Katsu Metadata service

Release 2.11.0

Ksenia Zaytseva, David Lougheed, Simon Chénard, Romain Grég

Jun 03, 2022

CONTENTS:

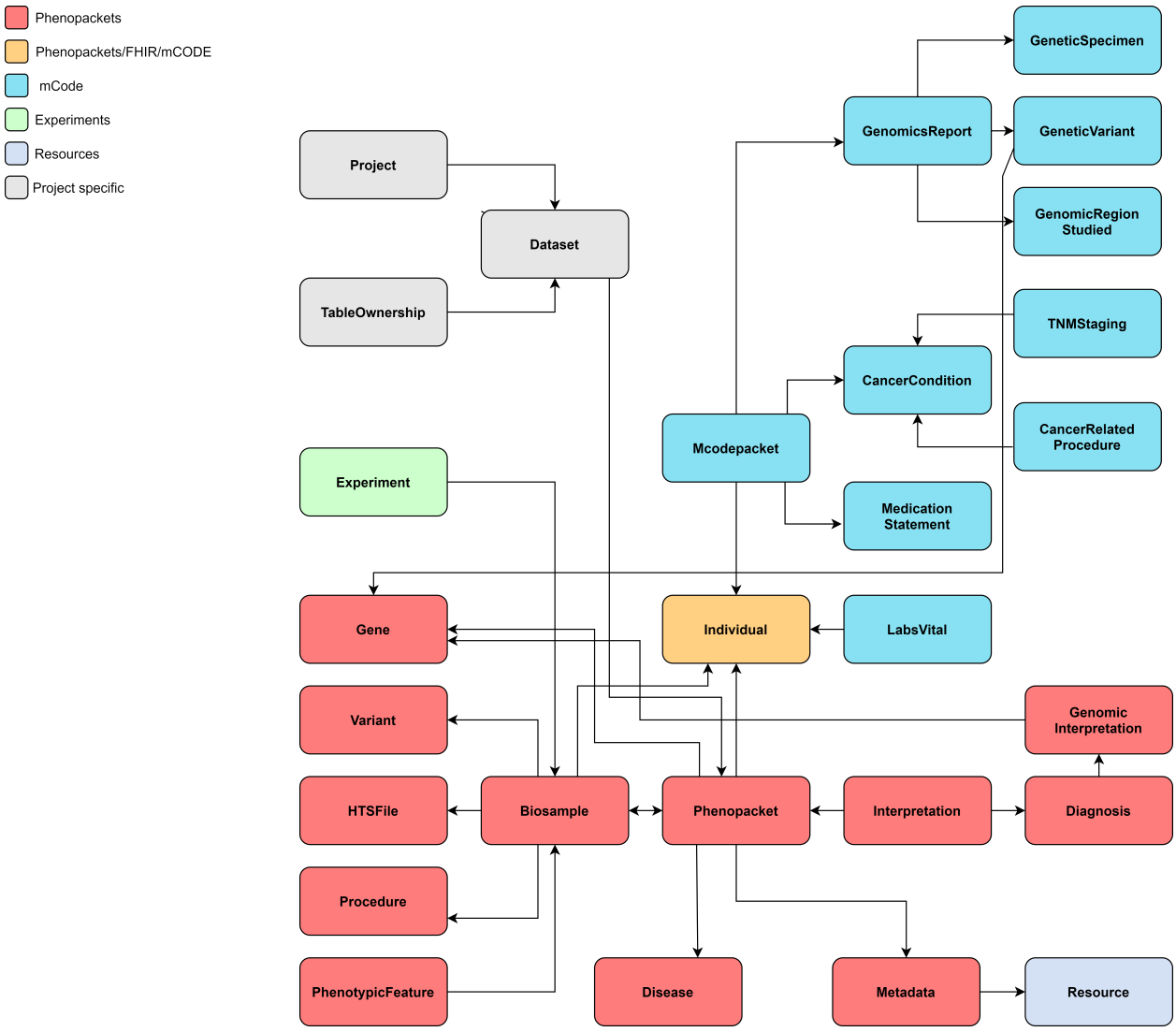
CHAPTER

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 and MINSEQE schema.

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.

