A RESTful API Design Specification

A.1 Train Creation Workflow API Specification

1. Upload an analysis task:

```
POST /upload/task HTTP/1.1
Content-Type: multipart/form-data
Content-Disposition: form-data; file="mnist_pca.py"
```

Example Response:

```
200 OK HTTP/1.1
{
    "file_type": "py",
    "message": "Data analysis task uploaded successfully",
    "train_id": "6939b765-a5da-11ec-b254-94e6f725bb14"
}
```

2. Upload a requirements file:

```
POST /upload/req-file/:train_id HTTP/1.1
Content-Type: multipart/form-data
Content-Disposition: form-data; file="requirements.txt"
```

Example Response:

```
200 OK HTTP/1.1
{
    "file_name": "requirements.txt",
    "message": "Requirements uploaded successfully",
    "train_id": "6939b765-a5da-11ec-b254-94e6f725bb14"
}
```

3. Upload a custom Dockerfile:

```
POST /upload/dockerfile/string:train_id HTTP/1.1
Content-Type: multipart/form-data
Content-Disposition: form-data; file="Dockerfile"
```

Example Response:

```
200 OK HTTP/1.1
{
    "file_name": "Dockerfile",
    "message": "Dockerfile uploaded successfully",
    "train_id": "6939b765-a5da-11ec-b254-94e6f725bb14"
}
```

4. Cancel Train creation process:

```
DELETE /upload/:train_id HTTP/1.1
```

Example Response:

```
200 OK HTTP/1.1
{
    "message": "Train image creation cancelled successfully",
    "train_id": "6939b765-a5da-11ec-b254-94e6f725bb14"
}
```

5. Get Dockerfile content:

```
GET /upload/template-dockerfile/:train_id HTTP/1.1
```

Example Response:

```
200 OK HTTP/1.1
{
    "docker_file_content": "FROM python:3.1\n\n ... CMD [\"python\"]",
    "message": "Dockerfile fetched successfully",
    "train_id": "01ede9b4-a5f6-11ec-85f9-94e6f725bb14"
}
```

Query Parameters:

Field	**Type**	**Description**
py_main	string	The name of the file that is the entry point for the analysis code. This file is included in the Dockerfile. (Optional)
custom_file	boolean	A boolean flag to differentiate if Dockerfile is custom or is created from a standard template. (Mandatory)

6. Create Dockerfile from a standard template:

```
POST /upload/template-dockerfile/:train_id HTTP/1.1
Content-Type: application/json
{
    "docker_file_content": "FROM python:3.1\n\n ... CMD [\"python\"]"
}
```

Example Response:

```
200 OK HTTP/1.1
{
    "message": "Dockerfile saved from template successfully"
}
```

7. Fetch code files for entry point selection:

```
GET /upload/task/entrypoint/:train_id HTTP/1.1
```

Example Response:

```
Content-Type: text/html; charset=utf-8
200 OK HTTP/1.1
["mnist_job/datasets/mnistdataset.py", "mnist_job/datasets/mnist_pca.py",
"mnist_job/datasets/__init__.py", "mnist_job/main.py"]
```

8. Get connection environment variables:

```
GET /connection-creds/:train_id HTTP/1.1
```

Example Response:

```
Content-Type: application/json
200 OK HTTP/1.1
{
    "connection_params": [{
        "name": "FHIR_SERVER",
        "required": true,
        "type": "string" }],
    "train_id": "01ede9b4-a5f6-11ec-85f9-94e6f725bb14"
}
```

9. Save connection environment variables:

```
POST /connection-creds/:train_id HTTP/1.1
Content-Type: application/json
{
    "connection_params": [{
        "name": "FHIR_SERVER_PORT",
        "required": false,
        "type": "int" }]
}
```

Example Response:

```
200 OK HTTP/1.1
{
    "connection_params": [{
        "name": "FHIR_SERVER_PORT",
        "required": false,
        "type": "int" }],
    "train_id": "01ede9b4-a5f6-11ec-85f9-94e6f725bb14"
}
```

10. <u>Get PHT metadata</u>:

```
GET /metadata/:train_id HTTP/1.1
```

Example Response:

```
Content-Type: application/json
200 OK HTTP/1.1
{
    "project_description": "Project Description",
    "project_name": "PHTProjectName",
    "project_type": "Public",
    "project_url": "https://git.example.de/max/train-store",
    "train_id": "01ede9b4-a5f6-11ec-85f9-94e6f725bb14",
    "additional_info": {
        "creator": "Max"
    }
}
```

