
Read the Docs Template Documentation

Release 1.0

Read the Docs

May 13, 2022

1	Collection API (Version 1.0)	3
1.1	Authors: Sabrina Chelbi, Thomas Jejkal	3
1.1.1	1. What is Collection API?	3
1.1.2	2. Installation	4
1.1.3	3. Collection API Implementation	4
1.1.4	4. How can the Collection API be used?	5

This documentation describes the single components of the KIT Data Manager Research Data Repository Platform. KIT Data Manager (KIT DM) is a domain-agnostic platform for building up research data repositories for managing research data according to the FAIR principles. Therefor, globally agreed recommendations and standards, e.g. outcomes of the Research Data Alliance, are implemented and integrated into the platform. Due to the different requirements for a domain-agnostic platform, all components are designed as independent as possible from each other. This allows to use only components required by a particular use case. This reduced complexity and offers a high level of flexibility for current and future challenges in the field of research data management.

1.1 Authors: Sabrina Chelbi, Thomas Jejkal

The Collection API is proposed by the RDA Recommendation on Research Data Collections doi: [10.15497/RDA00022](https://doi.org/10.15497/RDA00022). It can be used for building collections of digital objects independent from any repository in order to facilitate data interoperability, reuse and make collections actionable to be able to cope with ever increasing amounts and volumes of data.

1.1.1 1. What is Collection API?

A collection is a digital object which bears a unique identifier and binds a finite number of digital objects together, which have common concerns. The collection API is an interface specification for CRUD (Create, Read, Update and Delete) operations on collections in order to enable client-server interaction with particular observance of persistent identification. It can be used for building collections of digital objects independent from any repository in order to facilitate data interoperability, reuse and make collections actionable to be able to cope with ever increasing amounts and volumes of data. The Collection API has been proposed by [the RDA Recommendation on Research Data Collections](https://rdacollectionswg.github.io/apidocs/#/). The documentation of the different REST APIs is available under <http://rdacollectionswg.github.io/apidocs/#/>. Moreover, a JAVA implementation of the RDA Recommendation is provided by Steinbuch Centre for Computing (SCC) as a Spring Boot-based Microservice with a complete regard to the recommendation. As some aspects of Collection API are not clearly defined, the implementation contains some fixes [FIX], additions [ADD] and restrictions [RES]:

- [FIX] Return type inconsistencies have been fixed, e.g. in `/collections/{id}/members/{mid}`.
- [FIX] Delete operations return status 204 (NO_CONTENT) according to the HTTP specification.
- [FIX] Delete operations are realized idempotent following the HTTP specification. This means, that DELETE can be issued multiple times to a resource and returns HTTP 204 in all cases.
- [FIX] Collection operations allow navigation the same way all other operations do, e.g. via prev and next links.
- [RES] Listing a collection recursively does not consider the sorting of child elements.
- [RES] A recursive listing of a collection will also contain member items of expanded collections.

- [RES] There is currently no build-in PID support. If no PID are provided with a collection or member, a UUID is assigned.
- [ADD] Integrated ETag support in order to avoid concurrent modifications.
- [ADD] Navigation through a result set is realized using default Spring pagination, e.g. supporting page and size query parameters. The cursors (next and prev) of a result set are pointing to the next/prev page link.

1.1.2 2. Installation

For running the collection-api service you may either startup a docker container or you build and run the service from source. The source code is available at <https://github.com/kit-data-manager/collection-api> For more information about compiling and starting from source, please refer to the README located in the source repository. In the following, running the collection-api service using docker is explained.

Prerequisites:

- docker (tested with 19.03.8).

You can create an instance of the collection-api service by running docker:

```
$ docker run -d -p 8080:8080 kitdm/collection-api:latest

Unable to find image 'kitdm/collection-api:latest' locally
latest: Pulling from kitdm/collection-api
3192219afd04: Pull complete
17c160265e75: Pull complete
cc4fe40d0e61: Pull complete
9d647f502a07: Pull complete
[...]
Status: Downloaded newer image for kitdm/collection-api:latest
```

As soon as the microservice is started, you can browse to <http://localhost:8080/swagger-ui.html> in order to use the available REST APIs.

1.1.3 3. Collection API Implementation

3.1 Collection API Architecture

The architecture of the Collection API service is illustrated in the figure below. The bottom component is the Collection API core, which includes the collection implementation. The service contains various REST APIs responsible for interacting with users and thus enabling collections and collection items management. Moreover, the service offers a graphical web frontend in order to visualize managed collections, collection items and relationships between them. The web frontend is available under <http://{hostname}:{port}/static/overview.html>. In addition, an intuitive graphical user interface will be developed in the future by SCC.

3.2 Data Model

3.2.1 Service Features

The table below includes the different service-level features this implementation offers.