MINSEQE (Minimum Information About Sequencing Experiment) schema 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:
  - FHIR renderer uses SMART on FHIR python client for Phenopackets and based on GA4GH FHIR Implementation Guide.

    Currently, the following classes can be retrieved in FHIR format by appending `?format=fhir`: Phenopacket, Individual, Biosample, PhenotypicFeature, HtsFile, Gene, Variant, Disease, Procedure.
  - RDF and JSON-LD renderers for Dataset metadata, based on DATS metadata context.

    The 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.
  - Custom ARGO renderer which is based on CanDIG mCODE to ARGO mappings.

    Currently, the following classes can be retrieved in ARGO format by appending `?format=argo`: GeneticSpecimen, CancerCondition, CancerRelatedProcedure, MedicationStatement, MCodePacket.

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. Experiments data ingest

The data must follow Experiments schema in order to be ingested.

Example of Experiments data POST request body:

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

3. mCode data ingest

The data must follow Katsu's mcode schema in order to be ingested.

Example of mCode data POST request body:

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

4. FHIR 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"
   }
}
```

5. 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",
```

(continues on next page)

(continued from previous page)

```
    "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

PATIENTS API

3.1 Data types endpoints

Individuals

`api/individuals` GET: list of individuals

`api/individuals/{id}` GET: single individual

The following **filters** can be used:

- id (single or multiple ids can be sent): `/api/individuals?id=10001&id=10002`
- alternate_ids (case-insensitive partial match): `/api/individuals?alternate_ids=10002-23`
- sex (case-insensitive exact match): `/api/individuals?sex=female` options: female, male, unknown_sex, other_sex
- karyotypic_sex (case-insensitive exact match): `/api/individuals?karyotypic_sex=xx` options: unknown_karyotype, XX, XY, XO, XXY, XXX, XXYY, XXXY, XXXX, XYY, other_karyotype
- active status: `/api/individuals?active=true` options: true, false
- deceased status: `/api/individuals?deceased=false` options: true, false
- ethnicity (case-insensitive partial match): `/api/individuals?ethnicity={value}`
- race (case-insensitive partial match): `/api/individuals?race={value}`
- date_of_birth (range filter): `/api/individuals?date_of_birth_after=1987-01-01&date_of_birth_before`
- disease (case-insensitive partial match for disease term label or disease id represented by URI or CURIE): for example, a disease recorded as `{"id": "SNOMED:840539006", "label": "COVID-19"}` can be searched

  1. by its label

  `/api/individuals?disease=covid`

  or

  2. by its CURIE

  `/api/individuals?disease=SNOMED:840539006`

- found_phenotypic_feature (case-insensitive partial match for phenotypic feature type label or id represented by URI or CURIE), finds all phenotypic feature with negated set to False: for example, a phenotypic feature recorded as `{"id": "HP:0000822", "label": "Hypertension"}` can be searched

  1. by its label

`/api/individuals?found_phenotypic_feature=hypertension`

or

2. by its CURIE

`/api/individuals?found_phenotypic_feature=HP:0000822`

- phenopackets__biosamples (single or multiple biosample ids), returns individuals linked to those biosamples:

  `/api/individuals?phenopackets__biosamples=2615-01&phenopackets__biosamples=2390-11`

- phenopackets (single or multiple phenopacket ids): `/api/individuals?phenopackets=10080&phenopackets=12045`

CHAPTER

FOUR

PHENOPACKETS API

4.1 Data types endpoints

Phenotypic features

`api/phenotypicfeatures` GET: list of phenotypic features

`api/phenotypicfeatures/{id}` GET: single phenotypic feature

The following **filters** can be applied:

- id (exact match, single): `/api/phenotypicfeatures?id=112002`
- negated: `/api/phenotypicfeatures?negated=false` options: true, false
- description (case-insensitive partial match): `/api/phenotypicfeatures?description=test`
