datameta

Leon Kuchenbecker, Kersten Breuer, Moritz Hahn, Ryan Hutchins

Feb 21, 2023

CONTENTS

Data submission made easy! DataMeta allows you to easily define sample sheet columns, value constraints for the sample sheet and columns which are associated with raw data file names.

DataMeta is quick and easy to deploy on your local infrastructure and scales for high numbers of users!

Contents:

CHAPTER

ONE

APPLICATION FEATURES

1.1 Introduction

The DataMeta application stack comprises three main applications:

- The web application (aka DataMeta)
- A memcached server
- A PostgreSQL database server

1.2 Relational data model

erdiag

The relational database model comprises the following entities:

- **User**

  A database user, uniquely identifiable by their email address. The attributes are pretty self explanatory. A user can be disabled but should not be deleted to maintain tracability of submitted records.

- **Group**

  Every user is associated with exactly one group.

- **File**

  A file corresponds to a single file submitted by the user. It holds the following attributes:

  - `name` - The original name of the file as submitted by the user
  - `name_storage` - The name under which the file is stored in the storage backend. The webapp uses a fixed naming scheme for this purpose which ensures that the filename is unique on the storage backend: `{file_id.rjust(10,'0')}_{user_id}_{group_id}_{file_size}_{md5_checksum}`
  - `checksum` - The MD5 checksum, computed immediately after upload
  - `filesize` - The filesize of the file
  - `checksum_crypt` - The checksum of the file after encryption
  - `filesize_crypt` - The filesize of the file after encryption
  - `user_id` - The UID of the user who uploaded the file
  - `group_id` - The GID of the user who uploaded the file (see User and Group)

- **MetaDatum**

  A metadatum describes a datum conceptually, it does not hold any data. It thus corresponds to a column in the submitted sample sheet, but not to the data held in that column. The corresponding table in the database is filled when values when the DataMeta instance is *configured* and typically remains static throughout the lifetime of the instance. However, there are scenarios in which one may want to add a new column in a later stage, modify the linting constraints defined for a column etc. Implementing this will require migration concepts for these events, e.g. old data may no longer pass linting.A metadatum has the following attributes:

  - `name` - The name of the metadata. This defines the text required in the header of the corresponding sample sheet column.
  - `regexp` - A regular expression. If specified, it's applied to the field during linting.
  - `short_description` - A message to display in the interface when the verification based on `regexp` failes, e.g. "Only three digits are allowed in this field".
  - `datetimefmt` - C standard date time format code string (see e.g. here). If specified, the application assumes that this field holds a datetime value. When reading sample sheets from plain text formats (CSV, TSV) or if the corresponding column in a submitted Excel sheet is formatted as plain text, this format string is applied to parse the column values. Additionally, this format string is applied to display the values of this column in the user interface. *Note that date, time and datetime values are always stored as plain text datetime values in ISO 8601 format internally. The missing component falls back to 00:00 and 1900-01-01 respectively.*
  - `datetimemode` - An enum holding DATE, TIME or DATETIME. If set, denotes that the field should be treated as the corresponding type internally.
  - `mandatory` - A flag indicating whether this field may be empty in sample sheets or not
  - `order` - An integer. When displaying the sample sheet in the user interface, the columns are shown in the order of their `order` values.
  - `isfile` - A flag indicating whether this column corresponds to a file.

- **MetaDatumRecord**

  Corresponds to one captured value of one metadatum. Every filled field in the sample sheet (except the header) corresponds to one MetaDatumRecord. Attributes:

  - `metadatum_id` - The ID of the metadatum this record corresponds to
  - `metadataset_id` - The MetaDataSet (see below) this record is part of.
  - `file_id` - If this metadatum record corresponds to a file and it has been submitted (see *Conceptual Notes*)
  - `value` - The plain text value of this metadatum

- **MetaDataSet**

  A metadataset corresponds to *one row* in the sample sheet. It groups all metadatum records that correspond to this row and additionally holds information about the owner of the metadata and whether it has been submitted (aka committed) or not. Attributes are:

  - `user_id` - The owner's UID
  - `group_id` - The owner's GID at the time of creation
  - `submission_id` - The submission ID once this metadata has been committed (see below).

- **Submission**

  A submission corresponds to the event of a user clicking the `COMMIT` button on the `/submit` view. It holds the time at which the submission took place.

- **AppSettings**

  A table holding application settings configured by the administrator running the DataMeta instance. This is currently not used.

1.3 Conceptual Notes

1. *All captured metadata is treated as text internally*

   The generic application design, i.e. that the metadata fields can be defined dynamically at runtime, implies that there cannot be a 1:1 mapping of the sample sheet into the relational data model in form of a corresponding relation / table, unless one wants to go into the realm of using `CREATE / ALTER TABLE` at runtime. The current data model design stores all captured values in one attribute of one relation (`metadatumrecord.value`), which makes type reflection on the data model level impossible. At the same time, we're anyway accepting plain text formats such as CSV or TSV as input, thus all values that are to be held must be serializable anyway. Thus the `value` field of the MetaDatumRecord entity is `TEXT / VARCHAR`.

