Katsu Metadata service

Release 2.3.0

Ksenia Zaytseva, David Lougheed, Simon Chénard, Romain Grég

Aug 03, 2021

CONTENTS:

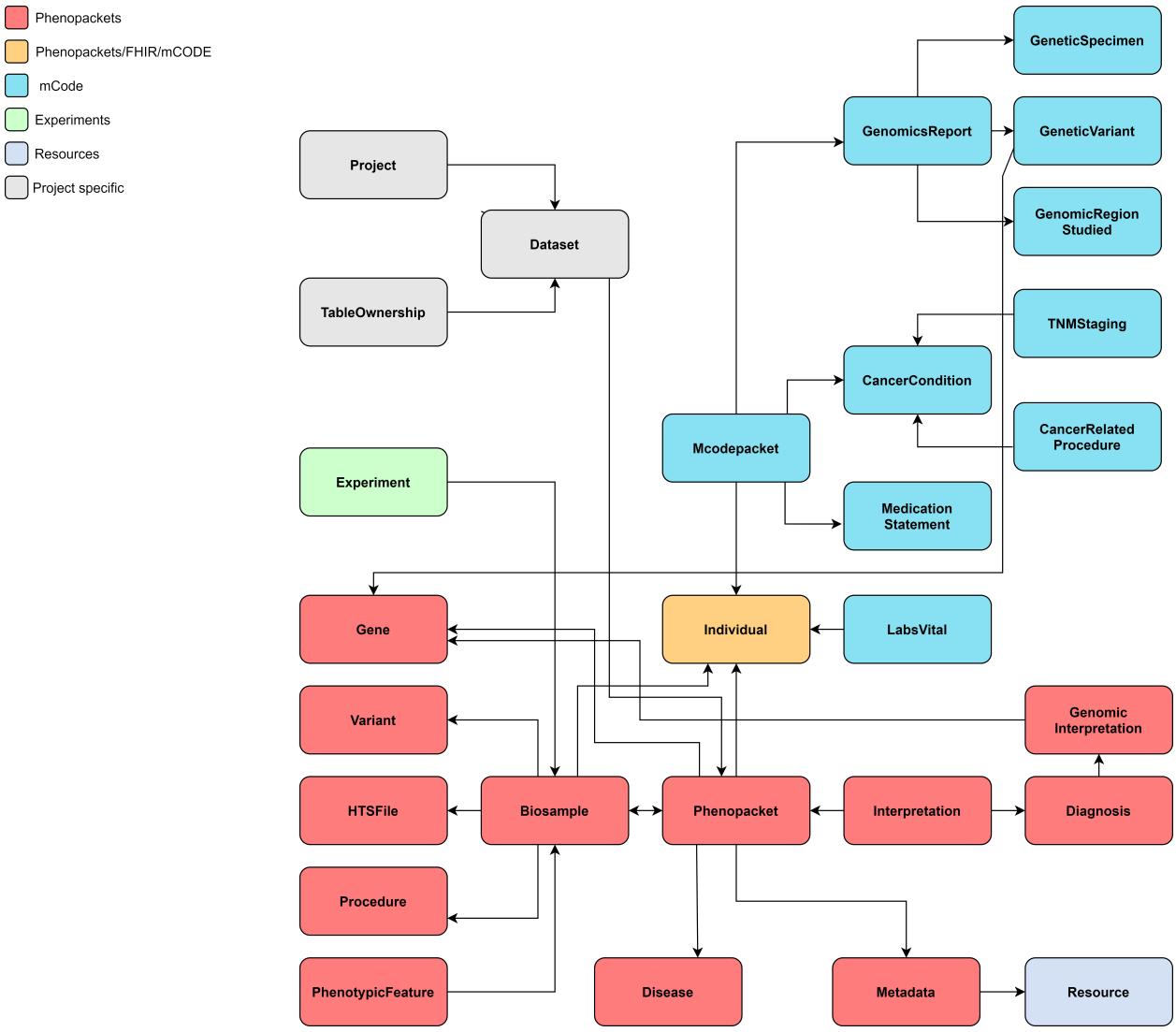
CHAPTER

ONE

INTRODUCTION

Katsu Metadata service is a service to store phenotypic and clinical metadata about the patient and/or biosample. The data model is partly based on GA4GH Phenopackets schema and extended to support oncology-related metadata and experiments metadata.

