RECEone Production, including

Raster and Vector Products based on Satellite Input Data from 2017/2018/2019

SYSTEM SPECIFICATION

Issue 1.0

European Environment Agency

Framework Service Contract No. EEA/DIS/R0/19/012

Issue:	1.0
Date:	xx/xx/2022
Compiled by:	GeoVille

DOCUMENT CONTROL INFORMATION

Document Description

Settings	Value
Document Title:	SYSTEM SPECIFICATION
Project Title:	Copernicus Land Monitoring Service – CLC+ Backbone Production, including Raster and Vector Products based on Satellite Input Data from 2017/2018/2019
Document Author(s):	GeoVille
Project Owner:	Hans Dufourmont (EEA)
Project Manager:	Tobias Langanke (EEA)
Doc. Issue/Version:	1.0
Date:	xx/xx/2022
Distribution List:	CLC+ Backbone project consortium; Project Officer and European Environment Agency services
Confidentiality:	Confidential to the above Distribution List

Document Release Sheet

Name	Role	Signature	Date
Johannes Schmid (GeoVille) Wolfgang Kapferer (GeoVille)	Creation		xx/xx/2022
Markus Probeck (GAF) Inés Ruiz Gomez (GAF)	Revision		
Tobias Langanke (EEA)	Approval		

Document History & Change Record

Issue / Version	Release Date	Created by	Page(s)	Description of Issue / Change(s)
1.0	20/07/2021	GAF / GeoVille	xx	First Issue of the CLC+ Backbone System Specification

Applicable Documents

Issue	Document
[AD-01]	EN-EEA.DIS.R0.19.012_Annex+I_Tender+Specifications_001
[AD-02]	EN-EEA.DIS.R0.19.012_Annex 7 to TS_Technical+specification_Land+monitoring+concept_EAGLE

Table of Contents

List of Figures

List of Tables

Acronyms and Abbreviations

AD	Applicable Document
AoI	Area of Interest
ARVI	Atmospherically Resistant Vegetation Index
CL	Confidence Layer
CLC, CLC+	CORINE Land Cover, CORINE Land Cover +
CLMS	Copernicus Land Monitoring Service
DEM	Digital Elevation Model
DOMs	DOMs French Overseas Departments
DSL	Data Score Layer
EAGLE	EIONET Action Group on Land monitoring in Europe
EEA	European Environment Agency
EEA-39	The 33 member and 6 cooperating countries of the EEA
EFFIS	European Forest Fire Information System
EIONET	European Environment Information and Observation Network
EO	Earth Observation
EPSG	European Petroleum Survey Group
ETRS89	European Terrestrial Reference System 1989
EU	European Union
EU28	The 28 member states of the European Union
EU-Hydro	European Hydrography Layer
EUROSTAT	European Statistical Office
FMask	Function of mask
GAFSEG	software developed by GAF AG
GIS	Geographic Information System
HDF5	Hierarchical Data Format 5
HR	High resolution
INSPIRE	INfrastructure for SPatial InfoRmation in Europe
JRC	Joint Research Centre
L2A	Level 2A
LAEA	Lambert Azimuthal Equal Area
LC	Land Cover
LC/LU	Land Cover/Land Use
LCC	Land Cover Component
LiDAR	Light detection and ranging
LPIS	Land parcel identification system
LUCAS	Land Use/Cover Area frame Survey
LZW	Lempel–Ziv–Welch data compression algorithm
MMU	Minimum Mapping Unit
MMW	Minimum Mapping Width
NBR	Normalized Burn Ratio
NDVI	Normalized Difference Vegetation Index
NDWI	NDWI Normalized Difference Water Index
OSM	Open Street Map
PU	Production Unit
QA	Quality Assurance
QC	Quality Control
S-2	Sentinel-2
SPU	Secondary Production Unit
TF	Time Feature
TIFF	Tagged image File Format
UK	United Kingdom
UTM	Universal Transverse Mercator
VHR	Very High Resolution
WGS84	World Geodetic System 1984
WISE	Water Information System for Europe
WKT	Well-known-Text
XML	Extensible Markup Language

0 Executive Summary

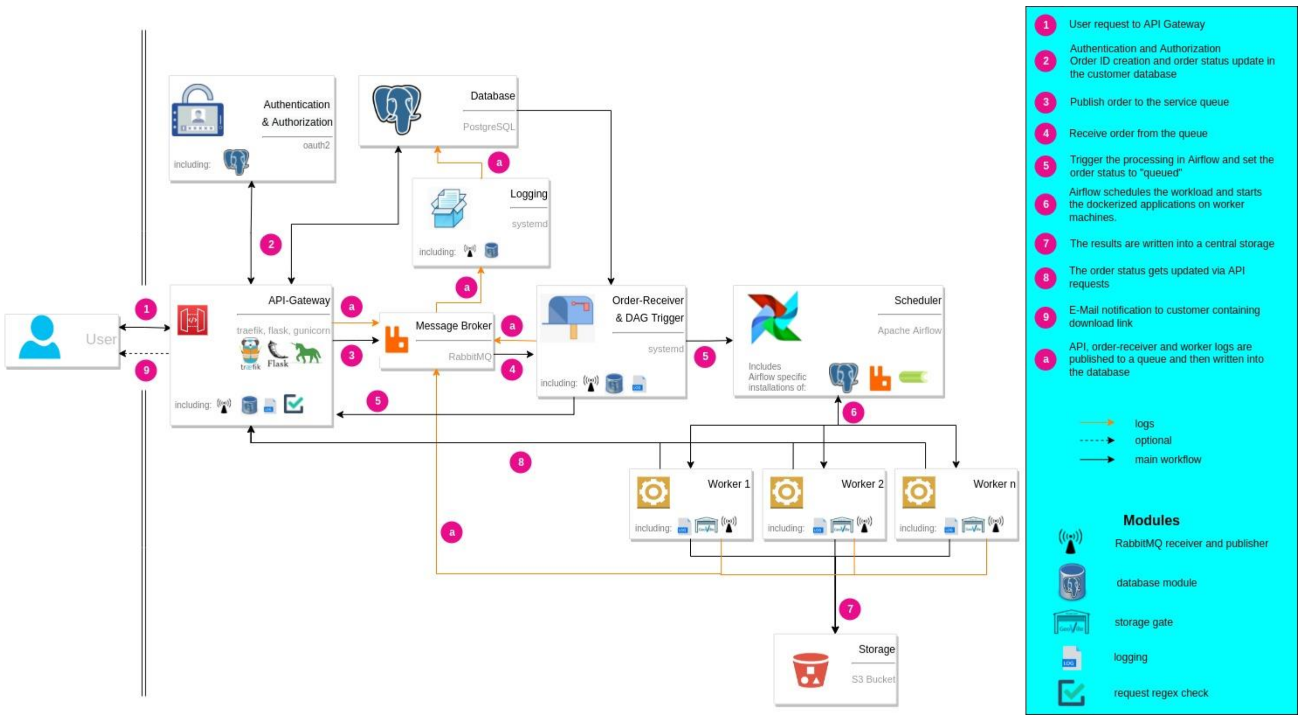
1 Background of the Document

1.1 Scope of the Document

1.2 Content and Structure

2 System components

The system is based on a distributed microservice architecture. Figure **?** shows the general workflow with the main system components (microservices) and how they link together. Besides the description of the workflow on the right-hand side, the Figure also displays the modules and software stack used by each component.



Authentication & Authorization
oauth2
including:
Database
PostgreSQL
Logging
systemd
including:
User
API-Gateway
traefik, flask, gunicorn
traefik
Flask
including:
Message Broker
RabbitMQ
Order-Receiver & DAG Trigger
systemd
including:
Scheduler
Apache Airflow
Includes Airflow specific installations of:
Worker 1
including:
Worker 2
including:
Worker n
including:
Storage
S3 Bucket
1
2
3
4
5
6
7
8
9
a
1 User request to API Gateway
2 Authentication and Authorization Order ID creation and order status update in the customer database
3 Publish order to the service queue
4 Receive order from the queue
5 Trigger the processing in Airflow and set the order status to "queued"
6 Airflow schedules the workload and starts the dockerized applications on worker machines.
7 The results are written into a central storage
8 The order status gets updated via API requests
9 E-Mail notification to customer containing download link
a API, order-receiver and worker logs are published to a queue and then written into the database
logs
optional
main workflow
Modules
RabbitMQ receiver and publisher
database module
storage gate
logging
request regex check

2.1 API Gateway

An API gateway sits between a client and a collection of backend services and serves as the entry point to the microservice infrastructure. An API gateway accepts all application programming interface (API) calls, aggregates the various services required to fulfil them, and returns the appropriate result. The RESTful API provides all endpoints which are required to interact with the system. The endpoints are divided into namespaces or sections to group common operations (e.g.: geo-services, custom-relation-management services, etc.). The API is documented with the help of the OpenAPI specification. The OpenAPI Specification (OAS) defines a standard, language-agnostic interface to RESTful APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection. When properly defined, a consumer can understand and interact with the remote service with a minimal amount of implementation logic. An OpenAPI document that conforms to the OpenAPI Specification is itself a JSON object, which may be represented either in JSON or YAML format. An OpenAPI definition can then be used by documentation generation tools to display the API, code generation tools to generate servers and clients in various programming languages, testing tools, and many other use cases. Moreover, the UI generated by these tools supports a straightforward workflow to run, debug and test all API endpoints. Please visit the generated web page for more details (https://api.clcplusbackbone.geoville.com/v1/).

2.2 Authentication and Authorization

User authentication, authorization and management are based on the OAuth2 framework. It is used to exchange data between client and server through authorization. The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. The Authorization Server provides several endpoints for authorization, issuing tokens, refreshing tokens and revoking tokens. When the resource owner (user)authenticates, this server issues an access token to the client. The resource owner is the user who is using a service. A resource owner can log in to a website with a username/email and password, or by other methods. A client is an application making protected resource requests on behalf of the resource owner and with its authorization. Any application that uses OAuth 2.0 to access CLC backbone API must have authorization credentials that identify the application to the OAuth 2.0 server. Therefore, the authorization server comes with a PostgreSQL database for managing users, clients, access permissions and access tokens. The API of the authorization server provides a set of endpoints which are required to perform common authorization operations and flows. Amongst others, this includes for example:

- Creating OAuth clients
- User login
- Access token generation
- Token validation
- Scopes creation and management

Scopes define which services a user has access to.

2.3 Order Status Updates

The status of an asynchronous order changes in the course of processing it. Possible order states can be the following:

- RECEIVED: the order has succeeded the validation and was accepted
- QUEUED: the order has been sent to the service queue
- RUNNING: the order started processing
- INVALID: there is no satellite data available for the requested date(s)
- SUCCESS: the order finished successfully
- FAILED: the order failed during processing
- ABORTED: the order was canceled manually by the user

The states are being updated by the API (RECEIVED, QUEUED and ABORTED) and by Airflow (RUNNING, INVALID, FAILED, SUCCESS). The overall workflow is illustraded in Figure 2-1.

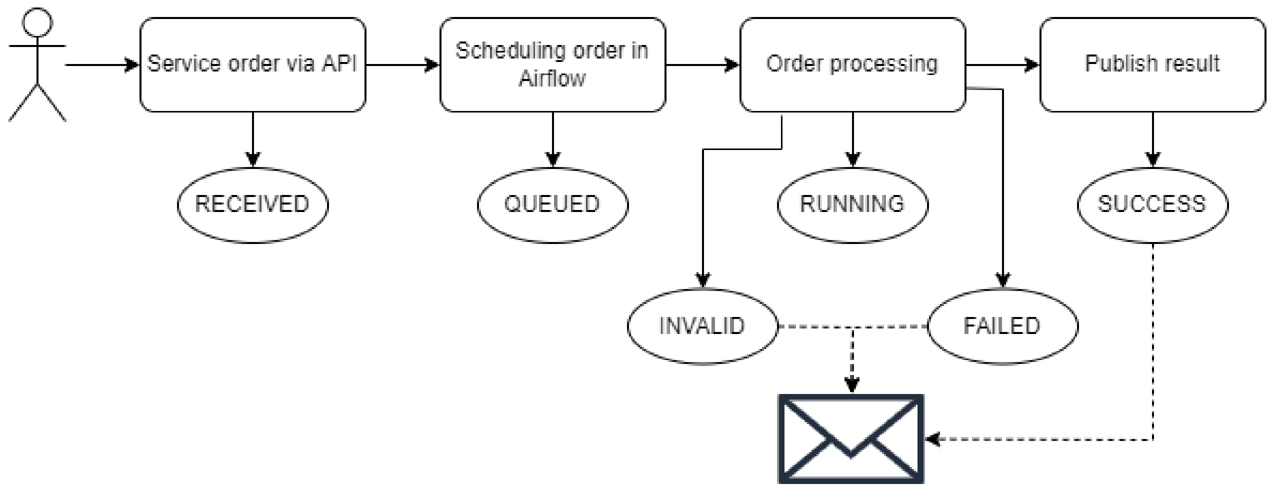


Figure 2-1: Overview of order status updating workflow

Updating a status is done by the API gateway, which offers respective routes for Airflow to access. For specific status updates such as "SUCCESS", "INVALID" and "FAILED" an e-mail notification is sent to the user, if the optional notification parameter in the service ordering payload is set to "true". By default no email is sent.

2.4 Message Broker

As already mentioned, the system infrastructure is based on a microservice architecture. The communication between these microservices is implemented by a message broker. specifically, the system uses the open-source message broker RabbitMQ together with a Python module that listens to the queues and triggers the respective services in the scheduler.

2.4.1 Queueing System

RabbitMQ is an open-source message broker that implements the Advanced Message Queuing Protocol (AMQP). Basically, the message broker software has two functions, to "publish" and to "receive" messages. By using the Python module pika, an AMQP connection to RabbitMQ can be created to send requests to and receive requests from the server. To ensure that the messages are secure - as they might include sensitive information - Cryptography is used to encrypt the messages.

The main advantages of using RabbitMQ are the high scalability, the ability to run on most operating systems and cloud environments and the regular updates from a large developer community.

2.4.2 Queue listener

Service orders that were published to a RabbitMQ service queue by the API gateway need to be received and trigger the respective processing chain in Airflow. This is done by a systemd service that is located on the same machine as the Airflow scheduler. It listens to all service queues and executes the respective Airflow trigger command if a new service order is received. The trigger command for each service is stored in a database table. The two functions; listening to the queue, and; executing the Airflow trigger, happen on different threads.

For efficient usage of RabbitMQ's functionalities from within our system, we created a Python module that simplifies accessing queues from our code. Details about the module can be found in section 3.3.

2.5 Scheduler

The scheduling system is one of the most important components of the system infrastructure. It allows the creation, execution and monitoring of multiple parallel workflows and tasks. The scheduler is based on the Apache Airflow workflow management platform. Airflow is an open-source platform based on Python that is designed under the principle of "configuration as code".

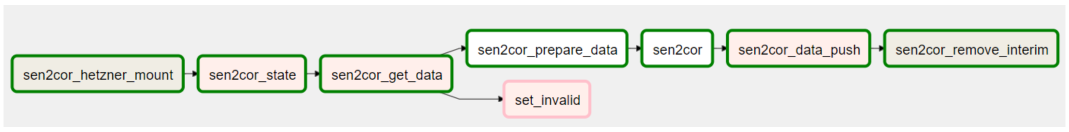
The following sub-sections address some of the most important Apache Airflow components.

2.5.1 DAG

Apache Airflow supports the creation of workflows. A workflow is formulated as a Directed Acyclic Graph (DAG). Usually, such a DAG is a collection of tasks and each DAG is represented by a Python script.

In Figure **?** an example DAG is visually presented. Each rectangle represents a task which can be either a Python function, a Bash command or a Docker container.

Figure ?: Graph of an example DAG



2.5.2 Operator

A DAG consists of several tasks which are also called operators. Airflow supports various types of operators. Common operators which are used in the current installation are listed below:

- Python Operator: Executes Python callable and commands.
- Bash operator: Executes commands in a Bash shell.
- Docker operator: Executes a command inside a docker container.

2.5.3 Worker

One advantage of Apache Airflow is the support of distributed system architectures. The system includes several so-called workers, which run on systems known as worker nodes. Jobs can be assigned by the main Airflow application to worker instances by using message protocols. Put simply, each production step can run on a separate virtual machine and is managed by the airflow scheduler machine.

2.5.4 Parallelism

Apache Airflow provides powerful tools and configuration options to run workflows (DAGs) and even single tasks (operators) in parallel. This helps to manage huge workloads.

2.5.5 Monitoring

Airflow provides a powerful monitoring tool which helps to monitor, start, delete and debug operators and DAGs.

2.6 Logging

The logging functionality of the system aims to provide a simple, yet flexible way to log events in a central database from each system component. It is based on:

1. a mechanism that receives logging messages and sends them into a queue
2. a service that consumes the log messages and stores them.

The receiving mechanism offers two options to be used for logging:

- A Python module which provides a command for logging. For details see section 3.1.
- A REST endpoint which allows logging via HTTP request, which provides a lot of flexibility because any client capable of sending HTTP requests can use it.

Generally, both logging mechanisms send messages into a queue.

The consumption mechanism is a standalone program (GeoVille_MS_Logging_Saver) which reads the log messages in batches from the queue and persists the logs in a relational database.

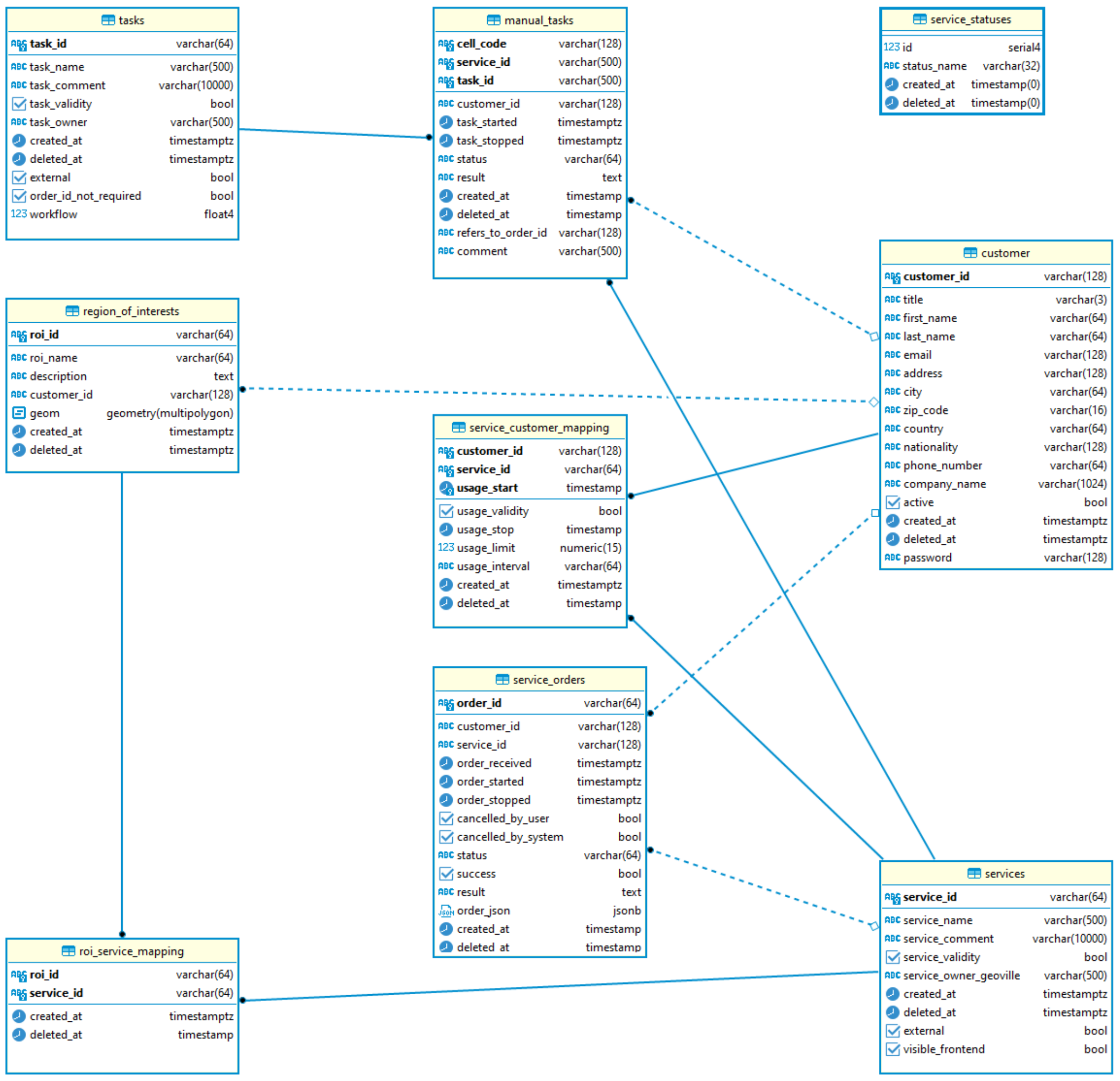
In order to support traceability and debugging, the logging module provides different log-levels:

- INFO: Confirmation that things are working as expected.
- WARNING: Indicates that something unexpected happened, but the software is still working as expected
- ERROR: Indicates a serious problem. The software has not been able to perform some function.

2.7 Database

Any modern backend solution needs a storage system to store data whilst processing particular tasks. As spatial information will be processed in the project, the tool chosen was a PostgreSQL database server with its extension PostGIS for spatial operations. PostgreSQL is a powerful, Open Source object-relational database system with over 30 years of active development. PostgreSQL supports both SQL (relational) and JSON (non-relational) querying. PostgreSQL is a highly stable database and used as a primary database for many web applications as well as mobile and analytics applications. PostGIS is a spatial database extender for PostgreSQL object-relational database. It adds support for geographic objects allowing location queries to be run in SQL and provides geospatial databases for geographic information systems (GIS). PostGIS follows the

Simple Features for SQL specification from the Open Geospatial Consortium (OGC). A relational database model is required to persist the processed information in a structured manner. The figure below shows the data model used by the backbone API.



tasks	
task_id	varchar(64)
task_name	varchar(500)
task_comment	varchar(10000)
task_validity	bool
task_owner	varchar(500)
created_at	timestamptz
deleted_at	timestamptz
external	bool
order_id_not_required	bool
workflow	float4

manual_tasks	
cell_code	varchar(128)
service_id	varchar(500)
task_id	varchar(500)
customer_id	varchar(128)
task_started	timestamptz
task_stopped	timestamptz
status	varchar(64)
result	text
created_at	timestamp
deleted_at	timestamp
refers_to_order_id	varchar(128)
comment	varchar(500)

service_statuses	
id	serial4
status_name	varchar(32)
created_at	timestamp(0)
deleted_at	timestamp(0)

region_of_interests	
roi_id	varchar(64)
roi_name	varchar(64)
description	text
customer_id	varchar(128)
geom	geometry(multipolygon)
created_at	timestamptz
deleted_at	timestamptz

customer	
customer_id	varchar(128)
title	varchar(3)
first_name	varchar(64)
last_name	varchar(64)
email	varchar(128)
address	varchar(128)
city	varchar(64)
zip_code	varchar(16)
country	varchar(64)
nationality	varchar(128)
phone_number	varchar(64)
company_name	varchar(1024)
active	bool
created_at	timestamptz
deleted_at	timestamptz
password	varchar(128)

service_customer_mapping	
customer_id	varchar(128)
service_id	varchar(64)
usage_start	timestamp
usage_validity	bool
usage_stop	timestamp
usage_limit	numeric(15)
usage_interval	varchar(64)
created_at	timestamptz
deleted_at	timestamp

service_orders	
order_id	varchar(64)
customer_id	varchar(128)
service_id	varchar(128)
order_received	timestamptz
order_started	timestamptz
order_stopped	timestamptz
cancelled_by_user	bool
cancelled_by_system	bool
status	varchar(64)
success	bool
result	text
order_json	jsonb
created_at	timestamp
deleted_at	timestamp

roi_service_mapping	
roi_id	varchar(64)
service_id	varchar(64)
created_at	timestamptz
deleted_at	timestamp

services	
service_id	varchar(64)
service_name	varchar(500)
service_comment	varchar(10000)
service_validity	bool
service_owner_geoville	varchar(500)
created_at	timestamptz
deleted_at	timestamptz
external	bool
visible_frontend	bool

2.8 Container virtualization

For OS-level virtualization the system uses Docker. As already mentioned, Apache Airflow provides an operator for efficiently running Docker containers. However, not just the single service tasks have been dockerized, but also the Airflow installation and configuration of each worker node.

Besides the Airflow components, the API gateway, the authentication and authorization as well as the central PostgreSQL database have also been dockerized.

The main reason why these system components have been dockerized is to enable simple reusability and scaling. When a system component needs to be migrated to another virtual machine or when another

Airflow worker needs to be deployed, the Docker image can be reused instead of spending hours on installation and configuration. This can be done by either pulling the Docker image from a private or public Docker registry or by building the Docker image locally.

Another big advantage of Docker is the independence from system updates. While the upgrade of a program or module by a system update could cause dependency problems if the system component runs as a daemonized systemd service, the installation and execution within a Docker container is not affected.

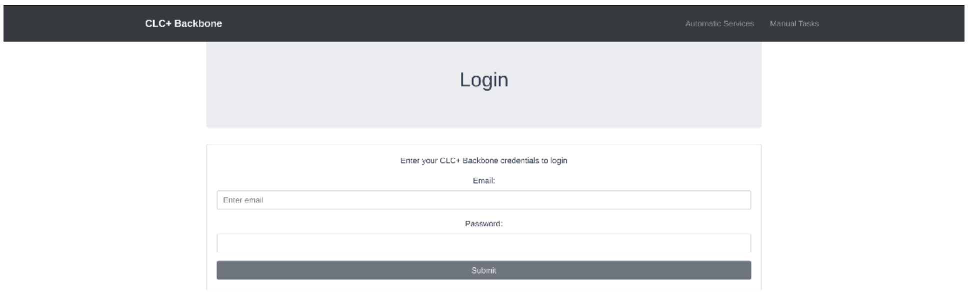
2.9 Monitoring

The monitoring system consists of three parts. Prometheus is a monitoring solution for storing time series data like system metrics. Grafana allows for the visualisation of the data stored in Prometheus. The Prometheus Alertmanager is the third part of the monitoring system. It handles any alerts sent by the Prometheus server.

2.10 Productive User interface

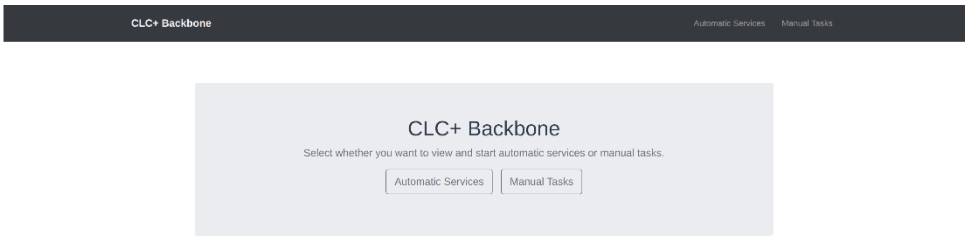
The productive user interface was developed for the operation of the CLC+ Backbone project. Here, various automatic processes as well as manual tasks can be started and monitored by GeoVille and GAF.

When entering the user interface at https://ui.clcplusbackbone.geoville.com/, the users are asked to insert their email address as well as their password (see Figure ?).



After a successful log in, the users can choose between "Automatic Services" and "Manual Tasks".

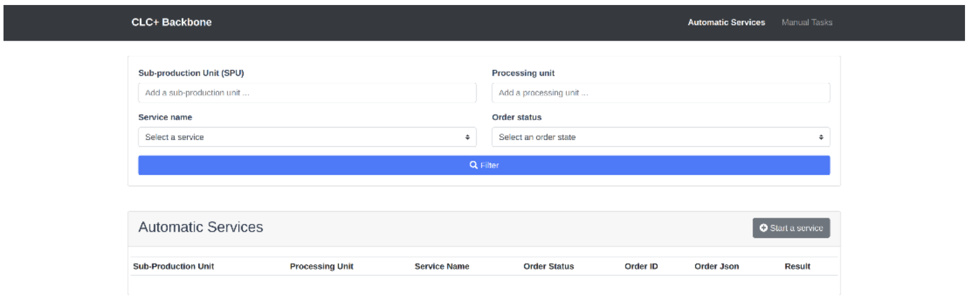
While automatic services represent processes in Airflow that are started by sending API requests, manual tasks are non-automatic parts of the operation workflow, for example quality control. Starting a manual task is nothing more than a status update in the database for monitoring all steps of the operations and not only the fully automatic steps.



CLC+ Backbone

Select whether you want to view and start automatic services or manual tasks.

When selecting “Automatic Services”, information such as the processing unit or the service name can be provided to search for the current operation status - as seen in the following Figure. In case a service was already successfully processed for a processing unit, it does not need to be processed again.

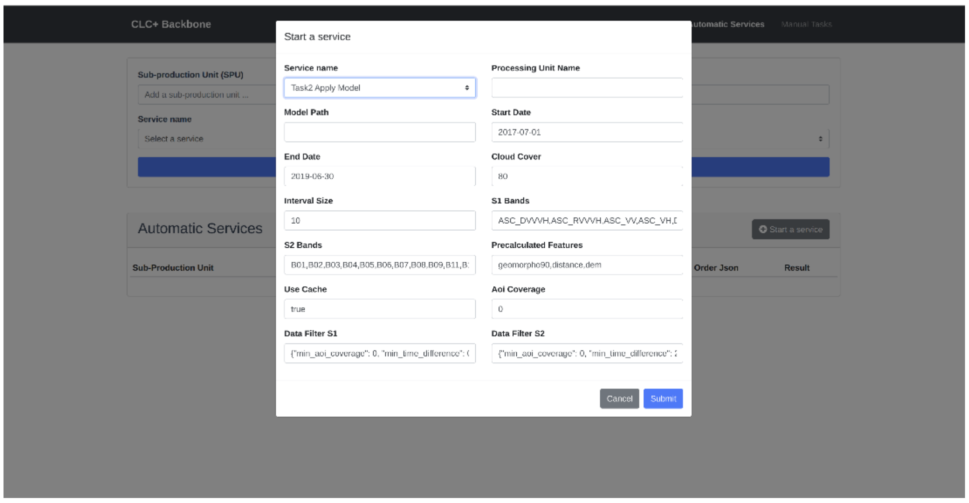


Sub-production Unit (SPU)	Processing unit
Add a sub-production unit ...	Add a processing unit ...
Service name	**Order status**
Select a service	Select an order state

Automatic Services

Sub-Production Unit	Processing Unit	Service Name	Order Status	Order ID	Order Json	Result

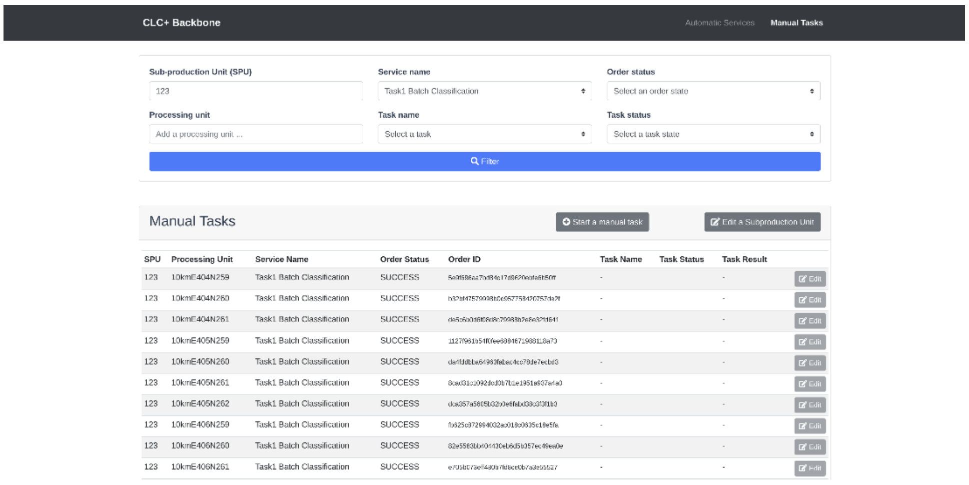
However, if an automatic service needs to be started, a click on the button “Start a service” will open a pop-up window (see Figure ?). After selecting a service, the relevant parameters such as the processing unit name can be filled in. A click on the “Submit” button will then send the request to the API gateway and where the request will be processed if validation succeeds.



Start a service

Service name: Task2 Apply Model	Processing Unit Name:
Model Path:	Start Date: 2017-07-01
End Date: 2019-06-30	Cloud Cover: 80
Interval Size: 10	S1 Bands: ASC_DVVVH,ASC_RVVVH,ASC_VV,ASC_VH,I
S2 Bands: B01,B02,B03,B04,B05,B06,B07,B08,B09,B11,B:	Precalculated Features: geomorpho90,distance,dem
Use Cache: true	Aoi Coverage: 0
Data Filter S1: {"min_aoi_coverage": 0, "min_time_difference": (	Data Filter S2: {"min_aoi_coverage": 0, "min_time_difference": 2

The functionality for manual tasks is similar to that for the automatic services. However, besides the button to “Start a manual task”, there is also a button to “Edit a Subproduction Unit”. With this button, the user can update the status of a specific task for an entire subproduction unit.



Sub-production Unit (SPU): 123	Service name: Task1 Batch Classification	Order status: Select an order state
Processing unit: Add a processing unit ...	Task name: Select a task	Task status: Select a task state

Manual Tasks

SPU	Processing Unit	Service Name	Order Status	Order ID	Task Name	Task Status	Task Result	
123	10kmE404N259	Task1 Batch Classification	SUCCESS	5e9f686aa7bd84c17d8620ebda6b50ff	-		-	Edit
123	10kmE404N260	Task1 Batch Classification	SUCCESS	b32bf47579998b0d957758420757da2f	-		-	Edit
123	10kmE404N261	Task1 Batch Classification	SUCCESS	de5c6b0d6f08d8c79988b2e8e321d641	-		-	Edit
123	10kmE405N259	Task1 Batch Classification	SUCCESS	1127f961b54f0fee6804671968118a73	-		-	Edit
123	10kmE405N260	Task1 Batch Classification	SUCCESS	da4fddbba64983fabac4cc78de7ecbd3	-		-	Edit
123	10kmE405N261	Task1 Batch Classification	SUCCESS	8cad31c1092dcd0b7b1e1951a837a4a0	-		-	Edit
123	10kmE405N262	Task1 Batch Classification	SUCCESS	dca357a5805b32b0e6fabd38c3f3f1b3	-		-	Edit
123	10kmE406N259	Task1 Batch Classification	SUCCESS	fb625c872894032ac018c0835c19e5fa	-		-	Edit
123	10kmE406N260	Task1 Batch Classification	SUCCESS	82e5583bb404430eb6d5b357ec49ea0e	-		-	Edit
123	10kmE406N261	Task1 Batch Classification	SUCCESS	e705b073eff480b7fd8ce0b7a3e55527	-		-	Edit

3 Modules

3.1 Logging module

The logging module is a Python module which simplifies logging from within Python code. It takes the log messages, including the log level, and sends them to the logging queue (RabbitMQ). This module helps to simplify the code, as it does not require communication with the logging API via HTTP. To send messages to the queue, the RabbitMQ module described in section 3.3 is used.

3.2 Database module

This module abstracts a PostgreSQL database connector and provides several functions to read from and write into tables. The provided methods are:

- read one row from a query
- read all rows from a query
- read many (a specific number of) rows from a query
- execute commands such as insert, update, create, drop, delete, etc.

3.3 RabbitMQ module

This Python module provides the basic implementation to retrieve messages from and publish messages to RabbitMQ queues. It consists of the classes BaseReceiver and Publisher. The Publisher makes it very simple to send a message to a queue, as can be identified by its name. A message can be everything from a string to a number to a more complex object like a dictionary. The BaseReceiver on the other hand can listen to a queue and retrieve messages whenever there are some in the queue.

3.4 Request validation module

Before a service request can be queued and sent to the scheduler, it has to be validated. If the validation fails, an error is instantly returned to the user. Hence, the user will know right away that invalid input parameters were provided or that parameters are missing. If no validation was done, the system would attempt to process the service request normally and it could take minutes or even hours before the code realizes that the input was incorrect.