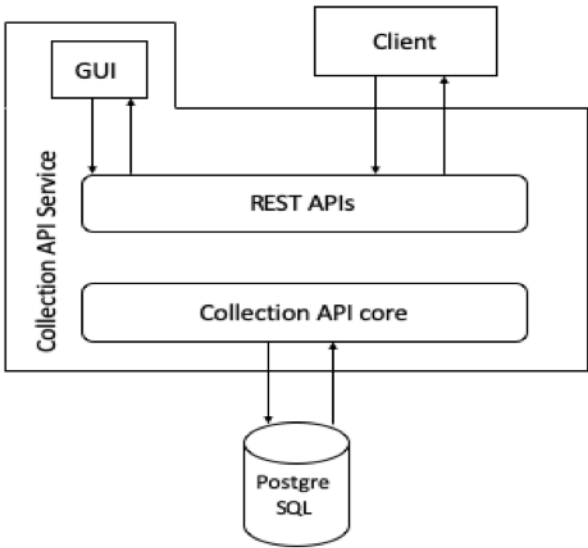


Fig. 1: Figure 1: Collection API architecture

serviceFeatures
<pre>providesCollectionPids: false collectionPidProviderType: null enforcesAccess: false supportsPagination: true asynchronousActions: false ruleBasedGeneration: false maxExpansionDepth: -1 providesVersioning: false supportedCollectionOperations: null supportedModelTypes: null</pre>

3.2.2 Collection Object

The service offers the possibility to create and manage collections and collection items. The figure below includes a data model of a collection, collection item and the relationship between them.

- Collection: includes the following attributes:
 1. Collection capabilities: comprise the following attributes, which determine the possible actions on a collection.
 2. Collection properties: include collection’s metadata.
- Collection Item: In order to create a new collection item, the following attributes are expected to be given by the user:
 1. Mappings: include the following attributes:

1.1.4 4. How can the Collection API be used?

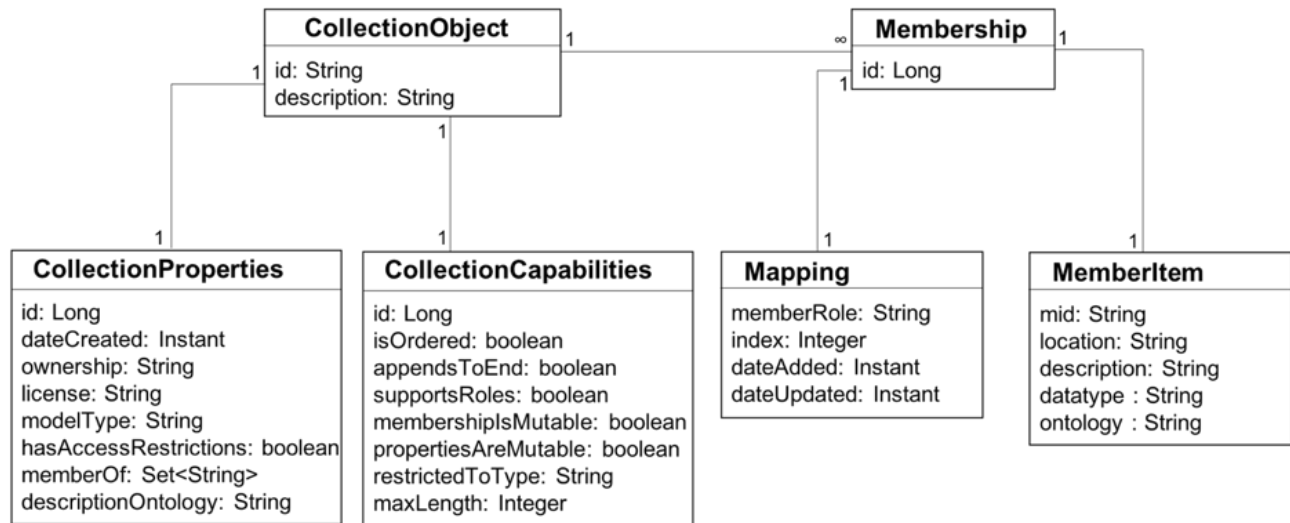


Fig. 2: Figure 2: Collections and collection items

Property Name	Description	Mandatory/Optional	Default Value
id	identifier for the collection	Optional	UUID
description	descriptive metadata about the collection	Optional	null
Collection capabilities	define the set of actions supported by a collection	Optional	—
Collection properties	functional metadata of a collection	Optional	—

Property Name	Description	Mandatory /Optional	Default Value
id	identifier of the collection capabilities	(*)	–
isOrdered	identifies if the collection items are ordered	optional	false
appendsToEnd	For an ordered collection, it indicates whether new items are appended to the end	optional	false
supportsRoles	identifies whether a collection supports assigning roles to its member items	optional	false
membershipsMutable	identifies whether a collection membership is mutable	optional	true
propertiesAreMutable	identifies whether a collection properties are mutable	optional	true
restrictedToType	indicates the type of the collection items	optional	null
maxLength	indicates the maximum length of the collection	optional	-1

(*): This value is automatically generated.