The simplified data model of the service is below.



1.1 Technical implementation

The service is implemented in Python and Django and uses PostgreSQL database to store the data. Besides PostgreSQL, the data can be indexed and queried in Elasticsearch.

1.2 Architecture

The Katsu Metadata Service contains several services that share one API. Services depend on each other and are separated based on their scope.

1. Patients service handles anonymized individual's data (e.g. individual id, sex, age, or date of birth).

- Data model: aggregated profile from GA4GH Phenopackets Individual, FHIR Patient, and mCODE Patient. It contains all fields of Phenopacket Individual and additional fields from FHIR and mCODE Patient.

2. Phenopackets service handles phenotypic and clinical data.

- Data model: GA4GH Phenopackets schema. Currently contains only two out of four Phenopackets top elements - Phenopacket and Interpretation.

3. mCode service handles patient's oncology-related data.

- Data model: mCODE data elements. mCODE data elements grouped in a mCodepacket (like Phenopacket) containing patient's cancer-related descriptions including genomics data, medication statements, and cancer-related procedures.

4. Experiments service handles experiment related data.

- Data model: derived from IHEC metadata Experiment specification.

5. Resources service handles metadata about ontologies used for data annotation.

- Data model: derived from the Phenopackets schema Resource profile.

6. CHORD service handles granular metadata about dataset (e.g. description, where the dataset is located, who are the creators of the dataset, licenses applied to the dataset, authorization policy, terms of use). The dataset in the current implementation is one or more phenopackets related to each other through their provenance.

- Data model:
  - DATS model used for dataset description;
  - GA4GH DUO is used to capture the terms of use applied to a dataset.

7. Restapi service handles all generic functionality shared among other services (e.g. renderers, common serializers, schemas, validators)

1.3 Metadata standards

Phenopackets schema is used for phenotypic description of patient and/or biosample.

mCODE data elements are used for oncology-related description of patient.

DATS standard is used for dataset description.

DUO ontology is used for describing terms of use for a dataset.

Phenopackets on FHIR Implementation Guide is used to map Phenopackets elements to FHIR resources.

IHEC Metadata Experiment is used for describing an experiment.

1.4 REST API highlights

Parsers and Renderers

- Standard API serves data in snake_case style.
- To retrieve the data in camelCase append `?format=phenopackets`.
- Data can be ingested in both snake_case or camelCase.
- Other available renderers:
  - Currently, the following classes can be retrieved in FHIR format by appending `?format=fhir`: Phenopacket, Individual, Biosample, PhenotypicFeature, HtsFile, Gene, Variant, Disease, Procedure.
  - JSON-LD context to schema.org provided for the Dataset class in order to allow for a Google dataset search for Open Access Data: append `?format=json-ld` when querying dataset endpoint.
  - Dataset description can also be retrieved in RDF format: append `?format=rdf` when querying the dataset endpoint.

Data ingest

Ingest workflows are implemented for different types of data within the service. Ingest endpoint is `/private/ingest`.

1. Phenopackets data ingest

The data must follow Phenopackets schema in order to be ingested. See full *Ingestion workflow example*.

Example of Phenopackets POST request body:

```
{
  "table_id": "table_unique_uuid",
  "workflow_id": "phenopackets_json",
  "workflow_params": {
    "phenopackets_json.json_document": "path/to/data.json"
  },
  "workflow_outputs": {
    "json_document": "path/to/data.json"
  }
}
```

2. mCode data ingest

mCODE data elements are based on FHIR datatypes. Only mCode related profiles will be ingested. It's expected that the data is compliant with FHIR Release 4 and provided in FHIR Bundles.