The validation consists of several parts such as checking whether the user or a processing unit exists. If dates are included, they get checked to ensure that they lie within an acceptable range (from the satellite start until the current date) and that the end date is later than the start date. Beside those basic content checks, a REGEX check is also performed. REGEX stands for REGular EXpression and can be described as a search pattern. This pattern can be used by a search algorithm to search a text for numbers, letters or specific characters. It was developed in theoretical computer science as well as formal language theory and is often used for input validation.

By executing the following command from the request validation module, a request payload can be validated:

```
check_message({"servicename": "test_service", "unit": "123", "begin":
"2019-06-30", "end": "2019-12-31"})
```

In this example, a request with four payload parameters (servicename, unit, begin and end) gets checked. The result is a boolean that states whether the request is valid (True) or invalid (False).

For example, the payload parameter “begin” should include a date. However, it has to be in the specific format “Year-Month-Day" (e.g. 2019-06-30). The REGEX for this would be

```
([12]\d{3}-(0[1-9]|1[0-2])-(0[1-9]|[12]\d|3[01]))
```

If the user does not provide the parameter “begin” or if the user provides an incorrectly formatted date such as “30.06.2019”, the validation fails and the request will not be processed. The user will receive anappropriate explanation as to why the request has failed, either because the parameter was missing or because it has the wrong format.

Note that this module expects that the service and its REGEX check rules are inserted in the "message_checker" table of the "postgres" database.

After extracting the list of parameters and regular expressions for the requested service from the database by using the module described in chapter 4.2, the parameters of the request and the database extraction will be compared. If a required parameter is missing, an error will be returned. In case of a success, the parameter values (e.g. the date of the parameter “begin”) will be validated using the regular expression. This part of the code mainly uses the Python module re[1].

3.5 netCDF storage module

The CLC+ Backbone products consist of a large amount of data and require an efficient storage system. Hence, it was decided to use netCDF files which are basically HDF5 files for scientific use. HDF stands for Hierarchical Data Format. As the name states, the files have an internal data structure. Each HDF5 or netCDF file can include several groups and even subgroups. Moreover, additional information on the groups as well as on the elements within the groups are stored within the hierarchical data format. One big advantage of this format (HDF5 and netCDF) is the memory efficient usage. Data is indexed so that it does not have to be loaded into memory until specifically required. Requesting different elements and their metadata is possible by using respective tools.

The Python based storage module is used for reading from and writing into netCDF files in various ways. Both general geospatial data types; raster and vector data; are supported by this module. While raster data can be

[1] https://docs.python.org/3/library/re.html

read and written by using netCDF groups or a region of interest (shapely Polygon), vector data can also by read and written by querying specific attributes (e.g. "area">5000).

Furthermore, the module provides general functions to obtain the group names of a netCDF file or the timestamps of a specific group in case of raster data.

3.6 Product integration module

This module serves as a utility that uploads quality-checked CLC+ Backbone products to netCDF files. The product's netCDF files can then be accessed via the get_product API in https://api.clcplusbackbone.geoville.com/v1/.

The code runs within a docker container. In case the docker image does not exist yet, go to the directory where the Dockerfile is stored and build it with the command:

```
docker build -t backbone_product_ingestion .
```

An image called backbone_product_ingestion will be created. To run the code with that image in a docker container, you need to use a docker-run-command similar to the following:

```
docker run -u $UID -v /mnt:/mnt backbone_product_ingestion /bin/bash -c
"python3 product_netcdf_upload.py -n Raster -u 161 -i
/mnt/in/task2/output/161/CLMS_CLCplus_RASTER_2018_010m_PU161_03035_V1_0.ti
f
```

Let's explain the command. The '-u $UID' uses the current user (gaf/geoville) within the docker container. This shall avoid file permission errors.
The flag -v mirrors (mounts) a local directory to the respective docker container directory. In this example, the local directory '/mnt', which contains the mounted s3 buckets (e.g. 'task2') of 'https://s3.waw2-1.cloudferro.com' as well as the output netCDF files, gets mirrored with the same path within the docker container.
Subsequently, the docker image name is defined (backbone_product_ingestion), before executing the code with '/bin/bash -c "python3 product_netcdf_upload.py"'. The command 'python3 product_netcdf_upload.py --help' can be executed in order to get help on the input parameters:

```
usage: product_netcdf_upload.py [-h] -n -u -i [-o]
This script converts CLC+ Backbone products to netCDF files.
optional arguments:
 -h, --help show this help message and exit
 -n , --Name Product Name
 -u , --Unit Unit for netCDF grouping (can be PU or SPU, but uniform
 across product)
 -i , --Input Path to the input raster (GeoTiff), input vector
 (Geopackage) or input vector attribute (csv)
 -o , --Output Not required, only for debugging! Path to the output netcdf
file.
```

In the example execution from above, the Raster product (-n) for the unit 161 (-u) shall be inserted. The relevant input path is given to the flag -i.

The code mainly consists of commands from the storage gate module that is described in chapter 3.5.

4 References

5 Annexes

5.1 Source code

5.1.1 bitbucket-pipelines.yml

```
image: node:latest

pipelines:
  branches:
        master:
          - step:
               deployment: clcplus_production
               script:
                 - pipe: atlassian/ssh-run:0.2.4
                   variables:
                     SSH_USER: $SSH_USER
                     SERVER: $SERVER_ADDRESS
                     COMMAND: 'cd clcplus_api && git pull && docker-compose build &&
docker-compose up -d'
                     MODE: 'command'
```

5.1.2 docker-compose.yml

```
version: '3.7'

services:
  reverse-proxy:
    image: traefik:latest
    container_name: proxy_server
    restart: always
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro
      - ./services/proxy_server/traefik/static_config/traefik.yml:/traefik.yml:ro
      - ./services/proxy_server/traefik/dynamic_config:/configs
      - traefik_cert_data:/letsencrypt
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.api.service=api@internal"
      - "traefik.http.routers.api.entryPoints=web_secure_https"
      - "traefik.http.routers.api.rule=Host(`$TRAEFIK_DOMAIN`) && (PathPrefix(`/api`) ||
PathPrefix(`/dashboard`))"
      - "traefik.http.routers.api.middlewares=dashboard_basic_auth"
      -
"traefik.http.middlewares.dashboard_basic_auth.basicauth.users=geoville:$$apr1$$GNDpC4ZG$$
xe16xZTnVOuUo3b4LpHcH0"
      - "traefik.http.routers.api.tls=true"
      - "traefik.http.routers.middlewares=secure-headers@file, compress-content@file"
      - "traefik.http.routers.api.tls.certresolver=myresolver"

  backend:
    build:
      context: ./services/backend_api
      args:
        - GIT_USER=$GIT_USER
        - GIT_PW=$GIT_PW
    container_name: api
    restart: always
    command: gunicorn -c gunicorn_config.py wsgi:app
    ports:
      - "8080:8080"
    env_file:
      - services/backend_api/.env
    logging:
      driver: "json-file"
      options:
        max-size: 10m
        max-file: "3"
    depends_on:
      - db
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.flask_backend.entryPoints=web_secure_https"
      - "traefik.http.routers.flask_backend.rule=Host(`$BACKEND_DOMAIN`)"
      - "traefik.http.routers.flask_backend.tls=true"
      - "traefik.http.routers.flask_backend.tls.certresolver=myresolver"

  db:
    build:
      context: ./services/database
    container_name: postgresql
    restart: always
    command: postgres -c config_file=/etc/postgresql/postgresql.conf
    ports:
      - "5432:5432"
    environment:
      - POSTGRES_USER=$POSTGRES_SUPER_USER
      - POSTGRES_PASSWORD=$POSTGRES_SUPER_USER_PASSWORD
      - DB_DATABASE_NAME=$ADDITIONAL_DATABASE_NAME
      - LC_ALL=C.UTF-8
      - LANG=C.UTF-8
    volumes:
      - ./services/database/db_init_script:/docker-entrypoint-initdb.d/
      - ./services/database/postgresql.conf:/etc/postgresql/postgresql.conf
      - postgresql_data:/var/lib/postgresql/data
    depends_on:
      - reverse-proxy

  message_queue:
    build:
      context: ./services/rabbitmq
    container_name: rabbitmq
    restart: always
    ports:
      - "15672:15672"
      - "15671:15671"
      - "5672:5672"
      - "5671:5671"
    volumes:
      - rabbitmq_data:/var/lib/rabbitmq
      - rabbitmq_logs:/var/log/rabbitmq
      - ./services/rabbitmq/config_files:/etc/rabbitmq

  frontend:
    image: imagehub.geoville.com/clcplus_frontend_web_app:latest
    container_name: frontend
    restart: always
    ports:
      - "9001:80"
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.frontend.entryPoints=web_secure_https"
      - "traefik.http.routers.frontend.rule=Host(`$FRONTEND_DOMAIN`)"
      - "traefik.http.routers.frontend.tls=true"
      - "traefik.http.routers.frontend.tls.certresolver=myresolver"

volumes:
  traefik_cert_data:
  postgresql_data:
  rabbitmq_data:
  rabbitmq_logs:
```

5.1.3 README.md

```
# The CLC+ API server based on Python

## What it offers
* Swagger UI
* API Access
* User based access on API endpoints

## Module description
This project is a complete Python implementation of an lightweight API server instance
based on Flask, Flask-Restx,
Authlib and gunicorn. It provides several API endpoints, contains a complete database
model and user-based access on API
routes via Authlib and the OAuth2 server instance.

Please note that the environment variable listed below are required:

    - FLASK_ENV: development or production
    - DATABASE_CONFIG_FILE: filename of the database.ini file
    - DATABASE_CONFIG_FILE_SECTION: section in the database.ini file which holds the db
connection parameters
    - DATABASE_CONFIG_FILE_SECTION_OAUTH: section in the database.ini file which holds the
oauth db connection parameters
    - OAUTH2_USER: OAuth2 user for the web interface
    - OAUTH2_PASSWORD: OAuth2 password for the web interface
    - OAUTH2_SERVER_BASE_URL: OAuth2 server base URL
    - BEARER_TOKEN_EXPIRATION_TIME: Bearer token expiration time in seconds
    - REFRESH_TOKEN_EXPIRATION_TIME: Refresh token expiration time in seconds
    - RABBIT_MQ_USER: RabbitMQ username
    - RABBIT_MQ_PASSWORD: RabbitMQ password
    - RABBIT_MQ_VHOST: RabbitMQ virtual host
    - RABBIT_MQ_MANAGEMENT_PORT: RabbitMQ port for the management UI
    - RABBIT_MQ_HOST: RabbitMQ host server address
    - LOGGER_QUEUE_NAME: Queue name which stores log messages

## Dependencies
* GeoVille_MS_RabbitMQ_Modul
* GeoVille_MS_Database_Modul
* Geoville_MS_Request_Check_Modul
* Geoville_MS_Logging_Modul
* Geoville_MS_Utils_Modul
* Authlib
* Flask
* flask-cors
* flask-restx
* flask_sqlalchemy
* gunicorn
* pyfiglet
* pyrabbit
* requests


## Acknowledgement
* Author: Michel Schwandner (schwandner@geoville.com)
* Date: 2021-02-01
* Version 21.02
```

5.1.4 services\backend_api\Dockerfile

```
FROM python:3.7

# Set build environment variables coming from the docker-compose env file
ARG GIT_USER
ARG GIT_PW

# Upgrade pip version
RUN /usr/local/bin/python -m pip install --upgrade pip

# Install GEMS modules stored in BitBucket
RUN pip install --upgrade --no-cache-dir
https://$GIT_USER:$GIT_PW@bitbucket.org/geoville/geoville_ms_request_check_modul/get/maste
r.zip
RUN pip install --upgrade --no-cache-dir
https://$GIT_USER:$GIT_PW@bitbucket.org/geoville/geoville_ms_database_modul/get/master.zip
RUN pip install --upgrade --no-cache-dir
https://$GIT_USER:$GIT_PW@bitbucket.org/geoville/geoville_ms_logging_modul/get/master.zip
RUN pip install --upgrade --no-cache-dir
https://$GIT_USER:$GIT_PW@bitbucket.org/geoville/geoville_ms_rabbitmq_modul/get/master.zip
RUN pip install --upgrade --no-cache-dir
https://$GIT_USER:$GIT_PW@bitbucket.org/geoville/geoville_ms_utils_modul/get/master.zip
RUN pip install --upgrade --no-cache-dir
https://$GIT_USER:$GIT_PW@bitbucket.org/geoville/geoville_ms_request_check_modul/get/maste
r.zip

# Copies the requirements file to the Docker image
COPY requirements.txt .

# Install the remaining Python package required
RUN pip install --no-cache-dir -r requirements.txt

# Sets current working directory
WORKDIR /app

# Copies the application code to the current working directory
COPY src/ /app
```

5.1.5 services\backend_api\requirements.txt

```
Authlib
Flask
Flask-Bcrypt
flask-cors
flask-restx
flask_sqlalchemy
gunicorn
pyfiglet
pyrabbit
requests
pyproj
shapely
```

5.1.6 services\backend_api\src\clcplus_API.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
```

```
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Flask App entry point
#
# Date created: 01.06.2020
# Date last modified: 07.07.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.07
#
###########################################################################################
##############################


from init.api_constructor import clcplus_api, clcplus_blueprint
from init.app_constructor import app
from init.namespace_constructor import *
from oauth.oauth2 import config_oauth
from resources.resources_auth.create_client.create_client import CreateOAuthClient
from resources.resources_auth.delete_clients.delete_clients import DeleteOAuthClients
from resources.resources_auth.delete_client_by_id.delete_client_by_id import
DeleteOAuthClient
from resources.resources_auth.delete_tokens.delete_tokens import DeleteTokens
from resources.resources_auth.delete_tokens_by_id.delete_tokens_by_id import
DeleteTokensByID
from resources.resources_auth.get_bearer_token.get_bearer_token import GetBearerToken
from resources.resources_auth.get_clients.get_clients import GetClients
from resources.resources_auth.get_client_by_id.get_client_by_id import GetClientByID
from resources.resources_auth.get_scopes.get_scopes import GetScopes
from resources.resources_auth.get_scope_by_id.get_scope_by_id import GetScopeByID
from resources.resources_auth.login.login import Login
from resources.resources_auth.set_scope_by_id.set_scope_by_id import UpdateScope
from resources.resources_auth.set_token_expiration_time.set_token_expiration_time import
UpdateTokenExpirationTime

from resources.resources_config.add_airflow_config.add_airflow_config import
AddAirflowConfig
from resources.resources_config.add_queue_config.add_queue_config import AddQueueConfig
from resources.resources_config.delete_airflow_config.delete_airflow_config import
DeleteAirflowConfiguration
from
resources.resources_config.delete_airflow_config_by_name.delete_airflow_config_by_name
import DeleteAirflowConfigByName
from resources.resources_config.delete_queue_config.delete_queue_config import
DeleteQueueConfiguration
from resources.resources_config.delete_queue_config_by_id.delete_queue_config_by_id import
DeleteQueueByID
from resources.resources_config.delete_queue_config_by_name.delete_queue_config_by_name
import DeleteQueueByName
from resources.resources_config.get_airflow_config.get_airflow_config import
GetAirflowConfig
from resources.resources_config.get_queue_config.get_queue_config import
GetMessageQueueConfig
from resources.resources_config.get_queue_config_by_id.get_queue_config_by_id import
GetMessageQueueConfigByID
from resources.resources_config.get_queue_config_by_name.get_queue_config_by_name import
GetMessageQueueConfigByName

from resources.resources_rabbitmq.delete_rabbitmq_queue.delete_rabbitmq_queue import
DeleteRabbitMQQueue
from resources.resources_rabbitmq.get_rabbitmq_queues.get_rabbitmq_queues import
RabbitMQListQueues
from resources.resources_rabbitmq.get_rabbitmq_message_count.get_rabbitmq_message_count
import RabbitMQMessageCount
from resources.resources_rabbitmq.get_rabbitmq_server_status.get_rabbitmq_server_status
import RabbitMQServerStatus
```

```
from resources.resources_rabbitmq.get_rabbitmq_users.get_rabbitmq_users import
RabbitMQUsers
from resources.resources_rabbitmq.get_rabbitmq_vhosts.get_rabbitmq_vhosts import
RabbitMQVHosts

from resources.resources_crm.create_customer.create_customer import CreateCustomer
from resources.resources_crm.create_service.create_service import CreateService
from resources.resources_crm.delete_all_customers.delete_all_customers import
DeleteAllCustomers
from resources.resources_crm.delete_all_services.delete_all_services import DeleteServices
from resources.resources_crm.delete_customer_by_id.delete_customer_by_id import
DeleteCustomerById
from resources.resources_crm.delete_customers_by_filter.delete_customers_by_filter import
DeleteCustomersByFilter
from resources.resources_crm.delete_service_by_id.delete_service_by_id import
DeleteServiceById
from resources.resources_crm.delete_service_by_name.delete_service_by_name import
DeleteServiceByName
from resources.resources_crm.get_all_customers.get_all_customers import GetAllCustomers
from resources.resources_crm.get_all_services.get_all_services import GetAllServices
from resources.resources_crm.get_customers_by_filter.get_customers_by_filter import
GetCustomersByFilter
from resources.resources_crm.get_customer_by_id.get_customer_by_id import GetCustomerById
from resources.resources_crm.get_service_by_id.get_service_by_id import GetServiceByID
from resources.resources_crm.get_service_by_name.get_service_by_name import
GetServiceByName
from resources.resources_crm.get_service_orders.get_service_orders import GetServiceOrders
from resources.resources_crm.get_manual_tasks.get_manual_tasks import GetManualTasks
from resources.resources_crm.get_all_tasks.get_all_tasks import GetAllTasks
from resources.resources_crm.create_manual_task.create_manual_task import CreateManualTask
from resources.resources_crm.update_manual_task.update_manual_task import UpdateManualTask
from resources.resources_crm.update_manual_task_spu.update_manual_task_spu import
UpdateManualTaskSPU
from resources.resources_crm.update_manual_task_order_id.update_manual_task_order_id
import UpdateManualTaskOrderID

from resources.resources_logging.log_error.log_error import LogError
from resources.resources_logging.log_info.log_info import LogInfo
from resources.resources_logging.log_warning.log_warning import LogWarning

from resources.resources_rois.create_roi.create_roi import CreateROI
from resources.resources_rois.delete_all_rois.delete_all_rois import DeleteAllROIs
from resources.resources_rois.delete_roi_by_id.delete_roi_by_id import DeleteROIByID
from resources.resources_rois.delete_roi_by_user_id.delete_roi_by_user_id import
DeleteROIByUserID
from resources.resources_rois.get_all_rois.get_all_rois import GetAllROIs
from resources.resources_rois.get_roi_by_id.get_roi_by_id import GetROIByID
from resources.resources_rois.get_roi_by_user_id.get_roi_by_user_id import GetROIByUserID
from resources.resources_rois.set_roi_attributes_by_id.set_roi_attributes_by_id import
UpdateROIAttributes
from resources.resources_rois.update_roi_entity_by_id.update_roi_entity_by_id import
UpdateROIEntity

from
resources.resources_services.batch_classification_production.batch_classification_producti
on import BatchClassificationProduction
from
resources.resources_services.batch_classification_staging.batch_classification_staging
import BatchClassificationStaging
from resources.resources_services.batch_classification_test.batch_classification_test
import BatchClassificationTest
from resources.resources_services.harmonics.harmonics import Harmonics
from resources.resources_services.retransformation.retransformation import
Retransformation
from resources.resources_services.service_order_status.order_status import OrderStatus
from resources.resources_services.task_1_batch_classification.task_1_batch_classification
import Task1BatchClassification
from
resources.resources_services.task_1_feature_classification.task_1_feature_classification
import Task1FeatureCalculation
```

```
from resources.resources_services.task_1_reprocessing.task_1_reprocessing import
Task1Reprocessing
from resources.resources_services.task_1_reprocessing_test.task_1_reprocessing_test import
Task1ReprocessingTest
from resources.resources_services.task_1_stitching.task_1_stitching import Task1Stitching
from resources.resources_services.task_2_apply_model.task_2_apply_model import
Task2ApplyModel
from
resources.resources_services.task_2_apply_model_fast_lane.task_2_apply_model_fast_lane
import Task2ApplyModelFastLane
from resources.resources_services.task_2_feature_calculation.task_2_feature_calculation
import Task2FeatureCalculation
from resources.resources_services.vector_class_attribution.vector_class_attribution import
VectorClassAttribution

from resources.resources_products.get_product.get_product import Products
from resources.resources_products.nations.nations import Nations
from resources.resources_products.get_national_product.get_national_product import
NationalProducts
from resources.resources_products.get_product_europe.get_product_europe import
ProductEurope

###########################################################################################
##############################
# Retrieving the API env variable
###########################################################################################
##############################

auth_namespace.add_resource(CreateOAuthClient, '/clients/create')
auth_namespace.add_resource(DeleteOAuthClients, '/clients/delete')
auth_namespace.add_resource(DeleteOAuthClient, '/clients/delete/<client_id>')
auth_namespace.add_resource(DeleteTokens, '/tokens/delete')
auth_namespace.add_resource(DeleteTokensByID, '/tokens/delete/<user_id>')
auth_namespace.add_resource(GetBearerToken, '/get_bearer_token')
auth_namespace.add_resource(GetClients, '/clients')
auth_namespace.add_resource(GetClientByID, '/clients/<user_id>')
auth_namespace.add_resource(GetScopes, '/scopes')
auth_namespace.add_resource(GetScopeByID, '/scopes/<user_id>')
auth_namespace.add_resource(Login, '/login')
auth_namespace.add_resource(UpdateScope, '/scopes/update')
auth_namespace.add_resource(UpdateTokenExpirationTime, '/tokens/update/expirationTime')

config_namespace.add_resource(AddAirflowConfig, '/airflow/create')
config_namespace.add_resource(AddQueueConfig, '/queues/create')
config_namespace.add_resource(DeleteAirflowConfiguration, '/airflow/delete')
config_namespace.add_resource(DeleteAirflowConfigByName, '/airflow/delete/<service_name>')
config_namespace.add_resource(DeleteQueueConfiguration, '/queues/delete')
config_namespace.add_resource(DeleteQueueByID, '/queues/delete/id/<service_id>')
config_namespace.add_resource(DeleteQueueByName, '/queues/delete/name/<queue_name>')
config_namespace.add_resource(GetAirflowConfig, '/airflow')
config_namespace.add_resource(GetMessageQueueConfig, '/queues')
config_namespace.add_resource(GetMessageQueueConfigByID, '/queues/id/<service_id>')
config_namespace.add_resource(GetMessageQueueConfigByName, '/queues/name/<queue_name>')

rabbitmq_namespace.add_resource(DeleteRabbitMQQueue, '/queues/delete/<queue_name>')
rabbitmq_namespace.add_resource(RabbitMQListQueues, '/queues')
rabbitmq_namespace.add_resource(RabbitMQUsers, '/users')
rabbitmq_namespace.add_resource(RabbitMQVHosts, '/virtualHosts')
rabbitmq_namespace.add_resource(RabbitMQMessageCount, '/messageCount/<queue_name>')
rabbitmq_namespace.add_resource(RabbitMQServerStatus, '/serverStatus')

crm_namespace.add_resource(CreateCustomer,  '/users/create')
crm_namespace.add_resource(GetAllCustomers, '/users')
crm_namespace.add_resource(GetCustomersByFilter, '/users/filter')
crm_namespace.add_resource(GetCustomerById, '/users/<user_id>')
crm_namespace.add_resource(DeleteAllCustomers, '/users/delete')
crm_namespace.add_resource(DeleteCustomerById, '/users/delete/<user_id>')
crm_namespace.add_resource(DeleteCustomersByFilter, '/users/delete/filter')
crm_namespace.add_resource(GetAllServices, '/services')
crm_namespace.add_resource(GetServiceByID, '/services/id/<service_id>')
crm_namespace.add_resource(GetServiceByName, '/services/name/<service_name>')
crm_namespace.add_resource(CreateService, '/services/create')
crm_namespace.add_resource(DeleteServices, '/services/delete')
crm_namespace.add_resource(DeleteServiceById, '/services/delete/id/<service_id>')
crm_namespace.add_resource(DeleteServiceByName, '/services/delete/name/<service_name>')
crm_namespace.add_resource(GetServiceOrders, '/services/order_query')
crm_namespace.add_resource(GetManualTasks, '/manual_tasks/task_query')
crm_namespace.add_resource(CreateManualTask,  '/manual_tasks/create')
crm_namespace.add_resource(UpdateManualTask,  '/manual_tasks/update_state')
crm_namespace.add_resource(UpdateManualTaskSPU,  '/manual_tasks/update_spu_state')
crm_namespace.add_resource(UpdateManualTaskOrderID,  '/manual_tasks/update_order_id')
crm_namespace.add_resource(GetAllTasks,  '/manual_tasks')

logging_namespace.add_resource(LogError, '/log_error')
logging_namespace.add_resource(LogInfo, '/log_info')
logging_namespace.add_resource(LogWarning, '/log_warning')

rois_namespace.add_resource(CreateROI, '/create')
rois_namespace.add_resource(DeleteAllROIs, '/delete')
rois_namespace.add_resource(DeleteROIByID, '/delete/id/<roi_id>')
rois_namespace.add_resource(DeleteROIByUserID, '/delete/user/<user_id>')
rois_namespace.add_resource(GetAllROIs, '/')
rois_namespace.add_resource(GetROIByID, '/id/<roi_id>')
rois_namespace.add_resource(GetROIByUserID, '/user/<user_id>')
rois_namespace.add_resource(UpdateROIAttributes, '/update/filter')
rois_namespace.add_resource(UpdateROIEntity, '/update')

service_namespace.add_resource(BatchClassificationProduction,
'/batch_classification_production')
service_namespace.add_resource(BatchClassificationStaging,
'/batch_classification_staging')
service_namespace.add_resource(BatchClassificationTest, '/batch_classification_test')
service_namespace.add_resource(Harmonics, '/harmonics')
service_namespace.add_resource(OrderStatus, '/order_status/<order_id>')
service_namespace.add_resource(Retransformation, '/retransformation')
service_namespace.add_resource(Task1BatchClassification, '/task1_batch_classification')
service_namespace.add_resource(Task1FeatureCalculation, '/task1_feature_calculation')
service_namespace.add_resource(Task1Reprocessing, '/task1_reprocessing')
service_namespace.add_resource(Task1ReprocessingTest, '/task1_reprocessing_test')
service_namespace.add_resource(Task1Stitching, '/task1_stitching')
service_namespace.add_resource(Task2ApplyModel, '/task2_apply_model')
service_namespace.add_resource(Task2ApplyModelFastLane, '/task2_apply_model_fast_lane')
service_namespace.add_resource(Task2FeatureCalculation, '/task2_feature_calculation')
service_namespace.add_resource(VectorClassAttribution, '/vector_class_attribution')

products_namespace.add_resource(Products, '/get_product')
products_namespace.add_resource(Nations, '/nations')
products_namespace.add_resource(NationalProducts, '/get_national_product')
products_namespace.add_resource(ProductEurope, '/get_product_europe')

#########################################################################################
##############################
# Adding all the required namespaces for internal GeoVille API
#########################################################################################
##############################

clcplus_api.add_namespace(auth_namespace)
clcplus_api.add_namespace(auth_header_namespace)
clcplus_api.add_namespace(general_error_namespace)
clcplus_api.add_namespace(config_namespace)
clcplus_api.add_namespace(crm_namespace)
clcplus_api.add_namespace(logging_namespace)
clcplus_api.add_namespace(rabbitmq_namespace)
clcplus_api.add_namespace(rois_namespace)
clcplus_api.add_namespace(service_namespace)
clcplus_api.add_namespace(products_namespace)

###########################################################################################
##############################
# Configures the Flask app object with the oauth instance running in the background
###########################################################################################
##############################

config_oauth(app)

###########################################################################################
##############################
# Registering the Blueprints of each API version
###########################################################################################
##############################

app.register_blueprint(clcplus_blueprint)

###########################################################################################
##############################
# Run the app in debug mode
###########################################################################################
##############################

# app.run(host=app.config['HOST'], debug=app.config['DEBUG'], port=app.config['PORT'])
```

5.1.7 services\backend_api\src\config.py

```
###########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# API Gateway config file
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
###########################################################################################
##############################

from lib.database_helper import get_database_connection_str
import os


###########################################################################################
##############################
# General configuration class
###########################################################################################
##############################

class Config(object):

    DEBUG = False
    TESTING = False
    DATABASE_CONFIG_FILE = os.environ.get('DATABASE_CONFIG_FILE')
    DATABASE_CONFIG_FILE_SECTION_API = os.environ.get('DATABASE_CONFIG_FILE_SECTION')
    DATABASE_CONFIG_FILE_SECTION_OAUTH =
os.environ.get('DATABASE_CONFIG_FILE_SECTION_OAUTH')

    BUNDLE_ERRORS = True

    RESTX_MASK_SWAGGER = False
    RESTX_INCLUDE_ALL_MODELS = False

    RABBIT_MQ_HOST = os.environ.get('RABBIT_MQ_HOST')
    RABBIT_MQ_USER = os.environ.get('RABBIT_MQ_USER')
    RABBIT_MQ_PASSWORD = os.environ.get('RABBIT_MQ_PASSWORD')
    RABBIT_MQ_MANAGEMENT_PORT = os.environ.get('RABBIT_MQ_MANAGEMENT_PORT')
    RABBIT_MQ_VIRTUAL_HOST = os.environ.get('RABBIT_MQ_VHOST')

    SCOPE_CONNECTOR = 'OR'

    OAUTH_USER = os.environ.get('OAUTH2_USER')
    OAUTH_PASSWORD = os.environ.get('OAUTH2_PASSWORD')
    OAUTH_CREATE_CLIENT_ADDRESS = os.environ.get('OAUTH2_SERVER_BASE_URL') +
'/oauth/create_client'
    OAUTH_GENERATE_TOKEN_ADDRESS = os.environ.get('OAUTH2_SERVER_BASE_URL') +
'/oauth/generate_token'
    OAUTH_VALIDATE_TOKEN_ADDRESS = os.environ.get('OAUTH2_SERVER_BASE_URL') +
'/oauth/validate_token'
    OAUTH_REVOKE_TOKEN_ADDRESS = os.environ.get('OAUTH2_SERVER_BASE_URL') +
'/oauth/revoke_token'
    OAUTH2_TOKEN_EXPIRES_IN = {
        'password': os.environ.get('BEARER_TOKEN_EXPIRATION_TIME'),
        'refresh_token': os.environ.get('REFRESH_TOKEN_EXPIRATION_TIME'),
    }


##########################################################################################
##############################
# Production configuration
##########################################################################################
##############################

class ProductionConfig(Config):

    SQLALCHEMY_DATABASE_URI = get_database_connection_str(Config.DATABASE_CONFIG_FILE,

Config.DATABASE_CONFIG_FILE_SECTION_OAUTH)
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    SQLALCHEMY_ECHO = False


##########################################################################################
##############################
# Development configuration
##########################################################################################
##############################

class DevelopmentConfig(Config):

    DEBUG = True
    PORT = 5001
    HOST = '0.0.0.0'

    SQLALCHEMY_DATABASE_URI = get_database_connection_str(Config.DATABASE_CONFIG_FILE,

Config.DATABASE_CONFIG_FILE_SECTION_OAUTH)
    SQLALCHEMY_TRACK_MODIFICATIONS = True
    SQLALCHEMY_ECHO = True
```

5.1.8 services\backend_api\src\database.ini-dist

```
[postgresql]
host=<database_host_address>
database=<database_name>
user=<database_user>
password=<database_password>
port=<database_port>

[postgresql_oauth]
host=<database_host_address>
database=<database_name>
user=<database_user>
password=<database_password>
port=<database_port>
```

5.1.9 services\backend_api\src\gunicorn_config.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Configuration file for the gunicorn server
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

pidfile = 'geoville_rest_api.pid'
proc_name = 'geoville_rest_api'
workers = 3
bind = '0.0.0.0:8080'
backlog = 2048
accesslog = '-'
errorlog = '-'
timeout = 1200
keepalive = 2
```

5.1.10 services\backend_api\src\wsgi.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# WSGI entry point for the Gunicorn server
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from clcplus_API import app

########################################################################################
##############################
# Starting the app with Gunicorn
########################################################################################
##############################

if __name__ == "__main__":
    app.run()
```

5.1.11 services\backend_api\src\blueprints\hello_Geoville\hello_geoville.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Index Page for the API Gateway
#
# Date created: 01.06.2020
# Date last modified: 07.07.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.07
#
########################################################################################
##############################

from flask import Blueprint
import os
import pyfiglet

########################################################################################
##############################
# Creation of the blueprint
########################################################################################
##############################

index_page = Blueprint('index_page', __name__)


########################################################################################
##############################
# Returns a static HTML page used as index page for API Gateway
########################################################################################
##############################

@index_page.route('/')
def api_hello_geoville():
    """ Returns static content for the index page
    This methods returns a static HTML page containing a nice figlet. The route is used to
serve the index page of
    API Gateway.

    Returns:
        (str): static HTML contents

    """

    ascii_banner = pyfiglet.figlet_format("CLCplus Backbone API\r\n2021")

    html_text_pre = '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html40/strict.dtd">' \
                    '<head>' \
                        '<title>Welcome to the CLCplus Backbone Service API</title>' \
                        '<LINK REL="StyleSheet" href="style1.css" type="text/css">' \
                    '</head>' \
                    '<body>' \
                        '<h1>Welcome to the CLCplus Backbone Service API</h1>' \
                        '<pre class="ascii">'

    html_text_post = f'</pre><p>(V 21.08)</p></body>'

    static_content = html_text_pre + ascii_banner + html_text_post

    return static_content
```

5.1.12 services\backend_api\src\configuration\configuration.py

```
class Configuration:
    def __init__(self, queue_host, queue_port, queue_name, message_key):
        self.queue_host = queue_host
        self.queue_port = queue_port
        self.queue_name = queue_name
        self.message_key = message_key
```

5.1.13 services\backend_api\src\configuration\get_configuration_from_database.py

```
import re
from configuration.configuration import Configuration
from geoville_ms_database.geoville_ms_database import *


def __get_message_key_from_database(databaseconfigfile, databaseconfigsection):
    """
    Extracts the message key from the database
    :param databaseconfigfile: the file path and name for the database.ini file
    :param databaseconfigsection: the section in the database.ini file for postgresql
    :return: a dictionary holding the key
    """
    sql = "SELECT \"name\", \"key\" FROM msgeovilleconfig.message_key"
    result = read_from_database_one_row(sql, None, databaseconfigfile,
databaseconfigsection, False)
    key_store = {result[0]: result[1]}

    return key_store


def __get_message_queue_config_from_database(servicename, databaseconfigfile,
databaseconfigsection):
    """
    Extracts the queue configuration for a given servicename
    :param servicename: the servicename
    :param databaseconfigfile: the file path and name for the database.ini file
    :param databaseconfigsection: the section in the database.ini file for postgresql
    :return: a dictionary holding the queue configuration
    """
    sql = "SELECT  \"key\", \"value\" FROM msgeovilleconfig.message_queue_config " \
          "where queue_name = '" + servicename + "'"
    result = read_from_database_all_rows(sql, None, databaseconfigfile,
databaseconfigsection, False)

    config = __get_dictionary_from_result(result)

    return config


def __get_dictionary_from_result(result):
    """
    returns a dictionary from a database result tuple
    :param result:
    :return: dictionary
    """
    dictionary = {}
    for x, y in result:
        if dictionary.keys() == x:
            dictionary[y].append(y)
        else:
            dictionary[x] = [y]

    return dictionary


def get_queue_configuration(service, databaseconfigfile, databaseconfigsection):
    """
    The public method to get the service configuration
    :param service: servicename
    :param databaseconfigfile: the file path and name for the database.ini file
    :param databaseconfigsection: the section in the database.ini file for postgresql
    :return: an object holding all necessary configuration information for rabbitMQ
    """
    key = __get_message_key_from_database(databaseconfigfile, databaseconfigsection)
    message_queue_config = __get_message_queue_config_from_database(service,
databaseconfigfile, databaseconfigsection)

    host = message_queue_config.get("host")
    port = str(message_queue_config.get("port"))
    port_int = [int(port) for port in re.findall('\\d+', port)]

    api_configuration = Configuration(host[0], port_int[0], service,
key.get("message_key"))

    return api_configuration
```

5.1.14 services\backend_api\src\error_classes\api_base_error\api_base_error.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# API base error class definition
#
```

```
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
#########################################################################################
##############################


class BaseError(Exception):
    """ API base error class

    This class definition serves as API base error class. It
    inherits from general Python Exception class.