Property Name	Description	Mandatory /Optional	Default Value
id	identifier of the collection properties	(*)	–
dateCreated	the date the collection was created	(*)	–
ownership	identifies the owner of the collection	optional	null
license	identifies the license that applies to the collection	optional	null
modelType	identifies the model that the collection adheres to	optional	null
hasAccessRestrictions	indicates whether the collection has access restrictions	optional	true
memberOf	includes a list of collection identifiers to which this collection belongs	(*)	–
descriptionOntology	identifies the ontology used for descriptive metadata	optional	null

(*): This value is automatically generated.

Property Name	Description	Mandatory /Optional	Default Value
id	identifier for the member	optional	UUID
location	location at which the item data can be retrieved	mandatory	–
description	human readable description	optional	null
datatype	URI of the data type of this item. If the value of the “restrictedToType” of the collection is not null, then the datatype of the member should have the same value as the “restrictedToType”	mandatory	–
ontology	URI of an ontology model class that applies to this item	optional	null
mappings	Collection item metadata	optional	true

Property Name	Description	Mandatory/ Optional	Default Value
role	The role of this item inside the collection	optional	null
index	The position of the item in the collection	optional	0
dateAdded	The date the item was added to the collection	(*)	–
dateUpdated	The date the item's metadata were last updated	(*)	–

(*): This value is automatically generated.

4.1 Example

In this section, an example of Collection API is introduced. Let's assume we want to publish a data set of an experiment. The set includes raw data, implementation and results. These collections are sub-collections of the “experiment” collection. The “rawData” collection includes one item called “images”, which represents a set of images used in the experiment and which is stored in a research data repository. As it exists two implementations of the experiment based on two different methods, the “implementation” collection includes two items “method1” and “method2”, which are stored in two Git repositories. The “result” collection includes two sub-collections “result1” and “result2”, which includes the results of the experiment based on both methods in form of documents such as images, Excel sheets, etc. Moreover, the user wants to store the item of the method-implementation in the results sub-collections to be able to check on which implementation the results are generated. As the Collection API offers the possibility to share items between different collections, each result sub-collection includes also the implementation item of the used method. The figure below describes how the collections and their items should look like.

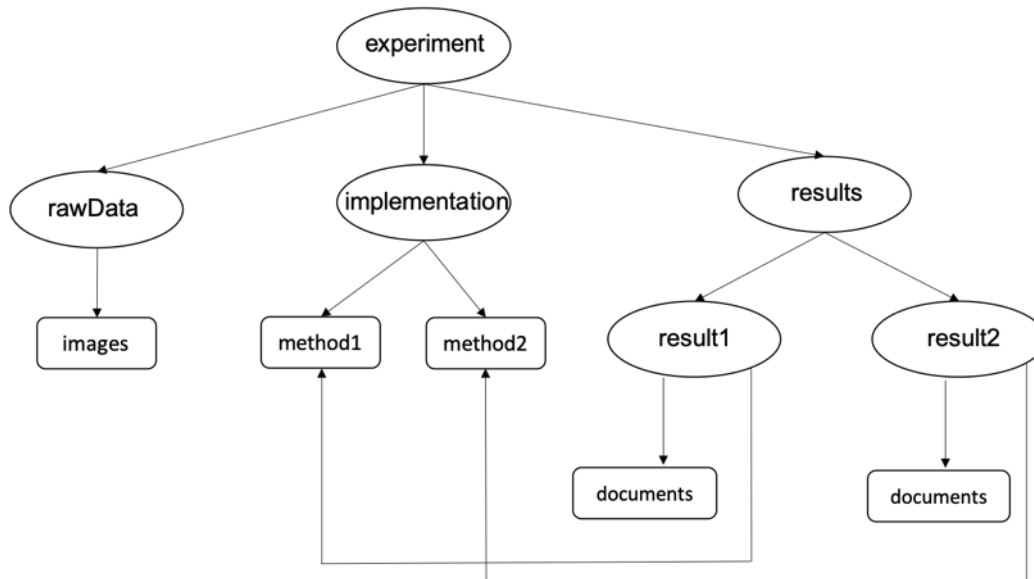


Fig. 3: Figure 4: Example

In the next sections, we will describe how to build up the above example using the Collection API service.

4.2 Creation of collections

Using the POST operation <http://localhost:8080/api/v1/collections>, we have the possibility to create collections one by one or all together. As an input, a JSON object including the collection attributes is needed. In the above described example, six collections should be created. Figure 5 below includes an example of creating the “experiment” collection and the response of this operation is represented in Figure 6. The non-given attributes are filled out with the default values. Moreover, the created date is automatically generated. As the experiment collection is not an item of another collection and has no items yet, the value of both attributes “memberOf” and “members” is an empty list. The five remaining collections “rawData”, “implementation”, “results”, “result1” and “result2” can be created in the same way.