- type (case-insensitive partial match): `/api/phenotypicfeatures?type=hypertension` or `/api/phenotypicfeatures?type=HP:0000822`
- severity (case-insensitive partial match): `/api/phenotypicfeatures?severity=mild` or `/api/phenotypicfeatures?severity=HP:0012825`
- onset (case-insensitive partial match): `/api/phenotypicfeatures?onset=adult` or `/api/phenotypicfeatures?onset=HP:0003581`
- evidence (case-insensitive partial match): `/api/phenotypicfeatures?evidence=author statement` or `/api/phenotypicfeatures?evidence=ECO:0006017`
- extra_properties (case-insensitive partial match): `/api/phenotypicfeatures?extra_properties=test`
- extra_properties_datatype (ONLY if "datatype" is present in extra_properties, case-insensitive partial match): `/api/phenotypicfeatures?extra_properties_datatype=comorbidities`
- individual (single or multiple individuals ids separated by comma), returns all phenotypic features for listed individuals: `/api/phenotypicfeatures?individual=10001,10002`
- biosample (single), returns phenotypic features that are related to a specified biosample: `/api/phenotypicfeatures?biosample=2615-01`
- phenopacket (single), returns phenotypic features that are related to a specified phenopacket: `/api/phenotypicfeatures?phenopacket=20110`
- datasets (single or multiple list of datasets titles separated by comma): `/api/phenotypicfeatures?datasets=dataset_1,dataset_2`
- authorized_datasets (single or multiple list of authorized datasets titles separated by comma): `/api/phenotypicfeatures?authorized_datasets=dataset_1,dataset_2`

Procedures

`api/procedures` GET: list of phenotypic features

`api/procedures/{id}` GET: single phenotypic feature

The following **filters** can be applied:

- id (exact match, single): `/api/procedures?id=112002`
- code (case-insensitive partial match): `/api/procedures?code=punch biopsy` or `/api/procedures?code=NCIT:C28743`
- body_site (case-insensitive partial match): `/api/procedures?body_site=skin of forearm` or `/api/procedures?body_site=UBERON:0003403`
- biosample (single), returns procedure that was performed on a specified biosample: `/api/procedures?biosample=2615-01`
- extra_properties (case-insensitive partial match): `/api/procedures?extra_properties=test`
- datasets (single or multiple list of datasets titles separated by comma): `/api/procedures?datasets=dataset_1,dataset_2`
- authorized_datasets (single or multiple list of authorized datasets titles separated by comma): `/api/procedures?authorized_datasets=dataset_1,dataset_2`

HTS Files

`api/htsfiles` GET: list of HTS files

`api/htsfiles/{uri}` GET: single HTS files

The following **filters** can be applied:

- uri (exact match, single): `/api/htsfiles?uri=drs://data/10001.vcf.gz`
- description (case-insensitive partial match): `/api/htsfiles?description=test`
- hts_format (case-insensitive exact match): `/api/htsfiles?hts_format=VCF` options: UNKNOWN, SAM, BAM, CRAM, VCF, BCF, GVCF
- genome_assembly (case-insensitive exact match): `/api/htsfiles?genome_assembly=GRCh37`
- extra_properties (case-insensitive partial match): `/api/htsfiles?extra_properties=test`
- datasets (single or multiple list of datasets titles separated by comma): `/api/htsfiles?datasets=dataset_1,dataset_2`
- authorized_datasets (single or multiple list of authorized datasets titles separated by comma): `/api/htsfiles?authorized_datasets=dataset_1,dataset_2`

Genes

`api/genes` GET: list of Genes

`api/genes/{id}` GET: single Gene

The following **filters** can be applied:

- id (single, exact match), takes an official identifier of the gene according to HGNC: `/api/genes?id=HGNC:347`
- symbol (single, exact match), takes an official symbol of the gene according to HGNC: `/api/genes?symbol=ETF1`
- extra_properties (case-insensitive partial match): `/api/genes?extra_properties=test`

- datasets (single or multiple list of datasets titles separated by comma): `/api/genes?datasets=dataset_1,dataset_2`
- authorized_datasets (single or multiple list of authorized datasets titles separated by comma): `/api/genes?authorized_datasets=dataset_1,dataset_2`