2. *Files and MetaDatumRecords are detached until submission*

   Until submission (aka commit), the data model does not link files and metadatumrecords. The integration of the file names and file uploads into the sample sheet (*Pending annotated submissions*) is purely visual on the client side through name-based matching. This is also utilized internally, i.e. to differentiate pending files from files that have been submitted, as files do not have a direct relation to a submission themselves. Only when a data record is submitted, files get linked to metadatumrecords and those get linked to a submission (via metadataset).

3. *Both files and metadatasets have owners*

   The previous point, i.e. pending files and metadatasets not being connected, requires that ownership is documented both on the file and on the metadataset level.

4. *Who has access to what data*

   What is shown on the `/submit` view is private to a UID/GID combination. Other members of the same group cannot access the pending submission, neither can the user himself in case they change their group. After submission, in the `/view` view, the user can see all submitted data from his group (not yet implemented).

1.4 Application Cases

1.4.1 Data File Submission

erdiag

1.4.2 Sample Sheet Submission

1.4.3 Commit Submission

1.4.4 Delete Pending Metadata Record

1.4.5 Delete Pending Data File

CHAPTER

TWO

LOCAL INSTALLATION

Install dependencies, e.g. via brew.

```
brew install postgresql memcached libmemcached npm
```

Register postgresql as a service

```
brew services start postgresql
```

Clone this repository:

```
git clone https://github.com/ghga-de/datameta.git
```

Change directory into your newly created project if not already there. Your current directory should be the same as this README.md file and setup.py.

```
cd datameta
```

Create a Python virtual environment, if not already created.

This can be done via `venv`:

```
python3 -m venv <environment_name>
```

or `conda`:

```
conda create -y -n <environment_name> 'python>3'
```

Activate the environment

With `venv` (docs):

```
source <path/to/environment>/bin/activate
```

With conda

```
conda activate <environment_name>
```

Upgrade packaging tools, if necessary.

```
pip install --upgrade pip setuptools
```

Install NPM dependencies

```
npm install --prefix datameta/static/
```

Install the project in editable mode with its testing requirements.

```
pip install -e ".[testing]"
```

Create a postgresql database, then add the path to the database to the `development.ini` found in `datameta/config`, e.g. `sqlalchemy.url = postgresql://localhost/<dbname>`. A copy of the `development.ini` can be placed at an arbitrary location named `/path/to/config/` in the following.

```
createdb <dbname>
```

Initialize the database using Alembic.

```
<path/to/environment>/bin/alembic -c <path/to/config>/development.ini upgrade head
```

Load default data into the database using a script. Change the initial user and group information to your requirements.

```
<path/to/environment>/bin/initialize_datameta_db \
        -c <path/to/config>/development.ini \
        --initial-user-fullname "First User Fullname" \
        --initial-user-email "first@user.email" \
        --initial-user-pass "initialPassword" \
        --initial-group "My Organization"
```

Run your project's tests.

```
<path/to/environment>/bin/pytest
```

Start the `memcached` process.

```
nohup memcached &
```

Run your project.

```
<path/to/environment>/bin/pserve /path/to/config/development.ini
```

CHAPTER

THREE

VISUAL STUDIO CODE / DOCKER INSTALLATION

Installation instructions for using the remote container feature of Visual Studio Code (vscode)

The remote container feature allows to run the editor's backend inside a docker container, which is readily set up for development.

3.1 Quickstart

Clone this repository:

```
git clone https://github.com/ghga-de/datameta.git
```

And open the created directory in vscode, for instance like that:

```
code ./datameta
```

Install the remote development extension:

- click on the extensions symbol in the side bar
- search for `Remote Development` and install it

To reopen vscode inside the dev container:

- select `View > Command Palette` in the dropdown menu
- then select (or type): `Remote-Containers: Reopen in Container`

If you are executing this for the first time, the containers will be set up via docker-compose. This might take some time.

3.2 Using the container

Once the build is successful, you will be able to use vscode as usual. The workspace will be mounted at `/workspace`.

However, before you start, you have to first install datameta in edit mode. Just type in the terminal:

```
dev_install
```

(this will execute the script at `/workspace/docker/dev_install`) You only have to run this once (unless you re-build the container or want to re-install datameta).

Every time you would like to deploy datameta, just type:

```
dev_launcher
```

(this will execute the script at `/workspace/docker/dev_launcher`)

The frontend should be available at `http://localhost:8080/` in your browser.

3.3 Configuration

Any configuration regarding the dev container environment can be found at `/.devcontainer`.

The environment includes a few useful vscode extensions out of the box. If you find a extension that might be of use to everybody, feel free to add it to the `/.devcontainer/devcontainer.json`.