    """


#########################################################################################
##########################
    # Constructor method

#########################################################################################
##########################

    def __init__(self, code=None, status=None, description=None, payload=None,
message=None, traceback=None):
        """ Constructor method

        This method is the is constructor method for the API
        error handling base class

        Arguments:

            code (str): HTTP error code
            status (str): HTTP error status
            description (str): HTTP error description
            payload (str): Payload of the current request
            message (str): Individual message derived from the resource
            traceback (str): Error traceback

        """

        Exception.__init__(self)

        self.code = code
        self.description = description
        self.traceback = traceback
        self.status = status
        self.payload = payload
        self.message = message


#########################################################################################
##########################
    # Method for returning a defined error dictionary

#########################################################################################
##########################

    def to_dict(self):
        """ Creates the error message dictionary

        This method returns a pre-defined error dictionary with all necessary
        information about the occurred error. The "error_description" section
        contains already existing information about the error derived from the
        Werkzeuge HTTP exception class. The other information are specific to
        the error message.

        """

        return {
            "error_description": {
                "code": self.code,
                "status": self.status,
                "description": self.description
            },
            "error_definition": {
                "payload": self.payload,
                "message": self.message,
                "traceback": self.traceback
            }

        }
```

5.1.15 services\backend_api\src\error_classes\http_error_400\http_error_400.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# HTTP error 400 (Bad Request) error class definition
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.api_base_error.api_base_error import BaseError
from werkzeug.exceptions import BadRequest


##########################################################################################
##############################
# Bad request error class
##########################################################################################
##############################

class BadRequestError(BaseError):
    """ Constructor method

    This method is the is constructor method for

    """

    def __init__(self, message, payload, traceback):
        """ Constructor method

        This method is the is constructor method for

        Arguments:
            payload (str): Payload of the current request
            message (str): Individual message derived from the resource
            traceback (str): Error traceback

        """

        BaseError.__init__(self)
        self.code = BadRequest.code
        self.status = 'BAD_REQUEST'
        self.description = BadRequest.description

        self.traceback = traceback
        self.payload = payload
        self.message = message
```

5.1.16 services\backend_api\src\error_classes\http_error_401\http_error_401.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# HTTP error 401 (Unauthorized) error class definition
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.api_base_error.api_base_error import BaseError
from werkzeug.exceptions import Unauthorized


##########################################################################################
##############################
# Bad request error class
##########################################################################################
##############################

class UnauthorizedError(BaseError):
    """ Constructor method

    This method is the is constructor method for

    """

    def __init__(self, message, payload, traceback):
        """ Constructor method

        This method is the constructor method for

        Arguments:

            payload (str): Payload of the current request
            message (str): Individual message derived from the resource
            traceback (str): Error traceback
        """

        BaseError.__init__(self)
        self.code = Unauthorized.code
        self.status = 'UNAUTHORIZED'
        self.description = Unauthorized.description

        self.traceback = traceback
        self.payload = payload
        self.message = message
```

5.1.17 services\backend_api\src\error_classes\http_error_403\http_error_403.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# HTTP error 403 (Forbidden) error class definition
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.api_base_error.api_base_error import BaseError
from werkzeug.exceptions import Forbidden


##########################################################################################
##############################
# Forbidden error class
##########################################################################################
##############################

class ForbiddenError(BaseError):
    """ Constructor method

    This method is the is constructor method for

    """

    def __init__(self, message, payload, traceback):
        """ Constructor method

        This method is the is constructor method for

        Arguments:

            payload (str): Payload of the current request
            message (str): Individual message derived from the resource
            traceback (str): Error traceback

        """
        BaseError.__init__(self)
        self.code = Forbidden.code
        self.status = 'FORBIDDEN'
        self.description = Forbidden.description

        self.traceback = traceback
        self.payload = payload
        self.message = message
```

5.1.18 services\backend_api\src\error_classes\http_error_404\http_error_404.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# HTTP error 404 (NotFound) error class definition
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.api_base_error.api_base_error import BaseError
from werkzeug.exceptions import NotFound


##########################################################################################
##############################
# NotFound error class
##########################################################################################
##############################

class NotFoundError(BaseError):
    """ Class definition

    This method is the is constructor method for

    """

    def __init__(self, message, payload, traceback):
        """ Constructor method

        This method is the is constructor method for

        Arguments:

            payload (str): Payload of the current request
            message (str): Individual message derived from the resource
            traceback (str): Error traceback

        """

        BaseError.__init__(self)
        self.code = NotFound.code
        self.status = 'NOT_FOUND'
        self.description = NotFound.description

        self.traceback = traceback
        self.payload = payload
        self.message = message
```

5.1.19 services\backend_api\src\error_classes\http_error_405\http_error_405.py

```
#########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# HTTP error 405 (Method Not Allowed) error class definition
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
#########################################################################################
##############################

from error_classes.api_base_error.api_base_error import BaseError
from werkzeug.exceptions import MethodNotAllowed


#########################################################################################
##############################
# MethodNotAllowed error class
#########################################################################################
##############################

class MethodNotAllowedError(BaseError):
    """ Class definition

    This method is the is constructor method for

    """

    def __init__(self, message, payload, traceback):
        """ Constructor method

        This method is the is constructor method for

        Arguments:

            payload (str): Payload of the current request
            message (str): Individual message derived from the resource
            traceback (str): Error traceback

        """

        BaseError.__init__(self)
        self.code = MethodNotAllowed.code
        self.status = 'METHOD_NOT_ALLOWED'
        self.description = MethodNotAllowed.description

        self.traceback = traceback
        self.payload = payload
        self.message = message
```

5.1.20 services\backend_api\src\error_classes\http_error_408\http_error_408.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# HTTP error 408 (Request Timeout) error class definition
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.api_base_error.api_base_error import BaseError
from werkzeug.exceptions import RequestTimeout


########################################################################################
##############################
# Request Timeout error class
########################################################################################
##############################

class RequestTimeoutError(BaseError):
    """ Class definition

    This class defines the error handler for a request timeout error.

    """

    def __init__(self, message, payload, traceback):
        """ Constructor method

        This method is the is constructor method for the request timeout error

        Arguments:
            payload (str): Payload of the current request
            message (str): Individual message derived from the resource
            traceback (str): Error traceback

        """

        BaseError.__init__(self)
        self.code = RequestTimeout.code
        self.status = 'REQUEST_TIMEOUT'
        self.description = RequestTimeout.description

        self.traceback = traceback
        self.payload = payload
        self.message = message
```

5.1.21 services\backend_api\src\error_classes\http_error_415\http_error_415.py

```
###########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# HTTP error 405 (Unsupported media type) error class definition
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
###########################################################################################
##############################

from error_classes.api_base_error.api_base_error import BaseError
from werkzeug.exceptions import UnsupportedMediaType


###########################################################################################
##############################
# UnsupportedMediaType error class
###########################################################################################
##############################

class UnsupportedMediaTypeError(BaseError):
    """ Class definition

    This method is the is constructor method for

    """

    def __init__(self, message, payload, traceback):
        """ Constructor method

        This method is the is constructor method for

        Arguments:

            payload (str): Payload of the current request
            message (str): Individual message derived from the resource
            traceback (str): Error traceback

        """

        BaseError.__init__(self)
        self.code = UnsupportedMediaType.code
        self.status = 'UNSUPPORTED_MEDIA_TYPE'
        self.description = UnsupportedMediaType.description

        self.traceback = traceback
        self.payload = payload
        self.message = message
```

5.1.22 services\backend_api\src\error_classes\http_error_422\http_error_422.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# HTTP error 422(Unsupported media type) error class definition
#
# Date created: 23.02.2021
# Date last modified: 23.02.2021
#
# __author__  = Patrick Wolf (wolf@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.api_base_error.api_base_error import BaseError
from werkzeug.exceptions import UnprocessableEntity


##########################################################################################
##############################
# UnsupportedMediaType error class
##########################################################################################
##############################

class UnprocessableEntityError(BaseError):
    """ Class definition

    This method is the is constructor method for

    """

    def __init__(self, message, payload, traceback):
        """ Constructor method

        This method is the is constructor method for

        Arguments:

            payload (str): Payload of the current request
            message (str): Individual message derived from the resource
            traceback (str): Error traceback

        """

        BaseError.__init__(self)
        self.code = UnprocessableEntity.code
        self.status = 'UNPROCESSABLE_ENTITY'
        self.description = UnprocessableEntity.description

        self.traceback = traceback
        self.payload = payload
        self.message = message
```

5.1.23 services\backend_api\src\error_classes\http_error_429\http_error_429.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# HTTP error 405 (Unsupported media type) error class definition
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.api_base_error.api_base_error import BaseError
from werkzeug.exceptions import TooManyRequests


##########################################################################################
##############################
# UnsupportedMediaType error class
##########################################################################################
##############################

class TooManyRequestsError(BaseError):
    """ Class definition

    This method is the is constructor method for

    """

    def __init__(self, message, payload, traceback):
        """ Constructor method

        This method is the is constructor method for

        Arguments:

            payload (str): Payload of the current request
            message (str): Individual message derived from the resource
            traceback (str): Error traceback

        """

        BaseError.__init__(self)
        self.code = TooManyRequests.code
        self.status = 'TOO_MANY_REQUESTS'
        self.description = TooManyRequests.description

        self.traceback = traceback
        self.payload = payload
        self.message = message
```

5.1.24 services\backend_api\src\error_classes\http_error_500\http_error_500.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# HTTP error 500 (Internal Server Error) error class definition
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.api_base_error.api_base_error import BaseError
from werkzeug.exceptions import InternalServerError


##########################################################################################
##############################
# InternalServerError error class
##########################################################################################
##############################

class InternalServerErrorAPI(BaseError):
    """ Class definition

    This method is the is constructor method for

    """

    def __init__(self, message, payload, traceback):
        """ Constructor method

        This method is the is constructor method for

        Arguments:

            payload (str): Payload of the current request
            message (str): Individual message derived from the resource
            traceback (str): Error traceback

        """

        BaseError.__init__(self)
        self.code = InternalServerError.code
        self.status = 'INTERNAL_SERVER_ERROR'
        self.description = InternalServerError.description

        self.traceback = traceback
        self.payload = payload
        self.message = message
```

5.1.25 services\backend_api\src\error_classes\http_error_501\http_error_501.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# HTTP error 501 (Not Implemented) error class definition
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.api_base_error.api_base_error import BaseError
from werkzeug.exceptions import NotImplemented


########################################################################################
##############################
# NotImplemented error class
########################################################################################
##############################

class NotImplementedError(BaseError):
    """ Class definition

    This method is the is constructor method for

    """

    def __init__(self, message, payload, traceback):
        """ Constructor method

        This method is the is constructor method for

        Arguments:

            payload (str): Payload of the current request
            message (str): Individual message derived from the resource
            traceback (str): Error traceback

        """

        BaseError.__init__(self)
        self.code = NotImplemented.code
        self.status = 'NOT_IMPLEMENTED'
        self.description = NotImplemented.description

        self.traceback = traceback
        self.payload = payload
        self.message = message
```

5.1.26 services\backend_api\src\error_classes\http_error_503\http_error_503.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# HTTP error 503 (Service Unavailable) error class definition
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.api_base_error.api_base_error import BaseError
from werkzeug.exceptions import ServiceUnavailable


##########################################################################################
##############################
# NotImplemented error class
##########################################################################################
##############################

class ServiceUnavailableError(BaseError):
    """ Class definition

    This method is the is constructor method for

    """

    def __init__(self, message, payload, traceback):
        """ Constructor method

        This method is the is constructor method for

        Arguments:

            payload (str): Payload of the current request
            message (str): Individual message derived from the resource
            traceback (str): Error traceback

        """

        BaseError.__init__(self)
        self.code = ServiceUnavailable.code
        self.status = 'SERVICE_UNAVAILABLE'
        self.description = ServiceUnavailable.description

        self.traceback = traceback
        self.payload = payload
        self.message = message
```

5.1.27 services\backend_api\src\init\api_constructor.py

```
#########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# API entry point
#
# Date created: 01.06.2020
# Date last modified: 07.07.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.07
#
#########################################################################################
##############################

from flask import Blueprint
from flask_restx import Api

#########################################################################################
##############################
# Definition of the Swagger UI for the internal GeoVille API
#########################################################################################
##############################

clcplus_blueprint = Blueprint('clcplus_api', __name__, url_prefix='/v1')

clcplus_api = Api(clcplus_blueprint,
                  title='CLCplus Backbone API',
                  version='21.08',
                  description='CLCplus Backbone Service API based on a Microservice
architecture',
                  contact='IT-Services@geoville.com'
                  )
```

5.1.28 services\backend_api\src\init\app_constructor.py

```
#########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Flask App entry point
#
# Date created: 01.06.2020
# Date last modified: 01.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
#########################################################################################
##############################

from blueprints.hello_Geoville.hello_geoville import index_page
from flask import Flask
from flask_bcrypt import Bcrypt
from flask_cors import CORS
from werkzeug.middleware.proxy_fix import ProxyFix
import os

##########################################################################################
##############################
# Template folder retrieval
##########################################################################################
##############################

template_dir = os.path.join(os.path.split(os.path.dirname(os.path.abspath(__file__)))[0],
'templates')

##########################################################################################
##############################
# Creation of the Flask App entry point
##########################################################################################
##############################

app = Flask(__name__)

##########################################################################################
##############################
# Configuration of the Flask App depending on ENV variable
##########################################################################################
##############################

if app.config["ENV"] == "production":
    app.config.from_object("config.ProductionConfig")

else:
    app.config.from_object("config.DevelopmentConfig")

##########################################################################################
##############################
# Creation of the Limiter object for limiting access to particular routes
##########################################################################################
##############################

app.wsgi_app = ProxyFix(app.wsgi_app, x_for=1, x_host=1)

##########################################################################################
##############################
# Added CORS for the app
##########################################################################################
##############################

CORS(app)

##########################################################################################
##############################
# Password encryption for the resource owner user creation
##########################################################################################
##############################

bcrypt = Bcrypt(app)

##########################################################################################
##############################
# Register the blueprints
##########################################################################################
##############################

app.register_blueprint(index_page)
```

5.1.29 services\backend_api\src\init\init_env_variables.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Initialising environment variables
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from init.app_constructor import app

##########################################################################################
##############################
# RabbitMQ environment variables
##########################################################################################
##############################

rabbitmq_host = app.config['RABBIT_MQ_HOST']
rabbitmq_user = app.config['RABBIT_MQ_USER']
rabbitmq_password = app.config['RABBIT_MQ_PASSWORD']
rabbitmq_management_port = app.config['RABBIT_MQ_MANAGEMENT_PORT']
rabbitmq_virtual_host = app.config['RABBIT_MQ_VIRTUAL_HOST']

##########################################################################################
##############################
# Database environment variables
##########################################################################################
##############################

database_config_file = app.config['DATABASE_CONFIG_FILE']
database_config_section_api = app.config['DATABASE_CONFIG_FILE_SECTION_API']
database_config_section_oauth = app.config['DATABASE_CONFIG_FILE_SECTION_OAUTH']

##########################################################################################
##############################
# OAuth2 environment variables
##########################################################################################
##############################

oauth2_create_client = app.config['OAUTH_CREATE_CLIENT_ADDRESS']
oauth2_generate_token = app.config['OAUTH_GENERATE_TOKEN_ADDRESS']
oauth2_validate_token = app.config['OAUTH_VALIDATE_TOKEN_ADDRESS']
oauth2_revoke_token = app.config['OAUTH_REVOKE_TOKEN_ADDRESS']
oauth2_user = app.config['OAUTH_USER']
oauth2_password = app.config['OAUTH_PASSWORD']
oauth2_bearer_expiration_time = app.config['OAUTH2_TOKEN_EXPIRES_IN']['password']
oauth2_refresh_expiration_time = app.config['OAUTH2_TOKEN_EXPIRES_IN']['refresh_token']
```

5.1.30 services\backend_api\src\init\namespace_constructor.py

```
##########################################################################################
##############################
```

```
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Namespace creation for the API definitions
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from flask_restx import Namespace

##########################################################################################
##############################
# Namespaces available in the Swagger UI
##########################################################################################
##############################

auth_namespace = Namespace('auth', description='Authentication related operations')
config_namespace = Namespace('config', description='Configuration related operations')
rabbitmq_namespace = Namespace('rabbitmq', description='RabbitMQ related operations')
crm_namespace = Namespace('crm', description='CRM related operations')
rois_namespace = Namespace('roi', description='Region of interest related operations')
service_namespace = Namespace('services', description='Service related operations')
logging_namespace = Namespace('logging', description='Logging related operations')
general_error_namespace = Namespace('error_models', description='Inlcudes all error
models')
auth_header_namespace = Namespace('auth', description='Authentication header model')
monitoring_namespace = Namespace('monitoring', description='Operations related to
monitoring')
products_namespace = Namespace('products', description='Order final products')
```

5.1.31 services\backend_api\src\lib\auth_header.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Initialising environment variables
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from init.namespace_constructor import auth_header_namespace as api
```

```
########################################################################################
##############################
# Authorization header for the bearer token
########################################################################################
##############################

auth_header_parser = api.parser()
auth_header_parser.add_argument('Authorization',
                                location='headers',
                                required=True,
                                help='An access token is required to query this resource')
```

5.1.32 services\backend_api\src\lib\database_helper.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Database helper methods
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from configparser import ConfigParser
from geoville_ms_database.geoville_ms_database import execute_database,
read_from_database_one_row, \
    read_from_database_all_rows
import json


########################################################################################
##############################
# Method definition for checking if a user exists
########################################################################################
##############################

def check_email_existence(email, database_config_file, database_config_section):
    """ Checks if a user exists

    This method returns a boolean. Depending on the check if the current user/customer
exists, it returns True or False.

    Arguments:
        email (str): unique identifier of customer
        database_config_file (str): database.ini file
        database_config_section (str): section of the database.ini file

    Returns:
        (bool): True if exists

    """

    db_query = """SELECT
                      customer_id, password, CONCAT(first_name, ' ',  last_name )
                      FROM
                          customer.customer
                      WHERE
                          lower(email) = lower(%s) AND
                          deleted_at IS NULL
                   """

    res = read_from_database_one_row(db_query, (email,), database_config_file,
database_config_section, True)

    if res is None or res is False:
        return False

    else:
        return [res[0], res[1], res[2]]


##########################################################################################
##############################
# Method definition for retrieving the client secret
##########################################################################################
##############################

def get_client_id_secret(client_id, database_config_file, database_config_section):
    """ Queries client secret from the database

    This method returns either the client secret for the corresponding client ID from the
database or a boolean (False),
    depending on if the current client ID exists.

    Arguments:
        client_id (str): unique identifier of a client
        database_config_file (str): database.ini file
        database_config_section (str): section of the database.ini file

    Returns:
        (str): current client secret
        (bool): False if the client ID does not exist

    """

    db_query = "SELECT client_secret FROM public.oauth2_client WHERE client_id = %s"
    res = read_from_database_one_row(db_query, (client_id,), database_config_file,
database_config_section, True)

    if res is None or res is False:
        return False

    else:
        return res[0]


##########################################################################################
##############################
# Method definition for retrieving the order status
##########################################################################################
##############################

def query_order_status(order_id, database_config_file, database_config_section):
    """ Returns the status of a submitted order

    This method returns the status of a submitted order, including the status, result path
if it exists and the success
    message, or None if the order ID does not exist

    Arguments:
        order_id (str): ID of the current order
        database_config_file (str): database.ini file
        database_config_section (str): section of the database.ini file

    Returns:
        (tuple): order status tuple

    """

    db_query = "SELECT status, result, success FROM customer.service_orders WHERE order_id
= %s"
    order_res = read_from_database_one_row(db_query, (order_id,), database_config_file,
database_config_section, True)

    if order_res:
        return order_res

    else:
        return None


##########################################################################################
##############################
# Method definition for retrieving the service ID
##########################################################################################
##############################

def get_service_id(service_name, database_config_file, database_config_section):
    """ Returns the service ID

    This method returns the service ID on the result of the database query. Either it will
be the service ID or None.

    Arguments:
        service_name (str): Name of the service to be requested
        database_config_file (str): database.ini file
        database_config_section (str): section of the database.ini file

    Returns:
        (str): service ID

    """

    db_query = "SELECT service_id FROM customer.services WHERE service_name = %s"
    service_id = read_from_database_one_row(db_query, (service_name,),
database_config_file, database_config_section,
                                            True)

    if service_id:
        return service_id[0]

    else:
        return None


#####################################
```

5.1.33 services\backend_api\src\lib\general_helper_methods.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
```

```
# General helper methods
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_401.http_error_401 import UnauthorizedError
from error_classes.http_error_403.http_error_403 import ForbiddenError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from geoville_ms_database.geoville_ms_database import read_from_database_one_row
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from geoville_ms_publisher.publisher import Publisher
from init.app_constructor import bcrypt
import json


##########################################################################################
##############################
# Method for the generation of a Bcrypt password hash
##########################################################################################
##############################

def generate_bcrypt_hash(input_str):
    """ Returns two lists for the SQL query statement

    This method returns depending on the content of the incoming request JSON, two lists
with the SQL query parameters
    and its corresponding values

    Arguments:
        input_str (dict): json content of the POST request

    Returns:
        (list): parameter and value list

    """

    return bcrypt.generate_password_hash(input_str).decode('utf-8')


##########################################################################################
##############################
# Method for the generation of the parameters and values of the SQL query
##########################################################################################
##############################

def parameter_and_value_list_generation(req_content):
    """ Returns two lists for the SQL query statement

    This method returns depending on the content of the incoming request JSON, two lists
with the SQL query parameters
    and its corresponding values

    Arguments:
        req_content (dict): json content of the POST request

    Returns:
        (list): parameter and value list

    """

    val_list = []
    param_list = []

    for key, value in req_content.items():
        if key:
            if key == 'geoJSON':
                param_list.append("geom = ST_Force_2D(ST_SetSRID(ST_GeomFromGeoJSON(%s),
4326))")
                val_list.append(json.dumps(req_content[key]))

            elif value == ('RECEIVED', 'QUEUED'):
                param_list.append(f"{key} in %s ")
                val_list.append(req_content[key])

            elif key != 'roi_id' and key != 'Authorization' and value is not None:
                param_list.append(f"{key} = %s ")
                val_list.append(req_content[key])

    return param_list, val_list


##########################################################################################
##############################
# Method definition for sending a JSON to a queue
##########################################################################################
##############################

def publish_to_queue(service_name, order_id, payload):
    """ Sends a JSON to queue

    This method publishes a JSON, containing the order ID and the payload of an incoming
request to a RabbitMQ queue of
    corresponding service.

    Arguments:
        service_name (str): name of the service
        order_id (str): current order ID
        payload (dict): cleaned payload of the request

    """

    publisher = Publisher(service_name)

    publisher.publish({
        "order_id": order_id,
        "parameters": json.dumps(payload)
    })


##########################################################################################
##############################
# Method definition for defining a HTTP response depending on the incoming request
##########################################################################################
##############################

def create_request_response(request_response, request_payload):
    """ Defines a response in JSON format

    This method returns a message dictionary with the contained error and the corrsponding
HTTP status code depending on
    the incoming request payload.

    Arguments:
        request_response (str): response of the request
        request_payload (dict): request payload

    Returns:
         (dict): HTTP status code and message dictionary
    """
    if request_response.status_code == 400:
        error = BadRequestError(request_response.json(), '', '') if request_response.json
is not None \
            else BadRequestError('Client ID does not exist', '', '')
        gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-validate_token')
        return {'message': error.to_dict()}, 400

    elif request_response.status_code == 401:
        error = UnauthorizedError(request_response.json(), "", "") if
request_response.json is not None \
            else UnauthorizedError('You are not authorized to access the resource', '',
'')
        gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-validate_token')
        return {'message': error.to_dict()}, 401

    elif request_response.status_code == 403:
        error = ForbiddenError(request_response.json(), '', '') if request_response.json
is not None \
            else ForbiddenError('You do not have sufficient access rights', '', '')
        gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-validate_token', )
        return {'message': error.to_dict()}, 403

    else:
        error = InternalServerErrorAPI(request_response.json(), request_payload,
                                       "") if request_response.json is not None \
            else InternalServerErrorAPI('Could not collect information about error
source', request_payload, '')
        gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-validate_token')
        return {'message': error.to_dict()}, 500


#########################################################################################
###########################################################################
# Method to validate a GeoJSON with an online validation tool
#########################################################################################
###########################################################################

def validate_geojson(json_file, db_config_file, db_config_section_api):
    """ Validates a GeoJSON with an online validation tool

    This method is using an online validation tool in order to check if the input file is
a correct GeoJSON file.
    Depending on the result of the request, a text message is returned.

    Arguments:
        json_file (str): GeoJSON representation of the file
        db_config_file (str):
        db_config_section_api (str):
    Returns:
         (bool): True if it is a correct GeoJSON

    Raises:
        Exception: returns False

    """

    db_query = """SELECT
                  valid(ST_IsValidDetail(ST_GeomFromGeoJSON(%s))),
                  reason(ST_IsValidDetail(ST_GeomFromGeoJSON(%s)))
               """

    try:
        data = read_from_database_one_row(db_query, (json.dumps(json_file),
json.dumps(json_file)), db_config_file,
                                          db_config_section_api, True)

        if data[0] is True:
            return [data[0]]

        else:
            return [data[0], data[1].split(":")[1].strip()]

    except Exception as err:
        return [False, f'Error: {err}']
```

5.1.34 services\backend_api\src\lib\hashing_helper.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Hashing methods
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from datetime import datetime
import hashlib


########################################################################################
##############################
# Method definition for generating hash string of the ROI ID
########################################################################################
##############################

def generate_roi_id_hash(customer_id, service_id):
    """ Generates a ROI ID hash string

    This method returns a sha256 hash string from the concatenation of
    the customer_id, the service_id and the current timestamp.

    Arguments:
        customer_id (str): send customer ID
        service_id (str): send service ID

    Returns:
        (str): sha256 hash string

    """

    hash_str =
f"{customer_id}{service_id}{datetime.now().strftime('%Y%m%d%H:%M:%S')}".strip().lower()
    return hashlib.sha256(hash_str.encode()).hexdigest()


########################################################################################
##############################
# Method definition for generating the service ID string
########################################################################################
##############################
```

```
def generate_service_id_hash(service_name, geoville_owner):
    """ Generates the service ID hash

    This method returns a sha256 hash string from the concatenation of
    the service name and the GeoVille owner of a customer.

    Arguments:
        service_name (str): first name of the customer
        geoville_owner (str): last name of the customer

    Returns:
        (str): sha256 hash string

    """

    hash_str = f"{service_name}{geoville_owner}".strip().lower()
    return hashlib.sha256(hash_str.encode()).hexdigest()


##########################################################################################
##############################
# Method definition for generating the service ID string
##########################################################################################
##############################

def generate_task_id_hash(task_name, task_owner):
    """ Generates the service ID hash

    This method returns a sha256 hash string from the concatenation of the task name and
the task owner.

    Arguments:
        task_name (str): name of the task
        task_owner (str): name of the task owner

    Returns:
        (str): sha256 hash string

    """

    hash_str = f"{task_name}{task_owner}".strip().lower()
    return hashlib.sha256(hash_str.encode()).hexdigest()
```

5.1.35 services\backend_api\src\lib\rabbitmq_helper.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Hashing methods
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################
```

```
##########################################################################################
##############################
# Method definition for retrieving all available virtual hosts
##########################################################################################
##############################

def list_virtual_host_names(client):
    """ Returns all available virtual hosts

    This method returns a list of all available virtual hosts by defining the connection
parameters of the RabbitMQ
    service.

    Arguments:
        client (obj): Pyrabbit connection object

    Returns:
        (list): all available virtual hosts

    Raises:
        HTTPError: aborts the process with detailed error information
        NetworkError: aborts the process with detailed error information

    """

    return client.get_vhost_names()


##########################################################################################
##############################
# Method definition for retrieving all available queue names
##########################################################################################
##############################

def list_queue_names(client):
    """ Returns all available queue names

    This method returns a list of all available queue names by defining the connection
parameters of the RabbitMQ
    service.

    Arguments:
        client (obj): Pyrabbit connection object

    Returns:
        (list): all available queue names

    Raises:
        HTTPError: aborts the process with detailed error information
        NetworkError: aborts the process with detailed error information

    """

    return [q['name'] for q in client.get_queues()]


##########################################################################################
##############################
# Method definition for counting messages for a specified queue name
##########################################################################################
##############################

def get_queue_message_count(client, virtual_host, queue_name):
    """ Counts messages for a specified queue name
    This method returns the amount of messages for a specified queue name and virtual host
address by defining the
    connection parameters of the RabbitMQ service.

    Arguments:
        client (obj): Pyrabbit connection object
        virtual_host (str): virtual host address
        queue_name (str): specified queue name

    Returns:
        (int): amount of messages in the queue

    Raises:
        HTTPError: aborts the process with detailed error information
        NetworkError: aborts the process with detailed error information

    """

    return client.get_queue_depth(virtual_host, queue_name)


##########################################################################################
##############################
# Method definition for purging a queue by name
##########################################################################################
##############################

def purge_queue(client, virtual_host, queue_name):
    """ Purges a specified queue name

    This method purges queues from the RabbitMQ service for specified queue name and the
virtual host by defining the
    connection parameters of the RabbitMQ service.

    Arguments:
        client (obj): Pyrabbit connection object
        virtual_host (str): virtual host address
        queue_name (str): specified queue name

    Returns:
        (list): contains a bool and the possible error message

    Raises:
        HTTPError: aborts the process with detailed error information
        NetworkError: aborts the process with detailed error information

    """

    try:
        client.delete_queue(virtual_host, queue_name)

    except Exception as err:
        return [False, err]

    else:
        return [True]
```

5.1.36 services\backend_api\src\lib\request_helper.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
```

```
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Foreign requests methods
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
#########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_401.http_error_401 import UnauthorizedError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_408.http_error_408 import RequestTimeoutError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import (database_config_file, database_config_section_oauth,
oauth2_generate_token,
                                     oauth2_create_client, oauth2_password, oauth2_user)
from lib.database_helper import get_scope_by_id
from requests.exceptions import HTTPError
import requests
import traceback


#########################################################################################
##############################
# Method definition for creating a OAuth client on the OAuth2 server
#########################################################################################
##############################

def create_oauth_client(client_name):
    """ Creates an OAuth2 client

    This method creates a HTTP POST request in order to create an OAuth2 client in the
OAuth2 database. The OAuth2 API
    returns in case of a success a dictionary with the client ID and the client secret
which is used for the customer
    creation resource route.

    Arguments:
        client_name (str): combination of first and last name of the customer

    Returns:
        (dict): response from the OAuth2 server as JSON

    """

    try:

        payload = {'client_name': client_name,
                   'grant_type': 'password\nrefresh_token',
                   'response_type': 'code',
                   'client_uri': '',
                   'redirect_uri': '',
                   'scope': '',
                   'token_endpoint_auth_method': 'client_secret_basic'}

        headers = {'Content-Type': 'application/x-www-form-urlencoded'}

        response = requests.request("POST", oauth2_create_client, headers=headers,
data=payload, timeout=15)
```

```
    except requests.exceptions.ReadTimeout:
        error = RequestTimeoutError('Connection timed out while contacting the 
Authorization server', '', '')
        gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
create_oauth_client_helper')
        return [None, {'message': error.to_dict()}, 408]

    except Exception:
        error = InternalServerErrorAPI('Could not get a response from the Authorisation 
server', '', '')
        gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
create_oauth_client_helper')
        return [None, {'message': error.to_dict()}, 500]

    else:
        if response.status_code == 200:
            return [response.json()]

        else:
            error = InternalServerErrorAPI(f'Status code is different: 
{response.status_code}', '', '')
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
create_oauth_client_helper')
            return [None, {'message': error.to_dict()}, 500]


##########################################################################################
##############################
# Method definition for creating a OAuth client on the OAuth2 server
##########################################################################################
##############################

def get_bearer_token(client_id, client_secret):
    """ Creates an OAuth2 client

    This method creates a HTTP POST request in order to create an OAuth2 client in the 
OAuth2 database. The OAuth2 API
    returns in case of a success a dictionary with the client ID and the client secret 
which is used for the customer
    creation resource route.

    Arguments:
        client_id (str): combination of first and last name of the customer
        client_secret (str):

    Returns:
        (dict): response from the OAuth2 server as JSON

    """

    try:
        scope = get_scope_by_id(client_id, database_config_file, 
database_config_section_oauth)

        files = {
            'grant_type': (None, "password"),
            'username': (None, oauth2_user),
            'password': (None, oauth2_password),
            'scope': (None, scope),
        }

        response = requests.post(oauth2_generate_token, files=files, auth=(client_id, 
client_secret), timeout=15)

    except KeyError as err:
        error = BadRequestError(f'Key error resulted in a BadRequest: {err}', '', 
traceback.format_exc())
        return {'message': error.to_dict()}, 400
    except requests.exceptions.ReadTimeout:
        error = RequestTimeoutError('Connection timed out while contacting the 
Authorization server', '', '')
        return {'message': error.to_dict()}, 408

    except HTTPError:
        error = NotFoundError(f'Could not connect to OAuth2 server', '', 
traceback.format_exc())
        return {'message': error.to_dict()}, 404

    except Exception:
        error = InternalServerErrorAPI('Unexpected error occurred', '', 
traceback.format_exc())
        return {'message': error.to_dict()}, 500

    else:
        if response.status_code == 200:
            return response.json()

        elif response.status_code == 401:
            error = UnauthorizedError('Submitted client is invalid', '', '')
            return {'message': error.to_dict()}, 401

        else:
            error = InternalServerErrorAPI(f'Unexpected error: {response.text}', '', '')
            return {'message': error.to_dict()}, 501
```

5.1.37 services\backend_api\src\models\general_models\general_models.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is 
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Service success response models for the Swagger UI
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import service_namespace as api

##########################################################################################
##############################
# Hypermedia service model
##########################################################################################
##############################

hypermedia_model = api.model('hypermedia_model',
                             {
                                 'href': fields.String(
                                     description='URI to linked resource',
                                     example='/services/order_status/<order_id>'
                                 ),
                                 'rel': fields.String(
                                     description='Linked resource',
                                        example='services'
                                    ),
                                    'type': fields.String(
                                        description='Type of HTTP method',
                                        example='GET'
                                    )
                                })

##########################################################################################
##############################
# Response model for the POST service request
##########################################################################################
##############################

service_success_response_model = api.model('service_success_response_model',
                                          {
                                              'message': fields.String(
                                                  description='Success message',
                                                  example='Order successfully received'
                                              ),
                                              'links': fields.Nested(
                                                  hypermedia_model,
                                                  description='Hypermedia model'
                                              )
                                          })
```

5.1.38 services\backend_api\src\models\models_auth\access_token_models\access_token_models.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Bearer token models for the Swagger UI
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import auth_namespace as api

##########################################################################################
##############################
# Request model for the POST request of the Bearer Token generation process
##########################################################################################
##############################

bearer_token_request_model = api.model('get_bearer_token_request_model',
                                       {
                                           'user_id': fields.String(
                                               description='User specific client id',
                                               example='8KfYSDj8Wq2iNtIly98M5ES4',
                                               required=True
```

```
                                                         ),
                                                         'client_secret': fields.String(
                                                             description='User specific client secret',
                                                             example='Cgc8lMd5LozQz5ifiqMks',
                                                             required=True
                                                         )
                                                     })

########################################################################################
##############################
# Response model for the POST request of the Bearer Token generation process
########################################################################################
##############################

bearer_token_response_model = api.model('bearer_token_response_model',
                                        {
                                            'access_token': fields.String(
                                                description='Bearer token for the user',
                                                example='ABCDEF1234'
                                            ),
                                            'expires_in': fields.Integer(
                                                description='The expiration duration',
                                                example=1000
                                            ),
                                            'refresh_token': fields.String(
                                                description='Refresh token for the user',
                                                example='ABCDEF1234'
                                            ),
                                            'token_type': fields.String(
                                                description='The type of the returned
token',

                                                example='Bearer'
                                            )
                                        })
```