Variants

`api/variants` GET: list of Variants

`api/variants/{id}` GET: single Variants

The following **filters** can be applied:

- id (single, exact match): `/api/variants?id=100`
- allele_type (single, case-insensitive exact match): `/api/variants?allele_type=spdiAllele`
- zygosity (case-insensitive partial match): `/api/variants?zygosity=heterozygous` or `/api/variants?zygosity=GENO:0000135`
- extra_properties (case-insensitive partial match): `/api/variants?extra_properties=test`
- datasets (single or multiple list of datasets titles separated by comma): `/api/variants?datasets=dataset_1,dataset_2`
- authorized_datasets (single or multiple list of authorized datasets titles separated by comma): `/api/variants?authorized_datasets=dataset_1,dataset_2`

Diseases

`api/diseases` GET: list of Diseases

`api/diseases/{id}` GET: single Disease

The following **filters** can be applied:

- id (single, exact match), disease id in Katsu database: `/api/diseases?id=1`
- term (case-insensitive partial match): `/api/diseases?term=COVID-19` or `/api/diseases?term=SNOMED:840539006`
- extra_properties (case-insensitive partial match): `/api/diseases?extra_properties=test`
- extra_properties_datatype (ONLY if "datatype" is present in extra_properties, case-insensitive partial match): `/api/diseases?extra_properties_datatype=comorbidities`
- extra_properties_comorbidities_group (ONLY if "comorbidities_group" is present in extra_properties, case-insensitive partial match): `/api/diseases?extra_properties_comorbidities_group=common`
- datasets (single or multiple list of datasets titles separated by comma): `/api/diseases?datasets=dataset_1,dataset_2`
- authorized_datasets (single or multiple list of authorized datasets titles separated by comma): `/api/diseases?authorized_datasets=dataset_1,dataset_2`

Biosamples

`api/biosamples` GET: list of Biosamples

`api/biosamples/{id}` GET: single Biosample

The following **filters** can be applied:

- id (single, exact match): `/api/biosamples?id=1`
- description (case-insensitive partial match): `/api/biosamples?description=test`

- sampled_tissue (case-insensitive partial match): `/api/biosamples?sampled_tissue=urinary bladder` or `/api/biosamples?sampled_tissue=UBERON:0001256`
- taxonomy (case-insensitive partial match): `/api/biosamples?taxonomy=homo sapiens` or `/api/biosamples?taxonomy=NCBITaxon:9606`
- histological_diagnosis (case-insensitive partial match): `/api/biosamples?histological_diagnosis=negative finding` or `/api/biosamples?histological_diagnosis=NCIT:C38757`
- tumor_progression (case-insensitive partial match): `/api/biosamples?tumor_progression=primary neoplasm` or `/api/biosamples?tumor_progression=NCIT:C8509`
- tumor_grade (case-insensitive partial match): `/api/biosamples?tumor_grade=healed` or `/api/biosamples?tumor_grade=NCIT:C41133`
- individual (single, exact match, biosample must be related to Individual via ForeignKey not via Phenopacket): `/api/biosamples?individual=10001`
- procedure (single, exact match, searches by procedure id): `/api/biosamples?procedure=1`
- is_control_sample: `/api/biosamples?is_control_sample=false` options: true, false
- extra_properties (case-insensitive partial match): `/api/biosamples?extra_properties=test`
- datasets (single or multiple list of datasets titles separated by comma): `/api/biosamples?datasets=dataset_1,dataset_2`
- authorized_datasets (single or multiple list of authorized datasets titles separated by comma): `/api/biosamples?authorized_datasets=dataset_1,dataset_2`

Phenopackets

`api/phenopackets` GET: list of Phenopackets

`api/phenopackets/{id}` GET: single Phenopacket

The following **filters** can be applied:

- id (single, exact match): `/api/phenopackets?id=12000`
- subject (single, exact match), returns all phenopackets for a single individual: `/api/phenopackets?subject=10001`
- disease (case-insensitive partial match): `/api/phenopackets?disease=COVID-19` or `/api/phenopackets?disease=SNOMED:840539006`
- found_phenotypic_feature (case-insensitive partial match): `/api/phenopackets?found_phenotypic_feature=hypertension` or `/api/phenopackets?found_phenotypic_feature=HP:0000822`