For general information on this vscode feature please look here.

CHAPTER

FOUR

REST API DESIGN

If not stated otherwise, all resources managed by the API are named according to the corresponding database model.

The first draft of the API focusses on managing resources which users typically iteract with during the data submission process. This primarily includes the following resources:

- metadatasets
- submissions

User/auth related resources (especially `users` and `groups` might follow later) for automation of adiministation might follow.

4.1 URL definitions by resource

4.1.1 metadatasets

`GET /api/metadatasets`

- get all metadatasets
- only allowed for admins (since it will return entries for all groups)

`POST /api/metadatasets`

- create new metadataset

`GET/PUT/DELETE /api/metadatasets/{metadataset_id}`

- get/update/delete metadataset

4.1.2 submissions

`GET /api/submissions`

- get all submissions
- only allowed for admins (since it will return entries for all groups)

`POST /api/submissions`

- create new submission that consists of a list of metadataset IDs and a list of file IDs

`GET/PUT/DELETE /api/submissions/{submission_id}`

- get/update/delete submission

4.1.3 files

Only the file entries in the database will be managed by the REST API. The actual file upload should take place by directly talking to another service like S3.

`GET /api/files`

- get all files
- only allowed for admins (since it will return entries for all groups)

`POST /api/files`

- create new file
- this will only create a new file entry in the database and for instance give back a presigned URL to S3 for upload
- the actual file content shall not be part of this request

`GET/PUT/DELETE /api/files/{file_id}`

- get/update/delete file

4.1.4 groups

`GET /api/groups/{group_id}`

- get info about a group

`GET /api/groups/{group_id}/users`

- lists the user's IDs which are part of this group

`GET /api/groups/{group_id}/submissions`

- lists the submission's IDs which are part of this group

4.1.5 users

`GET /api/users/{user_id}`

- get info about a user (e.g. which group he/she is part of)

`GET /api/users/{user_id}/pending_metadatasets`

- lists the pending metadatasets of that user

`GET /api/users/{user_id}/pending_files`

- lists the pending files of that user

CHAPTER

FIVE

OPEN API SPECS

`POST /keys`

Create new API Key/Token

Create new API Key/Token

Example request:

```
POST /keys HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "email": "string",
    "password": "string",
    "label": "string",
    "expires": "string"
}
```

Status Codes

- 200 OK – OK

  Example response:

  ```
  HTTP/1.1 200 OK
  Content-Type: application/json

  {
      "id": {
          "uuid": "string",
          "site": "string"
      },
      "userId": {
          "uuid": "string",
          "site": "string"
      },
      "token": "string",
      "label": "string",
      "expires": "string"
  }
  ```

- 401 Unauthorized – Unauthorized

- 403 Forbidden – Forbidden
- 400 Bad Request – Validation Error

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 500 Internal Server Error – Internal Server Error

`DELETE /keys/{id}`

Delete ApiKey by label

Delete ApiKey by label.

Parameters

- **id** (*string*) – ID (not label) of Apikey

Status Codes

- 200 OK – OK
- 401 Unauthorized – Unauthorized
- 403 Forbidden – Forbidden
- 404 Not Found – The specified key does not exist.
- 400 Bad Request – Validation Error

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
```

(continues on next page)

(continued from previous page)

```
        "field": "string"
    }
]
```

- 500 Internal Server Error – Internal Server Error

`GET /rpc/whoami`

[Not RESTful]: Returns information about the authenticated user

Returns the ids, name, groupAdmin, siteAdmin, email and groupName attributes for the logged in user. [Attention this endpoint is not RESTful, the result should not be cached.]

Example request:

```
GET /rpc/whoami HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – OK

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": {
        "uuid": "string",
        "site": "string"
    },
    "name": "string",
    "groupAdmin": true,
    "siteAdmin": true,
    "siteRead": true,
    "email": "string",
    "group": {
        "id": {
            "uuid": "string",
            "site": "string"
        },
        "name": "string"
    }
}
```

- 401 Unauthorized – Unauthorized
- 400 Bad Request – Validation Error

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
```

(continues on next page)

(continued from previous page)

```
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 500 Internal Server Error – Internal Server Error

`POST /rpc/delete-files`

Bulk-delete Staged Files

Bulk-delete Staged Files.

Example request:

```
POST /rpc/delete-files HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "fileIds": [
        "string"
    ]
}
```

Status Codes

- 204 No Content – Deletion successful
- 401 Unauthorized – Unauthorized
- 403 Forbidden – Forbidden
- 404 Not Found – File not found
- 400 Bad Request – Validation Error

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
```

(continues on next page)

(continued from previous page)

```
        },
        "field": "string"
    }
]
```

- 500 Internal Server Error – Internal Server Error

`POST /rpc/delete-metadatasets`

Bulk-delete Staged MetaDataSets

Bulk-delete Staged MetaDataSets.

