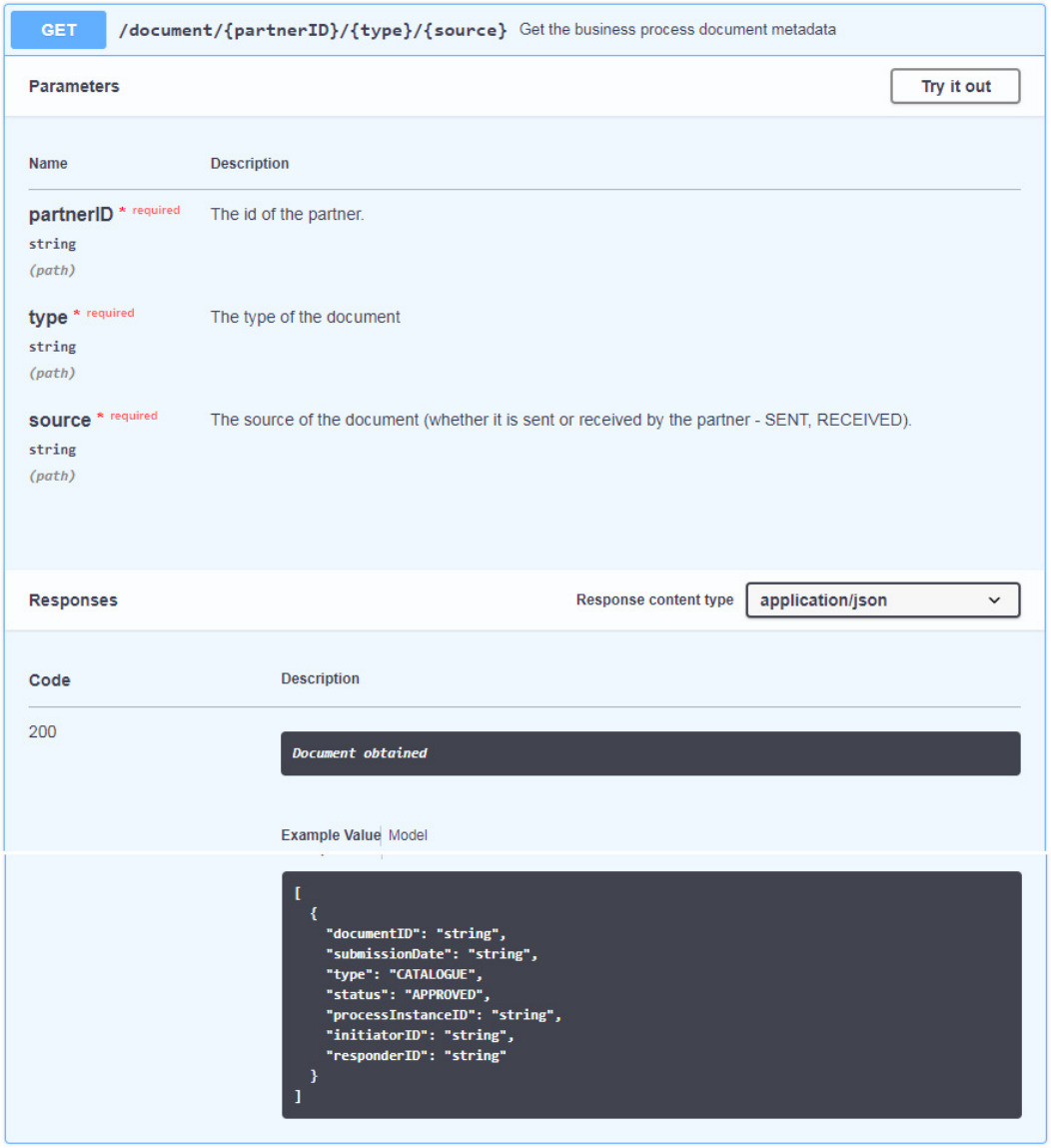

Figure 38: Get metadata API

5.2.4 Message exchange API

The Message exchange API provides developer access to data produced or stored on the platform. It involves the establishment of communication channels between different entities.

Registering interest in a topic

[REST_endpoint]/consumer/register?topic=[topic_name]

Messages will be made available to the interested subscriber via a provided call-back which will be invoked by the underlying messaging system.

Sending a message in a topic

[REST_endpoint]/producer/send?topic=[topic_name]&message=[my_message]

5.2.5 What's next

The detailed section above consists mostly of the API corresponding to capabilities of the first version of the platform consisting of the core services. As the platform matures and more capabilities are added it is to be expected that new families of APIs will be made available via the platform. The main categories which are expected at the moment to be further specified in the following iterations of the platform are briefly mentioned below.

Security API

A security related layer is embedded within the API processing to verify adequate access to NIMBLE entities and resources via the API. This component ensures that resources can only be accessed by users who have been granted with the corresponding privileges.

This component is responsible for authenticating the user behind the incoming request and provide credentials to be used within the platform.

Data management and sharing API

Providing the capability for different entities to share exclusively portions of their internal data, governed by a business process established between the two (or more) entities. In addition, data processing and storage directives as data flows into the system should be possible to define via the API.

Platform monitoring

The platform shall offer API for describing and defining monitoring requests, and the manner in which the information will be made available.

Regulation and legislation repository API

The platform shall offer API for searching for appropriate legislation and regulations. There shall be a capability to search by a variety of dimensions, such as title, description, code ,or keywords. This capability shall be realized via the development of RESTful API (GET) to access the regulations and legislation system database.

The search for documents about norms and laws to help SMEs adapt their products and services to the market conditions is a requirement to be met by the platform. This capability becomes even more important for companies at the stage in which they aim to target new markets. The regulation and legislation API is mainly focused on obtaining information via a GET action to search documents not only about regulation and legislation but also patents and sectorial reports stored in a specific document repository managed by AIDIMME. The API will enable the search based on specific document attributes according to the repository definition in order to retrieve more accurate results.

Furthermore, a frontend to consume this API will be developed to demonstrate the integration of this service inside NIMBLE.

6 Summary

This deliverable focuses on presenting the API of the different components comprising the NIMBLE B2B platform. Different user roles and the respective required capabilities were described.

Platform users will be able to discover capabilities and services of need provided by other platform entities. The complementary functionality is for entities to be able to publish capabilities they wish to be found by other entities. Building on these capabilities, entities on both sides of the process are able to participate in a business process. Finally, a federation enabling API will provide the capability to link between different instances of the NIMBLE platform, mostly for employing discovery and business process execution services across instances.

Through the API, developers should be able to manage the relations between their own applications and NIMBLE provided capabilities. The API in turn serves as a mediating layer between the external word and the internal components of the NIMBLE platform.