- biosamples (single or multiple, exact match), takes biosample id, returns phenopacket(s) containing specified biosample(s): `/api/phenopackets?biosamples=2231-20&biosamples=1289-21`
- genes (single or multiple, exact match), returns phenopacket(s) containing specified gene(s): `/api/phenopackets?genes=HGNC:347`
- variants (single or multiple, exact match), returns phenopacket(s) containing specified variant(s): `/api/phenopackets?variants=100&variants=101`
- hts_files (single or multiple, exact match), returns phenopacket(s) containing specified hts_file(s): `/api/phenopackets?hts_files=drs://data/10001.vcf.gz&hts_files=drs://data/10002.vcf.gz`

- extra_properties (case-insensitive partial match): `/api/phenopackets?extra_properties=test`
- datasets (single or multiple list of datasets titles separated by comma): `/api/phenopackets?datasets=dataset_1,dataset_2`
- authorized_datasets (single or multiple list of authorized datasets titles separated by comma): `/api/phenopackets?authorized_datasets=dataset_1,dataset_2`

Genomic Interpretations

`api/genomicinterpretations` GET: list of Genomic Interpretations

`api/genomicinterpretations/{id}` GET: single Genomic Interpretation

The following **filters** can be applied:

- id (single, exact match): `/api/genomicinterpretations?id=1`
- gene (single, exact match): `/api/genomicinterpretations?gene=HGNC:347`
- variant (single, exact match): `/api/genomicinterpretations?variant=100`
- status (case-insensitive, exact match): `/api/genomicinterpretations?status=causative` options: Unknown, Rejected, Candidate, Causative
- extra_properties (case-insensitive partial match): `/api/genomicinterpretations?extra_properties=test`

Diagnoses

`api/diagnoses` GET: list of Diagnoses

`api/diagnoses/{id}` GET: single Diagnosis

The following **filters** can be applied:

- id (single, exact match): `/api/diagnoses?id=1`
- disease_type (case-insensitive partial match): `/api/diagnoses?disease_type=COVID-19` or `/api/diagnoses?disease_type=SNOMED:840539006`
- extra_properties (case-insensitive partial match): `/api/diagnoses?extra_properties=test`
- datasets (single or multiple list of datasets titles separated by comma): `/api/diagnoses?datasets=dataset_1,dataset_2`
- authorized_datasets (single or multiple list of authorized datasets titles separated by comma): `/api/diagnoses?authorized_datasets=dataset_1,dataset_2`

Interpretations

`api/interpretations` GET: list of Interpretations

`api/interpretations/{id}` GET: single Interpretation

The following **filters** can be applied:

- id (single, exact match): `/api/interpretations?id=1`
- resolution_status (case-insensitive, exact match): `/api/interpretations?resolution_status=causative` options: Unknown, Solved, Unsolved, In_progress
- phenopacket (single, exact match, searches by phenopacket id), returns all interpretations made for a specified phenopacket: `/api/interpretations?phenopacket=12000`
- extra_properties (case-insensitive partial match): `/api/interpretations?extra_properties=test`

- datasets (single or multiple list of datasets titles separated by comma): `/api/interpretations?datasets=dataset_1,dataset_2`
- authorized_datasets (single or multiple list of authorized datasets titles separated by comma): `/api/interpretations?authorized_datasets=dataset_1,dataset_2`

CHAPTER

FIVE

EXPERIMENTS API

5.1 Data types endpoints

Experiments

`api/experiments` GET: list of Experiments

`api/experiments/{id}` GET: single Experiment

The following **filters** can be applied:

- id (exact match, single): `/api/experiments?id=100`
- reference_registry_id (single, case-sensitive, exact match): `/api/experiments?reference_registry_id=RR10015`
- study_type (single, case-insensitive, partial match): `/api/experiments?study_type=genomics`
- experiment_type (single, case-insensitive, partial match): `/api/experiments?experiment_type=wes`
- molecule (single, case-insensitive, partial match): `/api/experiments?molecule=protein`
- library_strategy (single, case-insensitive, partial match): `/api/experiments?library_strategy=wes`