Example of mCode FHIR data POST request body:

```
{
   "table_id":"table_unique_uuid",
   "workflow_id":"mcode_fhir_json",
   "workflow_params":{
      "mcode_fhir_json.json_document":"/path/to/data.json"
   },
   "workflow_outputs":{
      "json_document":"/path/to/data.json"
   }
}
```

3. FHIR data ingest

At the moment there is no implementation guide from FHIR to Phenopackets. FHIR data will only be ingested partially where it's possible to establish mapping between FHIR resource and Phenopackets element. The ingestion works for the following FHIR resources: Patient, Observation, Condition, Specimen. It's expected that the data is compliant with FHIR Release 4 and provided in FHIR Bundles.

```
{
  "table_id": "table_unique_uuid",
  "workflow_id": "fhir_json",
  "workflow_params": {
    "fhir_json.patients": "/path/to/patients.json",
    "fhir_json.observations": "/path/to/observations.json",
    "fhir_json.conditions": "/path/to/conditions.json",
    "fhir_json.specimens": "/path/to/specimens.json"
  },
  "workflow_outputs": {
    "patients": "/path/to/patients.json",
    "observations": "/path/to/observations.json",
    "conditions": "/path/to/conditions.json",
    "specimens": "/path/to/specimens.json"
  }
}
```

1.5 Elasticsearch index (optional)

Data in FHIR format can be indexed in Elasticsearch - this is optional. If an Elasticsearch instance is running on the server (so on `localhost:9000`) these models will be automatically indexed on creation/update. There are also two scripts provided to update these indexes all at once:

```
python manage.py patients_build_index
python manage.py phenopackets_build_index
```

Here is an example request for querying this information:

```
curl -X POST -H 'Content-Type: application/json' -d '{"data_type": "phenopacket",
↪"query": {"query": {"match": {"gender": "FEMALE"}}}}' http://127.0.0.1:8000/private/
↪fhir-search
```

CHAPTER

TWO

INSTALLATION

1. Clone the project from github (use `-r` to fetch submodules content)

```
git clone https://github.com/bento-platform/katsu.git
```

2. Install the git submodule for DATS JSON schemas (if did not clone recursively):

```
git submodule update --init
```

3. Create and activate a virtual environment
4. Move to the main directory and install required packages:

```
pip install -r requirements.txt
```

5. The service uses PostgreSQL database for data storage. Install PostgreSQL following this tutorial.
6. Configure database connection in settings.py

e.g. settings if running database on localhost, default port for PostgreSQL is 5432:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'database_name',
        'USER': 'user',
        'PASSWORD': 'password',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

7. From the main directory run (where the manage.py file located):

```
python manage.py makemigrations
python manage.py migrate
python manage.py runserver
```

8. Development server runs at `localhost:8000`

CHAPTER

THREE

INGESTION WORKFLOW EXAMPLE

1. Create a project at `/api/projects`:

```
{
  "title": "Test Project",
  "description": "About Test Project ..."
}
```

201 Response example:

```
{
    "identifier": "998a36b2-7251-445d-81de-01a5affc5523",
    "datasets": [],
    "title": "Test Project",
    "description": "About Test Project ...",
    "created": "2020-10-15T20:17:03.029395Z",
    "updated": "2020-10-15T20:17:03.029395Z"
}
```

2. Create a dataset at `/api/datasets`:

Add project identifier from project response.

```
{
  "project": "998a36b2-7251-445d-81de-01a5affc5523",
  "title": "Test Dataset",
  "description": "About Test Dataset ...",
  "data_use": {
    "consent_code": {
      "primary_category": {
        "code": "GRU"
      },
      "secondary_categories": [
        {
          "code": "RU"
        }
      ]
    },
    "data_use_requirements": [
      {
        "code": "COL"
      }
    ]
  }
}
```

201 Response example:


```
{
    "identifier": "86766cd6-f6bd-4d09-9d8a-308df4dd1fa1",
    "table_ownership": [],
    "title": "Test Dataset",
    "description": "About Test Dataset ...",
    "contact_info": "",
    "data_use": {
        "consent_code": {
            "primary_category": {
                "code": "GRU"
            },
            "secondary_categories": [
                {
                    "code": "RU"
                }
            ]
        },
        "data_use_requirements": [
            {
                "code": "COL"
            }
        ]
    },
    "linked_field_sets": [],
    "version": "version_2020-10-15 20:17:52.412173+00:00",
    "created": "2020-10-15T20:17:52.418029Z",
    "updated": "2020-10-15T20:17:52.418029Z",
    "project": "c488af39-d49b-4764-aa19-b86801220060"
}
```