5.1.39 services\backend_api\src\models\models_auth\client_models\client_models.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Client models for the Swagger UI
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import auth_namespace as api

########################################################################################
##############################
# Request model for the POST request of the OAuth2 client creation
########################################################################################
##############################
```

```
auth_client_request_model = api.model('auth_client_request_model',
                                      {
                                          'client_name': fields.String(
                                              description='Name of the new customer',
                                              example='Max Mustermann',
                                              required=True
                                          )
                                      })

##########################################################################################
##############################
# Response model for the POST request of the OAuth2 client creation
##########################################################################################
##############################

auth_client_response_model = api.model('auth_client_response_model',
                                       {
                                           'user_id': fields.String(
                                               description='Client ID returned from the
endpoint',

                                               example='8KfYSDj8Wq2iNtIly98M5ES4'
                                           ),
                                           'user_id': fields.String(
                                               description='Client secret returned from
the endpoint',

                                               example='Cgc8lMd5LozQz5ifiqMks'
                                           )
                                       })

##########################################################################################
##############################
# Client response model
##########################################################################################
##############################

client_response_model = api.model('client_response_model',
                                   {
                                       'user_id': fields.String(
                                           description='User specific client ID',
                                           example='8KfYSDj8Wq2iNtIly98M5ES4'
                                       ),
                                       'client_name': fields.String(
                                           description='Complete name of the client',
                                           example='Max Mustermann'
                                       ),
                                       'grant_type': fields.String(
                                           description='User specific grant type of the
OAuth flow',

                                           example='password'
                                       ),
                                       'response_type': fields.String(
                                           description='User specific response type of the
OAuth flow',

                                           example='code'
                                       ),
                                       'scope': fields.String(
                                           description='Authorization scope for the user',
                                           example='user'
                                       )
                                   })

##########################################################################################
##############################
# List of clients response model
##########################################################################################
##############################
```

```
clients_list_response_model = api.model('clients_list_response_model',
                                        {
                                            'oauth_clients': fields.List(
                                                fields.Nested(client_response_model)
                                            ),
                                        })
```

5.1.40 services\backend_api\src\models\models_auth\login_model\login_model.py

```
##########################################################################################
##############################
#
# Copyright (c) 2020, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Bearer token models for the Swagger UI
#
# Date created: 02.03.2020
# Date last modified: 01.07.2020
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 20.07
#
##########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import auth_namespace as api

##########################################################################################
##############################
# Request model for the POST request of the Bearer Token generation process
##########################################################################################
##############################

login_request_model = api.model('login_request_model',
                                {
                                    'email': fields.String(
                                        description='E-mail address of the user',
                                        example='max@mustermann.de',
                                        required=True
                                    ),
                                    'password': fields.String(
                                        description='Password of the user login',
                                        example='Cgc8lMd5LozQz5ifiqMks',
                                        required=True
                                    )
                                })

##########################################################################################
##############################
# Response model for the POST request of the Bearer Token generation process
##########################################################################################
##############################

login_response_model = api.model('login_response_model',
                                 {
                                     'access_token': fields.String(
                                         description='Bearer token for the user',
                                         example='ABCDEF1234'
                                     ),
                                     'expires_in': fields.Integer(
                                         description='The expiration duration in seconds',
                                         example=1000
                                     ),
                                     'refresh_token': fields.String(
                                         description='Refresh token for the user',
                                         example='ABCDEF1234'
                                     ),
                                     'token_type': fields.String(
                                         description='The type of the returned token',
                                         example='Bearer'
                                     ),
                                     'client_id': fields.String(
                                         description='Client ID returned from the
endpoint',
                                         example='8KfYSDj8Wq2iNtIly98M5ES4'
                                     ),
                                     'client_secret': fields.String(
                                         description='Client secret returned from the
endpoint',
                                         example='Cgc8lMd5LozQz5ifiqMks'
                                     )
                                 })
```

5.1.41 services\backend_api\src\models\models_auth\scope_models\scope_models.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Scope models for the Swagger UI
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import auth_namespace as api

########################################################################################
##############################
# Request model for the PUT request
########################################################################################
##############################

scope_update_request_model = api.model('scope_update_request_model',
                                       {
                                           'client_id': fields.String(
                                               description='User specific client ID',
                                               example='8KfYSDj8Wq2iNtIly98M5ES4',
                                               required=True
                                           ),
                                           'scope': fields.String(
                                               description='Authorization scope for the
user',
                                                    example='user',
                                                    required=True
                                                )
                                            })

#########################################################################################
##############################
# Response model for the POST request
#########################################################################################
##############################

scope_response_model = api.model('scope_response_model',
                                 {
                                     'client_id': fields.String(
                                         description='User specific client ID',
                                         example='8KfYSDj8Wq2iNtIly98M5ES4'
                                     ),
                                     'scope': fields.String(
                                         description='Authorization scope for the user',
                                         example='user'
                                     )
                                 })

#########################################################################################
##############################
# Response model for the GET request
#########################################################################################
##############################

scope_list_response_model = api.model('scope_list_response_model',
                                      {
                                          'scopes': fields.List(
                                              fields.Nested(scope_response_model)
                                          ),
                                      })
```

5.1.42 services\backend_api\src\models\models_auth\token_models\token_models.py

```
#########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# General token models for the Swagger UI
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
#########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import auth_namespace as api

#########################################################################################
##############################
# Request model for the token expiration time
```

```
##########################################################################################
##############################

exp_time_request_model = api.model('exp_time_request_model',
                                   {
                                       'bearer_token': fields.String(
                                           description='Bearer token for which the time
should be set',
                                           example='ABCDEF1234'
                                       ),
                                       'exp_time': fields.Integer(
                                           description='Expiration duration in seconds',
                                           example=1000,
                                           min=1,
                                           max=2000000000
                                       ),
                                   })
```

5.1.43 services\backend_api\src\models\models_config\airflow_models\airflow_config_models.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Airflow config models for the Swagger UI
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import config_namespace as api

##########################################################################################
##############################
# Request model for the POST request of adding a new Airflow entry
##########################################################################################
##############################

add_airflow_config_model = api.model('add_airflow_config_model',
                                     {
                                         'service_name': fields.String(
                                             description='Unique identifier of a service',
                                             example='service_name',
                                             required=True
                                         ),
                                         'command': fields.String(
                                             description='Command to trigger the DAG',
                                             example='airflow trigger_dag',
                                             required=True
                                         ),
                                         'description': fields.String(
                                                   description='Short description of the service
and command',
                                                   example='The command triggers the DAG for the
service',
                                                   required=True
                                               ),
                                           })

########################################################################################################################
# Request model for the POST request of deleting an Airflow entry by service name
########################################################################################################################

airflow_config_success_model = api.model('airflow_config_success_model',
                                         {
                                             'service_name': fields.String(
                                                 description='Name of the service',
                                                 example='service_name',
                                                 required=True
                                             )
                                         })

########################################################################################################################
# Response model for retrieving the entire configuration
########################################################################################################################

airflow_config_list_model = api.model('airflow_config_list_model',
                                      {
                                          'airflow_config': fields.List(fields.Nested(
                                              add_airflow_config_model,
                                              description='List of detailed airflow
configurations')
                                          ),
                                      })
```

5.1.44 services\backend_api\src\models\models_config\queue_config\queue_config_models.py

```
########################################################################################################################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Queue configuration models for the Swagger UI
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################################################

from flask_restx import fields
from init.namespace_constructor import config_namespace as api
```

```
##########################################################################################
##############################
# Request model for the POST request
##########################################################################################
##############################

add_queue_config_model = api.model('add_queue_config_model',
                                   {
                                       'service_id': fields.String(
                                           description='Unique identifier of a service',
                                           example='6237b6905d0d45',
                                           required=True
                                       ),
                                       'queue_name': fields.String(
                                           description='Name of the RabbitMQ queue',
                                           example='queue_name',
                                           required=True
                                       ),
                                       'host': fields.String(
                                           description='Host of the RabbitMQ instance',
                                           example='dev.services.geoville.com',
                                           required=True
                                       ),
                                       'port': fields.Integer(
                                           description='Port of the RabbitMQ instance',
                                           example=5672,
                                           required=True
                                       )
                                   })

##########################################################################################
##############################
# Success model for the POST request
##########################################################################################
##############################

queue_creation_success_model = api.model('queue_creation_success_model',
                                         {
                                             'service_id': fields.String(
                                                 description='Unique identifier of a
service',
                                                 example='6237b6905d0d45',
                                                 required=True
                                             ),
                                             'queue_name': fields.String(
                                                 description='Name of the RabbitMQ queue',
                                                 example='queue_name',
                                                 required=True
                                             )
                                         })

##########################################################################################
##############################
# Request model for the POST request
##########################################################################################
##############################

delete_queue_config_model = api.model('delete_queue_config_model',
                                      {
                                          'queue': fields.String(
                                              description='Rabbit MQ queue name to be
deleted',
                                              example='service_name',
                                              required=True
                                          )
                                      })
```

```
##########################################################################################
##############################
# Response model for retrieving the entire configuration
##########################################################################################
##############################

queue_config_list_model = api.model('queue_config_list_model',
                                    {
                                        'message_checker_config':
fields.List(fields.Nested(
                                            add_queue_config_model,
                                            description='List of detailed message checker
configurations')
                                        ),
                                    })
```

5.1.45 services\backend_api\src\models\models_crm\customer_models\customer_models.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Customer models for the Swagger UI
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import crm_namespace as api

##########################################################################################
##############################
# Request model for the POST request
##########################################################################################
##############################

customer_id_model = api.model('customer_id_model',
                              {
                                  'user_id': fields.String(
                                      description='Unique identifier of a customer',
                                      example="8KfYSDj8Wq2iNtIly98M5ES4",
                                      required=True
                                  ),
                              })

##########################################################################################
##############################
# Response model for the POST request
##########################################################################################
##############################

customer_creation_response_model = api.model('customer_creation_response_model',
                                             {
                                                 'client_id': fields.String(
                                                         description='Unique identifier of a
customer in OAuth2',
                                                         example='8KfYSDj8Wq2iNtIly98M5ES4'
                                                     ),
                                                     'client_secret': fields.String(
                                                         description='Secret for further
authorisation',
                                                         example='ece97e8804fb8239'
                                                     )
                                                 })

###########################################################################################
##############################
# Request model for POST and DELETE requests
###########################################################################################
##############################

customer_model = api.model('customer_model',
                           {
                               'title': fields.String(
                                   description='form of address',
                                   required=True,
                                   example='Mr'
                               ),
                               'first_name': fields.String(
                                   description='first name of the customer',
                                   required=True,
                                   example='Max'
                               ),
                               'last_name': fields.String(
                                   description='last name of the customer',
                                   required=True,
                                   example='Mustermann'
                               ),
                               'email': fields.String(
                                   description='e-mail address of the customer',
                                   required=True,
                                   pattern="(^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-
.]+$)",
                                   example='max_mustermann@mustermann.de'
                               ),
                               'password': fields.String(
                                   description='e-mail address of the customer',
                                   required=True,
                                   example='ABCD1234'
                               ),
                               'address': fields.String(
                                   description='1st address of the customer',
                                   example='Sparkassenplatz 1',
                                   required=True
                               ),
                               'zip_code': fields.String(
                                   description='zip code of the address',
                                   example='6020',
                                   required=True
                               ),
                               'city': fields.String(
                                   description='city of the customer',
                                   example='Innsbruck',
                                   required=True
                               ),
                               'country': fields.String(
                                   description='country of the customer',
                                   example='Austria',
                                   required=True
                               ),
                               'nationality': fields.String(
                                                      description='nationality of the customer',
                                                      example='German'
                                                  ),
                                                  'phone': fields.String(
                                                      description='phone number of the customer',
                                                      example='012345',
                                                      required=True
                                                  ),
                                                  'company_name': fields.String(
                                                      description='Company name of the customer',
                                                      example='GeoVille GmbH',
                                                      required=True
                                                  ),
                                              })

###########################################################################################
##############################
# Request model for POST and DELETE requests
###########################################################################################
##############################

customer_filter_model = api.model('customer_filter_model',
                                  {
                                      'title': fields.String(
                                          description='form of address',
                                          example='Mr'
                                      ),
                                      'first_name': fields.String(
                                          description='first name of the customer',
                                          example='Max'
                                      ),
                                      'last_name': fields.String(
                                          description='last name of the customer',
                                          example='Mustermann'
                                      ),
                                      'email': fields.String(
                                          description='e-mail address of the customer',
                                          pattern="(^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-
zA-Z0-9-.]+$)",

                                          example='max_mustermann@mustermann.de'
                                      ),
                                      'address': fields.String(
                                          description='1st address of the customer',
                                          example='Sparkassenplatz 1'
                                      ),
                                      'zip_code': fields.String(
                                          description='zip code of the address',
                                          example='6020'
                                      ),
                                      'city': fields.String(
                                          description='city of the customer',
                                          example='Innsbruck'
                                      ),
                                      'country': fields.String(
                                          description='country of the customer',
                                          example='Austria'
                                      ),
                                      'nationality': fields.String(
                                          description='nationality of the customer',
                                          example='German'
                                      ),
                                      'phone': fields.String(
                                          description='phone number of the customer',
                                          example='012345'
                                      ),
                                      'company_name': fields.String(
                                          description='Company name of the customer',
                                        example='GeoVille GmbH',
                                        required=True
                                    ),
                                })

###########################################################################################
##############################
# Response model for GET, POST and DELETE requests
###########################################################################################
##############################

customer_list_response_model = api.model('customer_list_response_model',
                                         {
                                             'customers': fields.List(
                                                 fields.Nested(
                                                     customer_filter_model
                                                 )
                                             ),
                                         })
```

5.1.46 services\backend_api\src\models\models_crm\manual_tasks_models\manual_tasks_models.py

```
###########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Service models for the Swagger UI
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
###########################################################################################
##############################

from datetime import datetime
from flask_restx import fields
from init.namespace_constructor import crm_namespace as api

###########################################################################################
##############################
# Request model for the POST request
###########################################################################################
##############################

manual_task_request_model = api.model('manual_task_request_model',
                                      {
                                          'processing_unit': fields.List(
                                              fields.String,
                                              description='Unique identifier of a cell',
                                              example=['10kmE108N256', '10kmE109N257'],
                                              required=False
                                          ),
                                          'subproduction_unit': fields.List(
                                              fields.String,
                                              description='Unique identifier of a cell',
                                              example=['64', '118'],
                                              required=False
                                          ),
                                          'task_id': fields.String(
                                              description='Unique identifier of the manual
task',

                                              example='12345',
                                              required=True
                                          ),
                                          'service_id': fields.String(
                                              description='Unique identifier of the
service',

example='046ab5de45ca923bb276f877845642a156d9589bf500bcf2756e3f38da0839fe',
                                              required=True

                                          ),

                                          # 'status': fields.String(
                                          #     description='Status of the manual task',
                                          #     example='TODO',
                                          #     required=True
                                          # ),
                                          # 'result': fields.String(
                                          #     description='Result of the manual task',
                                          #     example='test'
                                          # ),
                                          # 'comment': fields.String(
                                          #     description='Comment for the manual task',
                                          #
example='0259bc74928c7bf958be38c767c30ac57da4f9b543dc351ad1456df28014c992'
                                          # ),
                                          #'refers_to_order_id': fields.Integer(
                                          #    description='Order-ID which is connected to
the manual task',

                                          #    example='4a34a348500d7f3e799b3095df994602'
                                          # ),

                                          'client_id': fields.String(
                                              description='Unique identifier of a customer
in OAuth2',

                                              example='S6aIHB1NOSbaj1ghq99pXq9a',
                                              required=True
                                          ),
                                      })

###########################################################################################
##############################
# Hypermedia manual task model
###########################################################################################
##############################

hypermedia_model = api.model('hypermedia_model',
                             {
                                 'href': fields.String(
                                     description='URI to linked resource',

example='/crm/manual_tasks/task_query?processing_unit=<processing_unit>'
                                 ),
                                 'rel': fields.String(
                                     description='Linked resource',
                                     example='manual_tasks'
                                 ),
                                 'type': fields.String(
                                     description='Type of HTTP method',
                                     example='GET'
                                 )
                             })
```

```
############################################################################################
##############################
# Response model for the POST request
############################################################################################
##############################

manual_task_response_model = api.model('manual_task_response_model',
                                         {
                                             'message': fields.String(
                                                 description='Success message',
                                                 example='Manual task successfully
created'
                                             ),
                                             'links': fields.Nested(
                                                 hypermedia_model,
                                                 description='Hypermedia model'
                                             )
                                         })

############################################################################################
##############################
# Object model for the get_all_services GET request
############################################################################################
##############################

task_query_model = api.model('task_query_model',
                              {
                                  'subproduction_unit': fields.String(
                                          description='Unique identifier of a sub-
production unit',

                                          example='1_2'
                                      ),
                                  'processing_unit': fields.String(
                                      description='Unique identifier of a processing unit',
                                      example='10kmE108N256'
                                  ),
                                  'service_name': fields.String(
                                      description='Unique name of a service',
                                      example='service_name'
                                  ),
                                  'order_status': fields.String(
                                      description='Status of an order',
                                      example='order status'
                                  ),
                                  'order_id': fields.String(
                                      description='Unique identifier of an order',
                                      example='2bfedd049fb5cff841e4965fabbeec46'
                                  ),
                                  'task_name': fields.String(
                                      description='Unique name of a manual task',
                                      example='task_name'
                                  ),
                                  'task_status': fields.String(
                                      description='Status of a manual task',
                                      example='task status'
                                  ),
                                  'task_result': fields.String(
                                      description='Result of a manual task',
                                      example='task result'
                                  )

                              })

task_object_model = api.model('task_object_model',
                                  {
                                      'task_id': fields.String(
                                          description='Unique identifier of a service',

example='a798a06534046dadfac995b3f3806122317dba4391022a1f9296b911b789f344'
                                      ),
                                      'task_name': fields.String(
                                          description='Unique name of a service',
                                          example='service_name'
                                      ),
                                      'task_comment': fields.String(
                                          description='Description of what the service
offers',

                                          example='The service calculates...'
                                      ),
                                      'task_validity': fields.Boolean(
                                          description='Indicator if the service is active',
                                          example=True
                                      ),
                                      'task_owner': fields.String(
                                          description='Responsible person within GeoVille',
                                          example='IT-Services'
                                      ),
                                      'external': fields.Boolean(
                                          description='Internal or external service',
                                          example=True
                                      ),
                                      'date_of_creation': fields.DateTime(
                                          description='Date of creation of the service',
                                          example=datetime.now().strftime("%Y-%m-
%dT%H:%M:%S")
                                      ),
                                      'order_id_not_required': fields.DateTime(
                                          description='False, if manual task is based on a
service result',

                                          example=True
                                      )
                                  })

##########################################################################################
##############################
# Response model for the get_all_services GET request
##########################################################################################
##############################

task_list_model = api.model('task_list_model',
                              {
                                  'tasks': fields.List(fields.Nested(
                                      task_object_model,
                                      description='Detailed task information')
                                  ),
                              })

task_query_list_model = api.model('task_query_list_model',
                                 {
                                     'tasks': fields.List(fields.Nested(
                                         task_query_model,
                                         description='Detailed query information')
                                     ),
                                 })

##########################################################################################
##############################
# Request model for updating a task (state and result)
##########################################################################################
##############################


manual_task_update_model = api.model('manual_task_update_model',
                                 {
                                     'state': fields.String(
                                         description='Task state',
                                         example='finished',
                                         required=True
                                     ),
                                     'result': fields.String(
                                         description='Result path of the task (allowed 
states: not_started, '
                                                    'in_progress, failed, finished) ',
                                         example='/results/result_12345.tif',
                                         required=False
                                     ),
                                     'processing_unit': fields.String(
                                         description='Unique identifier of a processing 
unit',
                                         example='10kmE108N256',
                                         required=True
                                     ),
                                     'service_id': fields.String(
                                         description='Unique id of a service',

example='0259bc74928c7bf958be38c767c30ac57da4f9b543dc351ad1456df28014c992',
                                         required=True
                                     ),
                                     'task_id': fields.String(
                                         description='Unique id of a task',
                                         example='12345',
                                         required=True
                                     ),
                                     'client_id': fields.String(
                                         description='Unique identifier of a customer in 
OAuth2',
                                         example='S6aIHB1NOSbaj1ghq99pXq9a',
                                         required=False
                                     ),
                                 })

##########################################################################################
##############################
# Request model for updating a task (state and result) based on SPU
##########################################################################################
##############################


manual_task_update_spu_model = api.model('manual_task_update_spu_model',
                                 {
                                     'state': fields.String(
                                         description='Task state',
                                         example='finished',
                                         required=True
                                     ),
                                     'result': fields.String(
                                         description='Result path of the task (allowed 
states: not_started, '
                                                    'in_progress, failed, finished) ',
                                         example='/results/result_12345.tif',
                                         required=False
                                     ),
                                     'subproduction_unit': fields.String(
                                         description='Unique identifier of a 
subproduction unit',
                                         example='123',
                                         required=True
                                     ),
                                     'service_id': fields.String(
                                         description='Unique id of a service',

example='0259bc74928c7bf958be38c767c30ac57da4f9b543dc351ad1456df28014c992',
                                         required=True
                                     ),
                                     'task_id': fields.String(
                                         description='Unique id of a task',
                                         example='12345',
                                         required=True
                                     ),
                                     'client_id': fields.String(
                                         description='Unique identifier of a customer in
OAuth2',
                                         example='S6aIHB1NOSbaj1ghq99pXq9a',
                                         required=False
                                     ),
                                 })

##########################################################################################
##############################
# Request model for updating the order_id
##########################################################################################
##############################

manual_task_update_order_id_model = api.model('manual_task_update_order_id_model',
                                 {
                                     'refers_to_order_id': fields.String(
                                         description='Order-ID',
                                         example='035f4fc8b82544666c97357551441e32',
                                         required=True
                                     ),
                                     'processing_unit': fields.String(
                                         description='Unique identifier of a processing
unit',
                                         example='10kmE108N256',
                                         required=True
                                     ),
                                     'service_id': fields.String(
                                         description='Unique id of a service',
example='0259bc74928c7bf958be38c767c30ac57da4f9b543dc351ad1456df28014c992',
                                         required=True
                                     ),
                                     'task_id': fields.String(
                                         description='Unique id of a task',
                                         example='12345',
                                         required=True
                                     ),
                                     'client_id': fields.String(
                                         description='Unique identifier of a customer in
OAuth2',
                                         example='S6aIHB1NOSbaj1ghq99pXq9a',
                                         required=False
                                     ),
                                 })

##########################################################################################
##############################
# Response model for the PUT request (/crm/manual_tasks/update_order_id)
##########################################################################################
##############################

manual_task__update_order_id_response_model =
api.model('manual_task__update_order_id_response_model',
                                       {
                                           'message': fields.String(
                                               description='Success message',
                                               example='Your update has been
successfully submitted'
                                           ),
                                           'refers_to_order_id': fields.String(
                                                description='Order-ID',

example='92d32e1fbda9957138d6dde42d064a12'
                                            ),

                                        })

##########################################################################################
##############################
# Response model for the PUT request (/crm/manual_tasks/update_state)
##########################################################################################
##############################

manual_task_update_state_response_model =
api.model('manual_task_update_state_response_model',
                                        {
                                            'message': fields.String(
                                                description='Success message',
                                                example='Your update has been
successfully submitted'
                                            )
                                        })
```

5.1.47 services\backend_api\src\models\models_crm\service_models\service_models.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Service models for the Swagger UI
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from datetime import datetime
from flask_restx import fields
from init.namespace_constructor import crm_namespace as api

##########################################################################################
##############################
# Request model for the POST request
##########################################################################################
##############################

service_creation_model = api.model('service_creation_model',
                                  {
                                      'service_name': fields.String(
                                          description='Unique name of a service',
                                          example='service_name',
                                          required=True
                                      ),
                                      'service_comment': fields.String(
                                          description='Description of what the service
offers',
                                          example='The service calculates...'
```

```
                                        ),
                                        'service_validity': fields.Boolean(
                                            description='Indicator if the service is
active',

                                            example=True,
                                            required=True
                                        ),
                                        'service_owner_geoville': fields.String(
                                            description='Responsible person within
GeoVille',

                                            example='IT-Services',
                                            required=True
                                        ),
                                        'external': fields.Boolean(
                                            description='Internal or external service',
                                            example=True,
                                            required=True
                                        )
                                    })

##########################################################################################
##############################
# Response model for the POST request
##########################################################################################
##############################

service_id_model = api.model('service_id_model',
                             {
                                 'service_id': fields.String(
                                     description='Unique identifier of a service',

example='69179d1f69e3766406e7008500a1fe468652c951055b419d254c070b9e59c001',
                                     required=True
                                 ),
                             })

##########################################################################################
##############################
# Object model for the get_all_services GET request
##########################################################################################
##############################

service_query_model = api.model('service_query_model',
                                 {
                                     'subproduction_unit': fields.String(
                                         description='Unique identifier of a sub-
production unit',

                                         example='1_2'
                                     ),
                                     'processing_unit': fields.String(
                                         description='Unique identifier of a processing
unit',

                                         example='10kmE108N256'
                                     ),
                                     'service_name': fields.String(
                                         description='Unique name of a service',
                                         example='service_name'
                                     ),
                                     'order_status': fields.String(
                                         description='Status of an order',
                                         example='order status'
                                     ),
                                     'order_id': fields.String(
                                         description='Unique identifier of an order',
                                         example='2bfedd049fb5cff841e4965fabbeec46'
                                     ),
                                     'order_json': fields.String(
                                         description='Order payload in form of a json',
                                         example='{"key": value}'
                                     ),
                                     'order_result': fields.String(
                                         description='Path to the order result',
                                         example='/.../result.tif'
                                     )
                                 })

service_object_model = api.model('service_object_model',
                                 {
                                     'service_id': fields.String(
                                         description='Unique identifier of a service',

example='a798a06534046dadfac995b3f3806122317dba4391022a1f9296b911b789f344'
                                     ),
                                     'service_name': fields.String(
                                         description='Unique name of a service',
                                         example='service_name'
                                     ),
                                     'service_comment': fields.String(
                                         description='Description of what the service
offers',
                                         example='The service calculates...'
                                     ),
                                     'service_validity': fields.Boolean(
                                         description='Indicator if the service is active',
                                         example=True
                                     ),
                                     'service_owner_geoville': fields.String(
                                         description='Responsible person within GeoVille',
                                         example='IT-Services'
                                     ),
                                     'external': fields.Boolean(
                                         description='Internal or external service',
                                         example=True
                                     ),
                                     'date_of_creation': fields.DateTime(
                                         description='Date of creation of the service',
                                         example=datetime.now().strftime("%Y-%m-
%dT%H:%M:%S")
                                     )
                                 })

#########################################################################################
##############################
# Response model for the get_all_services GET request
#########################################################################################
##############################

service_list_model = api.model('service_list_model',
                              {
                                  'services': fields.List(fields.Nested(
                                      service_object_model,
                                      description='Detailed service information')
                                  ),
                              })

query_list_model = api.model('query_list_model',
                            {
                                 'services': fields.List(fields.Nested(
                                     service_query_model,
                                     description='Detailed query information')
                                 ),
                            })
```

5.1.48 services\backend_api\src\models\models_error\error_base_model.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Error base definition model for all kind of errors
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import general_error_namespace as api

##########################################################################################
##############################
# Error definition model for all kind of errors
##########################################################################################
##############################

error_definition = api.model('error_definition_model', {
    'message': fields.String,
    'payload': fields.String(),
    'traceback': fields.String()
})
```

5.1.49 services\backend_api\src\models\models_error\http_error_400.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# BadRequest error model
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import general_error_namespace as api
from models.models_error.error_base_model import error_definition
from werkzeug.exceptions import BadRequest
```

```
########################################################################################
##############################
# Nested response model for a BadRequest error
########################################################################################
##############################

error_description = api.model('error_description_400_model', {
    'code': fields.String(
        description='HTTP error status code',
        example=BadRequest.code,
        default=BadRequest.code
    ),
    'status': fields.String(
        description='HTTP error status',
        example='BAD_REQUEST',
        default='BAD_REQUEST'
    ),
    'description': fields.String(
        description='Detailed HTTP error description',
        example=BadRequest.description,
        default=BadRequest.description
    )
})

error_dicts = api.model('error_dict_400_model', {
    'error_description': fields.Nested(error_description),
    'error_definition': fields.Nested(error_definition)
})

error_400_model = api.model('error_400_model', {
    'message': fields.Nested(error_dicts)
})
```

5.1.50 services\backend_api\src\models\models_error\http_error_401.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Unauthorized error model
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import general_error_namespace as api
from models.models_error.error_base_model import error_definition
from werkzeug.exceptions import Unauthorized

########################################################################################
##############################
# Nested response model for Unauthorized errors
```

```
##########################################################################################
##############################

error_description = api.model('error_description_401_model', {
    'code': fields.String(
        description='HTTP error status code',
        example=Unauthorized.code,
        default=Unauthorized.code
    ),
    'status': fields.String(
        description='HTTP error status',
        example='UNAUTHORIZED',
        default='UNAUTHORIZED'
    ),
    'description': fields.String(
        description='Detailed HTTP error description',
        example=Unauthorized.description,
        default=Unauthorized.description
    )
})

error_dicts = api.model('error_dict_401_model', {
    'error_description': fields.Nested(error_description),
    'error_definition': fields.Nested(error_definition)
})

error_401_model = api.model('error_401_model', {
    'message': fields.Nested(error_dicts)
})
```

5.1.51 services\backend_api\src\models\models_error\http_error_403.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Forbidden error model
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import general_error_namespace as api
from models.models_error.error_base_model import error_definition
from werkzeug.exceptions import Forbidden

##########################################################################################
##############################
# Nested response model for Forbidden errors
##########################################################################################
##############################

error_description = api.model('error_description_403_model', {
    'code': fields.String(
        description='HTTP error status code',
        example=Forbidden.code,
        default=Forbidden.code
    ),
    'status': fields.String(
        description='HTTP error status',
        example='FORBIDDEN',
        default='FORBIDDEN'
    ),
    'description': fields.String(
        description='Detailed HTTP error description',
        example=Forbidden.description,
        default=Forbidden.description
    )
})

error_dicts = api.model('error_dict_403_model', {
    'error_description': fields.Nested(error_description),
    'error_definition': fields.Nested(error_definition)
})

error_403_model = api.model('error_403_model', {
    'message': fields.Nested(error_dicts)
})
```

5.1.52 services\backend_api\src\models\models_error\http_error_404.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# NotFound error model
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import general_error_namespace as api
from models.models_error.error_base_model import error_definition
from werkzeug.exceptions import NotFound

##########################################################################################
##############################
# Nested response model for the NotFound errors
##########################################################################################
##############################

error_description = api.model('error_description_404_model', {
    'code': fields.String(
        description='HTTP error status code',
        example=NotFound.code,
        default=NotFound.code
    ),
    'status': fields.String(
        description='HTTP error status',
        example='NOT_FOUND',
        default='NOT_FOUND'
    ),
    'description': fields.String(
        description='Detailed HTTP error description',
        example=NotFound.description,
        default=NotFound.description
    )
})

error_dicts = api.model('error_dict_404_model', {
    'error_description': fields.Nested(error_description),
    'error_definition': fields.Nested(error_definition),
})

error_404_model = api.model('error_404_model', {
    'message': fields.Nested(error_dicts),
})
```

5.1.53 services\backend_api\src\models\models_error\http_error_405.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Unauthorized model for the API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import general_error_namespace as api
from models.models_error.error_base_model import error_definition
from werkzeug.exceptions import MethodNotAllowed

########################################################################################
##############################
# Nested response model for the MethodNotAllowed errors
########################################################################################
##############################

error_description = api.model('error_description_405_model', {
    'code': fields.String(
        description='HTTP error status code',
        example=MethodNotAllowed.code,
        default=MethodNotAllowed.code
    ),
    'status': fields.String(
        description='HTTP error status',
        example='METHOD_NOT_ALLOWED',
        default='METHOD_NOT_ALLOWED'
    ),
    'description': fields.String(
        description='Detailed HTTP error description',
        example=MethodNotAllowed.description,
        default=MethodNotAllowed.description
    )
})

error_dicts = api.model('error_dict_405_model', {
    'error_description': fields.Nested(error_description),
    'error_definition': fields.Nested(error_definition),
})

error_405_model = api.model('error_405_model', {
    'message': fields.Nested(error_dicts),
})
```

5.1.54 services\backend_api\src\models\models_error\http_error_408.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Request timeout model for the API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import general_error_namespace as api
from models.models_error.error_base_model import error_definition
from werkzeug.exceptions import RequestTimeout

##########################################################################################
##############################
# Nested response model for the request timeout errors
##########################################################################################
##############################

error_description = api.model('error_description_408_model', {
    'code': fields.String(
        description='HTTP error status code',
        example=RequestTimeout.code,
        default=RequestTimeout.code
    ),
    'status': fields.String(
        description='HTTP error status',
        example='METHOD_NOT_ALLOWED',
        default='METHOD_NOT_ALLOWED'
    ),
    'description': fields.String(
        description='Detailed HTTP error description',
        example=RequestTimeout.description,
        default=RequestTimeout.description
    )
})

error_dicts = api.model('error_dict_408_model', {
    'error_description': fields.Nested(error_description),
    'error_definition': fields.Nested(error_definition),
})

error_408_model = api.model('error_408_model', {
    'message': fields.Nested(error_dicts),
})
```

5.1.55 services\backend_api\src\models\models_error\http_error_415.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# UnsupportedMediaType error model
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import general_error_namespace as api
from models.models_error.error_base_model import error_definition
from werkzeug.exceptions import UnsupportedMediaType

##########################################################################################
##############################
# Nested response model for the UnsupportedMediaType errors
##########################################################################################
##############################

error_description = api.model('error_description_415_model', {
    'code': fields.String(
        description='HTTP error status code',
        example=UnsupportedMediaType.code,
        default=UnsupportedMediaType.code
    ),
    'status': fields.String(
        description='HTTP error status',
        example='UNSUPPORTED_MEDIA_TYPE',
        default='UNSUPPORTED_MEDIA_TYPE'
    ),
    'description': fields.String(
        description='Detailed HTTP error description',
        example=UnsupportedMediaType.description,
        default=UnsupportedMediaType.description
    )
})
```

```
error_dicts = api.model('error_dict_415_model', {
    'error_description': fields.Nested(error_description),
    'error_definition': fields.Nested(error_definition),
})

error_415_model = api.model('error_415_model', {
    'message': fields.Nested(error_dicts),
})
```

5.1.56 services\backend_api\src\models\models_error\http_error_422.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# NotFound error model
#
# Date created: 23.02.2021
# Date last modified: v
#
# __author__  = Patrick wolf (wolf@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import general_error_namespace as api
from models.models_error.error_base_model import error_definition
from werkzeug.exceptions import UnprocessableEntity

########################################################################################
##############################
# Nested response model for a UnprocessableEntity error
########################################################################################
##############################

error_description = api.model('error_description_422_model', {
    'code': fields.String(
        description='HTTP error status code',
        example=UnprocessableEntity.code,
        default=UnprocessableEntity.code
    ),
    'status': fields.String(
        description='HTTP error status',
        example='UNPROCESSABLE_ENTITY',
        default='UNPROCESSABLE_ENTITY'
    ),
    'description': fields.String(
        description='Detailed HTTP error description',
        example=UnprocessableEntity.description,
        default=UnprocessableEntity.description
    )
})

error_dicts = api.model('error_dict_422_model', {
    'error_description': fields.Nested(error_description),
    'error_definition': fields.Nested(error_definition)
})

error_422_model = api.model('error_422_model', {
    'message': fields.Nested(error_dicts)
})
```

5.1.57 services\backend_api\src\models\models_error\http_error_429.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# UnsupportedMediaType error model
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import general_error_namespace as api
from models.models_error.error_base_model import error_definition
from werkzeug.exceptions import TooManyRequests