11. <u>Save PHT metadata</u>:

```
POST /metadata/:train_id HTTP/1.1
Content-Type: application/json
{
    "project_name": "Demo PHT Project",
    "project_type": "Public",
    "description": "PHT automatic workflow for train creation using docker",
    "additional_info": {
        "published_date": "01.03.2022"
    }
}
```

Example Response:

```
200 OK HTTP/1.1
{
    "project_name": "Demo PHT Project",
    "project_type": "Public",
    "description": "PHT automatic workflow for train creation using docker",
    "project_url": "https://git.example.de/max/train-store",
    "train_id": "01ede9b4-a5f6-11ec-85f9-94e6f725bb14",
    "additional_info": {
        "published_date": "01.03.2022"
    }
}
```

12. Get Train summary - task files, connection parameters and metadata:

```
GET /summary/:train_id HTTP/1.1
```

Example Response:

```
Content-Type: application/json
200 OK HTTP/1.1
{
    "connection_params": [{
        "name": "FHIR_SERVER_PORT",
        "required": false,
        "type": "int" }],
    "data_files": [ "Dockerfile", "mnist_pca.py", "requirements.txt" ],
    "metadata": {
        "additional_info": {
            "published_date": "01.03.2022"
        },
        "project_name": "Demo PHT Project",
        "project_type": "Public",
        "description": "PHT automatic workflow for train creation docker",
        "project_url": "https://git.example.de/max/train-store"
    },
    "train_id": "01ede9b4-a5f6-11ec-85f9-94e6f725bb14"
}
```

13. Get GitLab repository branches:

```
GET /git-op/git-info/:train_id HTTP/1.1
```

Example Response:

```
Content-Type: text/html; charset=utf-8
200 OK HTTP/1.1
["develop-max", "develop-peter", "develop-wizard", "develop-dummy", "developer-
branch"]
```

14. Create new GitLab branch and save commit data:

```
POST /git-op/git-branch/:train_id HTTP/1.1
Content-Type: application/json
{
    "new_branch": true,
    "new_branch_name": "test-branch",
    "commit_message": "add new train to repo"
}
```

Example Response:

```
200 OK HTTP/1.1
{
    "git_url": "https://git.example.de/",
    "project_id": "12345",
    "access_token": "abcdefgh123456",
    "branch": "",
    "commit_message": "add new train to repo",
    "new_branch": true,
    "new_branch_name": "test-branch"
}
```

15. Get GitLab access details:

```
GET /git-op/private-git-info/:train_id HTTP/1.1
```

Example Response:

```
Content-Type: application/json
200 OK HTTP/1.1
{
    "access_token": "abcdefgh123456",
    "git_url": "https://git.example.de/",
    "project_id": "12345"
}
```

16. Save GitLab access details:

```
POST /git-op/private-git-info/:train_id HTTP/1.1
Content-Type: application/json
{
    "access_token": "abcdefgh123456",
    "git_url": "https://git.example.de/",
    "project_id": "12345"
}
```

Example Response:

```
200 OK HTTP/1.1
```

17. Upload Train image to GitLab repository:

```
POST /git-op/git-repo/:train_id HTTP/1.1
```

Example Response:

```
200 OK HTTP/1.1
Content-Type: application/json
{
    "commit_message": "add new train to repo",
    "commit_created_at": "2022-03-03T09:26:24.000-07:00",
    "commit_url": "https://git.example.de/max/train-store/commit/1",
    "author_name": "Max",
    "commit_sha": "1260f8be67647431a7b5b2450a966d5fa8a587c2"
}
```

18. Train creation workflow module error codes:

- 404 Not Found: File not found in request body.
- 415 Unsupported Media Type: Invalid file extension.
- 400 Bad Request:
  - Invalid Train ID.
  - The file in request body is empty.
  - The personal access token is invalid.
- 401 Unauthorized: The personal access token isn't authenticated. A valid token is required.
- 500 Internal Server Error: Internal code failure.