3. Create a table ownership at `/api/table_ownership`:

Generate UUID for `table_id` and add dataset identifier from dataset response.


```
{
    "table_id": "e08ff220-0f26-11eb-adc1-0242ac120002",
    "service_id": "metadata_service",
    "service_artifact": "metadata",
    "dataset": "86766cd6-f6bd-4d09-9d8a-308df4dd1fa1"
}
```


201 Response example:


```
{
    "table_id": "e08ff220-0f26-11eb-adc1-0242ac120002",
    "service_id": "metadata_service",
    "service_artifact": "metadata",
    "dataset": "86766cd6-f6bd-4d09-9d8a-308df4dd1fa1"
}
```


4. Create a table at `/api/tables`:

Add table_id as `ownership_record`.


```
{
    "ownership_record": "e08ff220-0f26-11eb-adc1-0242ac120002",
    "name": "metadata",
```


(continues on next page)

(continued from previous page)

```
    "data_type": "phenopacket"
}
```

5. Ingest phenopackets at `/private/ingest`:

Add `table_id`.

Specify path to the phenopackets data.

```
{
  "table_id": "e08ff220-0f26-11eb-adc1-0242ac120002",
  "workflow_id": "phenopackets_json",
  "workflow_params": {
    "phenopackets_json.json_document": "path/to/phenopackets.json"
  },
  "workflow_outputs": {
    "json_document": "path/to/phenopackets.json"
  }
}
```

CHAPTER

FOUR

MODELS

4.1 Phenopackets service

class chord_metadata_service.phenopackets.models.**Biosample**(**args*, ***kwargs*)
Class to describe a unit of biological material

FHIR: Specimen

exception DoesNotExist

exception MultipleObjectsReturned

class chord_metadata_service.phenopackets.models.**Diagnosis**(**args*, ***kwargs*)
Class to refer to disease that is present in the individual analyzed

FHIR: Condition

exception DoesNotExist

exception MultipleObjectsReturned

class chord_metadata_service.phenopackets.models.**Disease**(**args*, ***kwargs*)
Class to represent a diagnosis and inference or hypothesis about the cause underlying the observed phenotypic abnormalities

FHIR: Condition

exception DoesNotExist

exception MultipleObjectsReturned

class chord_metadata_service.phenopackets.models.**Gene**(**args*, ***kwargs*)
Class to represent an identifier for a gene

FHIR: ? Draft extension for Gene is in development where Gene defined via class CodeableConcept

exception DoesNotExist

exception MultipleObjectsReturned

class chord_metadata_service.phenopackets.models.**GenomicInterpretation**(**args*, ***kwargs*)
Class to represent a statement about the contribution of a genomic element towards the observed phenotype

FHIR: Observation

exception DoesNotExist

exception MultipleObjectsReturned

clean()
Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

class chord_metadata_service.phenopackets.models.**HtsFile**(**args*, ***kwargs*)
Class to link HTC files with data

FHIR: DocumentReference

exception DoesNotExist

exception MultipleObjectsReturned

class chord_metadata_service.phenopackets.models.**Interpretation**(**args*, ***kwargs*)
Class to represent the interpretation of a genomic analysis

FHIR: DiagnosticReport

exception DoesNotExist

exception MultipleObjectsReturned

class chord_metadata_service.phenopackets.models.**MetaData**(**args*, ***kwargs*)
Class to store structured definitions of the resources and ontologies used within the phenopacket

FHIR: Metadata

exception DoesNotExist

exception MultipleObjectsReturned

class chord_metadata_service.phenopackets.models.**Phenopacket**(**args*, ***kwargs*)
Class to aggregate Individual's experiments data

FHIR: Composition

exception DoesNotExist

exception MultipleObjectsReturned

class chord_metadata_service.phenopackets.models.**PhenotypicFeature**(**args*, ***kwargs*)
Class to describe a phenotype of an Individual

FHIR: Condition or Observation

exception DoesNotExist

exception MultipleObjectsReturned

class chord_metadata_service.phenopackets.models.**Procedure**(**args*, ***kwargs*)
Class to represent a clinical procedure performed on an individual (subject) in order to extract a biosample