4.3 Creation of sub-collections

In order to add the relationship between “experiment” collection and other collections, we should add the sub-collections as items to the parent collection using the following POST operation: http://localhost:8080/api/v1/collections/{collection_id}/members. To run this operation, “id”, “location” and “datatype”

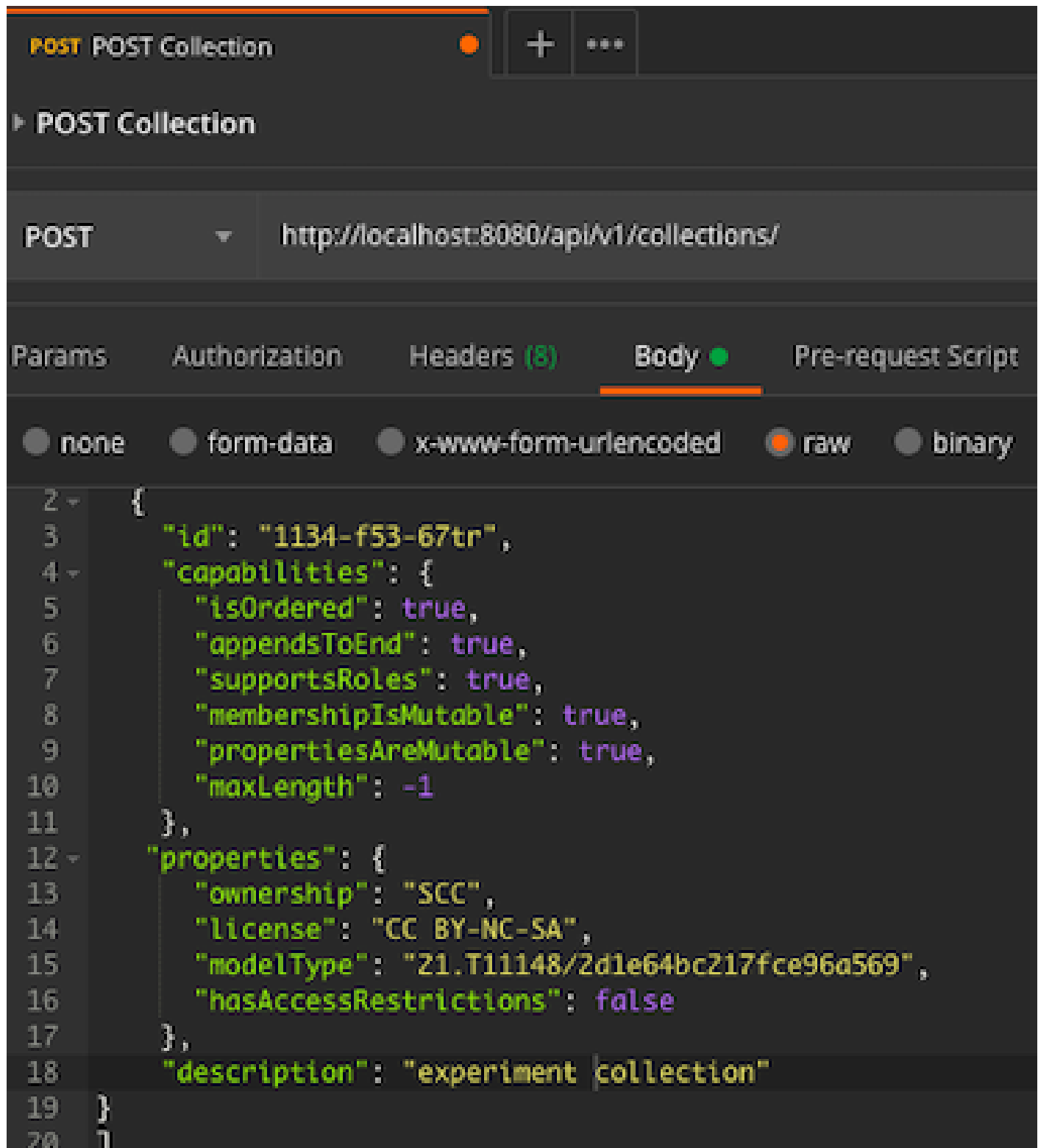


Fig. 4: Figure 5: Creation of the experiment collection

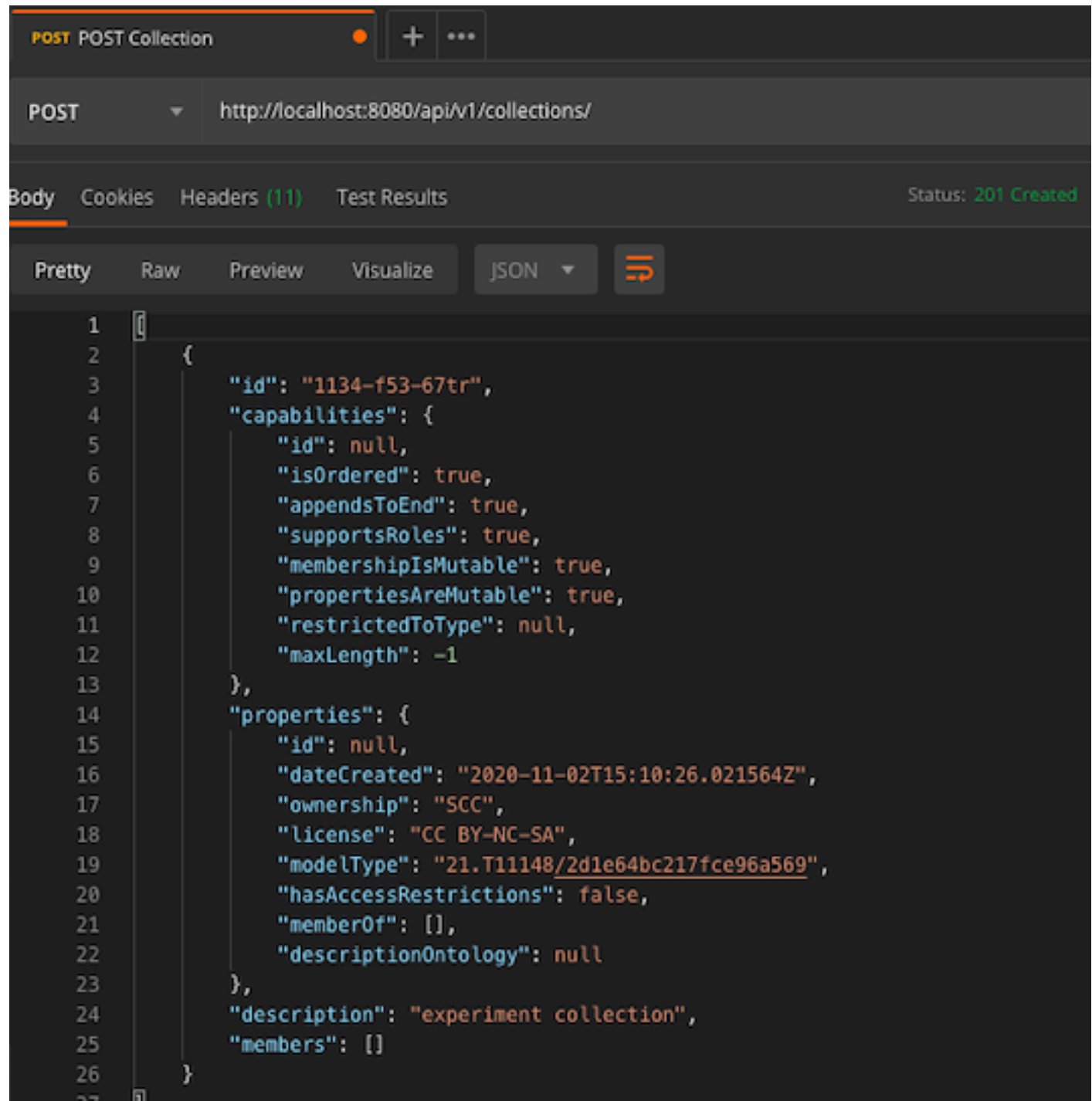


Fig. 5: Figure 6: Response of the POST experiment collection

are mandatory fields. Figure 7 includes an example of the JSON object needed while adding “rawData” collection to the “experiment” collection.