Example request:

```
POST /rpc/delete-metadatasets HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "metadatasetIds": [
        "string"
    ]
}
```

Status Codes

- 204 No Content – Deletion successful
- 401 Unauthorized – Unauthorized
- 403 Forbidden – Forbidden
- 404 Not Found – File not found
- 400 Bad Request – Validation Error

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 500 Internal Server Error – Internal Server Error

`GET /rpc/get-file-url/{id}`

[Not RESTful]: Redirects to a temporary, pre-signed HTTP-URL for downloading a file.

For the file with the given ID, this enpoint will redirect to a pre-signed HTTP URL for downloading the requested file. The pre-signed URL times out after a certain amount of time which can be configured with the "expires" query string. [Attention this endpoint is not RESTful, the result should not be cached.]

Parameters

- **id** (*string*) – ID of the file

Query Parameters

- **expires** (*integer*) – Minutes until the pre-signed URL will expire, defaults to 1
- **redirect** (*boolean*) – If set to true, the endpoint will return a 307 response, otherwise a 200 response.

Example request:

```
GET /rpc/get-file-url/{id} HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – OK

  Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "fileId": {
        "uuid": "string",
        "site": "string"
    },
    "fileUrl": "string",
    "expires": "2023-02-21T08:22:46.694531",
    "checksum": "string"
}
```

- 307 Temporary Redirect – Redirecting to the pre-signed URL of the file

  Example response:

```
HTTP/1.1 307 Temporary Redirect
Content-Type: application/json

{
    "fileId": {
        "uuid": "string",
        "site": "string"
    },
    "fileUrl": "string",
    "expires": "2023-02-21T08:22:46.694531",
    "checksum": "string"
}
```

- 400 Bad Request – Validation Error

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 401 Unauthorized – Unauthorized
- 403 Forbidden – Forbidden
- 404 Not Found – The specified file does not exist.
- 500 Internal Server Error – Internal Server Error

Response Headers

- **location** – Location to redirect to

GET /users/{id}/keys

All API keys for a user

Get a list of all API keys for a user. Please note that you cannot retrieve the tokens themselves because they are stored in a hashed format in our database as only the respective user is allowed to know them.

Parameters

- **id** (*string*) – ID of the User

Example request:

```
GET /users/{id}/keys HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – OK

  Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": {
```

(continues on next page)

(continued from previous page)

```
            "uuid": "string",
            "site": "string"
        },
        "label": "string",
        "expires": "string",
        "hasExpired": true
    }
]
```

- 401 Unauthorized – Unauthorized
- 403 Forbidden – Forbidden
- 404 Not Found – The specified user does not exist.
- 400 Bad Request – Validation Error

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 500 Internal Server Error – Internal Server Error

`PUT /users/{id}/password`

Update a user's password

Update a user's password. The user ID can be specified either as a UUID or as a site ID.

Parameters

- **id** (*string*) – User ID, either as UUID or as site ID. '0' for password reset token based access.

Example request:

```
PUT /users/{id}/password HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "passwordChangeCredential": "string",
```

(continues on next page)

(continued from previous page)

```
    "newPassword": "string"
}
```

Status Codes

- 200 OK – Password update successful

  Example response:

  ```
  HTTP/1.1 200 OK
  Content-Type: application/json

  {
      "tfaToken": "string"
  }
  ```

- 401 Unauthorized – Unauthorized
- 403 Forbidden – The specified user does not exist or is not the same user as the authorized user.
- 404 Not Found – Password reset token not found
- 410 Gone – Password reset token expired
- 400 Bad Request – Validation Error

  Example response:

  ```
  HTTP/1.1 400 Bad Request
  Content-Type: application/json

  [
      {
          "exception": "string",
          "error_code": "string",
          "message": "string",
          "entity": {
              "uuid": "string",
              "site": "string"
          },
          "field": "string"
      }
  ]
  ```

- 500 Internal Server Error – Internal Server Error

`DELETE /users/{id}/totp-secret`

Delete TOTP-secret for user

Delete TOTP-secret.

Parameters

- **id** (*string*) – User ID

Status Codes

- 200 OK – OK
- 401 Unauthorized – Unauthorized
- 403 Forbidden – Forbidden
- 404 Not Found – The specified user does not exist.
- 400 Bad Request – Validation Error

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 500 Internal Server Error – Internal Server Error

`GET /users/{id}`

Get user information

Get information about a user.

Parameters

- **id** (*string*) – ID of the user

Example request:

```
GET /users/{id} HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – OK

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": {
        "uuid": "string",
        "site": "string"
    },
    "name": "string",
```

(continues on next page)

(continued from previous page)

```
    "groupAdmin": true,
    "siteAdmin": true,
    "siteRead": true,
    "email": "string",
    "groupId": {
        "uuid": "string",
        "site": "string"
    }
}
```

- 400 Bad Request – Validation Error

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 401 Unauthorized – Unauthorized
- 403 Forbidden – Forbidden
- 404 Not Found – Not found
- 500 Internal Server Error – Internal Server Error

`PUT /users/{id}`

Update a user's credentials and status

Update a user's name, group, admin status and enabled status.