########################################################################################
##############################
# Nested response model for the UnsupportedMediaType errors
########################################################################################
##############################

error_description = api.model('error_description_429_model', {
    'code': fields.String(
        description='HTTP error status code',
        example=TooManyRequests.code,
        default=TooManyRequests.code
    ),
    'status': fields.String(
        description='HTTP error status',
        example='TOO_MANY_REQUESTS',
        default='TOO_MANY_REQUESTS'
    ),
    'description': fields.String(
        description='Detailed HTTP error description',
        example=TooManyRequests.description,
        default=TooManyRequests.description
    )
})

error_dicts = api.model('error_dict_429_model', {
    'error_description': fields.Nested(error_description),
    'error_definition': fields.Nested(error_definition),
})

error_429_model = api.model('error_429_model', {
    'message': fields.Nested(error_dicts),
})
```

5.1.58 services\backend_api\src\models\models_error\http_error_500.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# InternalServerError error model
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import general_error_namespace as api
from models.models_error.error_base_model import error_definition
from werkzeug.exceptions import InternalServerError

########################################################################################
##############################
# Nested response model for the InternalServerError errors
########################################################################################
##############################

error_description = api.model('error_description_500_model', {
    'code': fields.String(
        description='HTTP error status code',
        example=InternalServerError.code,
        default=InternalServerError.code
    ),
    'status': fields.String(
        description='HTTP error status',
        example='INTERNAL_SERVER_ERROR',
        default='INTERNAL_SERVER_ERROR'
    ),
    'description': fields.String(
        description='Detailed HTTP error description',
        example=InternalServerError.description,
        default=InternalServerError.description
    )
})

error_dicts = api.model('error_dict_500_model', {
    'error_description': fields.Nested(error_description),
    'error_definition': fields.Nested(error_definition),
})

error_500_model = api.model('error_500_model', {
    'message': fields.Nested(error_dicts),
})
```

5.1.59 services\backend_api\src\models\models_error\http_error_501.py

```
###############################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# NotImplemented error model
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
###############################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import general_error_namespace as api
from models.models_error.error_base_model import error_definition
from werkzeug.exceptions import NotImplemented

###############################################################################################
##############################
# Nested response model for the NotImplemented errors
###############################################################################################
##############################

error_description = api.model('error_description_501_model', {
    'code': fields.String(
        description='HTTP error status code',
        example=NotImplemented.code,
        default=NotImplemented.code
    ),
    'status': fields.String(
        description='HTTP error status',
        example='NOT_IMPLEMENTED',
        default='NOT_IMPLEMENTED'
    ),
    'description': fields.String(
        description='Detailed HTTP error description',
        example=NotImplemented.description,
        default=NotImplemented.description
    )
})

error_dicts = api.model('error_dict_501_model', {
    'error_description': fields.Nested(error_description),
    'error_definition': fields.Nested(error_definition),
})

error_501_model = api.model('error_501_model', {
    'message': fields.Nested(error_dicts),
})
```

5.1.60 services\backend_api\src\models\models_error\http_error_503.py

```
###############################################################################################
##############################
```

```
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# ServiceUnavailable error model
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import general_error_namespace as api
from models.models_error.error_base_model import error_definition
from werkzeug.exceptions import ServiceUnavailable

##########################################################################################
##############################
# Nested response model for the ServiceUnavailable errors
##########################################################################################
##############################

error_description = api.model('error_description_503_model', {
    'code': fields.String(
        description='HTTP error status code',
        example=ServiceUnavailable.code,
        default=ServiceUnavailable.code
    ),
    'status': fields.String(
        description='HTTP error status',
        example='SERVICE_UNAVAILABLE',
        default='SERVICE_UNAVAILABLE'
    ),
    'description': fields.String(
        description='Detailed HTTP error description',
        example=ServiceUnavailable.description,
        default=ServiceUnavailable.description
    )
})

error_dicts = api.model('error_dict_503_model', {
    'error_description': fields.Nested(error_description),
    'error_definition': fields.Nested(error_definition),
})

error_503_model = api.model('error_503_model', {
    'message': fields.Nested(error_dicts),
})
```

5.1.61 services\backend_api\src\models\models_logging\logging_models.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
```

```
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Models for the logger service
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import logging_namespace as api

########################################################################################
##############################
# Request model for the POST request
########################################################################################
##############################

logging_request_model = api.model('logging_request_model',
                                  {
                                      'service_module_name': fields.String(
                                          description='Name of the service and module
which triggers the logger call',
                                          example='service_module',
                                          required=True
                                      ),
                                      'log_message': fields.String(
                                          description='Detailed log description',
                                          example='This is a log message',
                                          required=True
                                      )
                                  })
```

5.1.62 services\backend_api\src\models\models_products\products_models.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Get Products model for the Swagger UI
#
# Date created: 23.07.2021
# Date last modified: 23.07.2021
#
# __author__  = Johannes Schmid (schmid@geoville.com)
# __version__ = 21.07
#
########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import service_namespace as api
```

```
##########################################################################################
##############################
# Request model for the POST request get_product
##########################################################################################
##############################

products_request_model = api.model('products_request_model',
                                    {
                                        'product': fields.String(
                                            description='Name of the CLC+ Backbone
product',
                                            example='Raster',
                                            required=True
                                        ),
                                        'aoi': fields.String(
                                            description='Area of Interest as MultiPolygon
(WKT) in WGS84 (EPSG:4326)',
                                            example='MULTIPOLYGON (((17.227309445590443
46.668932137424534, 17.2371487027879 47.10769387054001, 18.217384701084143
47.095973145761874, 18.19278655809051 46.65458260506014, 17.227309445590443
46.668932137424534)))',
                                            required=True
                                        ),
                                        'user_id': fields.String(
                                            description='ID of the current customer',
                                            example='S6aIHB1NOSbaj1ghq99pXq9a',
                                            required=True
                                        )
                                    })


##########################################################################################
##############################
# Request model for the POST request get_national_product
##########################################################################################
##############################

national_products_request_model = api.model('national_products_request_model',
                                    {
                                        'product': fields.String(
                                            description='Name of the CLC+ Backbone
product',
                                            example='Raster',
                                            required=True
                                        ),
                                        'nation': fields.String(
                                            description='Country name in English (e.g.
Austria)',
                                            example='Austria',
                                            required=True
                                        ),
                                        'user_id': fields.String(
                                            description='ID of the current customer',
                                            example='S6aIHB1NOSbaj1ghq99pXq9a',
                                            required=True
                                        )
                                    })


##########################################################################################
##############################
# Request model for the POST request get_product_europe
##########################################################################################
##############################

european_products_request_model = api.model('european_products_request_model',
                                    {
                                        'product': fields.String(
                                            description='Name of the CLC+ Backbone
product',
                                            example='Raster',
                                            required=True
                                        ),
                                        'user_id': fields.String(
                                            description='ID of the current customer',
                                            example='S6aIHB1NOSbaj1ghq99pXq9a',
                                            required=True
                                        )
                                    })


###########################################################################################
##############################
# Request model for the POST request get_national_product
###########################################################################################
##############################

nations_request_model = api.model('nations_request_model',
                                    {
                                        'user_id': fields.String(
                                            description='ID of the current customer',
                                            example='S6aIHB1NOSbaj1ghq99pXq9a',
                                            required=True
                                        )
                                    })

###########################################################################################
##############################
# Response model for getting national and europe products
###########################################################################################
##############################

products_success_response_model = api.model('products_success_response_model',
                                           {
                                               'result': fields.String(
                                                   description='Download Link of the
requested product',
                                                   example='https://s3.waw2-
1.cloudferro.com/swift/v1/AUTH_abc/'
                                                           'clcplus-public/products/'

'CLMS_CLCplus_RASTER_2018_010m_eu_03035_V1_1.tif'
                                               )
                                           })

###########################################################################################
##############################
# Response model for getting the nations
###########################################################################################
##############################

nations_success_response_model = api.model('nations_success_response_model',
                                           {
                                               'nations': fields.List(
                                                   fields.String,
                                                   description='List of nation names that
can be used as input for g'
                                                               'et_national_product',
                                                   example=["Austria", "Germany", "etc."]
                                               )
                                           })
```

5.1.63 services\backend_api\src\models\models_rabbitmq\list_queues\list_queues_model.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# RabbitMQ list queues model for the Swagger UI
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import rabbitmq_namespace as api

########################################################################################
##############################
# Response model for the POST and GET request
########################################################################################
##############################

rabbitmq_response_model = api.model('rabbitmq_list_queues_response_model',
                                    {
                                        'queues': fields.List(
                                            fields.String,
                                            description='List of all available RabbitMQ
queues on this host',
                                            example=['queue_1', 'queue_2', 'queue_3']
                                        )
                                    })
```

5.1.64 services\backend_api\src\models\models_rabbitmq\list_users\list_users_model.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# RabbitMQ list users model for the Swagger UI
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import rabbitmq_namespace as api

##########################################################################################
##############################
# Response model for the POST and GET request
##########################################################################################
##############################

users_object_model = api.model('rabbitmq_users_object',
                               {
                                   'name': fields.String(
                                       description='RabbitMQ user name',
                                       example='user_name'
                                   ),
                                   'permission': fields.String(
                                       description='Administration rights of the user',
                                       example='administrator'
                                   )
                               })

##########################################################################################
##############################
# Response model for the GET request
##########################################################################################
##############################

users_response_model = api.model('rabbitmq_users_response_model',
                                 {
                                     'users':fields.List(
                                         fields.Nested(
                                             users_object_model
                                         )
                                     )
                                 })
```

5.1.65 services\backend_api\src\models\models_rabbitmq\list_vhosts\list_vhosts_model.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# RabbitMQ list virtual hosts model for the Swagger UI
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import rabbitmq_namespace as api

##########################################################################################
##############################
# Response model for the POST and GET request
##########################################################################################
##############################
```

```
virtual_host_response_model = api.model('rabbitmq_virtual_hosts_response_model',
                                         {
                                             'virtual_hosts': fields.List(
                                                 fields.String,
                                                 description='List of all available
RabbitMQ queues on this host',
                                                 example=['virtual_host_1',
'virtual_host_2',
                                                          'virtual_host_3']
                                             )
                                         })

###########################################################################################
##############################
# Request model for the DELETE request
###########################################################################################
##############################

virtual_host_request_model = api.model('rabbitmq_queue_vhost_request_model',
                                    {
                                        'virtual_host': fields.String(
                                            description='RabbitMQ host address',
                                            example='/',
                                            required=True
                                        ),
                                        'queue_name': fields.String(
                                            description='RabbitMQ queue name',
                                            example='service_seasonality',
                                            required=True
                                        )
                                    })
```

5.1.66 services\backend_api\src\models\models_rabbitmq\message_count\message_count_model.py

```
###########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# RabbitMQ message count model for the Swagger UI
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
###########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import rabbitmq_namespace as api

###########################################################################################
##############################
# Response model for the POST request
###########################################################################################
##############################
```

```
message_count_model = api.model('rabbitmq_message_count_response_model',
                                {
                                    'virtual_host': fields.String(
                                        description='RabbitMQ virtual host address',
                                        example='/'
                                    ),
                                    'queue_name': fields.String(
                                        description='RabbitMQ queue name',
                                        example='queue_name'
                                    ),
                                    'message_count': fields.Integer(
                                        description='Number of messages in the specified
queue',
                                        example=5
                                    )
                                })
```

5.1.67 services\backend_api\src\models\models_rabbitmq\purge_queue\purge_queue_model.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, is prohibited for all commercial applications without
# licensing by GeoVille GmbH.
#
# RabbitMQ purge queue model for the API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import rabbitmq_namespace as api

##########################################################################################
##############################
# Response model for the DELETE request
##########################################################################################
##############################

purge_queue_response = api.model('rabbitmq_purge_queue_response_model',
                                 {
                                     'success': fields.String(
                                         description='Success message',
                                         example='Queue deleted'
                                     )
                                 })
```

5.1.68 services\backend_api\src\models\models_rabbitmq\server_status\server_status_model.py

```
############################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# RabbitMQ server status model for the Swagger UI
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
############################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import rabbitmq_namespace as api

############################################################################################
##############################
# Response model for the POST and GET request
############################################################################################
##############################

rabbitmq_response_model = api.model('rabbitmq_server_status_response_model',
                                    {
                                        'host': fields.String(
                                            description='RabbitMQ host address',
                                            example='0.0.0.0'
                                        ),
                                        'server_status': fields.String(
                                            description='The server status',
                                            example='up and running'
                                        )
                                   })
```

5.1.69 services\backend_api\src\models\models_rois\roi_models.py

```
############################################################################################
##############################
#
# Copyright (c) 2020, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Region of interest models for the Swagger UI
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
```

```
##########################################################################################
##############################

from datetime import datetime
from flask_restx import fields
from init.namespace_constructor import rois_namespace as api

##########################################################################################
##############################
# Request model for the POST request
##########################################################################################
##############################

roi_id_model = api.model('roi_id_model',
                         {
                             'roi_id': fields.String(
                                 description='Region of interest identifier',

example='aedadc0b181bca2166896cb737e99743de5a6df762e3c5d07b17b9dd6db86a1c',
                                 required=True
                             )
                         })

##########################################################################################
##############################
# Model for the geoJSON representation
##########################################################################################
##############################

geoJSON_model = api.model('geoJSON_model',
                          {
                              'type': fields.String(
                                  description='Type of geoJSON object',
                                  example='MultiPolygon',
                                  default='MultiPolygon',
                                  required=True
                              ),
                              'coordinates':
fields.List(fields.List(fields.List(fields.List(fields.Float(
                                  description='One coordinate pair of XY',
                                  required=True
                              )))))
                          })

##########################################################################################
##############################
# Request model for the ROI creation request
##########################################################################################
##############################

roi_request_model = api.model('roi_request_model',
                              {
                                  'name': fields.String(
                                      description='A name to identify the ROI',
                                      example='957653b708d9715d631',
                                      required=True
                                  ),
                                  'description': fields.String(
                                      description='Descriptive text about the ROI',
                                      example='An example region of interest'
                                  ),
                                  'user_id': fields.String(
                                      description='Unique identifier of a customer',
                                      example='8KfYSDj8Wq2iNtIly98M5ES4',
                                      required=True
                                  ),
                                  'geoJSON': fields.Nested(
                                      geoJSON_model,
                                          description='GeoJSON representation of the ROI',
                                          example={"type": "MultiPolygon",
                                                   "coordinates": [[[
                                                       [11.239013671875, 47.212105775622426],
                                                       [11.506805419921875,
47.212105775622426],
                                                       [11.506805419921875,
47.307171912070814],
                                                       [11.239013671875, 47.307171912070814],
                                                       [11.239013671875, 47.212105775622426]
                                                   ]]]},
                                          required=True)
                                  })

##########################################################################################
##############################
# ROI response model for a single ROI
##########################################################################################
##############################

single_roi_response_model = api.model('single_roi_response_model',
                                      {
                                          'name': fields.String(
                                              description='A name to identify the ROI',
                                              example='Region of Interest'
                                          ),
                                          'description': fields.String(
                                              description='Descriptive text about the
ROI',
                                              example='An example region of interest'
                                          ),
                                          'customer_id': fields.String(
                                              description='Unique identifier of a
customer',
                                              example='8KfYSDj8Wq2iNtIly98M5ES4'
                                          ),
                                          'geoJSON': fields.Nested(
                                              geoJSON_model,
                                              description='GeoJSON representation of the
ROI',
                                              example={"type": "MultiPolygon",
                                                       "coordinates": [[[
                                                           [11.239013671875,
47.212105775622426],
                                                           [11.506805419921875,
47.212105775622426],
                                                           [11.506805419921875,
47.307171912070814],
                                                           [11.239013671875,
47.307171912070814],
                                                           [11.239013671875,
47.212105775622426]
                                                       ]]]}
                                          ),
                                          'creation_date': fields.String(
                                              description='Date of creation of the ROI',
                                              example=datetime.now().strftime("%Y-%m-
%dT%H:%M:%S")
                                          ),
                                      })

##########################################################################################
##############################
# ROI response model for several ROI's
##########################################################################################
##############################

several_roi_response_model = api.model('roi_response_model',
                                       {
                                             'rois': fields.List(
                                                 fields.Nested(
                                                     single_roi_response_model,
                                                     description="List of ROI's",
                                                 )
                                             )
                                         })

##########################################################################################
##############################
# Model for the ROI update request
##########################################################################################
##############################

roi_attributes_request = api.model('roi_attributes_request',
                                   {
                                       'roi_id': fields.String(
                                           description='Unique identifier of a ROI',

example='aedadc0b181bca2166896cb737e99743de5a6df762e3c5d07b17b9dd6db86a1c',
                                           required=True
                                       ),
                                       'name': fields.String(
                                           description='A name to identify the ROI',
                                           example='Name identifier of a Region of
Interest'
                                       ),
                                       'description': fields.String(
                                           description='Descriptive ',
                                           example='An example region of interest'
                                       ),
                                       'customer_id': fields.String(
                                           description='Unique identifier of a customer',
                                           example='8KfYSDj8Wq2iNtIly98M5ES4'
                                       ),
                                       'geoJSON': fields.Nested(
                                           geoJSON_model,
                                           description='GeoJSON of the region of
interest',

                                           example={"type": "MultiPolygon",
                                                    "coordinates": [[[
                                                        [11.239013671875,
47.212105775622426],

                                                        [11.506805419921875,
47.212105775622426],

                                                        [11.506805419921875,
47.307171912070814],

                                                        [11.239013671875,
47.307171912070814],

                                                        [11.239013671875,
47.212105775622426]

                                                    ]]]}
                                       ),

                                   })

##########################################################################################
##############################
# Model for the ROI update request
##########################################################################################
##############################

roi_entity_request = api.model('roi_entity_request',
                               {
                                   'roi_id': fields.String(
                                       description='Unique identifier of a ROI',

example='aedadc0b181bca2166896cb737e99743de5a6df762e3c5d07b17b9dd6db86a1c',
                                       required=True
```

```
                                    ),
                                    'name': fields.String(
                                        description='Name to identify the ROI',
                                        example='Name of a Region of Interest',
                                        required=True
                                    ),
                                    'description': fields.String(
                                        description='Descriptive ',
                                        example='Example region of interest'
                                    ),
                                    'customer_id': fields.String(
                                        description='Unique identifier of a customer',
                                        example='8KfYSDj8Wq2iNtIly98M5ES4',
                                        required=True
                                    ),
                                    'geoJSON': fields.Nested(
                                        geoJSON_model,
                                        description='GeoJSON of the region of interest',
                                        example={"type": "MultiPolygon",
                                                 "coordinates": [[[
                                                     [11.239013671875, 47.212105775622426],
                                                     [11.506805419921875,
47.212105775622426],
                                                     [11.506805419921875,
47.307171912070814],
                                                     [11.239013671875, 47.307171912070814],
                                                     [11.239013671875, 47.212105775622426]
                                                 ]]]},
                                        required=True
                                    ),

                                })
```

5.1.70 services\backend_api\src\models\models_services\batch_classification\batch_classification_model.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Service success response models for the Swagger UI
#
# Date created: 01.06.2020
# Date last modified: 15.04.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.04
#
##########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import service_namespace as api

##########################################################################################
##############################
# Hypermedia service model
##########################################################################################
##############################
```

```
parameter_model = api.model('parameter_model',
                            {

                            })

##########################################################################################
##############################
# Response model for the POST service request
##########################################################################################
##############################

batch_classification_model = api.model('batch_classification_model',
                                       {
                                           'params': fields.Nested(
                                               parameter_model,
                                               required=True
                                           ),
                                           'service_name': fields.String(
                                               description='Name of the service to be
called',

                                               example='batch_classification_test',
                                               pattern='(batch_classification_test)',
                                               required=True
                                           )
                                       })

##########################################################################################
##############################
# Response model for the POST request
##########################################################################################
##############################

batch_classification_production_model = api.model('batch_classification_production_model',
                                       {
                                           'params': fields.Nested(
                                               parameter_model,
                                               required=True
                                           ),
                                           'service_name': fields.String(
                                               description='Name of the service to be
called',

                                               example='batch_classification_production',

pattern='(batch_classification_production)',
                                               required=True
                                           )
                                       })

##########################################################################################
##############################
# Response model for the POST request of the staging request
##########################################################################################
##############################

batch_classification_staging_model = api.model('batch_classification_staging_model',
                                       {
                                           'params': fields.Nested(
                                               parameter_model,
                                               required=True
                                           ),
                                           'service_name': fields.String(
                                               description='Name of the service to be
called',

                                               example='batch_classification_staging',
                                               pattern='(batch_classification_staging)',
                                               required=True
                                           )
```

```
                                            })
```

5.1.71 services\backend_api\src\models\models_services\harmonics\harmonics_model.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Harmonics model for the Swagger UI
#
# Date created: 01.06.2020
# Date last modified: 15.04.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.04
#
##########################################################################################
##############################

from datetime import date
from flask_restx import fields
from init.namespace_constructor import service_namespace as api

##########################################################################################
##############################
# Request model for the POST request
##########################################################################################
##############################

harmonics_request_model = api.model('harmonics_request_model',
                                    {
                                        'tile_id': fields.String(
                                            description='Sentinel-2 tile ID',
                                            example='33UXP',
                                            required=True
                                        ),
                                        'start_date': fields.Date(
                                            description='Start date to be requested',
                                            example=date.today().strftime('%Y-%m-%d'),
                                            required=True
                                        ),
                                        'end_date': fields.Date(
                                            description='End date to be requested',
                                            example=date.today().strftime('%Y-%m-%d'),
                                            required=True
                                        ),
                                        'band': fields.String(
                                            description='Processing level parameter',
                                            example='RED',
                                            required=True
                                        ),
                                        'resolution': fields.Integer(
                                            description='Resolution in meters',
                                            example=10,
                                            required=True
                                        ),
                                        'ndi_band': fields.String(
                                            description='Processing level parameter',
                                            example="None",
                                            required=True
                                        ),
                                        'user_id': fields.String(
                                            description='ID of the current customer',
                                            example='S6aIHB1NOSbaj1ghq99pXq9a',
                                            required=True
                                        ),
                                        'service_name': fields.String(
                                            description='Name of the service to be
called',
                                            example='harmonics',
                                            pattern='(harmonics)',
                                            required=True
                                        )
                                    })
```

5.1.72 services\backend_api\src\models\models_services\retransformation\retransformation_model.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Retransformation model for the Swagger UI
#
# Date created: 07.07.2021
# Date last modified: 07.07.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.07
#
##########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import service_namespace as api

##########################################################################################
##############################
# Retransformation request model
##########################################################################################
##############################

retransformation_request_model = api.model('retransformation_request_model',
                                           {
                                               'subproduction_unit_name ': fields.String(
                                                   example='1_1',
                                                   required=True
                                               ),
                                               'user_id': fields.String(
                                                   description='ID of the current user',
                                                   example='S6aIHB1NOSbaj1ghq99pXq9a',
                                                   required=True
                                               ),
                                               'service_name': fields.String(
                                                   description='Name of the service to be
called',
                                                   example='retransformation',
                                                   pattern='(retransformation)',
                                                   required=True
                                               )
                                           })
```

5.1.73 services\backend_api\src\models\models_services\service_order_status\order_status_model.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Service order status model for the Swagger UI
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import service_namespace as api

##########################################################################################
##############################
# Response model for the order status POST request
##########################################################################################
##############################

order_status_response_model = api.model('order_status_response_model',
                                        {
                                            'order_id': fields.String(
                                                description='ID of the created order',
                                                example='391d3b45f059f9fb74b79868f6e8511e'
                                            ),
                                            'status': fields.String(
                                                description='Status message',
                                                example='SUCCESS'
                                            ),
                                            'result': fields.String(
                                                description='Link to the result file',
                                                example='https://gems-
demo.s3.amazonaws.com'
                                            )
                                        })
```

5.1.74 services\backend_api\src\models\models_services\task_1_batch_classification\task_1_batch_classification_model.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
```

```
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Task 1 batch classification model for the Swagger UI
#
# Date created: 15.04.2021
# Date last modified: 15.04.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.04
#
##########################################################################################
##############################

from datetime import date
from flask_restx import fields
from init.namespace_constructor import service_namespace as api

##########################################################################################
##############################
# Additional data filter model
##########################################################################################
##############################

t1_batch_classification_data_filter_model =
api.model('t1_batch_classification_data_filter_model',
                                                     {
                                                         'min_time_difference':
fields.Integer(
                                                             example=10,
                                                             required=False
                                                         ),
                                                         'max_cc_local': fields.Integer(
                                                             description='Start date to
be requested',
                                                             minimum=0,
                                                             maximum=100,
                                                             default=0,
                                                             example=20,
                                                             required=False
                                                         )
                                                     })

##########################################################################################
##############################
# Request model for the POST request
##########################################################################################
##############################

t1_batch_classification_request_model = api.model('t1_batch_classification_request_model',
                                                 {
                                                     'start_date': fields.Date(
                                                         description='Oldest acquisition
date to consider',
                                                         example='2017-07-01',
                                                         default='2017-07-01',
                                                         required=True
                                                     ),
                                                     'end_date': fields.Date(
                                                         description='Newest acquisition
date to consider',
                                                         example='2019-06-30',
                                                         default='2019-06-30',
                                                         required=True
                                                     ),
                                                     'processing_unit_name':
fields.String(
                                                          description='Name of the
processing unit',
                                                          example='10kmE0N70',
                                                          required=True
                                                      ),
                                                      'cloud_cover': fields.Integer(
                                                          description='Maximum cloud cover
(in %) to consider',
                                                          example=80,
                                                          min=0,
                                                          max=100,
                                                          default=80,
                                                          required=True
                                                      ),
                                                      'features':
fields.List(fields.String(
                                                          description='Names of the
required auxiliary features',
                                                          example='B01_MIN',
                                                          required=True
                                                      )),
                                                      'use_cache': fields.Boolean(
                                                          description='Use cache results',
                                                          example=True,
                                                          required=True
                                                      ),
                                                      'data_filter': fields.Nested(

t1_batch_classification_data_filter_model,
                                                          description='Data filter',
                                                          required=True
                                                      ),
                                                      'rule_set':
fields.List(fields.String(
                                                          description='Apply rule set',
                                                          example="rule_1",
                                                          required=True
                                                      )),
                                                      'user_id': fields.String(
                                                          description='ID of the current
customer',

example='S6aIHB1NOSbaj1ghq99pXq9a',
                                                          required=True
                                                      ),
                                                      'service_name': fields.String(
                                                          description='Name of the service
to be called',

example='task1_batch_classification',

pattern='(task1_batch_classification)',
                                                          required=True
                                                      )
                                                  })
```

5.1.75 services\backend_api\src\models\models_services\task_1_feature_calculation\task_1_feature_calculation_model.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
```

```
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Task 1 feature calculation model for the Swagger UI
#
# Date created: 15.04.2021
# Date last modified: 15.04.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.04
#
##########################################################################################
##############################

from datetime import date
from flask_restx import fields
from init.namespace_constructor import service_namespace as api

##########################################################################################
##############################
# Additional data filter model
##########################################################################################
##############################

t1_feature_calculation_data_filter_model =
api.model('t1_feature_calculation_data_filter_model',
                                                    {
                                                        'min_time_difference':
fields.Integer(
                                                            example=10,
                                                            required=False
                                                        ),
                                                        'max_cc_local': fields.Integer(
                                                            description='Start date to be
requested',
                                                            minimum=0,
                                                            maximum=100,
                                                            default=0,
                                                            example=20,
                                                            required=False
                                                        )
                                                    })

##########################################################################################
##############################
# Request model for the POST request
##########################################################################################
##############################

t1_feature_calculation_request_model = api.model('t1_feature_calculation_request_model',
                                                {
                                                    'start_date': fields.Date(
                                                        description='Oldest acquisition
date to consider',
                                                        example='2017-07-01',
                                                        default='2017-07-01',
                                                        required=True
                                                    ),
                                                    'end_date': fields.Date(
                                                        description='Newest acquisition
date to consider',
                                                        example='2019-06-30',
                                                        default='2019-06-30',
                                                        required=True
                                                    ),
                                                    'processing_unit_name':
fields.String(
                                                            description='Name of the
processing unit',
                                                            example='10kmE0N70',
                                                            required=True
                                                        ),
                                                        'cloud_cover': fields.Integer(
                                                            description='Maximum cloud cover
(in %) to consider',
                                                            example=80,
                                                            min=0,
                                                            max=100,
                                                            default=80,
                                                            required=True
                                                        ),
                                                        'features':
fields.List(fields.String(
                                                            description='Names of the
required auxiliary features',
                                                            example='B01_MIN',
                                                            required=True
                                                        )),
                                                        'use_cache': fields.Boolean(
                                                            description='Use cache results',
                                                            example=True,
                                                            required=True
                                                        ),
                                                        'data_filter': fields.Nested(
t1_feature_calculation_data_filter_model,
                                                            description='Data filter',
                                                            required=True
                                                        ),
                                                        'user_id': fields.String(
                                                            description='ID of the current
customer',
example='S6aIHB1NOSbaj1ghq99pXq9a',
                                                            required=True
                                                        ),
                                                        'service_name': fields.String(
                                                            description='Name of the service
to be called',
example='task1_feature_calculation',
pattern='(task1_feature_calculation)',
                                                            required=True
                                                        )
                                                    })
```

5.1.76 services\backend_api\src\models\models_services\task_1_reprocessing\task_1_reprocessing_model.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Task 1 reprocessing model for the Swagger UI
#
# Date created: 15.04.2021
# Date last modified: 15.04.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.04
#
#########################################################################################
##############################

from datetime import date
from flask_restx import fields
from init.namespace_constructor import service_namespace as api

#########################################################################################
##############################
# Additional data filter model
#########################################################################################
##############################

task_1_reprocessing_request_data_filter_model =
api.model('t1_reprocessing_request_data_filter_model',
                                                      {
                                                          'min_time_difference':
fields.Integer(
                                                              example=10,
                                                              required=False
                                                          ),
                                                          'max_cc_local': fields.Integer(
                                                              description='Start date to
be requested',
                                                              minimum=0,
                                                              maximum=100,
                                                              default=0,
                                                              example=20,
                                                              required=False
                                                          )
                                                      })

#########################################################################################
##############################
# Request model for the POST request
#########################################################################################
##############################

task_1_reprocessing_request_model = api.model('t1_reprocessing_request_model',
                                                  {
                                                      'start_date': fields.Date(
                                                          description='Oldest acquisition
date to consider',
                                                          example='2017-07-01',
                                                          default='2017-07-01',
                                                          required=True
                                                      ),
                                                      'end_date': fields.Date(
                                                          description='Newest acquisition
date to consider',
                                                          example='2019-06-30',
                                                          default='2019-06-30',
                                                          required=True
                                                      ),
                                                      'processing_unit_name':
fields.String(
                                                          description='Name of the
processing unit',
                                                          example='10kmE0N70',
                                                          required=True
                                                      ),
                                                      'cloud_cover': fields.Integer(
                                                          description='Maximum cloud cover
(in %) to consider',
                                                          example=80,
                                                          min=0,
                                                          max=100,
                                                          default=80,
                                                          required=True
                                                      ),
                                                      'features':
fields.List(fields.String(
                                                          description='Names of the
required auxiliary features',
                                                          example='B01_MIN',
                                                          required=True
                                                      )),
                                                      'use_cache': fields.Boolean(
                                                          description='Use cache results',
                                                          example=True,
                                                          required=True
                                                      ),
                                                      'data_filter': fields.Nested(
task_1_reprocessing_request_data_filter_model,
                                                          description='Data filter',
                                                          required=True
                                                      ),
                                                      'rule_set':
fields.List(fields.String(
                                                          description='Apply rule set',
                                                          example="rule_1",
                                                          required=True
                                                      )),
                                                      'user_id': fields.String(
                                                          description='ID of the current
customer',
example='S6aIHB1NOSbaj1ghq99pXq9a',
                                                          required=True
                                                      ),
                                                      'service_name': fields.String(
                                                          description='Name of the service
to be called',
                                                          example='task1_reprocessing',
                                                          pattern='(task1_reprocessing)',
                                                          required=True
                                                      )
                                                  })
```

5.1.77 services\backend_api\src\models\models_services\task_1_reprocessing_test\task_1_reprocessing_test_model.py

```
###########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Task 1 reprocessing model for the Swagger UI
#
# Date created: 15.04.2021
# Date last modified: 15.04.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.04
#
###########################################################################################
##############################

from datetime import date
from flask_restx import fields
from init.namespace_constructor import service_namespace as api

###########################################################################################
##############################
# Additional data filter model
###########################################################################################
##############################

task_1_reprocessing_test_request_data_filter_model =
api.model('t1_reprocessing_test_request_data_filter_model',
                                                       {
                                                           'min_time_difference':
fields.Integer(
                                                               example=10,
                                                               required=False
                                                           ),
                                                           'max_cc_local': fields.Integer(
                                                               description='Start date to
be requested',
                                                               minimum=0,
                                                               maximum=100,
                                                               default=0,
                                                               example=20,
                                                               required=False
                                                           )
                                                       })

###########################################################################################
##############################
# Request model for the POST request
###########################################################################################
##############################

task_1_reprocessing_test_request_model = api.model('t1_reprocessing_test_request_model',
                                                   {
                                                       'start_date': fields.Date(
                                                           description='Oldest acquisition
date to consider',
                                                           example='2017-07-01',
                                                           default='2017-07-01',
                                                           required=True
                                                       ),
                                                       'end_date': fields.Date(
                                                           description='Newest acquisition
date to consider',
                                                           example='2019-06-30',
                                                           default='2019-06-30',
                                                           required=True
                                                       ),
                                                       'processing_unit_name':
fields.String(
                                                           description='Name of the
processing unit',
                                                           example='10kmE0N70',
                                                           required=True
                                                       ),
                                                       'cloud_cover': fields.Integer(
                                                           description='Maximum cloud cover
(in %) to consider',
                                                           example=80,
                                                           min=0,
                                                           max=100,
                                                           default=80,
                                                           required=True
                                                       ),
                                                       'features':
fields.List(fields.String(
                                                           description='Names of the
required auxiliary features',
                                                           example='B01_MIN',
                                                           required=True
                                                       )),
                                                       'use_cache': fields.Boolean(
                                                           description='Use cache results',
                                                           example=True,
                                                           required=True
                                                       ),
                                                       'data_filter': fields.Nested(
task_1_reprocessing_test_request_data_filter_model,
                                                           description='Data filter',
                                                           required=True
                                                       ),
                                                       'rule_set':
fields.List(fields.String(
                                                           description='Apply rule set',
                                                           example="rule_1",
                                                           required=True
                                                       )),
                                                       'user_id': fields.String(
                                                           description='ID of the current
customer',
example='S6aIHB1NOSbaj1ghq99pXq9a',
                                                           required=True
                                                       ),
                                                       'service_name': fields.String(
                                                           description='Name of the service
to be called',
example='task1_reprocessing_test',
pattern='(task1_reprocessing_test)',
                                                           required=True
                                                       )
                                                   })
```

5.1.78 services\backend_api\src\models\models_services\task_1_stitching\task_1_stitching_model.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Task 1 reprocessing model for the Swagger UI
#
# Date created: 02.08.2021
# Date last modified: 02.08.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.08
#
```

```
########################################################################################
##############################

from datetime import date
from flask_restx import fields
from init.namespace_constructor import service_namespace as api

########################################################################################
##############################
# Additional data filter model
########################################################################################
##############################

task_1_stitching_request_model = api.model('task_1_stitching_request_model',
                                           {
                                               'processing_unit_name': fields.String(
                                                   description='Input PU',
                                                   example='129_1',
                                                   required=True
                                               ),
                                               'surrounding_pus': fields.String(
                                                   description='List of surrounding PUs',
                                                   example='129_2, 174, 175',
                                                   required=True
                                               ),

                                               'user_id': fields.String(
                                                   description='ID of the current
customer',
                                                   example='S6aIHB1NOSbaj1ghq99pXq9a',
                                                   required=True
                                               ),
                                               'service_name': fields.String(
                                                   description='Name of the service to be
called',
                                                   example='task1_stitching',
                                                   pattern='(task1_stitching)',
                                                   required=True
                                               )
                                           })
```

5.1.79 services\backend_api\src\models\models_services\task_2_apply_model\task_2_apply_model_model.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Apply model for the Swagger UI
#
# Date created: 01.06.2020
# Date last modified: 15.04.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.04
#
########################################################################################
##############################
```

```
from flask_restx import fields
from init.namespace_constructor import service_namespace as api