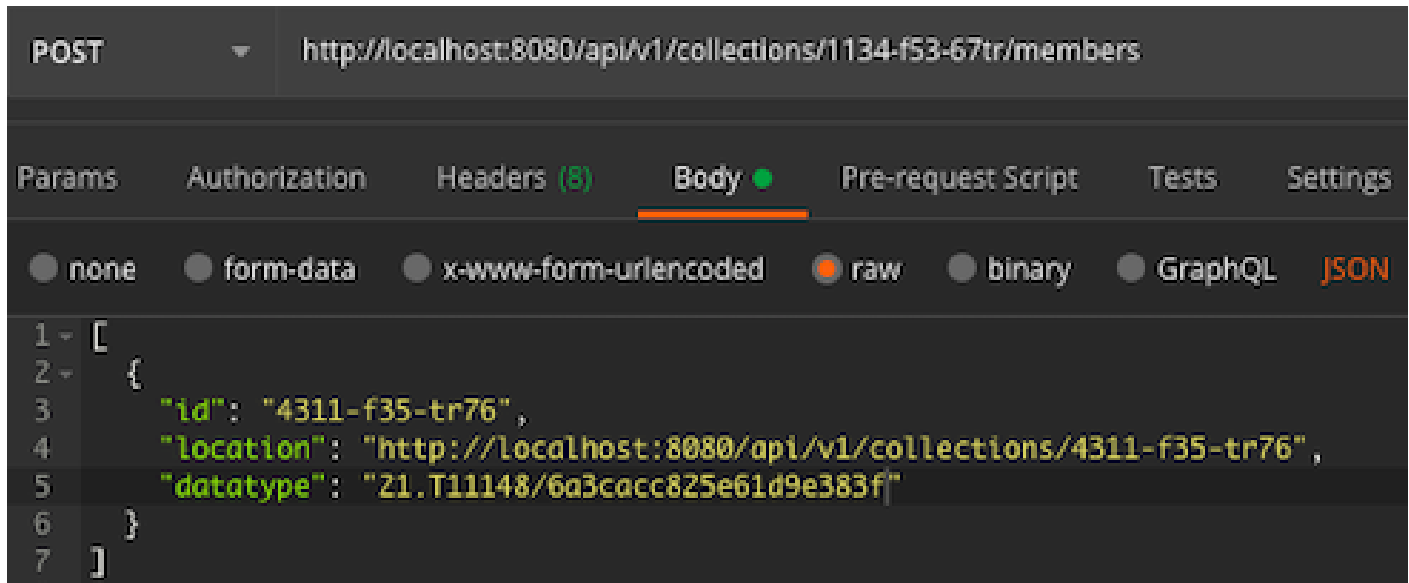


Fig. 6: Figure 7: Add „rawData“ as a sub-collection

After executing this operation, “rawData” collection is added to the member list of the “experiment” collection as shown in Figure 8, which includes a response of the GET collection operation.

The remaining collections can be added the same way as sub-collections to the “experiment” and “results” collection.

4.4 Creation of collection items

In order to create a new member and add it to a collection, the same POST operation mentioned in 4.3 should be performed: http://localhost:8080/api/v1/collections/{collection_identifier}/members. Figure 9 includes an example of adding item “images” to collection “rawData”.

Moreover, item “method1” is a shared item of two collections and Figure 10 includes an example of a JSON object, which should be added to both collections using the POST operation. Only the identifier of the collection, to which the item is added, should be modified.

Other REST APIS are available such as listing collections and items, updating or removing them. Moreover, to access the visualization of the above created collections, items and relationships between them, you can browse to <http://localhost:8080/static/overview.html>. Figure 11 includes the example overview. Blue ovals represent collections and orange ones represent member items.

To have more information about the collections or items, the user has just to click on the oval. Figure 12 includes an example of a collection description. Moreover, you can search a collection or an item by writing its identifier in the search box.

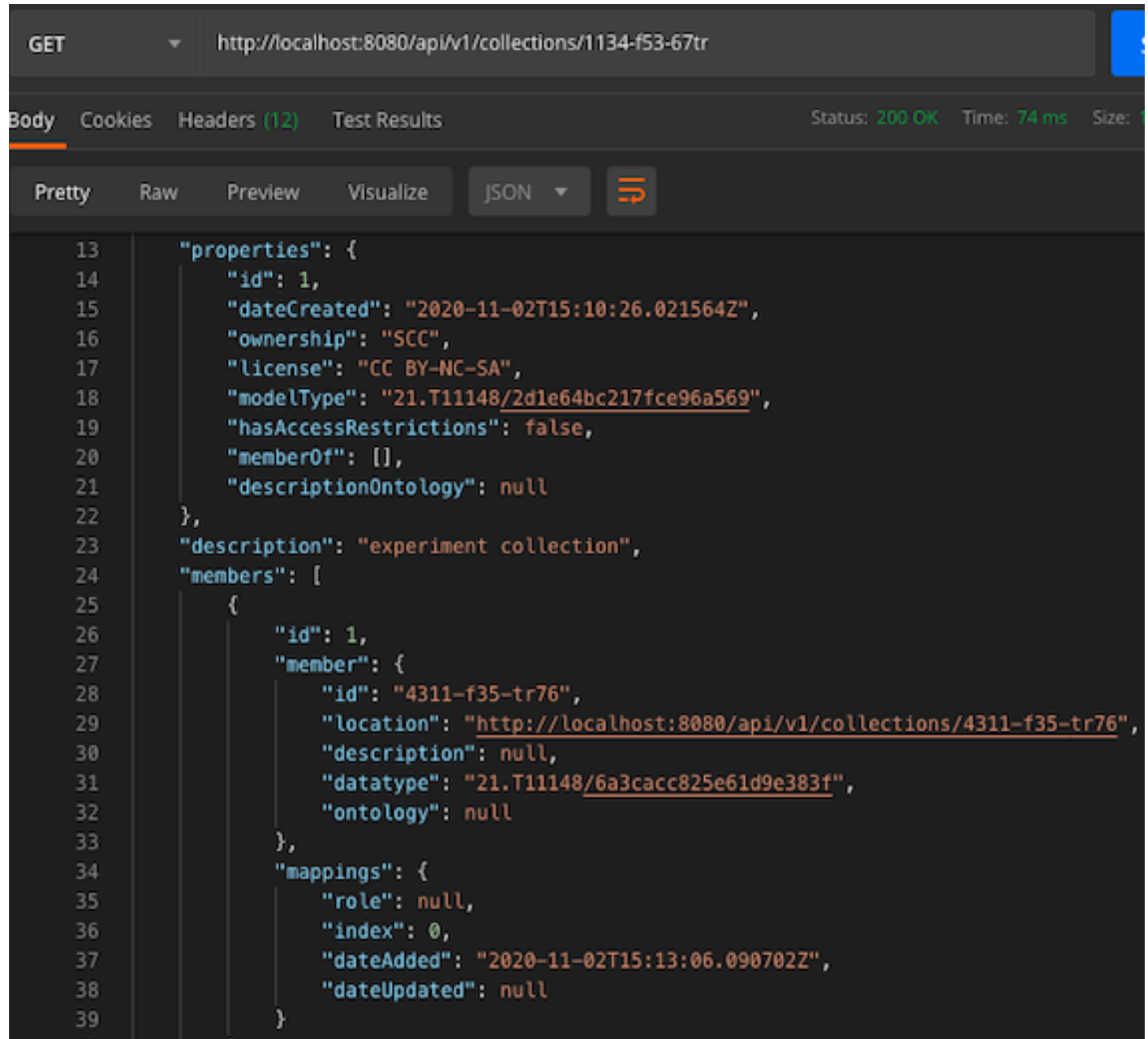
In order to build the above described example, a short tutorial is available under <https://www.katacoda.com/kitdm/scenarios/collection-api>