- library_source (single, case-insensitive, partial match): `/api/experiments?library_source=genomic`
- library_selection (single, case-insensitive, partial match): `/api/experiments?library_selection=random`
- library_layout (single, case-insensitive, partial match): `/api/experiments?library_layout=single`
- extraction_protocol (single, case-insensitive, partial match): `/api/experiments?extraction_protocol=exome capture`
- biosample (single, exact match), takes biosample id, returns all experimentsrelated to a specified biosample: `/api/experiments?biosample=1005`
- extra_properties (case-insensitive, partial match): `/api/experiments?extra_properties=test`
- datasets (single or multiple list of datasets titles separated by comma): `/api/experiments?datasets=dataset_1,dataset_2`

Experiment Results

`api/experimentresults` GET: list of Experiment Results

`api/experimentresults/{id}` GET: single Experiment Result

The following **filters** can be applied:

- identifier (single, exact match): `/api/experimentresults?identifier=RN-1001`
- description (single, case-insensitive, partial match): `/api/experimentresults?description=test`
- filename (single, case-insensitive, partial match): `/api/experimentresults?filename=1001_rnaseq.bw`
- genome_assembly_id (single, case-insensitive, exact match): `/api/experimentresults?genome_assembly_id=GRCh37` options: GRCh37, GRCh38, GRCm38, GRCm39
- file_format (single, case-insensitive, exact match): `/api/experimentresults?file_format=VCF`
- data_output_type (single, case-insensitive, partial match): `/api/experimentresults?data_output_type=raw data`
- usage (single, case-insensitive, partial match): `/api/experimentresults?usage=visualized`
- created_by (single, case-insensitive, partial match): `/api/experimentresults?created_by=Admin`
- extra_properties (case-insensitive, partial match): `/api/experimentresults?extra_properties=test`
- datasets (single or multiple list of datasets titles separated by comma): `/api/experimentresults?datasets=dataset_1,dataset_2`

CHAPTER

SIX

MCODE API ENDPOINTS

6.1 Data types endpoints

All data elements

`api/{data element plural form}` GET: list of objects

`api/{data element plural form}/{id}` GET: single object

The following **filters** can be applied:

- datasets (single or multiple list of datasets titles separated by comma): `/api/{data element plural form}?datasets=dataset_1,dataset_2`
- authorized_datasets (single or multiple list of authorized datasets titles separated by comma): `/api/{data element plural form}?authorized_datasets=dataset_1,dataset_2`

Example

`api/geneticspecimens` GET: list of Genetic Specimens

`api/geneticspecimens/{id}` GET: single Genetic Specimen

GET single object:

`api/geneticspecimens/100-1`

Filtering:

`/api/geneticspecimens?datasets=dataset_1,dataset_2`

CHAPTER

SEVEN

CHORD API

7.1 Data types endpoints

Projects

`api/projects` GET: list of Projects

`api/projects/{id}` GET: single Project

Datasets

`api/datasets` GET: list of Datasets

`api/datasets/{id}` GET: single Dataset

Table ownerships

`api/table_ownership` GET: list of Table ownerships

`api/table_ownership/{id}` GET: single Table ownership

Tables

`api/tables` GET: list of Tables

`api/tables/{id}` GET: single Table

or

`tables` GET: list of Tables

`tables/{id}` GET: single Table

`tables/{id}/summary` GET: summary about data in the table

`tables/{id}/search` POST: query data in the table

7.2 Schemas for Data types

`data-types` GET: list of all data types available for ingestion

`data-types/{data_type_name}` GET: single data type schema

For example: `data-types/experiment`

`data-types/{data_type_name}/schema` GET: same as above but just data type schema, without data type id

7.3 Private search endpoints

`private/search` POST: returns phenopackets that fit the conditions, works on all phenopackets in database

`private/tables/{id}/search` POST: returns phenopackets from a specified table that fit the conditions

Example of POST request to search for all phenopackets that have disease Carcinoma