Parameters

- **id** (*string*) – User ID

Example request:

```
PUT /users/{id} HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "name": "string",
    "groupId": "string",
```

(continues on next page)

(continued from previous page)

```
    "groupAdmin": true,
    "siteAdmin": true,
    "siteRead": true,
    "enabled": true
}
```

Status Codes

- 204 No Content – User update successful
- 401 Unauthorized – Unauthorized
- 403 Forbidden – This user does not have the rights to perform this action.
- 404 Not Found – The specified user does not exist.
- 400 Bad Request – Validation Error

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 500 Internal Server Error – Internal Server Error

`GET /metadata`

Get metadata definitions

Get the metadata definitions that are configured for this site.

Example request:

```
GET /metadata HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – OK

  Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json
```

(continues on next page)

(continued from previous page)


```
[
    {
        "id": {
            "uuid": "string",
            "site": "string"
        },
        "name": "string",
        "regexDescription": "string",
        "longDescription": "string",
        "example": "string",
        "regExp": "string",
        "dateTimeFmt": "string",
        "isMandatory": true,
        "order": 1,
        "isFile": true,
        "isSubmissionUnique": true,
        "isSiteUnique": true,
        "serviceId": {
            "uuid": "string",
            "site": "string"
        }
    }
]
```


- 401 Unauthorized – Unauthorized
- 500 Internal Server Error – Internal Server Error

`POST /metadata`

Create a New MetaDatum

Create a new MetaDatum. This is an administrative Endpoint that is not accessible for regular users.

Example request:

```
POST /metadata HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "name": "string",
    "regexDescription": "string",
    "longDescription": "string",
    "example": "string",
    "regExp": "string",
    "dateTimeFmt": "string",
    "isMandatory": true,
    "order": 1,
    "isFile": true,
    "isSubmissionUnique": true,
    "isSiteUnique": true,
    "serviceId": "string"
}
```

Status Codes

- 200 OK – OK

  Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": {
        "uuid": "string",
        "site": "string"
    },
    "name": "string",
    "regexDescription": "string",
    "longDescription": "string",
    "example": "string",
    "regExp": "string",
    "dateTimeFmt": "string",
    "isMandatory": true,
    "order": 1,
    "isFile": true,
    "isSubmissionUnique": true,
    "isSiteUnique": true,
    "serviceId": {
        "uuid": "string",
        "site": "string"
    }
}
```

- 401 Unauthorized – Unauthorized
- 403 Forbidden – Forbidden
- 404 Not Found – The specified Service does not exist.
- 400 Bad Request – Validation Error

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 500 Internal Server Error – Internal Server Error

`PUT /metadata/{id}`

Update a MetaDatum

Update a MetaDatum. This is an administrative Endpoint that is not accessible for regular users.

Parameters

- **id** (*string*) – User ID

Example request:

```
PUT /metadata/{id} HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "name": "string",
    "regexDescription": "string",
    "longDescription": "string",
    "example": "string",
    "regExp": "string",
    "dateTimeFmt": "string",
    "isMandatory": true,
    "order": 1,
    "isFile": true,
    "isSubmissionUnique": true,
    "isSiteUnique": true,
    "serviceId": "string"
}
```

Status Codes

- 200 OK – OK

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": {
        "uuid": "string",
        "site": "string"
    },
    "name": "string",
    "regexDescription": "string",
    "longDescription": "string",
    "example": "string",
    "regExp": "string",
    "dateTimeFmt": "string",
    "isMandatory": true,
    "order": 1,
    "isFile": true,
    "isSubmissionUnique": true,
```

(continues on next page)

(continued from previous page)

```
    "isSiteUnique": true,
    "serviceId": {
        "uuid": "string",
        "site": "string"
    }
}
```

- 401 Unauthorized – Unauthorized
- 403 Forbidden – This user does not have the rights to perform this action.
- 404 Not Found – The specified service does not exist.
- 400 Bad Request – Validation Error

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 500 Internal Server Error – Internal Server Error

`POST /metadatasets`

Create a New MetaDataSet

Create a new MetaDataSet

Example request:

```
POST /metadatasets HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "record": {}
}
```

Status Codes

- 200 OK – OK

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "record": {},
    "fileIds": {},
    "serviceExecutions": {},
    "id": {
        "uuid": "string",
        "site": "string"
    },
    "submissionId": {
        "uuid": "string",
        "site": "string"
    },
    "userId": {
        "uuid": "string",
        "site": "string"
    }
}
```

- 401 Unauthorized – Unauthorized
- 400 Bad Request – Validation Error

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 500 Internal Server Error – Internal Server Error

`GET /metadatasets`

Query metadatasets

Query all metadatasets

Query Parameters

- **submittedAfter** (*string*) – ISO datetime string specifying an exclusive lower bound for the submission date of the returned metadatasets.

- **submittedBefore** (*string*) – ISO datetime string specifying an exclusive upper bound for the submission date of the returned metadatasets.
- **awaitingService** (*string*) – Identifier for a service. Restricts the result to metadatsets for which the specified service has not been executed yet.

Example request:

```
GET /metadatasets HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – OK

  Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "record": {},
        "fileIds": {},
        "serviceExecutions": {},
        "id": {
            "uuid": "string",
            "site": "string"
        },
        "submissionId": {
            "uuid": "string",
            "site": "string"
        },
        "userId": {
            "uuid": "string",
            "site": "string"
        }
    }
]
```

- 401 Unauthorized – Unauthorized
- 404 Not Found – Not Found
- 400 Bad Request – Validation Error

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
```

(continues on next page)

(continued from previous page)

```
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 500 Internal Server Error – Internal Server Error

`GET /metadatasets/{id}`

Get Details for a MetaDataSet

Get details for a metadataset.

Parameters

- **id** (*string*) – ID of the MetaDataSet

Example request:

```
GET /metadatasets/{id} HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – OK

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "record": {},
    "fileIds": {},
    "serviceExecutions": {},
    "id": {
        "uuid": "string",
        "site": "string"
    },
    "submissionId": {
        "uuid": "string",
        "site": "string"
    },
    "userId": {
        "uuid": "string",
        "site": "string"
    }
}
```

- 401 Unauthorized – Unauthorized
- 403 Forbidden – Forbidden
- 400 Bad Request – Validation Error

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 404 Not Found – The specified metadataset does not exist.
- 500 Internal Server Error – Internal Server Error

`DELETE /metadatasets/{id}`

Delete Not-Submitted Metadataset

Delete File. Please note: This is only allowed if the metadataset has not been part of a Submission, yet.

Parameters

- **id** (*string*) – ID of the Metadataset

Status Codes

- 204 No Content – Deletion successful
- 401 Unauthorized – Unauthorized
- 403 Forbidden – Either forbidden or the resource is not modifiable

Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 404 Not Found – The specified metadataset does not exist.
- 400 Bad Request – Validation Error

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 500 Internal Server Error – Internal Server Error

`POST /service-execution/{serviceId}/{metadatasetId}`

Endpoint to store the result of a service execution for a single metadataset

This endpoint is used to report the result of a service execution in the form of metadatum key - value pairs for the service-related metadata and corresponding files if file metadata are involved.

Parameters

- **serviceId** (*string*) – ID of the service
- **metadatasetId** (*string*) – ID of the metadataset

Example request:

```
POST /service-execution/{serviceId}/{metadatasetId} HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "record": {},
    "fileIds": [
        "string"
    ]
}
```

Status Codes

- 200 OK – Update successful

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "record": {},
    "fileIds": {},
```

(continues on next page)

(continued from previous page)

```
    "serviceExecutions": {},
    "id": {
        "uuid": "string",
        "site": "string"
    },
    "submissionId": {
        "uuid": "string",
        "site": "string"
    },
    "userId": {
        "uuid": "string",
        "site": "string"
    }
}
```

- 401 Unauthorized – Unauthorized
- 403 Forbidden – Forbidden
- 404 Not Found – The specified metadataset or service does not exist.
- 400 Bad Request – Validation Error

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 500 Internal Server Error – Internal Server Error

POST /files

Create a New File

Creates a new empty file object. Attention: this endpoint does not take the file content for upload. Instead, it will respond with a presigned URL which you can use to upload (PUT) your file content.

Example request:

```
POST /files HTTP/1.1
Host: example.com
Content-Type: application/json
```

(continues on next page)

(continued from previous page)

```
{
    "name": "string",
    "checksum": "string"
}
```

Status Codes

- 200 OK – OK

  Example response:

  ```
  HTTP/1.1 200 OK
  Content-Type: application/json

  {
      "id": {
          "uuid": "string",
          "site": "string"
      },
      "name": "string",
      "urlToUpload": "string",
      "requestHeaders": {},
      "userId": {
          "uuid": "string",
          "site": "string"
      },
      "expires": "string"
  }
  ```

- 400 Bad Request – Validation Error

  Example response:

  ```
  HTTP/1.1 400 Bad Request
  Content-Type: application/json

  [
      {
          "exception": "string",
          "error_code": "string",
          "message": "string",
          "entity": {
              "uuid": "string",
              "site": "string"
          },
          "field": "string"
      }
  ]
  ```

- 401 Unauthorized – Unauthorized
- 500 Internal Server Error – Internal Server Error

GET /files/{id}

Get Details for A File

Get details for a file.