FHIR: Procedure

exception DoesNotExist

exception MultipleObjectsReturned

class chord_metadata_service.phenopackets.models.**Variant**(**args*, ***kwargs*)
Class to describe Individual variants or diagnosed causative variants

FHIR: Observation ? Draft extension for Variant is in development

exception DoesNotExist

exception MultipleObjectsReturned

4.2 Patients service

class chord_metadata_service.patients.models.**Individual**(**args*, ***kwargs*)
Class to store demographic information about an Individual (Patient)

exception DoesNotExist

exception MultipleObjectsReturned

4.3 Mcode service

class chord_metadata_service.mcode.models.**CancerCondition**(**args*, ***kwargs*)
Class to record the history of primary or secondary cancer conditions.

exception DoesNotExist

exception MultipleObjectsReturned

class chord_metadata_service.mcode.models.**CancerGeneticVariant**(**args*, ***kwargs*)
Class to record an alteration in DNA.

exception DoesNotExist

exception MultipleObjectsReturned

class chord_metadata_service.mcode.models.**CancerRelatedProcedure**(**args*, ***kwargs*)
Class to represent radiological treatment or surgical action addressing a cancer condition.

exception DoesNotExist

exception MultipleObjectsReturned

class chord_metadata_service.mcode.models.**GeneticSpecimen**(**args*, ***kwargs*)
Class to describe a biosample used for genomics testing or analysis.

exception DoesNotExist

exception MultipleObjectsReturned

class chord_metadata_service.mcode.models.**GenomicRegionStudied**(**args*, ***kwargs*)
Class to describe the area of the genome region referenced in testing for variants.

exception DoesNotExist

exception MultipleObjectsReturned

class chord_metadata_service.mcode.models.**GenomicsReport**(**args*, ***kwargs*)
Genetic Analysis Summary.

exception DoesNotExist

exception MultipleObjectsReturned

class chord_metadata_service.mcode.models.**LabsVital**(**args*, ***kwargs*)
Class to record tests performed on patient.

exception DoesNotExist

exception MultipleObjectsReturned

class chord_metadata_service.mcode.models.**MCodePacket**(**args*, ***kwargs*)
Class to aggregate Individual's cancer related metadata

exception DoesNotExist

exception MultipleObjectsReturned

class chord_metadata_service.mcode.models.**MedicationStatement**(**args*, ***kwargs*)
Class to record the use of a medication.

exception DoesNotExist

exception MultipleObjectsReturned

class chord_metadata_service.mcode.models.**TNMStaging**(**args*, ***kwargs*)
Class to describe the spread of cancer in a patient's body.

exception DoesNotExist

exception MultipleObjectsReturned

4.4 Experiments service

class chord_metadata_service.experiments.models.**Experiment**(**args*, ***kwargs*)
Class to store Experiment information

exception DoesNotExist

exception MultipleObjectsReturned

class chord_metadata_service.experiments.models.**ExperimentResult**(**args*, ***kwargs*)
Class to represent information about analysis of sequencing data in a file format.

exception DoesNotExist

exception MultipleObjectsReturned

class chord_metadata_service.experiments.models.**Instrument**(**args*, ***kwargs*)
Class to represent information about instrument used to perform a sequencing experiment.

exception DoesNotExist

exception MultipleObjectsReturned

4.5 Resources service

class chord_metadata_service.resources.models.**Resource**(**args*, ***kwargs*)
Class to represent a description of an external resource used for referencing an object

FHIR: CodeSystem

exception DoesNotExist

exception MultipleObjectsReturned

clean()
Hook for doing any extra model-wide validation after clean() has been called on every field by

self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

save(**args*, ***kwargs*)
Save the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

4.6 CHORD service

class chord_metadata_service.chord.models.**Project**(**args*, ***kwargs*)
Class to represent a Project, which contains multiple Datasets which are each a group of Phenopackets.

exception DoesNotExist

exception MultipleObjectsReturned

class chord_metadata_service.chord.models.**Dataset**(**args*, ***kwargs*)
Class to represent a Dataset, which contains multiple Phenopackets.

exception DoesNotExist

exception MultipleObjectsReturned

clean()
Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

save(**args*, ***kwargs*)
Save the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

class chord_metadata_service.chord.models.**TableOwnership**(**args*, ***kwargs*)
Class to represent a Table, which are organizationally part of a Dataset and can optionally be attached to a Phenopacket (and possibly a Biosample).