```
{
    "data_type": "phenopacket",
    "query": ["#ico", ["#resolve", "diseases", "[item]", "term", "label"], "Carcinoma
↪"]
}
```

Example of POST request to search for all experiments that have experiments results in VCF format

```
{
    "data_type": "experiment",
    "query": ["#eq", ["#resolve", "experiment_results", "[item]", "file_format"], "VCF
↪"]
}
```

7.4 Ingest endpoint

`private/ingest` POST: ingests data to database

Example of POST request to ingest phenopackets file

```
{
  "table_id": "{table_id}",
  "workflow_id": "phenopackets_json",
  "workflow_params": {
    "phenopacket_json.json_document": "path/phenopackets.json"
  },
  "workflow_outputs": {
    "json_document": "path/path.json"
  }
}
```

Example of POST request to ingest experiments file

```
{
  "table_id": "{table_id}",
  "workflow_id": "experiments_json",
  "workflow_params": {
    "experiments_json.json_document": "path/experiments.json"
  },
  "workflow_outputs": {
    "json_document": "path/experiments.json"
  }
}
```

Example of POST request to ingest mcodepackets file

```
{
  "table_id": "{table_id}",
  "workflow_id": "mcode_json",
  "workflow_params": {
    "mcode_json.json_document": "path/mcodepackets.json"
  },
  "workflow_outputs": {
    "json_document": "path/mcodepackets.json"
  }
}
```

7.5 Export endpoint

`private/export` POST: retrieves data from database

Example of POST request to retrieve data formatted in cbioportal format

```
{
  "format": "cbioportal",
  "object_type": "dataset",
  "object_id": "{dataset_id}",
  "output_path": "{path_to_local_directory_optional}"
}
```

7.6 Workflows endpoints

`workflows` GET: list of all available workflows

`workflows/{slug:workflow_id}` GET: single workflow schema

`workflows/{slug:workflow_id}.wdl` GET: returns a wdl file for a given workflow

CHAPTER

EIGHT

RESOURCES API

8.1 Data types endpoints

Resources

`api/resources` GET: list of resources

`api/resources/{id}` GET: single resource

The following **filters** can be used:

- name (single, case-insensitive, partial match): `/api/resources?name=NCBI Taxonomy`
- namespace_prefix (single, case-insensitive, exact match): `/api/resources?namespace_prefix=NCBITaxon`
- url (single, case-insensitive, exact match): `/api/resources?url=http://purl.obolibrary.org/obo/ncbitaxon.owl`
- iri_prefix (single, case-insensitive, exact match): `/api/resources?url=http://purl.obolibrary.org/obo/NCBITaxon_`

CHAPTER

NINE

OVERVIEW API

`api/overview` GET: returns an overview of all phenopackets, individuals and other related data types. The overview includes counts for individuals, unique diseases, phenotypic features, experiments and other information.

`api/mcode_overview` GET: returns an overview of mcode-based data. The overview includes counts for individuals, cancer conditions, cancer related procedures and cancer status.

CHAPTER

TEN

PUBLIC API

There are several public APIs to return data overview and perform a search that returns only objects count. The implementation of public APIs relies on a project customized configuration file (config.json) that must be placed in the base directory. Currently, there is an `example.config.json` located in `/katsu/chord_metadata_service` directory which is set to be the project base directory. The file can be copied, renamed to `config.json` and modified.

The `config.json` file contains fields that data providers would like to open for public access. If the `config.json` file is not set up/created it means there is no public data and no data will be available via these APIs.

10.1 Config file specification

The `config.json` file follows jsonschema specifications: it includes fields from katsu data model, defines their type and other attributes that determine how the data from these fields will be presented in the public response.

Jsonschema properties:

- "type" - defines a data type for this field, e.g. "number" or "string" (katsu's config accepts only number and string types)
- "format" - defines a string format, e.g. "date" to record date in the format of "2021-12-31"
- "enum" - defines a list of options for this field
- "title" - field's user-friendly name
- "description" - field's description

Custom properties:

- "bin_size" (number) - defines a bin size for numeric fields (where "type" is set to "number"), by default bin size is set to 10
- "queryable" (true/false) - defines if the field should be included in search, if set to false the field will only be shown as a chart
- "is_range" (true/false) - defines if this field can be searched using range search (e.g.min value and max value)
- "chart" (options: pie, bar)- defines a type of the chart to be used to visualize the data
- "taper_left" and "taper_right" (number) - defines the cut offs for the data to be shown in charts
- "units" (string) - defines unit value for numeric fields (e.g. "years", "mg/L")
- "minimum" (number) - defines the minimum value in this field
- "maximum" (number) - defines the maximum value in this field
- "index" (number) - defines properties display order in the UI

Example of the config.json