A.2 Train Storehouse Platform API Specification

1. Logon to storehouse platform:

```
POST /authentication HTTP/1.1
Content-Type: application/json
{
    "username": "some_user_name",
    "pat": "abcdefgh123456"
}
```

Example Response:

```
200 OK HTTP/1.1
{
    "status_code": 200,
    "message": "Valid user!!!"
}
```

2. Logoff from storehouse platform:

```
DELETE /authentication/logoff HTTP/1.1
```

Example Response:

```
200 OK HTTP/1.1
{
    "status_code": 200,
    "message": "Logoff successful"
}
```

3. Get GitLab repository branches:

```
GET /gitlab/branches HTTP/1.1
```

Example Response:

```
Content-Type: text/html; charset=utf-8
200 OK HTTP/1.1
["develop-max", "develop-peter", "develop-wizard", "develop-dummy", "develop-
branch"]
```

4. Get GitLab repository tree for a given branch:

```
GET /gitlab/images/:branch_name HTTP/1.1
```

Example Response:

```
Content-Type: text/html; charset=utf-8
200 OK HTTP/1.1
["pht-train-image", "pht-new-usecase"]
```

5. Get Train image complete information:

```
POST /gitlab/images/info HTTP/1.1
Content-Type: application/json
{
    "branch_name": "test-branch",
    "project_name": "pht-train-image"
}
```

Example Response:

```
Content-Type: application/json
200 OK HTTP/1.1
{
    "approval_permission": true,
    "feedback_permission": true,
    "branch_name": "develop-wizard",
    "connection_params": [{
        "name": "FHIR_SERVER_PORT",
        "required": false,
        "type": "int" }],
    "feedback": [{
        "member_name": "Some User",
        "rating": 4,
        "comment": "The train works fine at the stations!",
        "date_time": "2022-03-03T09:26:24.000-07:00" }],
    "member_name": "Some User",
    "metadata": {
        "creator": "Max",
        "published_date": "01.03.2022",
        "project_description": "Testing official train store",
        "project_name": "Test_PHT_Train_Image",
        "project_type": "Public",
        "project_url": "https://git.example.de/max/train-store",
        "size": "100 MiB"
    },
    "project_name": "Test_PHT_Train_Image",
    "project_url": "https://git.example.de/max/train-store/-/tree/develop-
wizard/Test_PHT_Train_Image"
}
```

6. Save Train image user ratings and feedback:

```
POST /gitlab/save-feedback HTTP/1.1
Content-Type: application/json
{
    "branch_name": "develop-wizard",
    "project_name": "Test_PHT_Train_Image",
    "member_name": "Some User",
    "rating": 4,
    "comment": "The train works fine at the stations!"
}
```

Example Response:

```
Content-Type: application/json
```

```
200 OK HTTP/1.1
{
    "commit_message": "add user rating and feedback",
    "commit_created_at": "2022-03-03T09:26:24.000-07:00",
    "commit_url": "https://git.example.de/max/train-store/commit/1",
    "author_name": "Max",
    "commit_sha": "1260f8be"
}
```

7. Create GitLab merge request:

```
POST /gitlab/merge-request HTTP/1.1
Content-Type: application/json
{
    "mr_title": "merge new pht train to main branch",
    "branch_name": "develop-wizard"
}
```

Example Response:

```
Content-Type: application/json
200 OK HTTP/1.1
{
    "mr_iid": 2,
    "mr_title": "merge new pht train to main branch",
    "mr_state": "created",
    "mr_created_at": "2022-03-03T09:26:24.000-07:00",
    "mr_source_branch": "develop-wizard",
    "mr_target_branch": "main",
    "mr_url": "https://git.example.de/max/train-store/merge-request/2",
    "pipeline_url": "https://git.example.de/max/train-store/pipeline/5"
}
```

8. Approve and push GitLab merge request:

```
POST /gitlab/merge-request/merge HTTP/1.1
Content-Type: application/json
{
    "mr_iid": 2
}
```

Example Response:

```
Content-Type: application/json
200 OK HTTP/1.1
{
    "mr_push_created_at": "2022-03-03T09:26:24.000-07:00",
    "mr_push_url": "https://git.example.de/max/train-store/merge-request/2"
}
```

9. Train storehouse platform module error codes:

- 400 Bad Request:
  - Logoff failed due to internal error.
  - A merge request for a given branch already exists.
  - No merge request exists for approval.
  - Git commit to save user ratings and feedback failed.
- 401 Unauthorized: The personal access token isn't authenticated. A valid token is required.
- 403 Forbidden: The user is not allowed to access the git repository.
- 500 Internal Server Error: Internal code failure.