1. What is Collection API? A short overview of the Collection API.

2. Installation Installation instructions.

3. Collection API Implementation illustration of the service architecture.

4. How can the Collection API be used? Steps towards realizing an example use case.



```
GET http://localhost:8080/api/v1/collections/1134-f53-67tr

Body Cookies Headers (12) Test Results Status: 200 OK Time: 74 ms Size: 1

Pretty Raw Preview Visualize JSON

{
  "properties": {
    "id": 1,
    "dateCreated": "2020-11-02T15:10:26.021564Z",
    "ownership": "SCC",
    "license": "CC BY-NC-SA",
    "modelType": "21.T11148/2d1e64bc217fce96a569",
    "hasAccessRestrictions": false,
    "memberOf": [],
    "descriptionOntology": null
  },
  "description": "experiment collection",
  "members": [
    {
      "id": 1,
      "member": {
        "id": "4311-f35-tr76",
        "location": "http://localhost:8080/api/v1/collections/4311-f35-tr76",
        "description": null,
        "datatype": "21.T11148/6a3cacc825e61d9e383f",
        "ontology": null
      },
      "mappings": {
        "role": null,
        "index": 0,
        "dateAdded": "2020-11-02T15:13:06.090702Z",
        "dateUpdated": null
      }
    }
  ]
}
```