```
{
  "age": {
    "type": "number",
    "title": "Age",
    "bin_size": 10,
    "is_range": true,
    "queryable": true,
    "taper_left": 40,
    "taper_right": 60,
    "units": "years",
    "minimum": 0,
    "description": "Age at arrival"
  },
  "sex": {
    "type": "string",
    "enum": [
      "Male",
      "Female"
    ],
    "title": "Sex",
    "queryable": true,
    "description": "Sex at birth"
  },
  "extra_properties": {
    "date_of_consent": {
      "type": "string",
      "format": "date",
      "title": "Verbal consent date",
      "chart": "bar",
      "queryable": true,
      "description": "Date of initial verbal consent(participant, legal␣
↪representative or tutor), yyyy-mm-dd"
    }
  }
}
```

10.2 Public endpoints

The public APIs include the following endpoints:

1. `/api/public_search_fields` GET: returns `config.json` contents in a form of jsonschema.

   The response when public fields are not configured and config file is not provided: `{"message": "No public fields configured."}`

2. `/api/public_overview` GET: returns an overview that contains counts for each field of interest.

   The response when there is no public data available and config file is not provided: `{"message": "No public data available."}`

3. `/api/public` GET: returns a count of all individuals in database.

   The response when there is no public data available and config file is not provided: `{"message": "No public data available."}`

The response when there is no enough data that passes the project-custom threshold: `{"message": "Insufficient data available."}`

When count is less or equal to a project's custom threshold returns message that insufficient data available. Accepts search filters on the fields that are specified in the `config.json` file and set to "queryable". Currently, the following filters are written for the Individual model:

- sex: e.g. `/api/public?sex=female`
- age: search by age ranges e.g. `/api/public?age_range_min=20&age_range_max=30`
- extra_properties: e.g. `/api/public?extra_properties=[{"smoking": "Non-smoker"},{"covidstatus": "positive"}]`

The `extra_properties` is a JSONField without a schema. To allow searching content in this field the nested fields have to be added to the config file (see the config file example above). The query string must contain a list of objects where each object has a key-value pair representing a nested field name and a search value.

Examples of extra properties searches:

Search for items that have a type of string:

```
/api/public?extra_properties=[{"smoking": "Non-smoker"},{"death_dc": "deceased"},{
↪"covidstatus": "positive"}]
```

Search for items that contain date ranges:

```
/api/public?extra_properties=[{"date_of_consent": {"after": "2020-03-01", "before
↪": "2021-05-01"}}]
```

Search for items that contain numeric ranges:

```
/api/public?extra_properties=[{"lab_test_result_value": {"rangeMin": 5, "rangeMax
↪": 900}}]
```

Examples of combining extra properties search with other fields:

```
/api/public?sex=female&extra_properties=[{"covidstatus": "positive"}]
```

CHAPTER

ELEVEN

SCHEMAS API

`api/chord_phenopacket_schema` GET: returns katsu's phenopackets schema.

`api/experiment_schema` GET: returns katsu's experiments schema.

`api/mcode_schema` GET: returns katsu's mcode schema.

CHAPTER

TWELVE

AUTOCOMPLETE API

The autocomplete APIs can be used for autocomplete suggestions in search UI.

`api/disease_term_autocomplete` GET: returns all disease terms available in database.

`api/phenotypic_feature_type_autocomplete` GET: returns all phenotypic feature types available in database.

`api/biosample_sampled_tissue_autocomplete` GET: returns all biosample sample tissues available in database.

CHAPTER

THIRTEEN

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

FOURTEEN

MODELS

14.1 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

14.2 Patients service

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

exception DoesNotExist

exception MultipleObjectsReturned

14.3 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

14.4 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

14.5 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.

14.6 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

FIFTEEN

VIEWS

15.1 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.

15.2 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

class chord_metadata_service.patients.api_views.**PublicListIndividuals**(**kwargs)
View to return only count of all individuals after filtering.

15.3 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

15.4 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

15.5 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

15.6 CHORD service

CHAPTER

SIXTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

E

G

H

I

L

M

P

R

S

T

V