#########################################################################################
##############################
# Additional data filter model
#########################################################################################
##############################

t2_apply_model_data_filter_model = api.model('t2_apply_model_data_filter_model',
                                             {
                                                 'min_time_difference': fields.Integer(
                                                     example=10,
                                                     required=False
                                                 ),
                                                 'max_cc_local': fields.Integer(
                                                     description='Start date to be
requested',
                                                     minimum=0,
                                                     maximum=100,
                                                     default=0,
                                                     example=20,
                                                     required=False
                                                 )
                                             })

#########################################################################################
##############################
# Request model for the POST request
#########################################################################################
##############################

t2_apply_model_request_model = api.model('t2_apply_model_request_model',
                                         {
                                             'model_path': fields.String(
                                                 description='parameter for saving data',
                                                 example='path/to/my/model.pkl',
                                                 required=True
                                             ),
                                             'start_date': fields.Date(
                                                 description='Oldest acquisition date to
consider',
                                                 example='2017-07-01',
                                                 default='2017-07-01',
                                                 required=True
                                             ),
                                             'end_date': fields.Date(
                                                 description='Newest acquisition date to
consider',
                                                 example='2019-06-30',
                                                 default='2019-06-30',
                                                 required=True
                                             ),
                                             'processing_unit_name': fields.String(
                                                 description='Name of the processing
unit',
                                                 example='10kmE0N70',
                                                 required=True
                                             ),
                                             'cloud_cover': fields.Integer(
                                                 description='Maximum cloud cover (in %)
to consider',
                                                 example=80,
                                                 min=0,
                                                 max=100,
                                                 default=80,
                                                 required=True
                                             ),
                                             'interval_size': fields.Integer(
                                                 description='Time difference in days for
temporal interpolation',
                                                 example=10,
                                                 default=10,
                                                 required=True
                                             ),
                                             's1_bands': fields.List(fields.String(
                                                 description='Names of the required
Sentinel-1 bands or indices',
                                                 example='ASC_DVVVH',
                                                 required=False
                                             )),
                                             's2_bands': fields.List(fields.String(
                                                 description='Names of the required
Sentinel-2 bands or indices',
                                                 example='B01',
                                                 required=False
                                             )),
                                             'precalculated_features':
fields.List(fields.String(
                                                 description='Names of the required
auxiliary features',
                                                 example='geomorpho90',
                                                 required=False
                                             )),
                                             'use_cache': fields.Boolean(
                                                 description='Use cache results',
                                                 example=True,
                                                 required=False
                                             ),
                                             'data_filter_s1': fields.Nested(
                                                 t2_apply_model_data_filter_model,
                                                 description='Use cache results',
                                                 required=False
                                             ),
                                             'data_filter_s2': fields.Nested(
                                                 t2_apply_model_data_filter_model,
                                                 description='Use cache results',
                                                 required=False
                                             ),
                                             'aoi_coverage': fields.Integer(
                                                 description='min coverage in percent for
one scene of aoi',
                                                 example=20,
                                                 default=0,
                                                 minimum=0,
                                                 maximum=100,
                                                 required=False
                                             ),
                                             'user_id': fields.String(
                                                 description='ID of the current customer',
                                                 example='S6aIHB1NOSbaj1ghq99pXq9a',
                                                 required=True
                                             ),
                                             'service_name': fields.String(
                                                 description='Name of the service to be
called',
                                                 example='task2_apply_model',
                                                 pattern='(task2_apply_model)',
                                                 required=True
                                             )
                                         })
```

5.1.80 services\backend_api\src\models\models_services\task_2_apply_model_fast_lane\task_2_apply_model_fast_lane_model.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Apply model for the Swagger UI
#
# Date created: 01.06.2020
# Date last modified: 15.04.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.04
#
##########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import service_namespace as api

##########################################################################################
##############################
# Additional data filter model
##########################################################################################
##############################

t2_apply_model_fast_lane_data_filter_model =
api.model('t2_apply_model_fast_lane_data_filter_model',
                                                       {
                                                           'min_time_difference':
fields.Integer(
                                                               example=10,
                                                               required=False
                                                           ),
                                                           'max_cc_local': fields.Integer(
                                                               description='Start date to
be requested',
                                                               minimum=0,
                                                               maximum=100,
                                                               default=0,
                                                               example=20,
                                                               required=False
                                                           )
                                                       })

##########################################################################################
##############################
# Request model for the POST request
##########################################################################################
##############################

t2_apply_model_fast_lane_request_model =
api.model('t2_apply_model_fast_lane_request_model',
                                                   {
                                                       'model_path': fields.String(
                                                           description='parameter for
saving data',
                                                           example='path/to/my/model.pkl',
                                                           required=True
                                                       ),
                                                       'start_date': fields.Date(
        description='Oldest acquisition
date to consider',
        example='2017-07-01',
        default='2017-07-01',
        required=True
    ),
    'end_date': fields.Date(
        description='Newest acquisition
date to consider',
        example='2019-06-30',
        default='2019-06-30',
        required=True
    ),
    'processing_unit_name':
fields.String(
        description='Name of the
processing unit',
        example='10kmE0N70',
        required=True
    ),
    'cloud_cover': fields.Integer(
        description='Maximum cloud
cover (in %) to consider',
        example=80,
        min=0,
        max=100,
        default=80,
        required=True
    ),
    'interval_size': fields.Integer(
        description='Time difference in
days for temporal interpolation',
        example=10,
        default=10,
        required=True
    ),
    's1_bands':
fields.List(fields.String(
        description='Names of the
required Sentinel-1 bands or indices',
        example='ASC_DVVVH',
        required=False
    )),
    's2_bands':
fields.List(fields.String(
        description='Names of the
required Sentinel-2 bands or indices',
        example='B01',
        required=False
    )),
    'precalculated_features':
fields.List(fields.String(
        description='Names of the
required auxiliary features',
        example='geomorpho90',
        required=False
    )),
    'use_cache': fields.Boolean(
        description='Use cache
results',
        example=True,
        required=False
    ),
    'data_filter_s1': fields.Nested(
t2_apply_model_fast_lane_data_filter_model,
        description='Use cache
results',
        required=False
```

```
                                                                                                    ),
                                                                                                    'data_filter_s2': fields.Nested(

t2_apply_model_fast_lane_data_filter_model,
                                                                                                        description='Use cache
results',
                                                                                                        required=False
                                                                                                    ),
                                                                                                    'aoi_coverage': fields.Integer(
                                                                                                        description='min coverage in
percent for one scene of aoi',
                                                                                                        example=20,
                                                                                                        default=0,
                                                                                                        minimum=0,
                                                                                                        maximum=100,
                                                                                                        required=False
                                                                                                    ),
                                                                                                    'user_id': fields.String(
                                                                                                        description='ID of the current
customer',

example='S6aIHB1NOSbaj1ghq99pXq9a',
                                                                                                        required=True
                                                                                                    ),
                                                                                                    'service_name': fields.String(
                                                                                                        description='Name of the
service to be called',

example='task2_apply_model_fast_lane',

pattern='(task2_apply_model_fast_lane)',
                                                                                                        required=True
                                                                                                    )
                                                                                                })
```

5.1.81 services\backend_api\src\models\models_services\task_2_feature_calculation\task_2_feature_calculation_model.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Feature calculation model for the Swagger UI
#
# Date created: 01.06.2020
# Date last modified: 15.04.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.04
#
##########################################################################################
##############################

from flask_restx import fields
from init.namespace_constructor import service_namespace as api

##########################################################################################
##############################
# Additional data filter model
```

```
##########################################################################################
##############################

t2_feature_calculation_data_filter_model =
api.model('t2_feature_calculation_data_filter_model',
                                                     {
                                                         'min_time_difference':
fields.Integer(
                                                             example=10,
                                                             required=False
                                                         ),
                                                         'max_cc_local': fields.Integer(
                                                             description='Start date to be
requested',
                                                             minimum=0,
                                                             maximum=100,
                                                             default=0,
                                                             example=20,
                                                             required=False
                                                         )
                                                     })

##########################################################################################
##############################
# Request model for the POST request
##########################################################################################
##############################

t2_feature_calculation_request_model = api.model('t2_feature_calculation_request_model',
                                                 {
                                                     'start_date': fields.Date(
                                                         description='Oldest acquisition
date to consider',
                                                         example='2017-07-01',
                                                         default='2017-07-01',
                                                         required=True
                                                     ),
                                                     'end_date': fields.Date(
                                                         description='Newest acquisition
date to consider',
                                                         example='2019-06-30',
                                                         default='2019-06-30',
                                                         required=True
                                                     ),
                                                     'processing_unit_name':
fields.String(
                                                         description='Name of the
processing unit',
                                                         example='10kmE0N70',
                                                         required=True
                                                     ),
                                                     'cloud_cover': fields.Integer(
                                                         description='Maximum cloud cover
(in %) to consider',
                                                         example=80,
                                                         min=0,
                                                         max=100,
                                                         default=80,
                                                         required=True
                                                     ),
                                                     'interval_size': fields.Integer(
                                                         description='Time difference in
days for temporal interpolation',
                                                         example=10,
                                                         default=10,
                                                         required=True
                                                     ),
                                                     's1_bands':
fields.List(fields.String(
                                                         description='Names of the
required Sentinel-1 bands or indices',

                                                         example='ASC_DVVVH',
                                                         required=False
                                                     )),
                                                     's2_bands':
fields.List(fields.String(
                                                         description='Names of the
required Sentinel-2 bands or indices',

                                                         example='B01',
                                                         required=False
                                                     )),
                                                     'precalculated_features':
fields.List(fields.String(
                                                         description='Names of the
required auxiliary features',

                                                         example='geomorpho90',
                                                         required=False
                                                     )),
                                                     'use_cache': fields.Boolean(
                                                         description='Use cache results',
                                                         example=True,
                                                         required=False
                                                     ),
                                                     'data_filter_s1': fields.Nested(

t2_feature_calculation_data_filter_model,

                                                         description='Use cache results',
                                                         required=False
                                                     ),
                                                     'data_filter_s2': fields.Nested(

t2_feature_calculation_data_filter_model,

                                                         description='Use cache results',
                                                         required=False
                                                     ),
                                                     'aoi_coverage': fields.Integer(
                                                         description='min coverage in
percent for one scene of aoi',

                                                         example=20,
                                                         default=0,
                                                         minimum=0,
                                                         maximum=100,
                                                         required=False
                                                     ),
                                                     'user_id': fields.String(
                                                         description='ID of the current
customer',

example='S6aIHB1NOSbaj1ghq99pXq9a',

                                                         required=True
                                                     ),
                                                     'service_name': fields.String(
                                                         description='Name of the service
to be called',

example='task2_feature_calculation',

                                                         required=True
                                                     )
                                                 })
```

5.1.82 services\backend_api\src\models\models_services\vector_class_attribution\vector_class_attribution_model.py

```
##########################################################################################
##############################
```

```
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Vector class attribution model for the Swagger UI
#
# Date created: 15.04.2021
# Date last modified: 15.04.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.04
#
##########################################################################################
##############################

from datetime import date
from flask_restx import fields
from init.namespace_constructor import service_namespace as api

##########################################################################################
##############################
# Request model for the POST request
##########################################################################################
##############################

vector_class_attribution_request_model =
api.model('vector_class_attribution_request_model',
                                                   {
                                                       'vector': fields.String(
                                                           description='Vector path',

example='/vsis3/task22/tests/in/test1.shp',

                                                           required=True
                                                       ),
                                                       'raster': fields.String(
                                                           description='List of raster
path',

example='/vsis3/task22/tests/in/test.tif',

                                                           required=True
                                                       ),
                                                       'subproduction_unit_name':
fields.Integer(
                                                           description='Subproduction Unit
Identifier',
                                                           example=123,
                                                           required=True
                                                       ),
                                                       'config': fields.String(
                                                           description='Config parameters
for GDAL',
                                                           example="AWS_SECRET_ACCESS_KEY
123abc "
                                                                   "AWS_S3_ENDPOINT
cf2.cloudferro.com:8080 "
                                                                   "AWS_VIRTUAL_HOSTING
FALSE "
                                                                   "AWS_ACCESS_KEY_ID
abc123",
                                                           required=True
                                                       ),
                                                       'method': fields.String(
                                                           description='Extraction
Method',
                                                           example="relative_count",
                                                           required=True
                                                       ),
                                                       'method_params': fields.String(
                                                           description='Extraction
Method',
                                                           example="1 7",
                                                           required=False
                                                       ),
                                                       'na_value': fields.Integer(
                                                           description='na value',
                                                           example=0,
                                                           required=False
                                                       ),
                                                       'col_names': fields.String(
                                                           description='List of column
names',
                                                           example="occurrence_class_1
occurrence_class_7",
                                                           required=False
                                                       ),
                                                       'id_column': fields.String(
                                                           description='Column name of
polygon id',
                                                           example="id",
                                                           required=True
                                                       ),
                                                       'bucket_path': fields.String(
                                                           description='Path to S3
bucket',

example="bucketname/folder/subfolder/",
                                                           required=True
                                                       ),
                                                       'reference_year': fields.Date(
                                                           description='Reference year to
be requested',
                                                           example='2018',
                                                           required=True
                                                       ),
                                                       'user_id': fields.String(
                                                           description='ID of the current
customer',

example='S6aIHB1NOSbaj1ghq99pXq9a',
                                                           required=True
                                                       ),
                                                       'service_name': fields.String(
                                                           description='Name of the
service to be called',

example='vector_class_attribution',

pattern='(vector_class_attribution)',
                                                           required=True
                                                       )
                                                   })
```

5.1.83 services\backend_api\src\oauth\oauth2.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
```

```
# applications without licensing by GeoVille GmbH.
#
# OAuth2 method collection for the GEMS API
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
#########################################################################################
##############################

from authlib.integrations.flask_oauth2 import AuthorizationServer
from authlib.integrations.sqla_oauth2 import (create_query_client_func,
create_save_token_func,
                                              create_revocation_endpoint,
create_bearer_token_validator)
from oauth.oauth_models import db, OAuth2Client, OAuth2Token
from oauth.resource_protector import ResourceProtector

#########################################################################################
##############################
# Setting up additional parameters
#########################################################################################
##############################

query_client = create_query_client_func(db.session, OAuth2Client)
save_token = create_save_token_func(db.session, OAuth2Token)
authorization = AuthorizationServer(query_client=query_client, save_token=save_token, )
require_oauth = ResourceProtector()


#########################################################################################
##############################
# Method for configuring the Flask app object
#########################################################################################
##############################

def config_oauth(app):
    """ Configures the FLASK app

    This method registers the OAuth2 instance on the FLASK app object. Thus, all OAuth2
functionalities can be accessed
    through the FLASk app.

    Arguments:
        app (obj): FLASK app object

    """

    # initialise app
    authorization.init_app(app)

    # support revocation
    revocation_cls = create_revocation_endpoint(db.session, OAuth2Token)
    authorization.register_endpoint(revocation_cls)

    # protect resource
    bearer_cls = create_bearer_token_validator(db.session, OAuth2Token)
    require_oauth.register_token_validator(bearer_cls())
```

5.1.84 services\backend_api\src\oauth\oauth_models.py

```
#########################################################################################
##############################
```

```
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# OAuth2 database model descriptions for SQLAlchemy
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from authlib.integrations.sqla_oauth2 import OAuth2ClientMixin, OAuth2TokenMixin
from flask_sqlalchemy import SQLAlchemy
from init.app_constructor import app
import time

########################################################################################
##############################
# Initialising the SQLAlchemy object
########################################################################################
##############################

db = SQLAlchemy(app)


########################################################################################
##############################
# Database model definition for the user table
########################################################################################
##############################

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(40), unique=True)

    def __str__(self):
        return self.username

    def get_user_id(self):
        return self.id

    def check_password(self, password):
        return password == 'valid'


########################################################################################
##############################
# Database model definition for the OAuth2 client table
########################################################################################
##############################

class OAuth2Client(db.Model, OAuth2ClientMixin):

    __tablename__ = 'oauth2_client'

    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id', ondelete='CASCADE'))
    user = db.relationship('User')
```

```
########################################################################################
##############################
# Database model definition for the OAuth2 token table
########################################################################################
##############################

class OAuth2Token(db.Model, OAuth2TokenMixin):

    __tablename__ = 'oauth2_token'

    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id', ondelete='CASCADE'))
    user = db.relationship('User')

    def is_refresh_token_active(self):
        if self.revoked:
            return False
        expires_at = self.issued_at + self.expires_in * 2
        return expires_at >= time.time()
```

5.1.85 services\backend_api\src\oauth\resource_protector.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Implementation of an own Resource Protector
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from authlib.integrations.flask_oauth2 import ResourceProtector as _ResourceProtector
from authlib.oauth2 import OAuth2Error
from authlib.oauth2.rfc6749 import MissingAuthorizationError, HttpRequest
from error_classes.http_error_401.http_error_401 import UnauthorizedError
from error_classes.http_error_403.http_error_403 import ForbiddenError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from flask import _app_ctx_stack, abort, request as _req
from flask.signals import Namespace
from init.app_constructor import app
import functools

########################################################################################
##############################
# OAuth signal definition
########################################################################################
##############################

_signal = Namespace()
client_authenticated = _signal.signal('client_authenticated')
token_revoked = _signal.signal('token_revoked')
token_authenticated = _signal.signal('token_authenticated')
```

```
##########################################################################################
##############################
# class definition for the Resource Protector
##########################################################################################
##############################

class ResourceProtector(_ResourceProtector):
    """ Class definition for an OAuth decorator

    This class defines a protecting method for resource servers. It is creating
    a "require_oauth" decorator easily with the definition of a ResourceProtector.

    """


##########################################################################################
##########################
    # Method for raising the error message

##########################################################################################
##########################

    def raise_error_response(self, error):
        """ Custom error response for OAuth2 errors

        This method raises an individual exception for OAuth2 errors. It is a re-
implementation
        of the original method in order to customize the error response.

        Arguments:
            error (obj): OAuth2Error

        Raises:
            Exception depending one the status code

        """

        status = error.status_code
        body = dict(error.get_body())

        if status == 401:
            if 'error_description' in body:
                error = UnauthorizedError(f"{body['error']}: {body['error_description']}",
"", "")

            else:
                error = UnauthorizedError(f"{body['error']}", "", "")

        elif status == 403:
            if 'error_description' in body:
                error = ForbiddenError(f"{body['error']}: {body['error_description']}",
"", "")

            else:
                error = ForbiddenError(f"{body['error']}", "", "")

        else:
            error = InternalServerErrorAPI(f"{body['error']}:
{body['error_description']}", "", "")
            abort(500, error.to_dict())

        abort(status, error.to_dict())


##########################################################################################
##########################
    # Method for acquiring the token
```

```
##########################################################################################
##########################

    def acquire_token(self, scope=None, operator=app.config['SCOPE_CONNECTOR']):
        """A method to acquire current valid token with the given scope.

        Arguments:
            scope (str, list): scope values
            operator (str): value of "AND" or "OR"

        Returns:
             token (obj): object
        """

        request = HttpRequest(
            _req.method,
            _req.full_path,
            _req.data,
            _req.headers
        )

        if not callable(operator):
            operator = operator.upper()
        token = self.validate_request(scope, request, operator)
        token_authenticated.send(self, token=token)
        ctx = _app_ctx_stack.top
        ctx.authlib_server_oauth2_token = token
        return token


##########################################################################################
##########################
    # Method for defining the decorator object

##########################################################################################
##########################

    def __call__(self, scope=None, operator=app.config['SCOPE_CONNECTOR'],
optional=False):
        def wrapper(f):
            @functools.wraps(f)
            def decorated(*args, **kwargs):
                try:
                    self.acquire_token(scope, operator)
                except MissingAuthorizationError as error:
                    if optional:
                        return f(*args, **kwargs)
                    self.raise_error_response(error)
                except OAuth2Error as error:
                    self.raise_error_response(error)
                return f(*args, **kwargs)

            return decorated

        return wrapper
```

5.1.86 services\backend_api\src\resources\resources_auth\create_client\create_client.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
```

```
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Create OAuth2 client API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
#########################################################################################
##############################

from error_classes.http_error_408.http_error_408 import RequestTimeoutError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from init.init_env_variables import oauth2_create_client
from init.namespace_constructor import auth_namespace as api
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from lib.auth_header import auth_header_parser
from models.models_auth.client_models.client_models import auth_client_request_model,
auth_client_response_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_408 import error_408_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from requests.exceptions import HTTPError
from oauth.oauth2 import require_oauth
import requests
import traceback


#########################################################################################
##############################
# Resource definition for the create OAuth2 client API call
#########################################################################################
##############################

@api.expect(auth_client_request_model)
@api.header('Content-Type', 'application/json')
class CreateOAuthClient(Resource):
    """ Class for handling the POST request

    This class defines the API call for the create OAuth2 client script. The class
consists of one method which accepts
    a POST request. For the POST request a JSON with several parameters is required and
defined in the corresponding
    model.

    """


#########################################################################################
##########################
    # Method for handling the POST request

#########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Operation successful', auth_client_response_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(408, 'Request Timeout', error_408_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for creating an OAuth2 client

        <p style="text-align: justify">This method defines the handler for the POST
request of the create OAuth2 client
        script. It returns the client ID and client secret wrapped into a Python
dictionary of the newly created OAuth2
        client.</p>

        <br><b>Description:</b>
        <p style="text-align: justify">The request call sends a HTTP POST request in the
background to the OAuth2 server
        with the full name of the client / user. In OAuth2 database a new client will be
created and the credentials for
        authentication will be returned. The generated client ID and secret should be kept
and not passed to anyone.
        Client ID and secret are used to generate Bearer tokens for the authentication
process. This request does not
        create a new customer in the GEMS customer database.</p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>client_name (str): Full name of the client</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the GET request is a JSON that
contains 2 key value pairs. The
        client ID is a unique identifier of a client in the OAuth2 database and in the
GEMS database for GEMS user. The
        client secret could be seen as a password for the GEMS user, used only for the
Bearer token generation.</p>
        <ul>
        <li><p><i>client_id (str): Unique client ID of a GEMS customer for
authentication</i></p></li>
        <li><p><i>client_secret (str): Unique client secret of a GEMS customer for
authentication</i></p></li>
        </ul>

        """

        try:
            req_args = api.payload
            gemslog(LogLevel.INFO, f'Request payload: {req_args}', 'API-
create_oauth_client')

            payload = {'client_name': req_args['client_name'],
                       'grant_type': 'password\nrefresh_token',
                       'response_type': 'code',
                       'client_uri': '',
                       'redirect_uri': '',
                       'scope': '',
                       'token_endpoint_auth_method': 'client_secret_basic'}

            headers = {'Content-Type': 'application/x-www-form-urlencoded'}

            response = requests.request('POST', oauth2_create_client, headers=headers,
data=payload, timeout=15)

        except requests.exceptions.ReadTimeout:
            error = RequestTimeoutError('Connection timed out while contacting the 
Authorization server', '', '')
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
create_oauth_client')
            return {'message': error.to_dict()}, 408

        except HTTPError:
            error = ServiceUnavailableError('Could not connect to OAuth2 server', '', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
create_oauth_client')
            return {'message': error.to_dict()}, 404

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload, 
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
create_oauth_client')
            return {'message': error.to_dict()}, 500

        else:
            if response.status_code == 200:
                gemslog(LogLevel.INFO, f'Request response: {response.json()}', 'API-
create_oauth_client')
                return response.json()

            else:
                error = ServiceUnavailableError('Could not contact the Authorization 
Server', api.payload, '')
                gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
create_oauth_client')
                return {'message': error.to_dict()}, 503
```

5.1.87 services\backend_api\src\resources\resources_auth\delete_clients\delete_clients.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is 
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Delete all OAuth2 clients API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_oauth
from init.namespace_constructor import auth_namespace as api
from lib.auth_header import auth_header_parser
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


##########################################################################################
##############################
# Resource definition for the delete OAuth2 clients API call
##########################################################################################
##############################

@api.header('Content-Type', 'application/json')
class DeleteOAuthClients(Resource):
    """ Class for handling the DELETE request

    This class defines the API call for the delete OAuth2 client script. The class
consists of one method which accepts a
    DELETE request. For the DELETE request a JSON with several parameters is required and
defined in the corresponding
    model.

    """


##########################################################################################
##########################
    # Method for handling the DELETE request

##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation successful')
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def delete(self):
        """ DELETE definition for removing all OAuth2 clients

        <p style="text-align: justify">This method defines the handler for the DELETE
request of the delete OAuth2
        clients script. It returns no message body and thus no contents. In contrast it
returns the HTTP status code
        204.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the DELETE request does not contain
any object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any
other response status code
        than 204, indicates an error during the execution.</p>

        """

        try:
            db_query = "UPDATE public.oauth2_client SET deleted_at = NOW()"
            execute_database(db_query, (), database_config_file,
database_config_section_oauth, True)
```

```
        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-delete_clients')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', '',
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-delete_clients')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Successfully removed all OAuth clients', 'API-
delete_clients')
            return '', 204
```

5.1.88 services\backend_api\src\resources\resources_auth\delete_client_by_id\delete_client_by_id.py

```
###########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Delete an OAuth2 client by ID API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
###########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_oauth
from init.namespace_constructor import auth_namespace as api
from lib.auth_header import auth_header_parser
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


###########################################################################################
##############################
# Resource definition for the OAuth2 client by ID API call
###########################################################################################
##############################
```

```
@api.header('Content-Type', 'application/json')
@api.param('user_id', 'User ID to be deleted')
class DeleteOAuthClient(Resource):
    """ Class for handling the DELETE request

    This class defines the API call for the delete OAuth2 client by ID script. The class
consists of one method which
    accepts a DELETE request. For the DELETE request an additional path parameter is
required.

    """


###############################################################################################
###########################
    # Method for handling the DELETE request

###############################################################################################
###########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation successful')
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def delete(self, user_id):
        """ DELETE definition for removing an OAuth2 client by ID

        <p style="text-align: justify">This method defines the handler for the DELETE
request of the delete OAuth2
        client by ID script. It returns no message body and thus no contents. In contrast
it returns the HTTP status
        code 204.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameter:</b>
        <ul>
        <li><p><i>client_id (str): </i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the DELETE request does not contain
any object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any
other response status code
        than 204, indicates an error during the execution.</p>

        """

        try:
            gemslog(LogLevel.INFO, f'Request path parameter: {user_id}', 'API-
delete_client_id')

            db_query = "UPDATE public.oauth2_client SET deleted_at = NOW() WHERE client_id
= %s"
            execute_database(db_query, (user_id,), database_config_file,
database_config_section_oauth, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_client_id')
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_client_id')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', '',
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_client_id')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Successfully removed OAuth client with ID:
{user_id}', 'API-delete_client_id')
            return '', 204
```

5.1.89 services\backend_api\src\resources\resources_auth\delete_tokens\delete_tokens.py

```
###########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Delete all OAuth2 tokens API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
###########################################################################################
##############################

from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_oauth
from init.namespace_constructor import auth_namespace as api
from lib.auth_header import auth_header_parser
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


###########################################################################################
##############################
```

```
# Resource definition for the delete OAuth2 clients API call
############################################################################################
##############################

@api.header('Content-Type', 'application/json')
class DeleteTokens(Resource):
    """ Class for handling the DELETE request

    This class defines the API call for the delete OAuth2 token script. The class consists
of one method which accepts a
    DELETE request. For the DELETE request a JSON with several parameters is required and
defined in the corresponding
    model.

    """


############################################################################################
##########################
    # Method for handling the DELETE request

############################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation successful')
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def delete(self):
        """ DELETE definition for removing all OAuth2 tokens

        <p style="text-align: justify">This method defines the handler for the DELETE
request of the delete OAuth2
        tokens script. It returns no message body and thus no contents. In contrast it
returns the HTTP status code 204.
        </p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the DELETE request does not contain
any object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any
other response status code
        than 204, indicates an error during the execution.</p>

        """

        try:
            db_query = "UPDATE public.oauth2_token SET deleted_at = NOW()"
            execute_database(db_query, (), database_config_file,
database_config_section_oauth, True)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-delete_tokens')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', '', 
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-delete_tokens')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Successfully removed all OAuth clients', 'API-
delete_tokens')
            return '', 204
```

5.1.90 services\backend_api\src\resources\resources_auth\delete_tokens_by_id\delete_tokens_by_id.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, is prohibited for all commercial applications without
# licensing by GeoVille GmbH.
#
# Delete OAuth2 tokens by ID API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_oauth
from init.namespace_constructor import auth_namespace as api
from lib.auth_header import auth_header_parser
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


##########################################################################################
##############################
# Resource definition for the OAuth2 client by ID API call
##########################################################################################
##############################

@api.header('Content-Type', 'application/json')
@api.param('user_id', 'User ID to be deleted')
class DeleteTokensByID(Resource):
    """ Class for handling the DELETE request

    This class defines the API call for the delete OAuth2 client by ID script. The class 
consists of one method which
```

```
    accepts a DELETE request. For the DELETE request an additional path parameter is
required.

    """


##########################################################################################
##########################
    # Method for handling the DELETE request

##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation successful')
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def delete(self, user_id):
        """ DELETE definition for removing an OAuth2 Token by ID

        <p style="text-align: justify">This method defines the handler for the DELETE
request of the delete OAuth2 token
        by ID script. It returns no message body and thus no contents. In contrast it
returns the HTTP status code 204.
        </p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameter:</b>
        <ul>
        <li><p><i>client_id (str): </i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the DELETE request does not contain
any object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any
other response status code
        than 204, indicates an error during the execution.</p>

        """

        try:
            gemslog(LogLevel.INFO, f'Request path parameter: {client_id}', 'API-
delete_tokens_id')

            db_query = "UPDATE public.oauth2_token SET deleted_at = NOW() WHERE client_id
= %s"
            execute_database(db_query, (user_id,), database_config_file,
database_config_section_oauth, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_tokens_id')
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server', 
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_tokens_id')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', '', 
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_tokens_id')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Successfully removed OAuth client with ID: 
{user_id}', 'API-delete_tokens_id')
            return '', 204
```

5.1.91 services\backend_api\src\resources\resources_auth\get_bearer_token\get_bearer_token.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is 
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Get Bearer token API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_401.http_error_401 import UnauthorizedError
from error_classes.http_error_408.http_error_408 import RequestTimeoutError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from flask_restx import Resource
from init.init_env_variables import (database_config_file, database_config_section_oauth, 
oauth2_bearer_expiration_time,
                                     oauth2_generate_token, oauth2_password, oauth2_user)
from init.namespace_constructor import auth_namespace as api
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from lib.database_helper import (check_client_id_existence, 
check_client_id_secret_existence,
                                 check_client_secret_existence, get_scope_by_id)
from models.models_auth.access_token_models.access_token_models import 
bearer_token_request_model, bearer_token_response_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_408 import error_408_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from requests.exceptions import HTTPError
```

```
import requests
import traceback


###############################################################################################
###############################
# Resource definition for the get Bearer token API call
###############################################################################################
###############################

@api.header('Content-Type', 'application/json')
class GetBearerToken(Resource):
    """ Class for handling the GET request

    This class defines the API call for the get Bearer token script. The class consists of
one method which accepts a
    GET request. For the GET request two path parameters are required.
    """


###############################################################################################
###########################
    # Method for handling the GET request

###############################################################################################
###########################

    @api.expect(bearer_token_request_model)
    @api.response(200, 'Operation successful', bearer_token_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(404, 'Not Found Error', error_404_model)
    @api.response(408, 'Request Timeout', error_408_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for retrieving an OAuth2 Bearer token

        <p style="text-align: justify">This method defines the handler for the GET request
of the get Bearer token
        request. It returns all the information from the OAuth2 server</p>

        <br><b>Description:</b>
        <p style="text-align: justify">The GEMS microservice architecture has foreseen an
OAuth2 authentication and
        authorisation server for handling the GEMS costumers login and the access to GEMS
services. To gain access of
        the services of the GEMS API it is necessary for each consumer to create a new
Bearer token from this service
        route. The token type and the token itself are needed for service routes which
require an authorisation header.
        In the authorisation header, the API consumer need to write the token in the
following format "Bearer XXXX".</p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>None</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>client_id (str): User specific client ID provided by the GEMS
administration team</i></p></li>
        <li><p><i>client_secret (str): User specific client secret provided by the GEMS
administration team</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the submitted request is an
automatically created JSON response of
        the OAuth2 server which returns the following parameters:</p>
        <ul>
        <li><p><i>access_token: Token needed for all the authorisation steps</i></p></li>
        <li><p><i>expires_in: Expiration time of the token in seconds</i></p></li>
        <li><p><i>refresh_token: Can be used to obtain a renewed access token</i></p></li>
        <li><p><i>token_type: Type of the token, in case of GEMS, it is always Bearer.
Needed in the authorisation header</i></p></li>
        </ul>

        """

        try:
            req_args = api.payload
            gemslog(LogLevel.INFO, f'Request payload: {req_args}', 'API-get_bearer_token')

            if not check_client_id_existence(req_args['user_id'], database_config_file,
database_config_section_oauth):
                error = NotFoundError('User ID does not exist', '', '')
                gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_bearer_token')
                return {'message': error.to_dict()}, 404

            if not check_client_secret_existence(req_args['client_secret'],
database_config_file, database_config_section_oauth):
                error = NotFoundError('Client secret does not exist', '', '')
                gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_bearer_token')
                return {'message': error.to_dict()}, 404

            if not check_client_id_secret_existence(req_args['user_id'],
req_args['client_secret'],
                                                    database_config_file,
database_config_section_oauth):
                error = BadRequestError('Invalid client credentials combination', '', '')
                gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_bearer_token')
                return {'message': error.to_dict()}, 400

            scope = get_scope_by_id(req_args['user_id'], database_config_file,
database_config_section_oauth)

            files = {
                'grant_type': (None, "password"),
                'username': (None, oauth2_user),
                'password': (None, oauth2_password),
                'scope': (None, scope),
            }

            response = requests.post(oauth2_generate_token, files=files,
auth=(req_args['user_id'],

req_args['client_secret']), timeout=15)

            token_response = response.json()
            update_query = """UPDATE public.oauth2_token
                              SET
                                  expires_in = %s
                              WHERE
                                  access_token like %s;
                           """

            execute_database(update_query, (oauth2_bearer_expiration_time,
token_response['access_token']),
                             database_config_file, database_config_section_oauth, True)

            token_response['expires_in'] = oauth2_bearer_expiration_time
        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}', '',
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_bearer_token')
            return {'message': error.to_dict()}, 400

        except requests.exceptions.ReadTimeout:
            error = RequestTimeoutError('Connection timed out while contacting the
Authorization server', '', '')
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
get_bearer_token')
            return {'message': error.to_dict()}, 408

        except HTTPError:
            error = NotFoundError(f'Could not connect to OAuth2 server', '',
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_bearer_token')
            return {'message': error.to_dict()}, 404

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_bearer_token')
            return {'message': error.to_dict()}, 500

        else:
            if response.status_code == 200:
                gemslog(LogLevel.INFO, f'Request response: {response.json()}', 'API-
get_bearer_token')
                return token_response

            elif response.status_code == 401:
                error = UnauthorizedError('Submitted client is invalid', '', '')
                gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_bearer_token')
                return {'message': error.to_dict()}, 401

            else:
                error = InternalServerErrorAPI(f'Unexpected error: {response.text}', '',
'')
                gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_bearer_token')
                return {'message': error.to_dict()}, 501
```

5.1.92 services\backend_api\src\resources\resources_auth\get_clients\get_clients.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Get OAuth2 clients API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
```

```
#############################################################################################
##############################

from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import read_from_database_all_rows
from init.init_env_variables import database_config_file, database_config_section_oauth
from init.namespace_constructor import auth_namespace as api
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from lib.auth_header import auth_header_parser
from models.models_auth.client_models.client_models import clients_list_response_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import json
import traceback


#############################################################################################
##############################
# Resource definition for the get scopes API call
#############################################################################################
##############################

@api.header('Content-Type', 'application/json')
class GetClients(Resource):
    """ Class for handling the GET request

    This class defines the API call for the get OAuth2 clients script. The class consists
of one method which accepts a
    GET request. For the GET request no more additional parameters are required.

    """


#############################################################################################
##########################
    # Method for handling the GET request

#############################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Operation successful', clients_list_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self):
        """ GET definition for retrieving all OAuth2 clients

        <p style="text-align: justify">This method defines the handler for the GET request
of the get OAuth2 clients
        script. It returns a message wrapped into a dictionary with all the necessary
client information.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify"></p>

        """

        try:
            db_query = "SELECT client_id, client_metadata FROM public.oauth2_client WHERE
deleted_at IS NULL"
            clients = read_from_database_all_rows(db_query, (), database_config_file,
database_config_section_oauth,
                                                  True)

            if clients is None or clients is False:
                gemslog(LogLevel.INFO, 'No client data found', 'API-get_clients')
                return {'oauth_clients': None}, 200

            res_array = []

            for single_client in clients:
                additional_data = json.loads(single_client[1])
                client_obj = {'client_id': single_client[0],
                              'client_name': additional_data['client_name'],
                              'grant_type': additional_data['grant_types'],
                              'response_type': additional_data['response_types'],
                              'scope': additional_data['scope']
                              }

                res_array.append(client_obj)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-get_clients')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', '',
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-get_clients')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, 'Request successful', 'API-get_clients')
            return {'oauth_clients': res_array}, 200
```

5.1.93 services\backend_api\src\resources\resources_auth\get_client_by_id\get_client_by_id.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Get OAuth2 client by ID API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
#########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import read_from_database_one_row
from init.init_env_variables import database_config_file, database_config_section_oauth
from init.namespace_constructor import auth_namespace as api
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from lib.auth_header import auth_header_parser
from lib.database_helper import check_client_id_existence
from models.models_auth.client_models.client_models import client_response_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import json
import traceback


#########################################################################################
##############################
# Resource definition for the get OAuth2 client by ID API call
#########################################################################################
##############################

@api.header('Content-Type', 'application/json')
@api.param('user_id', 'User ID to be requested')
class GetClientByID(Resource):
    """ Class for handling the GET request

    This class defines the API call for the get OAuth2 client by ID script. The class
consists of one method which accepts a
    GET request. For the GET request an additional path parameter is required..