Fig. 7: Figure 8: Get „experiment“ collection

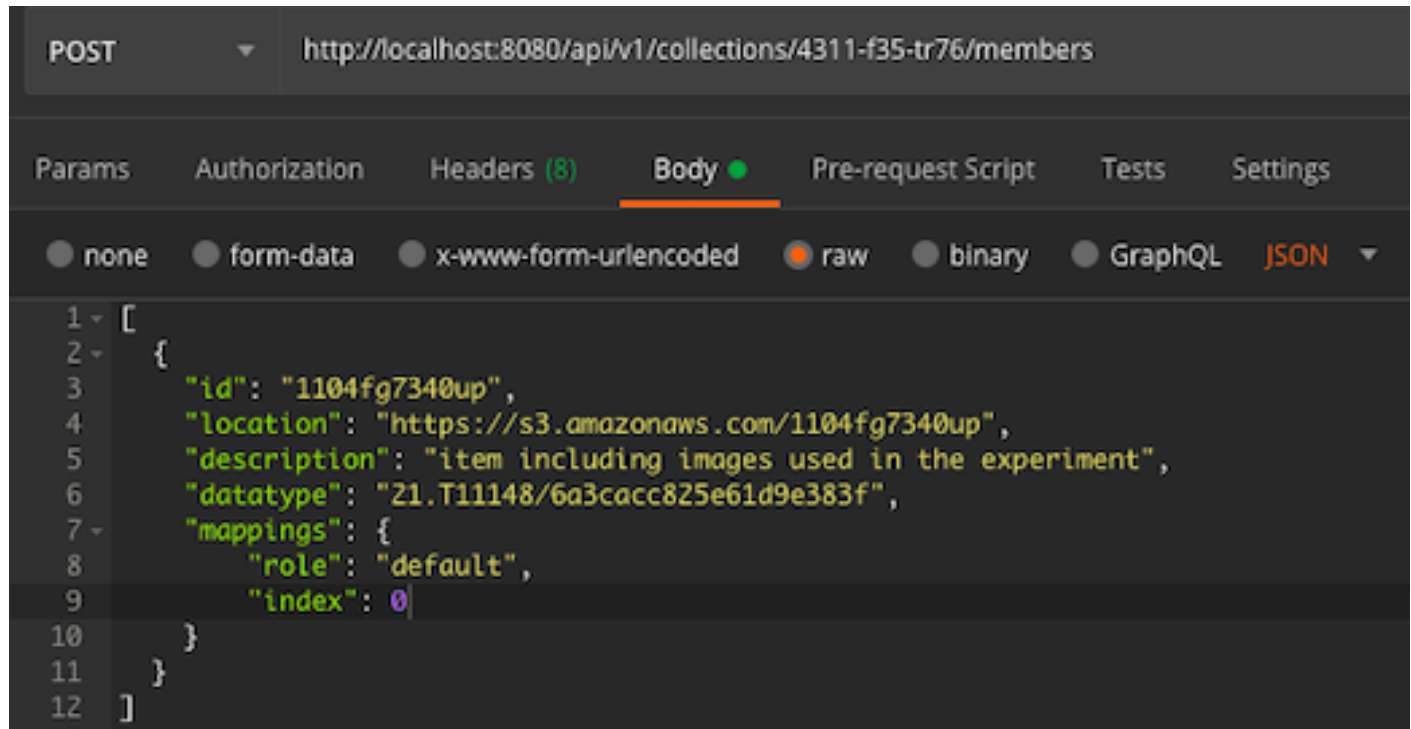


Fig. 8: Figure 9: Creation of „images“ item

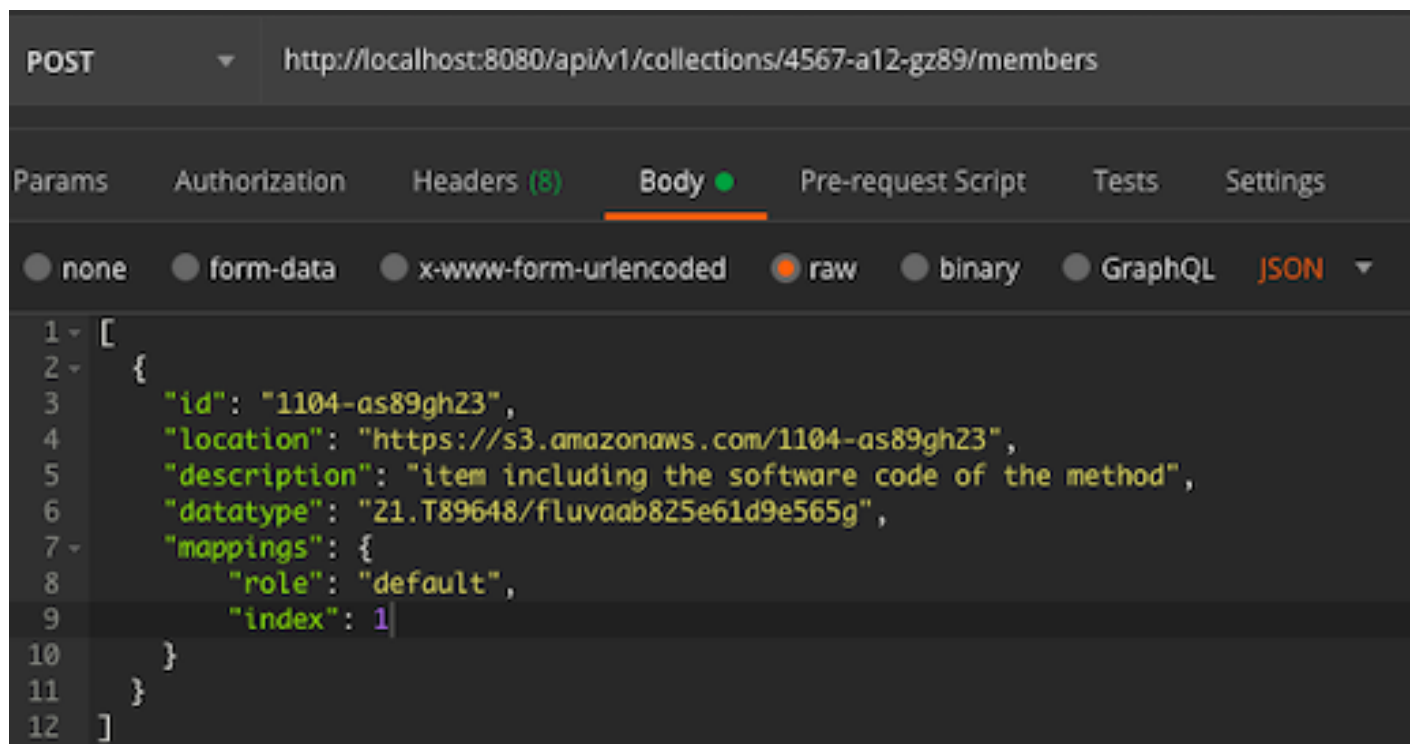


Fig. 9: Figure 10: Add “method1” item to the “implementation” collection



Fig. 10: Figure 11: Overview

Search

Collection info

Identifier

4567-a12-gz89

Description

Implementation collection

properties

capabilities

Fig. 11: Figure 12: Collection description