Parameters

- **id** (*string*) – ID of the File

Example request:

```
GET /files/{id} HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – OK

  Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": {
        "uuid": "string",
        "site": "string"
    },
    "name": "string",
    "contentUploaded": true,
    "checksum": "string",
    "filesize": 1,
    "userId": {
        "uuid": "string",
        "site": "string"
    },
    "expires": "string"
}
```

- 401 Unauthorized – Unauthorized
- 403 Forbidden – Either forbidden or the resource is not modifiable

  Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
```

(continues on next page)

(continued from previous page)

```
        "field": "string"
    }
]
```

- 404 Not Found – File not found
- 400 Bad Request – Validation Error

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 500 Internal Server Error – Internal Server Error

`PUT /files/{id}`

Update File Details

Update details for a File. E.g. to indicate that the File content has been uploaded (set contentUploaded=true). Please note: this only works for Files that have not been submitted, yet. Other file attributes (checksum and name) can only be updated until contentUploaded has been set to 'true'.

Parameters

- **id** (*string*) – ID of the File

Example request:

```
PUT /files/{id} HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "name": "string",
    "contentUploaded": true,
    "checksum": "string"
}
```

Status Codes

- 200 OK – OK

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": {
        "uuid": "string",
        "site": "string"
    },
    "name": "string",
    "contentUploaded": true,
    "checksum": "string",
    "filesize": 1,
    "userId": {
        "uuid": "string",
        "site": "string"
    },
    "expires": "string"
}
```

- 401 Unauthorized – Unauthorized
- 403 Forbidden – Either forbidden or the resource is not modifiable

Example response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 404 Not Found – File not found
- 409 Conflict – Mismatch between uploaded data checksum and announced checksum
- 400 Bad Request – Validation Error

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
```

(continues on next page)

(continued from previous page)

```
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 500 Internal Server Error – Internal Server Error

`DELETE /files/{id}`

Delete Not-Submitted File

Delete File. Please note: This is only allowed if the File has not been part of a Submission, yet.

Parameters

- **id** (*string*) – ID of the File

Status Codes

- 204 No Content – Deletion successful
- 401 Unauthorized – Unauthorized
- 403 Forbidden – Forbidden
- 404 Not Found – File not found
- 400 Bad Request – Validation Error

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 500 Internal Server Error – Internal Server Error

`POST /submissions`

Create a New Submission

Creates a new Submission. A submission consists of a list of metadatasets and a list of files.

Example request:

```
POST /submissions HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "metadatasetIds": [
        "string"
    ],
    "fileIds": [
        "string"
    ],
    "label": "string"
}
```

Status Codes

- 200 OK – OK

  Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": {
        "uuid": "string",
        "site": "string"
    },
    "label": "string",
    "metadatasetIds": [
        {
            "uuid": "string",
            "site": "string"
        }
    ],
    "fileIds": [
        {
            "uuid": "string",
            "site": "string"
        }
    ]
}
```

- 400 Bad Request – Validation Error

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
```

(continues on next page)

(continued from previous page)

```
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 401 Unauthorized – Unauthorized
- 500 Internal Server Error – Internal Server Error

POST /presubvalidation

Pre-validate a submission

Pre-validates a submission request without actually creating a submission. A submission request consists of a list of metadatasets and a list of files.

Example request:

```
POST /presubvalidation HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "metadatasetIds": [
        "string"
    ],
    "fileIds": [
        "string"
    ],
    "label": "string"
}
```

Status Codes

- 204 No Content – OK
- 400 Bad Request – Validation Error

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
```

(continues on next page)

(continued from previous page)

```
        "field": "string"
    }
]
```

- 401 Unauthorized – Unauthorized
- 500 Internal Server Error – Internal Server Error

GET /groups/{id}/submissions

Get A List of All Submissions of A Group.

Get a list of all submissions of a group.

Parameters

- **id** (*string*) – ID of the group

Example request:

```
GET /groups/{id}/submissions HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – OK

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": {
            "uuid": "string",
            "site": "string"
        },
        "label": "string",
        "metadatasetIds": [
            {
                "uuid": "string",
                "site": "string"
            }
        ],
        "fileIds": [
            {
                "uuid": "string",
                "site": "string"
            }
        ]
    }
]
```

- 401 Unauthorized – Unauthorized
- 403 Forbidden – Forbidden

- 404 Not Found – The specified group does not exist.
- 400 Bad Request – Validation Error

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 500 Internal Server Error – Internal Server Error

`GET /groups/{id}`

Get group information

Get information about a group.

Parameters

- **id** (*string*) – ID of the group

Example request:

```
GET /groups/{id} HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – OK

  Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": {
        "uuid": "string",
        "site": "string"
    },
    "name": "string"
}
```

- 400 Bad Request – Validation Error

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 401 Unauthorized – Unauthorized
- 403 Forbidden – Forbidden
- 404 Not Found – Not found
- 500 Internal Server Error – Internal Server Error

`PUT /groups/{id}`

Change the name of a group.

Change the name of a group.