    """


#########################################################################################
##########################
    # Method for handling the GET request

#########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Operation successful', client_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self, user_id):
        """ GET definition for retrieving an OAuth2 client by ID
        <p style="text-align: justify">This method defines the handler for the GET request
of the get OAuth2 client by
        ID script. It returns a message wrapped into a dictionary with all the necessary
client information.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameter:</b>
        <ul>
        <li><p><i>client_id (str): </i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify"></p>

        """

        try:
            if not check_client_id_existence(user_id, database_config_file,
database_config_section_oauth):
                error = NotFoundError('Client ID does not exist', '', '')
                gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_client_by_id')
                return {'message': error.to_dict()}, 404

            db_query = """SELECT
                                client_id, client_metadata
                            FROM
                                public.oauth2_client
                            WHERE
                                client_id = %s AND
                                deleted_at IS NULL
                        """

            client_data = read_from_database_one_row(db_query, (user_id,),
database_config_file,
                                                     database_config_section_oauth, True)
            additional_data = json.loads(client_data[1]) if client_data is not None else
None

            client_obj = {'client_id': client_data[0],
                          'client_name': additional_data['client_name'],
                          'grant_type': additional_data['grant_types'],
                          'response_type': additional_data['response_types'],
                          'scope': additional_data['scope']
                          } if client_data is not None else {}

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}', '',
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_client_by_id')
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_client_by_id')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', '',
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_client_by_id')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, 'Request successful', 'API-get_client_by_id')
            return client_obj, 200
```

5.1.94 services\backend_api\src\resources\resources_auth\get_scopes\get_scopes.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Get all OAuth2 scopes API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import read_from_database_all_rows
from init.namespace_constructor import auth_namespace as api
from init.init_env_variables import database_config_file, database_config_section_oauth
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from lib.auth_header import auth_header_parser
from models.models_auth.scope_models.scope_models import scope_list_response_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import json
import traceback


##########################################################################################
##############################
# Resource definition for the get scopes API call
##########################################################################################
##############################

@api.header('Content-Type', 'application/json')
class GetScopes(Resource):
    """ Class for handling the GET request

    This class defines the API call for the get OAuth2 scopes script. The class consists
of one method which accepts a
```

```
    POST request. For the GET request a JSON with several parameters is required and
defined in the corresponding model.

    """


##########################################################################################
##########################
    # Method for handling the GET request

##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.doc(parser=auth_header_parser)
    @api.response(200, 'Operation successful', scope_list_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self):
        """ GET definition for retrieving all OAuth2 scopes

        <p style="text-align: justify">This method defines the handler for the GET request
of the get all OAuth2 scopes
        script. It returns a message wrapped into a dictionary with the actual scope
definition.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify"></p>

        """

        try:
            db_query = "SELECT client_id, client_metadata FROM public.oauth2_client WHERE
deleted_at IS NULL"
            scope = read_from_database_all_rows(db_query, (), database_config_file,
database_config_section_oauth, True)

            if scope is None or scope is False:
                gemslog(LogLevel.INFO, 'No scope data found', 'API-get_scopes')
                return {'scopes': None}, 200

            res_array = []

            for single_scope in scope:
                additional_data = json.loads(single_scope[1])
                scope_obj = {'client_id': single_scope[0], 'scope':
additional_data['scope']}
                res_array.append(scope_obj)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-get_scopes')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', '',
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-get_scopes')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Request successful: {scope_obj}', 'API-get_scopes')
            return {'scopes': res_array}, 200
```

5.1.95 services\backend_api\src\resources\resources_auth\get_scope_by_id\get_scope_by_id.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Get OAuth2 scope by ID API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import read_from_database_one_row
from init.init_env_variables import database_config_file, database_config_section_oauth
from init.namespace_constructor import auth_namespace as api
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from lib.auth_header import auth_header_parser
from lib.database_helper import check_client_id_existence
from models.models_auth.scope_models.scope_models import scope_response_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import json
import traceback


########################################################################################
##############################
# Resource definition for the get OAuth2 scope by ID API call
########################################################################################
##############################

@api.header('Content-Type', 'application/json')
@api.param('user_id', 'User ID to be requested')
class GetScopeByID(Resource):
    """ Class for handling the GET request

    This class defines the API call for the get scope by ID script. The class consists of
one method which accepts a
    POST request. For the GET request an additional path parameter is required.

    """


##########################################################################################
##########################
    # Method for handling the GET request

##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Operation successful', scope_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self, user_id):
        """ GET definition for retrieving the OAuth2 scope by ID

        <p style="text-align: justify">This method defines the handler for the GET request
of the get OAuth2 scope by
        ID script. It returns a message wrapped into a dictionary with the actual scope
definition.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameter:</b>
        <ul>
        <li><p><i>client_id (str): </i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify"></p>

        """

        try:
            if not check_client_id_existence(user_id, database_config_file,
database_config_section_oauth):
                error = NotFoundError('Client ID does not exist', '', '')
                gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_scope_by_id')
                return {'message': error.to_dict()}, 404

            db_query = """SELECT
                              client_id, client_metadata
                          FROM
                              public.oauth2_client
                          WHERE
                              client_id = %s AND
                              deleted_at IS NULL
                       """
            scope_data = read_from_database_one_row(db_query, (user_id,),
database_config_file,
                                                    database_config_section_oauth, True)

            additional_data = json.loads(scope_data[1]) if scope_data is not None else
None
            scope_obj = {'client_id': scope_data[0],
                         'scope': additional_data['scope']} if scope_data is not None else
None

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}', '',
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}")
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_scope_by_id', )
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', '',
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_scope_by_id', )
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Request successful: {scope_obj}', 'API-
get_scope_by_id')
            return scope_obj, 200
```

5.1.96 services\backend_api\src\resources\resources_auth\login\login.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Login API Call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_408.http_error_408 import RequestTimeoutError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from flask_restx import Resource
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.app_constructor import bcrypt
from init.init_env_variables import (database_config_file, database_config_section_api,
database_config_section_oauth)
```

```
from init.namespace_constructor import auth_namespace as api
from lib.database_helper import (check_email_existence, get_client_id_secret)
from lib.request_helper import get_bearer_token
from models.models_auth.login_model.login_model import login_request_model,
login_response_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_408 import error_408_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from requests.exceptions import HTTPError
import requests
import traceback


###########################################################################################
##############################
# Resource definition for the get bearer token API call
###########################################################################################
##############################

@api.header('Content-Type', 'application/json')
class Login(Resource):
    """ Class for handling the GET request

    This class defines the API call for the get Bearer token script. The class consists of
one method which accepts a
    GET request. For the GET request two path parameters are required.
    """


###########################################################################################
##########################
    # Method for handling the GET request

###########################################################################################
##########################

    @api.expect(login_request_model)
    @api.response(200, 'Operation successful', login_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(404, 'Not Found Error', error_404_model)
    @api.response(408, 'Request Timeout', error_408_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for logging in

        <p style="text-align: justify">This method defines the handler for the GET request
of the get Bearer token
        request. It returns all the information from the OAuth2 server</p>

        <br><b>Description:</b>
        <p style="text-align: justify">The GEMS microservice architecture has foreseen an
OAuth2 authentication and
        authorisation server for handling the GEMS costumers login and the access to GEMS
services. To gain access of
        the services of the GEMS API it is necessary for each consumer to create login and
gain a new Bearer token from
        this service route. The token type and the token itself are needed for service
routes which require an
        authorisation header. In the authorisation header, the API consumer need to write
the token in the following
        format "Bearer XXXX".</p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>None</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>email (str): User specific client ID provided by the GEMS administration
team</i></p></li>
        <li><p><i>password (str): User specific client secret provided by the GEMS
administration team</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the submitted request is an
automatically created JSON response of
        the OAuth2 server which returns the following parameters:</p>
        <ul>
        <li><p><i>access_token: Token needed for all the authorisation steps</i></p></li>
        <li><p><i>expires_in: Expiration time of the token in seconds</i></p></li>
        <li><p><i>refresh_token: Can be used to obtain a renewed access token</i></p></li>
        <li><p><i>scope: All grants the specific user has</i></p></li>
        <li><p><i>token_type: Type of the token, in case of GEMS, it is always Bearer.
Needed in the authorisation header</i></p></li>
        </ul>

        """

        try:
            req_args = api.payload

            user_id = check_email_existence(req_args['email'], database_config_file,
database_config_section_api)
            if not user_id:
                error = BadRequestError(f'User does not exist', api.payload, '')
                gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-login')
                return {'message': error.to_dict()}, 400

            if not bcrypt.check_password_hash(user_id[1], req_args['password']):
                error = BadRequestError(f'Wrong password submitted', api.payload, '')
                gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-login')
                return {'message': error.to_dict()}, 400

            client_secret = get_client_id_secret(user_id[0], database_config_file,
database_config_section_oauth)
            token_response = get_bearer_token(user_id[0], client_secret)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}', '',
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-login')
            return {'message': error.to_dict()}, 400

        except requests.exceptions.ReadTimeout:
            error = RequestTimeoutError('Connection timed out while contacting the
Authorization server', '', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-login')
            return {'message': error.to_dict()}, 408

        except HTTPError:
            error = NotFoundError(f'Could not connect to OAuth2 server', '',
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-login')
            return {'message': error.to_dict()}, 404

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-login')
            return {'message': error.to_dict()}, 500
        else:
            token_response.update([('client_id', user_id[0]), ('client_secret',
client_secret)])
            gemslog(LogLevel.INFO, 'Login successful', 'API-login')
            return token_response
```

5.1.97 services\backend_api\src\resources\resources_auth\set_scope_by_id\set_scope_by_id.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Set OAuth2 scope by ID API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database,
read_from_database_one_row
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_oauth
from init.namespace_constructor import auth_namespace as api
from lib.auth_header import auth_header_parser
from models.models_auth.scope_models.scope_models import scope_update_request_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import json
import traceback


##########################################################################################
##############################
# Resource definition for the set scope API call
##########################################################################################
##############################

@api.expect(scope_update_request_model)
@api.header('Content-Type', 'application/json')
class UpdateScope(Resource):
    """ Class for handling the PATCH request

    This class defines the API call for the set OAuth2 scope by ID script. The class
consists of one method which
```

```
    accepts a PATCH request. For the PATCH request a JSON with several parameters is
required and defined in the
    corresponding model.

    """


##########################################################################################
##########################
    # Method for handling the PATCH request

##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation successful')
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def patch(self):
        """ PATCH definition for setting the OAuth2 scope by ID

        <p style="text-align: justify">This method defines the handler for the PATCH
request of the update OAuth2 scope by ID script. It returns a
        message wrapped into a dictionary about the status of the update operation. The
scope parameter is a string
        which separates the scope values with a whitespace. Please note that the route
overrides the current scope and
        does no concatenation with already existing scope values.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameter:</b>
        <ul>
        <li><p><i>client_id (str): </i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the PATCH request does not contain
any object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any
other response status code
        than 204, indicated an error during the execution.</p>

        """

        try:
            req_args = api.payload
            gemslog(LogLevel.INFO, f'Request payload: {req_args}', 'API-set_scope_by_id')

            db_query = "SELECT client_metadata FROM public.oauth2_client WHERE client_id =
%s"
            gemslog(LogLevel.INFO,
                    db_query % req_args['client_id'],
                    'API-set_scope_by_id')
            scope_data = read_from_database_one_row(db_query, (req_args['client_id'],),
database_config_file,
                                                    database_config_section_oauth, True)

            additional_data = json.loads(scope_data[0])
            additional_data['scope'] = req_args['scope']

            db_query = """UPDATE
                                public.oauth2_client
                            SET
                                client_metadata= %s
                            WHERE
                                client_id = %s AND
                                deleted_at IS NULL
                                     """
            gemslog(LogLevel.INFO,
                    db_query % (json.dumps(additional_data), req_args['client_id']),
                    'API-set_scope_by_id')
            execute_database(db_query, (json.dumps(additional_data),
req_args['client_id']), database_config_file,
                              database_config_section_oauth, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
set_scope_by_id')
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
set_scope_by_id')
            return {'message': error.to_dict()}, 503

        except Exception as err:
            error = InternalServerErrorAPI(f'Unexpected error occurred {err}',
api.payload, traceback.format_exc())
            gemslog('API-set_scope_by_id', LogLevel.ERROR, f"'message':
{error.to_dict()}")
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f"Updated the scope entry for ID:
{req_args['client_id']}", 'API-set_scope_by_id')
            return '', 204
```

5.1.98 services\backend_api\src\resources\resources_auth\set_token_expiration_time\set_token_expiration_time.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Set OAuth2 token expiration time API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################
```

```
from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_oauth
from init.namespace_constructor import auth_namespace as api
from lib.auth_header import auth_header_parser
from models.models_auth.token_models.token_models import exp_time_request_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


#########################################################################################
##############################
# Resource definition for the set OAuth2 token expiration time API call
#########################################################################################
##############################

@api.expect(exp_time_request_model)
@api.header('Content-Type', 'application/json')
class UpdateTokenExpirationTime(Resource):
    """ Class for handling the PATCH request

    This class defines the API call for the set OAuth 2 token expiration time script. The
class consists of one method which
    accepts a PATCH request. For the PATCH request a JSON with several parameters is
required and defined in the
    corresponding model.

    """


#########################################################################################
##########################
    # Method for handling the PATCH request

#########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation successful')
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def patch(self):
        """ PATCH definition for setting the OAuth2 token expiration time

        <p style="text-align: justify">This method defines the handler for the PATCH
request of the set OAuth2 token
        expiration time script. It returns a no message about the status of the update
operation. In contrast it returns
        the HTTP status code 204.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameter:</b>
        <ul>
        <li><p><i>client_id (str): </i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the PATCH request does not contain
any object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any
other response status code
        than 204, indicated an error during the execution.</p>

        """

        try:
            req_args = api.payload
            gemslog(LogLevel.INFO, f'Request payload: {req_args}', 'API-token_time')

            db_query = """UPDATE
                                        public.oauth2_token
                                    SET
                                        expires_in = %s,
                                        updated_at = NOW()
                                    WHERE
                                        access_token = %s AND
                                        deleted_at IS NULL
                                 """
            execute_database(db_query, (req_args['exp_time'], req_args['bearer_token']),
database_config_file,
                             database_config_section_oauth, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-token_time')
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-token_time')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-token_time')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, 'Set new token expiration time', 'API-token_time')
            return '', 204
```

5.1.99 services\backend_api\src\resources\resources_config\add_airflow_config\add_airflow_config.py

```
##########################################################################################
##############################
```

```
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Adds a new Airflow configuration to the database
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import config_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_airflow_service_existence
from models.models_config.airflow_models.airflow_config_models import
add_airflow_config_model, airflow_config_success_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


########################################################################################
##############################
# Resource definition for the inserting a new Airflow configuration API call
########################################################################################
##############################

@api.expect(add_airflow_config_model)
@api.header('Content-Type', 'application/json')
class AddAirflowConfig(Resource):
    """ Class for handling the POST request

    This class defines the API call for the add Airflow configuration script. The class
consists of one method which
    accepts a POST request. For the POST request a JSON with several parameters is
required and defined in the
    corresponding model.

    """


########################################################################################
##########################
    # Method for handling the POST request

########################################################################################
##########################
```

```
    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(201, 'Operation successful', airflow_config_success_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for adding a new Airflow configuration

        <p style="text-align: justify">This method defines the handler for the POST
request of the add Airflow
        configuration script. It returns a message wrapped into a dictionary about the
status of the insertion
        operation.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>service_name (str): Full name of the client</i></p></li>
        <li><p><i>command (str): Full name of the client</i></p></li>
        <li><p><i>description (str): Full name of the client</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify"></p>

        """

        try:
            req_args = api.payload
            gemslog(LogLevel.INFO, f'Request: {req_args}', 'API-add_airflow_config')

            if check_airflow_service_existence(req_args['service_name'],
database_config_file, database_config_section_api):
                error = NotFoundError('Service name exists already', '', '')
                gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
add_airflow_config')
                return {'message': error.to_dict()}, 404

            db_query = """INSERT INTO msgeovilleconfig.airflow_config
                           (
                               service_name, command, description
                           )
                           VALUES
                           (
                               %s, %s, %s
                           )
                       """

            execute_database(db_query, (req_args['service_name'], req_args['command'],
req_args['description']),
                                 database_config_file, database_config_section_api, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
```

```
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
add_airflow_config')
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
add_airflow_config')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
add_airflow_config')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Request: {req_args}', 'API-add_airflow_config')
            return {'service_name': req_args['service_name']}, 201
```

5.1.100 services\backend_api\src\resources\resources_config\add_queue_config\add_queue_config.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Adds a new API queue configuration to the database
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import config_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_service_existence, check_queue_existence
from models.models_config.queue_config.queue_config_models import add_queue_config_model,
queue_creation_success_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


###########################################################################################
##############################
# Resource definition for the inserting a new queue configuration API call
###########################################################################################
##############################

@api.expect(add_queue_config_model)
@api.header('Content-Type', 'application/json')
class AddQueueConfig(Resource):
    """ Class for handling the POST request

    This class defines the API call for the add queue configuration script. The class
consists of one method which
    accepts a POST request. For the POST request a JSON with several parameters is
required and defined in the
    corresponding model.

    """


###########################################################################################
##########################
    # Method for handling the POST request

###########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(201, 'Operation successful', queue_creation_success_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for adding a new queue configuration

        <p style="text-align: justify">This method defines the handler for the POST
request of the add queue
        configuration script. It returns a message wrapped into a dictionary about the
status of the insertion
        operation.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>service_id (str): Full name of the client</i></p></li>
        <li><p><i>queue_name (str): Full name of the client</i></p></li>
        <li><p><i>host (str): Full name of the client</i></p></li>
        <li><p><i>port (str): Full name of the client</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify"></p>
        """

        try:
            req_args = api.payload
            gemslog(LogLevel.INFO, f'Request: {req_args}', 'API-add_queue_config')

            if check_queue_existence(req_args['queue_name'], database_config_file,
database_config_section_api):
                error = NotFoundError('Queue name exists already', '', '')
                gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
add_queue_config')
                return {'message': error.to_dict()}, 404

            if not check_service_existence(req_args['service_id'], database_config_file,
database_config_section_api):
                error = NotFoundError('Service ID does not exist', '', '')
                gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
add_queue_config')
                return {'message': error.to_dict()}, 404

            db_query = """INSERT INTO msgeovilleconfig.message_queue_config
                          (
                              queue_name, host, port, service_id
                          )
                          VALUES
                          (
                              %s, %s, %s, %s
                          )
                       """

            execute_database(db_query, (req_args['queue_name'], req_args['host'],
req_args['port'],
                                        req_args['service_id']), database_config_file,
database_config_section_api, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
add_queue_config')
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
add_queue_config')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
add_queue_config')
            return {'message': error.to_dict()}, 500

        else:
            return {'service_id': req_args['service_id'],
                    'queue_name': req_args['queue_name']}, 201
```

5.1.101 services\backend_api\src\resources\resources_config\delete_airflow_config\delete_airflow_config.py

```
########################################################################################
##############################
#
```

```
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Delete Airflow configuration API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import config_namespace as api
from lib.auth_header import auth_header_parser
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


##########################################################################################
##############################
# Resource definition for the delete Airflow configuration API call
##########################################################################################
##############################

@api.header('Content-Type', 'application/json')
class DeleteAirflowConfiguration(Resource):
    """ Class for handling the DELETE request

    This class defines the API call for the delete Airflow configuration script. The class
consists of one method which
    accepts a DELETE request. For the DELETE request no parameters are required.

    """


##########################################################################################
##########################
    # Method for handling the DELETE request

##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation successful')
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def delete(self):
        """ DELETE definition for removing the Airflow configuration
        <p style="text-align: justify">This method defines the handler for the DELETE 
request of the delete Airflow
        configuration script. It returns no message body and thus no contents. In contrast 
it returns the HTTP status
        code 204.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the DELETE request does not contain 
any object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any 
other response status code
        than 204, indicates an error during the execution.</p>

        """

        try:
            db_query = "UPDATE msgeovilleconfig.airflow_config SET deleted_at = NOW()"
            execute_database(db_query, (), database_config_file, 
database_config_section_api, True)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server', 
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_airflow_config')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload, 
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_airflow_config')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, 'Deleted entire airflow config table', 'API-
delete_airflow_config')
            return '', 204
```

5.1.102 services\backend_api\src\resources\resources_config\delete_airflow_config_by_name\delete_airflow_config_by_name.py

```
###########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is 
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Delete Airflow configuration by name API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import config_namespace as api
from lib.auth_header import auth_header_parser
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


##########################################################################################
##############################
# Resource definition for the delete Airflow configuration by name API call
##########################################################################################
##############################

@api.header('Content-Type', 'application/json')
@api.param('service_name', 'Service name to be deleted')
class DeleteAirflowConfigByName(Resource):
    """ Class for handling the DELETE request

    This class defines the API call for the delete Airflow configuration by name script.
The class consists of one
    method which accepts a DELETE request. For the DELETE request no parameters are
required.

    """


##########################################################################################
##########################
    # Method for handling the DELETE request

##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation successful')
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def delete(self, service_name):
        """ DELETE definition for removing the Airflow configuration by name

        <p style="text-align: justify">This method defines the handler for the DELETE
request of the delete Airflow
        configuration script. It returns no message body and thus no contents. In contrast
it returns the HTTP status code 204.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameter:</b>
        <ul>
        <li><p><i>service_name (str): </i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the DELETE request does not contain
any object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any
other response status code
        than 204, indicates an error during the execution.</p>

        """

        try:
            gemslog(LogLevel.INFO, f'Request path parameter: {service_name}', 'API-
delete_airflow_config_by_name')

            db_query = "UPDATE msgeovilleconfig.airflow_config SET deleted_at = NOW()
WHERE service_name = %s"
            execute_database(db_query, (service_name, ), database_config_file,
database_config_section_api, True)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_airflow_config_by_name')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_airflow_config_by_name')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Deleted airflow config for: {service_name}', 'API-
delete_airflow_config_by_name')
            return '', 204
```

5.1.103 services\backend_api\src\resources\resources_config\delete_logging_entries\delete_logging_entries.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Delete logging entries API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
```

```
##########################################################################################
##############################

from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import config_namespace as api
from lib.auth_header import auth_header_parser
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


##########################################################################################
##############################
# Resource definition for the delete logging entries API call
##########################################################################################
##############################

@api.header('Content-Type', 'application/json')
class DeleteLoggingEntries(Resource):
    """ Class for handling the DELETE request

    This class defines the API call for the delete logging entries script. The class
consists of one method which
    accepts a DELETE request. For the DELETE request no parameters are required.

    """


##########################################################################################
##########################
    # Method for handling the DELETE request

##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation successful')
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def delete(self):
        """ DELETE definition for removing all logging entries

        <p style="text-align: justify">This method defines the handler for the DELETE
request of the delete logging
        entries script. It returns no message body and thus no contents. In contrast it
returns the HTTP status code
        204.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the DELETE request does not contain
any object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any
other response status code
        than 204, indicates an error during the execution.</p>

        """

        try:
            db_query = "UPDATE logging.logging SET deleted_at = NOW()"
            execute_database(db_query, (), database_config_file,
database_config_section_api, True)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_logging_entries')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_logging_entries')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, 'Successfully deleted all logging entries', 'API-
delete_logging_entries')
            return '', 204
```

5.1.104 services\backend_api\src\resources\resources_config\delete_queue_config\delete_queue_config.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Delete message queue configuration API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import config_namespace as api
from lib.auth_header import auth_header_parser
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


##########################################################################################
##############################
# Resource definition for the delete message queue configuration API call
##########################################################################################
##############################

@api.header('Content-Type', 'application/json')
class DeleteQueueConfiguration(Resource):
    """ Class for handling the DELETE request

    This class defines the API call for the delete message queue configuration script. The
class consists of one method
    which accepts a DELETE request. For the DELETE request no parameters are required.

    """


##########################################################################################
##########################
    # Method for handling the DELETE request

##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation successful')
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def delete(self):
        """ DELETE definition for removing the message queue configuration

        <p style="text-align: justify">This method defines the handler for the DELETE
request of the message delete
        queue configuration script. It returns no message body and thus no contents. In
contrast it returns the HTTP
        status code 204.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the DELETE request does not contain
any object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any
other response status code
        than 204, indicates an error during the execution.</p>

        """

        try:
            db_query = "UPDATE msgeovilleconfig.message_queue_config SET deleted_at =
NOW()"
            execute_database(db_query, (), database_config_file,
database_config_section_api, True)
```

```
        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_queue_config')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_queue_config')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Successfully deleted the queue configuration table',
'API-delete_queue_config')
            return '', 204
```

5.1.105 services\backend_api\src\resources\resources_config\delete_queue_config_by_id\delete_queue_config_by_id.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Delete message queue configuration by ID API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import config_namespace as api
from lib.auth_header import auth_header_parser
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


########################################################################################
##############################
# Resource definition for the delete queue configuration by ID API call
```

```
##########################################################################################
##############################

@api.header('Content-Type', 'application/json')
@api.param('service_id', 'Service ID to be deleted')
class DeleteQueueByID(Resource):
    """ Class for handling the DELETE request

    This class defines the API call for the delete queue configuration by ID script. The
class consists of one method
    which accepts a DELETE request. For the DELETE request one path parameter is required.

    """


##########################################################################################
##########################
    # Method for handling the DELETE request

##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation successful')
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def delete(self, service_id):
        """ DELETE definition for removing a message queue configuration by ID

        <p style="text-align: justify">This method defines the handler for the DELETE
request of the delete message
        queue configuration by ID script. It returns no message body and thus no contents.
In contrast it returns the
        HTTP status code 204.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameter:</b>
        <ul>
        <li><p><i>client_id (str): </i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the DELETE request does not contain
any object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any
other response status code
        than 204, indicates an error during the execution.</p>

        """

        try:
            gemslog(LogLevel.INFO, f'Request path parameters: {service_id}', 'API-
delete_queue_config_by_id')

            db_query = "UPDATE msgeovilleconfig.message_queue_config SET deleted_at =
NOW() WHERE service_id = %s"
```

```
            execute_database(db_query, (service_id,), database_config_file,
database_config_section_api, True)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_queue_config_by_id')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_queue_config_by_id')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Deleted queue config for: {service_id}', 'API-
delete_queue_config_by_id')
            return '', 204
```

5.1.106 services\backend_api\src\resources\resources_config\delete_queue_config_by_name\delete_queue_config_by_name.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Delete message queue configuration by name API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import config_namespace as api
from lib.auth_header import auth_header_parser
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback
```

```
##########################################################################################
##############################
# Resource definition for the delete queue configuration by name API call
##########################################################################################
##############################

@api.header('Content-Type', 'application/json')
@api.param('queue_name', 'Queue name to be deleted')
class DeleteQueueByName(Resource):
    """ Class for handling the DELETE request

    This class defines the API call for the delete queue configuration by name script. The
class consists of one method
    which accepts a DELETE request. For the DELETE request a JSON with one parameter is
required and defined in the
    corresponding model.

    """


##########################################################################################
##########################
    # Method for handling the DELETE request

##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation successful')
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def delete(self, queue_name):
        """ DELETE definition for removing a message queue configuration by name

        <p style="text-align: justify">This method defines the handler for the DELETE
request of the delete message
        queue configuration by name script. It returns no message body and thus no
contents. In contrast it returns the
        HTTP status code 204.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameter:</b>
        <ul>
        <li><p><i>client_id (str): </i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the DELETE request does not contain
any object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any
other response status code
        than 204, indicates an error during the execution.</p>

        """

        try:
            gemslog(LogLevel.INFO, f'Request path parameters: {queue_name}', 'API-
delete_queue_config_by_name')

            db_query = "UPDATE msgeovilleconfig.message_queue_config SET deleted_at =
NOW() WHERE queue_name = %s"
            execute_database(db_query, (queue_name,), database_config_file,
database_config_section_api, True)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_queue_config_by_name')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_queue_config_by_name')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Deleted queue config for: {queue_name}', 'API-
delete_queue_config_by_name')
            return '', 204
```

5.1.107 services\backend_api\src\resources\resources_config\get_airflow_config\get_airflow_config.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Retrieves the entire Airflow configuration from the database
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import read_from_database_all_rows
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import config_namespace as api
from lib.auth_header import auth_header_parser
from models.models_config.airflow_models.airflow_config_models import
airflow_config_list_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
```

```
import traceback


##########################################################################################
##############################
# Resource definition for the get Airflow configuration API call
##########################################################################################
##############################

@api.header('Content-Type', 'application/json')
class GetAirflowConfig(Resource):
    """ Class for handling the GET request

    This class defines the API call for the get Airflow configuration script. The class
consists of one method which
    accepts a GET request. For the GET request no additional parameters are required.

    """


##########################################################################################
##########################
    # Method for handling the GET request

##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Operation successful', airflow_config_list_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self):
        """ GET definition for retrieving the Airflow configuration

        <p style="text-align: justify">This method defines the handler for the GET request
of the get Airflow
        configuration script. It returns a message wrapped in a dictionary with the
configuration from the database.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify"></p>

        """

        try:
            db_query = """SELECT
                                service_name, command, description
                            FROM
                                msgeovilleconfig.airflow_config
                            WHERE
                                deleted_at IS NULL;
                       """

            conf_data = read_from_database_all_rows(db_query, (), database_config_file,
database_config_section_api, True)

            res_array = []
            for config in conf_data:
                config_obj = {
                    'service_name': config[0],
                    'command': config[1],
                    'description': config[2]
                }

                res_array.append(config_obj)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_airflow_config')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_airflow_config')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Request successful', 'API-get_airflow_config')
            return {'airflow_configuration': res_array}, 200
```

5.1.108 services\backend_api\src\resources\resources_config\get_airflow_config_by_name\get_airflow_config_by_name.py

```
###########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Retrieves the entire Airflow configuration from the database for a
# particular service name
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
###########################################################################################
##############################

from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import read_from_database_one_row
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import config_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_airflow_service_existence
from models.models_config.airflow_models.airflow_config_models import
add_airflow_config_model
```

```
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


##########################################################################################
##############################
# Resource definition for the get Airflow configuration by name API call
##########################################################################################
##############################

@api.header('Content-Type', 'application/json')
@api.param('service_name', 'Service name to be requested')
class GetAirflowConfigByServiceName(Resource):
    """ Class for handling the GET request

    This class defines the API call for the get Airflow configuration by name script. The
class consists of one method
    which accepts a GET request. For the GET request one additional path parameter is
required and defined in the
    resource definition

    """


##########################################################################################
##########################
    # Method for handling the GET request

##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(201, 'Operation successful', add_airflow_config_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self, service_name):
        """ GET definition for retrieving the Airflow configuration by name

        This method defines the handler for the GET request of the aget Airflow
configuration by name script. It returns
        a message wrapped into a dictionary with the configuration data from the database.

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameters:</b>
        <ul>
        <li><p><i>service_name (str): </i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify"></p>

        """
        try:
            gemslog(LogLevel.INFO, f'Request: {service_name}', 'API-
get_airflow_config_by_name')

            if not check_airflow_service_existence(service_name, database_config_file,
database_config_section_api):
                error = NotFoundError('The service name exists already', '', '')
                gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_airflow_config_by_name')
                return {'message': error.to_dict()}, 404

            db_query = """SELECT
                              service_name, command, description
                          FROM
                              msgeovilleconfig.airflow_config
                          WHERE
                              service_name = %s AND
                              deleted_at IS NULL
                       """

            conf_data = read_from_database_one_row(db_query, (service_name, ),
database_config_file,
                                                         database_config_section_api, True)

            config_obj = {
                'service_name': conf_data[0],
                'command': conf_data[1],
                'description': conf_data[2]
            }

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_airflow_config_by_name')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_airflow_config_by_name')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, 'Request successful', 'API-get_airflow_config_by_name')
            return config_obj, 200
```

5.1.109 services\backend_api\src\resources\resources_config\get_queue_config\get_queue_config.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Retrieves the entire message queue configuration from the database
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import read_from_database_all_rows
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import config_namespace as api
from lib.auth_header import auth_header_parser
from models.models_config.queue_config.queue_config_models import queue_config_list_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


########################################################################################
##############################
# Resource definition for the get message queue configuration API call
########################################################################################
##############################

@api.header('Content-Type', 'application/json')
class GetMessageQueueConfig(Resource):
    """ Class for handling the GET request

    This class defines the API call for the get message queue configuration script. The
class consists of one method
    which accepts a GET request. For the GET request no additional parameters are
required.

    """


########################################################################################
##########################
    # Method for handling the GET request

########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Operation successful', queue_config_list_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self):
        """ GET definition for retrieving the message queue configuration

        <p style="text-align: justify">This method defines the handler for the GET request
of the get message queue
        configuration script. It returns a message wrapped in a dictionary with the
configuration from the database.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>
        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify"></p>

        """

        try:
            db_query = """SELECT
                              service_id, queue_name, host, port
                          FROM
                              msgeovilleconfig.message_queue_config
                          WHERE
                              deleted_at IS NULL;
                       """

            conf_data = read_from_database_all_rows(db_query, (), database_config_file,
database_config_section_api, True)

            res_array = []

            for config in conf_data:
                config_obj = {
                    'service_id': config[0],
                    'queue_name': config[1],
                    'host': config[2],
                    'port': config[3]
                }

                res_array.append(config_obj)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_queue_config', )
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_queue_config', )
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Request successful', 'API-get_queue_config')
            return {'message_queue_configuration': res_array}, 200
```

5.1.110 services\backend_api\src\resources\resources_config\get_queue_config_by_id\get_queue_config_by_id.py

```
###########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
```

```
# Retrieves the entire message queue configuration by ID from the database
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import read_from_database_one_row
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file as db_file,
database_config_section_api as db_section
from init.namespace_constructor import config_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_service_existence
from models.models_config.queue_config.queue_config_models import add_queue_config_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


########################################################################################
##############################
# Resource definition for the get message queue configuration by ID API call
########################################################################################
##############################

@api.header('Content-Type', 'application/json')
@api.param('service_id', 'Service ID to be requested')
class GetMessageQueueConfigByID(Resource):
    """ Class for handling the GET request

    This class defines the API call for the get message queue configuration by ID script.
The class consists of one
    method which accepts a GET request. For the GET request no additional parameters are
required.