exception DoesNotExist

exception MultipleObjectsReturned

class chord_metadata_service.chord.models.**Table**(*ownership_record*, *name*, *data_type*, *created*, *updated*)

exception DoesNotExist

exception MultipleObjectsReturned

CHAPTER

FIVE

VIEWS

5.1 Phenopackets service

class chord_metadata_service.phenopackets.api_views.**BiosampleViewSet**(***kwargs*)
get: Return a list of all existing biosamples

post: Create a new biosample

serializer_class
alias of chord_metadata_service.phenopackets.serializers.BiosampleSerializer

class chord_metadata_service.phenopackets.api_views.**DiagnosisViewSet**(***kwargs*)
get: Return a list of all existing diagnoses

post: Create a new diagnosis

serializer_class
alias of chord_metadata_service.phenopackets.serializers.DiagnosisSerializer

class chord_metadata_service.phenopackets.api_views.**DiseaseViewSet**(***kwargs*)
get: Return a list of all existing diseases

post: Create a new disease

serializer_class
alias of chord_metadata_service.phenopackets.serializers.DiseaseSerializer

class chord_metadata_service.phenopackets.api_views.**ExtendedPhenopacketsModelViewSet**(***kwarg*

class chord_metadata_service.phenopackets.api_views.**GeneViewSet**(***kwargs*)
get: Return a list of all existing genes

post: Create a new gene

serializer_class
alias of chord_metadata_service.phenopackets.serializers.GeneSerializer

class chord_metadata_service.phenopackets.api_views.**GenomicInterpretationViewSet**(***kwargs*)
get: Return a list of all existing genomic interpretations

post: Create a new genomic interpretation

serializer_class
alias of chord_metadata_service.phenopackets.serializers.GenomicInterpretationSerializer

class chord_metadata_service.phenopackets.api_views.**HtsFileViewSet**(***kwargs*)
get: Return a list of all existing HTS files

post: Create a new HTS file

serializer_class
alias of chord_metadata_service.phenopackets.serializers.HtsFileSerializer

class chord_metadata_service.phenopackets.api_views.**InterpretationViewSet**(***kwargs*)
get: Return a list of all existing interpretations

post: Create a new interpretation

serializer_class
alias of chord_metadata_service.phenopackets.serializers.InterpretationSerializer

class chord_metadata_service.phenopackets.api_views.**MetaDataViewSet**(***kwargs*)
get: Return a list of all existing metadata records

post: Create a new metadata record

serializer_class
alias of chord_metadata_service.phenopackets.serializers.MetaDataSerializer

class chord_metadata_service.phenopackets.api_views.**PhenopacketViewSet**(***kwargs*)
get: Return a list of all existing phenopackets

post: Create a new phenopacket

serializer_class
alias of chord_metadata_service.phenopackets.serializers.PhenopacketSerializer

class chord_metadata_service.phenopackets.api_views.**PhenopacketsModelViewSet**(***kwargs*)

dispatch(**args*, ***kwargs*)
.dispatch() is pretty much the same as Django's regular dispatch, but with extra hooks for startup, finalize, and exception handling.

pagination_class
alias of chord_metadata_service.restapi.pagination.LargeResultsSetPagination

class chord_metadata_service.phenopackets.api_views.**PhenotypicFeatureViewSet**(***kwargs*)
get: Return a list of all existing phenotypic features

post: Create a new phenotypic feature

serializer_class
alias of chord_metadata_service.phenopackets.serializers.PhenotypicFeatureSerializer

class chord_metadata_service.phenopackets.api_views.**ProcedureViewSet**(***kwargs*)
get: Return a list of all existing procedures

post: Create a new procedure

serializer_class
alias of chord_metadata_service.phenopackets.serializers.ProcedureSerializer

class chord_metadata_service.phenopackets.api_views.**VariantViewSet**(**kwargs)
get: Return a list of all existing variants

post: Create a new variant

serializer_class
alias of chord_metadata_service.phenopackets.serializers.VariantSerializer

chord_metadata_service.phenopackets.api_views.**get_chord_phenopacket_schema**(*self*, *request*, **args*, ***kwargs*)
get: Chord phenopacket schema that can be shared with data providers.