Parameters

- **id** (*string*) – ID of the group

Example request:

```
PUT /groups/{id} HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "name": "string"
}
```

Status Codes

- 204 No Content – No Content
- 401 Unauthorized – Unauthorized
- 403 Forbidden – Forbidden
- 404 Not Found – The specified group does not exist.
- 400 Bad Request – Validation Error

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 500 Internal Server Error – Internal Server Error

`GET /appsettings`

GET all AppSettings

GET all AppSettings. This is an administrative Endpoint that is not accessible for regular users.

Example request:

```
GET /appsettings HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – OK

  Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": {
            "uuid": "string",
            "site": "string"
        },
        "key": "string",
        "valueType": "string",
        "value": "string"
    }
]
```

- 401 Unauthorized – Unauthorized
- 403 Forbidden – Forbidden
- 400 Bad Request – Validation Error

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 500 Internal Server Error – Internal Server Error

`POST /services`

Create a new service.

Create a new service.

Example request:

```
POST /services HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "name": "string"
}
```

Status Codes

- 200 OK – OK

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": {
        "uuid": "string",
        "site": "string"
    },
    "name": "string",
    "userIds": [
        {
            "uuid": "string",
            "site": "string"
        }
    ]
}
```

- 400 Bad Request – Validation Error

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 401 Unauthorized – Unauthorized
- 403 Forbidden – Forbidden
- 500 Internal Server Error – Internal Server Error

`GET /services`

Get Services information

Get names and IDs for all services

Example request:

```
GET /services HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – OK

  Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": {
            "uuid": "string",
            "site": "string"
        },
        "name": "string",
        "userIds": [
            {
                "uuid": "string",
                "site": "string"
```

(continues on next page)

(continued from previous page)

```
            }
        ]
    }
]
```

- 401 Unauthorized – Unauthorized
- 403 Forbidden – Forbidden
- 500 Internal Server Error – Internal Server Error

`PUT /services/{id}`

Update a specific service.

Update the name and/or the users of this service. If the name, or the userIds are omitted, those will not be updated upon request. To remove all users, an empty array must be submitted for userIds.

Parameters

- **id** (*string*) – ID of the service

Example request:

```
PUT /services/{id} HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "name": "string",
    "userIds": [
        "string"
    ]
}
```

Status Codes

- 200 OK – OK

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": {
        "uuid": "string",
        "site": "string"
    },
    "name": "string",
    "userIds": [
        {
            "uuid": "string",
            "site": "string"
        }
    ]
}
```

- 400 Bad Request – Validation Error

  Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 401 Unauthorized – Unauthorized
- 403 Forbidden – Forbidden
- 404 Not Found – Service does not exist
- 500 Internal Server Error – Internal Server Error

`GET /server`

Get DataMeta server information

Get information about the DataMeta server serving this API

Example request:

```
GET /server HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – OK

  Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "apiVersion": "string",
    "datametaVersion": "string"
}
```

- 500 Internal Server Error – Internal Server Error

`POST /registrations`

Create a new user registration request

Create a new user registration request

Example request:

```
POST /registrations HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "name": "string",
    "email": "string",
    "org_select": "string",
    "org_create": "string",
    "org_new_name": "string",
    "check_user_agreement": true
}
```

Status Codes

- 204 No Content – New User Registration Request created
- 400 Bad Request – Validation Error

  Example response:

  ```
  HTTP/1.1 400 Bad Request
  Content-Type: application/json

  [
      {
          "exception": "string",
          "error_code": "string",
          "message": "string",
          "entity": {
              "uuid": "string",
              "site": "string"
          },
          "field": "string"
      }
  ]
  ```

- 500 Internal Server Error – Internal Server Error

`GET /registrationsettings`

Get details for the registration view

Get all available groups and the user agreement for the registration view

Example request:

```
GET /registrationsettings HTTP/1.1
Host: example.com
```

Status Codes

- 200 OK – OK

  Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "userAgreement": "string",
    "groups": [
        {
            "id": {
                "uuid": "string",
                "site": "string"
            },
            "name": "string"
        }
    ]
}
```

- 500 Internal Server Error – Internal Server Error

`PUT /appsettings/{id}`

Update a specific appsetting. This is an administrative Endpoint that is not accessible for regular users.

Update a specific appsetting

Parameters

- **id** (*string*) – ID of the appsetting

Example request:

```
PUT /appsettings/{id} HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "value": "string"
}
```

Status Codes

- 204 No Content – App Settings Updated
- 401 Unauthorized – Unauthorized
- 403 Forbidden – Forbidden
- 404 Not Found – Setting does not exist
- 400 Bad Request – Validation Error

Example response:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

[
    {
        "exception": "string",
```

(continues on next page)

(continued from previous page)

```
        "error_code": "string",
        "message": "string",
        "entity": {
            "uuid": "string",
            "site": "string"
        },
        "field": "string"
    }
]
```

- 500 Internal Server Error – Internal Server Error

HTTP ROUTING TABLE