B GitLab REST API Resources

The GitLab REST API[1] resources and endpoints used in this thesis are documented below.

1. List repository branches: Get a list of branches from a project repository.

```
GET /projects/:id/repository/branches
PRIVATE-TOKEN: <your_access_token>
```

Path Parameters:

Attribute	Type	Required	Description
id	integer/string	yes	ID or URL-encoded path of the project owned by the user.

2. Create repository branch: Create a new branch in the project repository.

```
POST /projects/:id/repository/branches?branch=:branch&ref=:ref
PRIVATE-TOKEN: <your_access_token>
```

Query Parameters:

Attribute	Type	Required	Description
branch	string	yes	Name of the branch.
ref	string	yes	Branch name or commit SHA to create branch from.

3. Create a commit with multiple files and actions: Create a project repository commit by providing a JSON payload. The JSON payload comprises of the branch name to commit to, the commit message, and the file information (file name and path, content, and the action).

```
POST /projects/:id/repository/commits
PRIVATE-TOKEN: <your_access_token>
Content-Type: application/json
Accept-Charset: UTF-8
{
    "branch": <branch_name>,
    "commit_message": <commit_message>,
```

[1] https://docs.gitlab.com/ee/api/api_resources.html

```
    "actions": [ {...}, {...} ]
}
```

Request Body Attributes:

Attribute	**Type**	**Required**	**Description**
branch	string	yes	Name of the branch to commit into.
commit_message	string	yes	Commit message
actions[ ]	array	yes	An array of action hashes to commit as a batch.

Actions attributes and their description:

actions[ ] Attribute	**Type**	**Required**	**Description**
action	string	yes	The action to perform: create, delete, move, update, chmod.
file_path	string	yes	Full path to the file.
content	string	no	File content, required for all except delete, chmod, and move.
encoding	string	no	text or base64. text is default.

4. List all members of a project: Get a list of project members viewable by the authenticated user.

```
GET /projects/:id/members?per_page=:per_page
PRIVATE-TOKEN: <your_access_token>
```

5. List repository tree: Get a list of repository files and directories in a project.

```
GET /projects/:id/repository/tree?per_page=:per_page&ref=:ref
PRIVATE-TOKEN: <your_access_token>
```

Query Parameters:

Attribute	**Type**	**Required**	**Description**
per_page	integer	no	Number of records to return per page.
ref	string	no	The name of a repository branch or tag.

6. Get file from repository: Fetches the information about file in repository like name, size, content. The file content is Base64 encoded.

```
GET /projects/:id/repository/files/:file_path?ref=:ref
PRIVATE-TOKEN: <your_access_token>
```

Path and Query Parameters:

Attribute	Type	Required	Description
file_path	string	yes	URL encoded full path to new file.
ref	string	no	The name of branch, tag or commit.

7. Create merge request: Creates a merge request from a source branch to a target branch.

```
POST /projects/:id/merge_requests?source_branch=:source_branch&
target_branch=:target_branch&title=:title
PRIVATE-TOKEN: <your_access_token>
```

Query Parameters:

Attribute	Type	Required	Description
source_branch	string	yes	The source branch for merge request.
target_branch	string	yes	The target branch for merge request.
title	string	yes	Title of the merge request.

8. Merge a merge request: Accept and merge changes submitted with the merge request to a target branch.

```
PUT /projects/:id/merge_requests/:merge_request_iid/merge?
merge_when_pipeline_succeeds=true&should_remove_source_branch=true
PRIVATE-TOKEN: <your_access_token>
```

Query Parameters:

Attribute	Type	Required	Description
merge_request_iid	integer	yes	The internal ID of the merge request.
merge_when_pipeline_succeeds	boolean	no	If 'true' the merge request is merged when the pipeline succeeds.
should_remove_source_branch	boolean	no	If 'true' removes the source branch.

9. List project pipelines: List pipelines of a project. It does not includes the child pipelines, and has to be fetched individually.

```
GET /projects/:id/pipelines
PRIVATE-TOKEN: <your_access_token>
```