5.2 Patients service

class chord_metadata_service.patients.api_views.**IndividualViewSet**(**kwargs)
get: Return a list of all existing individuals

post: Create a new individual

dispatch(**args*, ***kwargs*)
.dispatch() is pretty much the same as Django's regular dispatch, but with extra hooks for startup, finalize, and exception handling.

pagination_class
alias of chord_metadata_service.restapi.pagination.LargeResultsSetPagination

serializer_class
alias of chord_metadata_service.patients.serializers.IndividualSerializer

5.3 Mcode service

class chord_metadata_service.mcode.api_views.**CancerConditionViewSet**(**kwargs)

serializer_class
alias of chord_metadata_service.mcode.serializers.CancerConditionSerializer

class chord_metadata_service.mcode.api_views.**CancerGeneticVariantViewSet**(**kwargs)

serializer_class
alias of chord_metadata_service.mcode.serializers.CancerGeneticVariantSerializer

class chord_metadata_service.mcode.api_views.**CancerRelatedProcedureViewSet**(**kwargs)

serializer_class
alias of chord_metadata_service.mcode.serializers.CancerRelatedProcedureSerializer

class chord_metadata_service.mcode.api_views.**GeneticSpecimenViewSet**(**kwargs)

serializer_class
alias of chord_metadata_service.mcode.serializers.GeneticSpecimenSerializer

class chord_metadata_service.mcode.api_views.**GenomicRegionStudiedViewSet**(***kwargs*)

serializer_class
alias of chord_metadata_service.mcode.serializers.GenomicRegionStudiedSerializer

class chord_metadata_service.mcode.api_views.**GenomicsReportViewSet**(***kwargs*)

serializer_class
alias of chord_metadata_service.mcode.serializers.GenomicsReportSerializer

class chord_metadata_service.mcode.api_views.**LabsVitalViewSet**(***kwargs*)

serializer_class
alias of chord_metadata_service.mcode.serializers.LabsVitalSerializer

class chord_metadata_service.mcode.api_views.**MCodePacketViewSet**(***kwargs*)

serializer_class
alias of chord_metadata_service.mcode.serializers.MCodePacketSerializer

class chord_metadata_service.mcode.api_views.**McodeModelViewSet**(***kwargs*)

dispatch(**args*, ***kwargs*)
.dispatch() is pretty much the same as Django's regular dispatch, but with extra hooks for startup, finalize, and exception handling.

pagination_class
alias of chord_metadata_service.restapi.pagination.LargeResultsSetPagination

class chord_metadata_service.mcode.api_views.**MedicationStatementViewSet**(***kwargs*)

serializer_class
alias of chord_metadata_service.mcode.serializers.MedicationStatementSerializer

class chord_metadata_service.mcode.api_views.**TNMStagingViewSet**(***kwargs*)

serializer_class
alias of chord_metadata_service.mcode.serializers.TNMStagingSerializer

chord_metadata_service.mcode.api_views.**get_mcode_schema**(*self*, *request*, **args*, ***kwargs*)
get: Mcodepacket schema

5.4 Experiments service

class chord_metadata_service.experiments.api_views.**ExperimentViewSet**(***kwargs*)

get: Return a list of all existing experiments

post: Create a new experiment

dispatch(**args*, ***kwargs*)

.dispatch() is pretty much the same as Django's regular dispatch, but with extra hooks for startup, finalize, and exception handling.

pagination_class

alias of chord_metadata_service.restapi.pagination.LargeResultsSetPagination

serializer_class

alias of chord_metadata_service.experiments.serializers.ExperimentSerializer

chord_metadata_service.experiments.api_views.**get_experiment_schema**(*self*, *request*, **args*, ***kwargs*)

get: Experiment schema

5.5 Resources service

class chord_metadata_service.resources.api_views.**ResourceViewSet**(***kwargs*)

get: Return a list of all existing resources

post: Create a new resource

pagination_class

alias of chord_metadata_service.restapi.pagination.LargeResultsSetPagination

serializer_class

alias of chord_metadata_service.resources.serializers.ResourceSerializer

5.6 CHORD service

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

E

G

H

I

T

V