    """


########################################################################################
##########################
    # Method for handling the GET request

########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Operation successful', add_queue_config_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self, service_id):
        """ GET definition for retrieving the message queue configuration by ID

        <p style="text-align: justify">This method defines the handler for the GET request 
of the get queue
        configuration by ID script. It returns a message wrapped in a dictionary with the 
configuration from the
        database.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameters:</b>
        <ul>
        <li><p><i>service_name (str): </i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify"></p>

        """

        db_query = """SELECT
                          service_id, queue_name, host, port
                      FROM
                          msgeovilleconfig.message_queue_config
                      WHERE
                          service_id = %s AND
                          deleted_at IS NULL
                   """

        try:
            gemslog(LogLevel.INFO, f'Request path parameters: {service_id}', 'API-
get_queue_config_by_id')

            if not check_service_existence(service_id, db_file, db_section):
                error = NotFoundError('Service ID does not exist', '', '')
                gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_queue_config_by_id')
                return {'message': error.to_dict()}, 404

            conf_data = read_from_database_one_row(db_query, (service_id, ), db_file, 
db_section, True)

            config_obj = {
                'service_id': conf_data[0],
                'queue_name': conf_data[1],
                'host': conf_data[2],
                'port': conf_data[3]
            }

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server', 
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_queue_config_by_id')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload, 
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_queue_config_by_id')
```

```
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Request successful', 'API-get_queue_config_by_id')
            return config_obj, 200
```

5.1.111 services\backend_api\src\resources\resources_config\get_queue_config_by_name\get_queue_config_by_name.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Retrieves the entire message queue configuration by name from the database
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import read_from_database_one_row
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file as db_file,
database_config_section_api as db_section
from init.namespace_constructor import config_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_queue_existence
from models.models_config.queue_config.queue_config_models import add_queue_config_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


########################################################################################
##############################
# Resource definition for the get queue configuration by name API call
########################################################################################
##############################

@api.header('Content-Type', 'application/json')
@api.param('queue_name', 'Queue_name to be requested')
class GetMessageQueueConfigByName(Resource):
    """ Class for handling the GET request

    This class defines the API call for the get queue configuration by name script. The
class consists of one method
```

```
    which accepts a GET request. For the GET request no additional parameters are
required.

    """


########################################################################################
##########################
    # Method for handling the GET request

########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Operation successful', add_queue_config_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self, queue_name):
        """ GET definition for retrieving the message queue configuration by name

        <p style="text-align: justify">This method defines the handler for the GET request
of the get queue
        configuration by name script. It returns a message wrapped in a dictionary with
the configuration from the
        database.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameters:</b>
        <ul>
        <li><p><i>service_name (str): </i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify"></p>

        """

        db_query = """SELECT
                          service_id, queue_name, host, port
                      FROM
                          msgeovilleconfig.message_queue_config
                      WHERE
                          queue_name = %s AND
                          deleted_at IS NULL
                   """

        try:
            gemslog(LogLevel.INFO, f'Request path parameters: {queue_name}', 'API-
get_queue_config_by_name')

            if not check_queue_existence(queue_name, db_file, db_section):
                error = NotFoundError('Queue name does not exist', '', '')
                gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_queue_config_by_name')
                return {'message': error.to_dict()}, 404
            conf_data = read_from_database_one_row(db_query, (queue_name, ), db_file,
db_section, True)

            config_obj = {
                'service_id': conf_data[0],
                'queue_name': conf_data[1],
                'host': conf_data[2],
                'port': conf_data[3]
            }

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_queue_config_by_name')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_queue_config_by_name')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Request successful', 'API-get_queue_config_by_name')
            return config_obj, 200
```

5.1.112 services\backend_api\src\resources\resources_crm\create_customer\create_customer.py

```
###########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Create customer API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
###########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import crm_namespace as api
from lib.auth_header import auth_header_parser
from lib.general_helper_methods import generate_bcrypt_hash
from lib.request_helper import create_oauth_client
from models.models_crm.customer_models.customer_models import customer_model,
customer_creation_response_model
```

```
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_408 import error_408_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


##########################################################################################
##############################
# Resource definition for the create customer API call
##########################################################################################
##############################

@api.expect(customer_model)
@api.header('Content-Type', 'application/json')
class CreateCustomer(Resource):
    """ Class for handling the POST request

    This class defines the API call for the create customer script. The class consists of
one method which accepts a
    POST request. For the POST request a JSON with several parameters is required and
defined in the corresponding
    model.

    """


##########################################################################################
##########################
    # Method for handling the POST request

##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(201, 'Operation successful', customer_creation_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(408, 'Request Timeout', error_408_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for creating a new user

        <p style="text-align: justify">This method defines the handler for the POST
request of the create user script.
        It returns a message wrapped into a dictionary containing the user-specific client
ID and secret.</p>

        <br><b>Description:</b>
        <p style="text-align: justify">This service creates a GEMS user in the GEMS
customer database and in the OAuth2
        database. First a POST request to the OAuth2 server will be send with the full
name of the client to be created.
        A new OAuth2 user is created and the OAuth2 client ID and client secret is
returned. The client ID will be
        further used as primary key for the customer table. The new customer with all
submitted request parameters will
        be created and the client ID and client secret returned.</p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>
        <br><b>Request payload:</b>
        <ul>
        <li><p><i>title (str): </i></p></li>
        <li><p><i>first_name (str): </i></p></li>
        <li><p><i>last_name (str): </i></p></li>
        <li><p><i>email (str): </i></p></li>
        <li><p><i>password (str): </i></p></li>
        <li><p><i>address (str): </i></p></li>
        <li><p><i>city (str): </i></p></li>
        <li><p><i>zip_code (str): </i></p></li>
        <li><p><i>country (str): </i></p></li>
        <li><p><i>nationality (str): optional</i></p></li>
        <li><p><i>phone_number (str): optional</i></p></li>
        <li><p><i>company_name (str): </i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the GET request is a JSON that
contains 2 key value pairs. The
        client ID is a unique identifier of a client in the OAuth2 database and in the
GEMS database for GEMS user. The
        client secret could be seen as a password for the GEMS user, used only for the
Bearer token generation.</p>
        <ul>
        <li><p><i>client_id (str): Unique client ID of a GEMS customer for
authentication</i></p></li>
        <li><p><i>client_secret (str): Unique client secret of a GEMS customer for
authentication</i></p></li>
        </ul>

        """

        try:
            req_args = api.payload
            gemslog(LogLevel.INFO, f'Request payload: {req_args}', 'API-create_customer')

            request_response = create_oauth_client(f"{req_args['first_name']}
{req_args['last_name']}")

            if request_response[0] is None:
                return request_response[1], request_response[2]

            nationality = None if 'phone' not in req_args else req_args['nationality']
            phone = None if 'phone' not in req_args else req_args['phone']

            db_query = """INSERT INTO customer.customer
                          (
                              customer_id, title, first_name, last_name, email, password,
address, city, zip_code,
                              country, nationality, phone_number, company_name
                          )
                          VALUES
                          (
                              %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s
                          );
                       """

            execute_database(db_query,
                             (request_response[0]['client_id'], req_args['title'],
req_args['first_name'],
                              req_args['last_name'], req_args['email'],
generate_bcrypt_hash(req_args['password']),
                              req_args['address'], req_args['city'], req_args['zip_code'],
req_args['country'],
                              nationality, phone, req_args['company_name']),
                             database_config_file, database_config_section_api, True)
        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
create_customer')
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
create_customer')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
create_customer')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f"Successfully created new customer
{req_args['last_name']}", 'API-create_customer')
            return {
                       'client_id': request_response[0]['client_id'],
                       'client_secret': request_response[0]['client_secret']
                   }, 201
```

5.1.113 services\backend_api\src\resources\resources_crm\create_manual_task\create_manual_task.py

```
###########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Create customer API call
#
# Date created: 22.02.2021
# Date last modified: 23.02.2021
#
# __author__  = Patrick Wolf (wolf@geoville.com)
# __version__ = 21.02
#
###########################################################################################
##############################

import datetime
from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_422.http_error_422 import UnprocessableEntityError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database,
execute_values_database
# from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import crm_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_user_existence
from lib.database_helper import check_processing_unit_exists
from lib.database_helper import get_processing_units_for_spu
from lib.database_helper import check_sup_exists
from lib.database_helper import check_order_id_required
from lib.database_helper import get_order_id_for_tasks
from lib.database_helper import check_production_unit_already_inserted
from models.models_crm.manual_tasks_models.manual_tasks_models import
manual_task_request_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_422 import error_422_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import psycopg2
import traceback


#########################################################################################
##############################
# Resource definition for the create customer API call
#########################################################################################
##############################

@api.expect(manual_task_request_model)
@api.header('Content-Type', 'application/json')
class CreateManualTask(Resource):
    """ Class for handling the POST request

    This class defines the API call for the create manual task script. The class consists
of one method which accepts a
    POST request. For the POST request a JSON with several parameters is required and
defined in the corresponding
    model.

    """


#########################################################################################
##########################
    # Method for handling the POST request

#########################################################################################
##########################

    @require_oauth(['admin', 'user'])
    @api.expect(auth_header_parser)
    @api.response(201, 'Operation successful', manual_task_request_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(422, 'Unprocessable Entity', error_422_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for creating a new user

        <p style="text-align: justify">This method defines the handler for the POST
request of the create manual task
        script.</p>

        <br><b>Description:</b>
        <p style="text-align: justify">This service adds a new manual task entry to the
database. In addition to the
        parameters which define a manual task, the service requires a client-id and an
access token. As a result the
        service responses the HTTP code 201 combined with the information to further
access the manual task entity. If
```

```
        an error occurs, the service returns on of the appropriate error status codes
(400, 401, 403, 404, 500, 503)</p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>processing_unit (list): List with unique processing units)</i></p></li>
        <li><p><i>subproduction_unit (str): Unique subprocessing unit
identifier</i></p></li>
        <li><p><i>task_id (str): Unique task identifier</i></p></li>
        <li><p><i>service_id (str): Unique service identifier</i></p></li>
        <li><p><i>comment (str): optional</i></p></li>
        <li><p><i>client_id (str): User client-id</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">SUCCESS: If the entry was added successfully to the
database, the service returns
        the HTTP code 201 as well an a dictionary containing two keys (message, links).
The value links holds a list of
        all inserted processing_units.</p>
        <p style="text-align: justify">ERROR: In case of an error, the appropriate HTTP
error code is returned (400, 401, 403,
        404, 500, 503). Additionally, the service returns a dictionary with two keys
(message, error_definition)</p>

        """

        # Get request parameters and check if the payload parameter names are correct
        try:
            req_args = api.payload

            request_processing_unit = None if "processing_unit" not in req_args else
req_args['processing_unit']
            request_subproduction_unit = None if "subproduction_unit" not in req_args else
req_args[
                'subproduction_unit']

            request_client_id = req_args['client_id']
            # request_service_id = req_args['service_id']
            # request_task_id = req_args['task_id']

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            return {'message': error.to_dict()}, 400

        try:
            # Check if user exists
            if not check_user_existence(request_client_id, database_config_file,
database_config_section_api):
                error = UnprocessableEntityError('User ID does not exist', '', '')
                return {'message': error.to_dict()}, 422

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            return {'message': error.to_dict()}, 500

        if request_processing_unit is not None and request_subproduction_unit is not None:
            error = UnprocessableEntityError('Multiple region of interest types are 
provided. Only one '
                                             'type (processing_unit or subproduction_unit) 
is supported. ', '', '')
            return {'message': error.to_dict()}, 422

        if request_processing_unit is None and request_subproduction_unit is None:
            error = UnprocessableEntityError('Region of interest (processing_unit, 
subproduction_unit) is missing', '',
                                             '')
            return {'message': error.to_dict()}, 422

        if request_processing_unit is not None and request_subproduction_unit is None:
            response_dictionary, error_code = 
self.submit_manual_task_preprocessing_unit(req_args)

        if request_subproduction_unit is not None and request_processing_unit is None:
            response_dictionary, error_code = self.submit_manual_task_spu(req_args)

        return response_dictionary, error_code

    @staticmethod
    def submit_manual_task_preprocessing_unit(req_args):
        """
        This method stores a list of manual tasks based on preprocessing units into the 
database.
        @param req_args: Request arguments dictionary
        @return: Return a tuple containing the response dictionary and the response code
        """

        try:
            request_processing_unit = req_args['processing_unit']
            request_task_id = req_args['task_id']
            request_service_id = req_args['service_id']
            request_client_id = req_args['client_id']
            # request_status = req_args['status']

            request_comment = None if "comment" not in req_args else req_args['comment']
            # request_result = None if "result" not in req_args else req_args['result']

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}', 
api.payload, traceback.format_exc())
            return {'message': error.to_dict()}, 400

        try:
            # Firstly, check if the value for request_processing_unit is from type 'list'
            if not isinstance(request_processing_unit, list):
                error = UnprocessableEntityError('Processing unit value is not from type 
list'.format(
                    request_processing_unit), '', '')
                return {'message': error.to_dict()}, 422

            # Unique set
            request_processing_unit = list(set(request_processing_unit))

            # Secondly, check if all elements in the list are correct
            incorrect_processing_units = []
            for elem in request_processing_unit:
                processing_unit_exists = check_processing_unit_exists(elem, 
database_config_file,

database_config_section_api)
                if not processing_unit_exists:
                    incorrect_processing_units.append(elem)

            if len(incorrect_processing_units) >= 1:
                error = UnprocessableEntityError('Processing units ({0}) do not exist in
database'.format(
                    str(incorrect_processing_units)), '', '')
                return {'message': error.to_dict()}, 422

            # Check if one of the submitted production units was already inserted
            # pu_already_inserted, list_of_pus =
check_production_unit_already_inserted(request_processing_unit,
            #
database_config_file,
            #
database_config_section_api)
            # if pu_already_inserted:
            #     error = UnprocessableEntityError('Production units already exists in the
manual tasks table. List of'
            #                                       'affected units:
{0}'.format(list_of_pus), '', '')
            #     return {'message': error.to_dict()}, 422

            # Check if order id is required for the task
            is_order_id_required = check_order_id_required(request_task_id,
database_config_file,
                                                           database_config_section_api)

            print("is required: ", is_order_id_required)
            if is_order_id_required:
                # If the order id is required, check ud the order-is is valid. Therefore,
get the order-ids for the
                # processing units. If one order-is is missing, an error should be
returned

                order_ids_ok, v = get_order_id_for_tasks(request_processing_unit,
request_service_id, request_task_id,
                                                         database_config_file,
database_config_section_api)
                if not order_ids_ok:
                    incorrect_units = v
                    error = UnprocessableEntityError('The order-id for one or multiple
processing units does not exist '
                                                     'in the database - affected units:
{0}'
                                                     .format(incorrect_units), '', '')
                    return {'message': error.to_dict()}, 422
                else:
                    order_ids = v

            # Insert the manual task into the database
            for element in request_processing_unit:
                task_started = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                state_running = "RUNNING"

                if is_order_id_required:
                    refers_to_order_id = order_ids[element]
                else:
                    refers_to_order_id = None

                print(refers_to_order_id)

                values = (
                    element, request_client_id, request_service_id, request_task_id,
task_started, None,
                    state_running, None, None, refers_to_order_id, request_comment
                )

                sql = """
                    INSERT INTO customer.manual_tasks
                    (
                        cell_code, customer_id, service_id, task_id, task_started,
task_stopped, status,  "result",
                        deleted_at, refers_to_order_id, "comment"
                    )
                    VALUES
                    (
                        %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s
                    );
                """

                try:
                    execute_database(sql, values, database_config_file,
database_config_section_api, True)
                except Exception:
                    error = InternalServerErrorAPI(f'Unexpected error occurred for unit
{element}',
                                                   api.payload,
                                                   traceback.format_exc())
                    return {'message': error.to_dict()}, 500

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            return {'message': error.to_dict()}, 500

        # Create final response
        response_links = []
        for element in request_processing_unit:
            d = {
                'href': f'/crm/manual_tasks/task_query?processing_unit={element}',
                'rel': 'manual_tasks',
                'type': 'GET'
            }
            response_links.append(d)

        response = {
            'message': 'Your order has been successfully submitted',
            'links': response_links
        }

        return response, 200

    @staticmethod
    def submit_manual_task_spu(req_args):
        """
        This method stores a manual tasks based on a subproduction unit into the database.
        @param req_args: Request arguments dictionary
        @return: Return a tuple containing the response dictionary and the response code
        """

        # Get the request parameters and set default values for missing values
        try:
            request_subproduction_units = req_args['subproduction_unit']
            request_task_id = req_args['task_id']
            request_service_id = req_args['service_id']
            request_client_id = req_args['client_id']

            request_comment = None if "comment" not in req_args else req_args['comment']

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            return {'message': error.to_dict()}, 400
        try:

            for request_subproduction_unit in request_subproduction_units:
                # Check if sup exists and retrieve a list of all processing units for the
subproduction unit
                sup_exists = check_sup_exists(request_subproduction_unit,
database_config_file,
                                              database_config_section_api)

                if not sup_exists:
                    error = UnprocessableEntityError('Subproduction unit ({0}) does not
exist in database'.format(
                        request_subproduction_unit), '', '')
                    return {'message': error.to_dict()}, 422

                # Get a list of all processing units for the subproduction unit
                processing_units =
get_processing_units_for_spu(request_subproduction_unit, database_config_file,

database_config_section_api)

                # Unique set
                processing_units = list(set(processing_units))

                # Check if one of the submitted production units was already inserted
                # pu_already_inserted, list_of_pus =
check_production_unit_already_inserted(processing_units,
                #
database_config_file,
                #
database_config_section_api)
                # if pu_already_inserted:
                #     error = UnprocessableEntityError('Production units already exists in
the manual tasks table. List of'
                #                                      'affected units:
{0}'.format(list_of_pus), '', '')
                #     return {'message': error.to_dict()}, 422

                # Check if order-id is required for the task
                is_order_id_required = check_order_id_required(request_task_id,
database_config_file,

database_config_section_api)
                if is_order_id_required:
                    # If the order id is required, check if the order-is is valid.
Therefore, get the order-ids for the
                    # processing units. If one order-is is missing, an error should be
returned

                    order_ids_ok, v = get_order_id_for_tasks(processing_units,
request_service_id, request_task_id,
                                                             database_config_file,
database_config_section_api)
                    if not order_ids_ok:
                        incorrect_units = v
                        error = UnprocessableEntityError('The order-id for one or multiple
processing units does not exist '
                                                         'in the database - affected unis:
{0}'
                                                         .format(incorrect_units), '', '')
                        return {'message': error.to_dict()}, 422
                    else:
                        order_ids = v

                # Insert the manual task (based on processing_unit) into the database
                for element in processing_units:
                    task_started = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                    state_running = "RUNNING"

                    if is_order_id_required:
                        refers_to_order_id = order_ids[element]
                    else:
                        refers_to_order_id = None

                    values = (
                        element, request_client_id, request_service_id, request_task_id,
task_started,
                        state_running, refers_to_order_id, request_comment
                    )

                    sql = """
                        INSERT INTO customer.manual_tasks
                        (
                            cell_code, customer_id, service_id, task_id, task_started,
status,
                            refers_to_order_id, comment
                        )
                        VALUES
                        (%s, %s,%s,%s,%s,%s,%s, %s)
                    """

                    try:
                        execute_database(sql, values, database_config_file,
database_config_section_api, True)

                    except psycopg2.errors.UniqueViolation as err:
                        error = BadRequestError(
                            f'Unique Key constraint {element}: {err}',
                            api.payload,
                            traceback.format_exc())
                        return {'message': error.to_dict()}, 400

                    except Exception as err:
                        error = InternalServerErrorAPI(
                            f'Unexpected error occurred for unit {element}: {err}',
                            api.payload,
                            traceback.format_exc())
                        return {'message': error.to_dict()}, 500

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            return {'message': error.to_dict()}, 503

        except Exception as err:
            error = InternalServerErrorAPI(f'Unexpected error occurred: {err}',
api.payload, traceback.format_exc())
            return {'message': error.to_dict()}, 500

        # Create final response
        response_links = []
        for element in processing_units:
            d = {
                'href': f'/crm/manual_tasks/task_query?processing_unit={element}',
                'rel': 'manual_tasks',
                'type': 'GET'
            }
            response_links.append(d)

        response = {
            'message': 'Your order has been successfully submitted',
            'links': response_links
        }

        return response, 200
```

5.1.114 services\backend_api\src\resources\resources_crm\create_service\create_service.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Create service API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import crm_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_service_name_existence
from lib.hashing_helper import generate_service_id_hash
from models.models_crm.service_models.service_models import service_creation_model,
service_id_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


##########################################################################################
##############################
# Resource definition for the create customer API call
##########################################################################################
##############################

@api.expect(service_creation_model)
@api.header('Content-Type', 'application/json')
class CreateService(Resource):
    """ Class for handling the POST request

    This class defines the API call for the create customer script. The class consists of
one method which accepts a
    POST request. For the POST request a JSON with several parameters is required and
defined in the corresponding model.

    """
```

```
##########################################################################################
##########################
    # Method for handling the POST request

##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(201, 'Operation successful', service_id_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for creating a new service

        <p style="text-align: justify">This method defines the handler for the POST
request of the create service
        script. It returns a message wrapped into a dictionary about the process of the
insertion operation.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>service_name (str): </i></p></li>
        <li><p><i>service_validity (str): </i></p></li>
        <li><p><i>service_owner_geoville (str): </i></p></li>
        <li><p><i>service_comment (str): </i></p></li>
        <li><p><i>external (str): </i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify"></p>

        """

        try:
            req_args = api.payload
            gemslog(LogLevel.INFO, f'Request payload: {req_args}', 'API-create_service')

            if check_service_name_existence(req_args['service_name'],
database_config_file, database_config_section_api):
                error = BadRequestError('Service name exists already', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
create_service')
                return {'message': error.to_dict()}, 400

            service_comment = None if 'service_comment' not in req_args else
req_args['service_comment']
            service_id = generate_service_id_hash(req_args['service_name'],
req_args['service_owner_geoville'])

            db_query = """INSERT INTO customer.services
                          (
                            service_id, service_name, service_validity,
service_owner_geoville, service_comment, external
                          )
                          VALUES
```

```
                        (
                          %s, %s, %s, %s, %s, %s
                        );
                     """

            execute_database(db_query, (service_id, req_args['service_name'],
req_args['service_validity'],
                                        req_args['service_owner_geoville'],
service_comment, req_args['external']),
                             database_config_file, database_config_section_api, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
create_service')
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-create_service')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-create_service')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, 'Successfully created new service', 'API-
create_service')
            return {'service_id': service_id}, 201
```

5.1.115 services\backend_api\src\resources\resources_crm\delete_all_customers\delete_all_customers.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Delete all customers API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api,
database_config_section_oauth
```

```
from init.namespace_constructor import crm_namespace as api
from lib.auth_header import auth_header_parser
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


############################################################################################
##############################
# Resource  definition for the delete all customers API call
############################################################################################
##############################

@api.header('Content-Type', 'application/json')
class DeleteAllCustomers(Resource):
    """ Class for handling the DELETE request

    This class defines the API call for the delete all customers script. The class
consists of one method which accepts
    a DELETE request. For the DELETE request a JSON no additional parameters are required.

    """


############################################################################################
##########################
    # Method for handling the DELETE request

############################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation successful')
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def delete(self):
        """ DELETE definition for removing all customers

        <p style="text-align: justify">This method defines the handler for the DELETE
request of the delete all
        customers script. It returns no message body and thus no contents. In contrast it
returns the HTTP status
        code 204.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameter:</b>
        <ul>
        <li><p><i>client_id (str): </i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the DELETE request does not contain
any object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any
other response status code
```

```
        than 204, indicates an error during the execution.</p>

        """

        db_query_api = "UPDATE customer.customer SET deleted_at = NOW()"
        db_query_oauth = "UPDATE public.oauth2_client SET deleted_at = NOW()"
        db_query_token = "UPDATE public.oauth2_token SET deleted_at = NOW()"

        try:
            execute_database(db_query_api, (), database_config_file,
database_config_section_api, True)
            execute_database(db_query_oauth, (), database_config_file,
database_config_section_oauth, True)
            execute_database(db_query_token, (), database_config_file,
database_config_section_oauth, True)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database servers',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_customers')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_customers')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, 'Successfully deleted the entire customer table', 'API-
delete_customers')
            return '', 204
```

5.1.116 services\backend_api\src\resources\resources_crm\delete_all_services\delete_all_services.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Delete services API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import crm_namespace as api
from lib.auth_header import auth_header_parser
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


##########################################################################################
##############################
# Resource definition for the delete Airflow configuration API call
##########################################################################################
##############################

@api.header('Content-Type', 'application/json')
class DeleteServices(Resource):
    """ Class for handling the DELETE request

    This class defines the API call for the delete services script. The class consists of
one method which accepts a
    DELETE request. For the DELETE request no parameters are required.

    """


##########################################################################################
##########################
    # Method for handling the DELETE request

##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation successful')
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def delete(self):
        """ DELETE definition for removing all services

        <p style="text-align: justify">This method defines the handler for the DELETE
request of the delete services
        script. It returns no message body and thus no contents. In contrast it returns
the HTTP status code 204.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameter:</b>
        <ul>
        <li><p><i>client_id (str): </i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the DELETE request does not contain
any object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any
other response status code
        than 204, indicates an error during the execution.</p>
        """

        try:
            db_query = "UPDATE customer.services SET deleted_at = NOW()"
            execute_database(db_query, (), database_config_file,
database_config_section_api, True)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_services')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_services')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, 'Successfully deleted the entire service table', 'API-
delete_services')
            return '', 204
```

5.1.117 services\backend_api\src\resources\resources_crm\delete_customers_by_filter\delete_customers_by_filter.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Delete customer by filter API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database,
read_from_database_all_rows
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api,
database_config_section_oauth
from init.namespace_constructor import crm_namespace as api
from lib.auth_header import auth_header_parser
from lib.general_helper_methods import parameter_and_value_list_generation
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback

##########################################################################################
##############################
# Query parameter definition of the GET request
##########################################################################################
##############################

query_param_parser = auth_header_parser.copy()
query_param_parser.add_argument('title', location='args', type=str, help='Title',
trim=True)
query_param_parser.add_argument('first_name', location='args', type=str, help='First
name')
query_param_parser.add_argument('last_name', location='args', type=str, help='Last name')
query_param_parser.add_argument('email', location='args', type=str, help='E-mail address')
query_param_parser.add_argument('address', location='args', type=str, help='Address 1')
query_param_parser.add_argument('zip_code', location='args', type=str, help='Zip Code')
query_param_parser.add_argument('city', location='args', type=str, help='City')
query_param_parser.add_argument('country', location='args', type=str, help='Country')
query_param_parser.add_argument('nationality', location='args', type=str,
help='Nationality')
query_param_parser.add_argument('phone', location='args', type=str, help='Phone number')
query_param_parser.add_argument('company_name', location='args', type=str, help='Company
name')


##########################################################################################
##############################
# Resource definition for the delete customer by filter API call
##########################################################################################
##############################

@api.header('Content-Type', 'application/json')
class DeleteCustomersByFilter(Resource):
    """ Class for handling the DELETE request

    This class defines the API call for the delete customer by filter script. The class
consists of one method which
    accepts a DELETE request. For the DELETE request a JSON with several additional
parameter is required, defined
    in the corresponding request model.

    """


##########################################################################################
##########################
    # Method for handling the DELETE request

##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(query_param_parser)
    @api.response(204, 'Operation successful')
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def delete(self):
        """ DELETE definition for removing customers by filter

        <p style="text-align: justify">This method defines the handler for the DELETE
request of the delete all
        customers by filter script. It returns no message body and thus no contents. In
contrast it returns the HTTP
```

```
        status code 204. If no filter option is specified all customers will be
deleted.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameter:</b>
        <ul>
        <li><p><i>client_id (str): </i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the DELETE request does not contain
any object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any
other response status code
        than 204, indicates an error during the execution.</p>

        """

        try:
            req_args = query_param_parser.parse_args()
            gemslog(LogLevel.INFO, f'Request payload: {req_args}', 'API-
delete_customer_filter')

            param_list, val_list = parameter_and_value_list_generation(req_args)

            if not val_list and not param_list:
                db_query_api = "UPDATE customer.customer SET deleted_at = NOW()"
                db_query_oauth = "UPDATE public.oauth2_client SET deleted_at = NOW()"
                db_query_token = "UPDATE public.oauth2_token SET deleted_at = NOW()"

                execute_database(db_query_api, val_list, database_config_file,
database_config_section_api, True)
                execute_database(db_query_oauth, val_list, database_config_file,
database_config_section_oauth, True)
                execute_database(db_query_token, val_list, database_config_file,
database_config_section_oauth, True)

            else:
                db_query_api = f"UPDATE customer.customer SET deleted_at = NOW() WHERE
{'and '.join(param_list)} RETURNING user_id"
                db_query_oauth = f"UPDATE public.oauth2_client SET deleted_at = NOW()
WHERE client_id = %s"
                db_query_token = f"UPDATE public.oauth2_token SET deleted_at = NOW() WHERE
client_id = %s"

                returned_client_ids = read_from_database_all_rows(db_query_api, val_list,
database_config_file,

database_config_section_api, True)

                if returned_client_ids is not None or returned_client_ids is not False:
                    for client_tuple in returned_client_ids:
                        execute_database(db_query_oauth, (client_tuple[0], ),
database_config_file,
                                         database_config_section_oauth, True)
                        execute_database(db_query_token, (client_tuple[0],),
database_config_file,
                                         database_config_section_oauth, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
```

```
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
delete_customer_filter')
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_customer_filter')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_customer_filter')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, 'The filtered list of customer data has been deleted',
'API-delete_customer_filter')
            return '', 204
```

5.1.118 services\backend_api\src\resources\resources_crm\delete_customer_by_id\delete_customer_by_id.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Delete customer by ID API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api,
database_config_section_oauth
from init.namespace_constructor import crm_namespace as api
from lib.auth_header import auth_header_parser
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback
```

```
##########################################################################################
##############################
# Resource definition for the delete customer by ID API call
##########################################################################################
##############################

@api.header('Content-Type', 'application/json')
@api.param('user_id', 'User ID to be deleted')
class DeleteCustomerById(Resource):
    """ Class for handling the DELETE request

    This class defines the API call for the delete customer by ID script. The class
consists of one method which accepts
    a DELETE request. For the DELETE request a JSON with one additional parameter is
required.

    """


##########################################################################################
##########################
    # Method for handling the DELETE request

##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation successful')
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def delete(self, user_id):
        """ DELETE definition for removing a customer by ID

        <p style="text-align: justify">This method defines the handler for the DELETE
request of the delete all
        customers by ID script. It returns no message body and thus no contents. In
contrast it returns the HTTP status
        code 204.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameter:</b>
        <ul>
        <li><p><i>client_id (str): </i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the DELETE request does not contain
any object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any
other response status code
        than 204, indicates an error during the execution.</p>

        """
        db_query_api = "UPDATE customer.customer SET deleted_at = NOW() WHERE customer_id
= %s"
        db_query_oauth = "UPDATE public.oauth2_client SET deleted_at = NOW() WHERE
client_id = %s"
        db_query_token = "UPDATE public.oauth2_token SET deleted_at = NOW() WHERE
client_id = %s"

        try:
            gemslog(LogLevel.INFO, f'Request path parameter: {user_id}', 'API-
delete_customer_by_id')

            execute_database(db_query_api, (user_id,), database_config_file,
database_config_section_api, True)
            execute_database(db_query_oauth, (user_id,), database_config_file,
database_config_section_oauth, True)
            execute_database(db_query_token, (user_id,), database_config_file,
database_config_section_oauth, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}', '',
traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
delete_customer_by_id')
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_customer_by_id')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_customer_by_id')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'User ID {user_id} has been deleted', 'API-
delete_customer_by_id')
            return '', 204
```

5.1.119 services\backend_api\src\resources\resources_crm\delete_service_by_id\delete_service_by_id.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Delete service by ID API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################
```

```
from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import crm_namespace as api
from lib.auth_header import auth_header_parser
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


###########################################################################################
##############################
# Resource definition for the delete service by ID API call
###########################################################################################
##############################

@api.header('Content-Type', 'application/json')
@api.param('service_id', 'Service ID to be deleted')
class DeleteServiceById(Resource):
    """ Class for handling the DELETE request

    This class defines the API call for the delete service by ID script. The class
consists of one method which accepts
    a DELETE request. For the DELETE request a JSON with one additional parameter is
required.

    """


###########################################################################################
##########################
    # Method for handling the DELETE request

###########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation successful')
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def delete(self, service_id):
        """ DELETE definition for removing a service by ID

        <p style="text-align: justify">This method defines the handler for the DELETE
request of the delete all service
        by ID script. It returns no message body and thus no contents. In contrast it
returns the HTTP status code 204.
        </p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameter:</b>
        <ul>
        <li><p><i>client_id (str): </i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the DELETE request does not contain
any object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any
other response status code
        than 204, indicates an error during the execution.</p>

        """

        try:
            gemslog(LogLevel.INFO, f'Request path parameter: {service_id}', 'API-
delete_service_by_id')

            db_query_api = "Update customer.services SET deleted_at = NOW() WHERE
service_id = %s"
            execute_database(db_query_api, (service_id,), database_config_file,
database_config_section_api, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}', '',
traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
delete_service_by_id')
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_service_by_id')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', '',
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_service_by_id')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Service ID {service_id} has been deleted', 'API-
delete_service_by_id')
            return '', 204
```

5.1.120 services\backend_api\src\resources\resources_crm\delete_service_by_name\delete_service_by_name.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Delete service by name API call
#
```

```
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import crm_namespace as api
from lib.auth_header import auth_header_parser
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


##########################################################################################
##############################
# Resource definition for the delete service by ID API call
##########################################################################################
##############################

@api.header('Content-Type', 'application/json')
class DeleteServiceByName(Resource):
    """ Class for handling the DELETE request

    This class defines the API call for the delete service by name script. The class
consists of one method which
    accepts a DELETE request. For the DELETE request a JSON with one additional parameter
is required.

    """


##########################################################################################
##########################
    # Method for handling the DELETE request

##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation successful')
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def delete(self, service_name):
        """ DELETE definition for removing a service by name

        <p style="text-align: justify">This method defines the handler for the DELETE
request of the delete all service
        by name script. It returns no message body and thus no contents. In contrast it
returns the HTTP status code
        204.</p>
        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameter:</b>
        <ul>
        <li><p><i>client_id (str): </i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the DELETE request does not contain
any object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any
other response status code
        than 204, indicates an error during the execution.</p>

        """

        try:
            gemslog(LogLevel.INFO, f'Request path parameter: {service_name}', 'API-
delete_service_name')

            db_query_api = "UPDATE customer.services SET deleted_at = NOW() WHERE
service_name = %s"
            execute_database(db_query_api, (service_name,), database_config_file,
database_config_section_api, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}', '',
traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
delete_service_name')
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_service_name')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', '',
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_service_name')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f"Service name '{service_name}' has been deleted",
'API-delete_service_name')
            return '', 204
```

5.1.121 services\backend_api\src\resources\resources_crm\get_all_customers\get_all_customers.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
```

```
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Get all customers API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import read_from_database_all_rows
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import crm_namespace as api
from lib.auth_header import auth_header_parser
from models.models_crm.customer_models.customer_models import customer_list_response_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


########################################################################################
##############################
# Resources definition for the get all customers API call
########################################################################################
##############################

@api.header('Content-Type', 'application/json')
class GetAllCustomers(Resource):
    """ Class for handling the GET request

    This class defines the API call for the get all customers script. The class consists
of one method which accepts a
    GET request. For the GET request no additional parameter are required.

    """


########################################################################################
##########################
    # Method for handling the GET request

########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Operation was successful', customer_list_response_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self):
        """ GET definition for retrieving all customers
        <p style="text-align: justify">This method defines the handler for the GET request
of the get all customers
        script. It returns all customer data stored in the database wrapped into a
dictionary defined by corresponding
        model.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify"></p>

        """

        db_query = """SELECT
                          title, first_name, last_name, email, address, city, zip_code,
country, nationality,
                          phone_number, company_name
                      FROM
                          customer.customer
                      WHERE
                          deleted_at IS NULL
                   """

        try:
            customer_data = read_from_database_all_rows(db_query, (),
database_config_file, database_config_section_api,
                                                          True)
            res_array = []

            for customer in customer_data:
                customer_obj = {
                    'title': customer[0],
                    'first_name': customer[1],
                    'last_name': customer[2],
                    'email': customer[3],
                    'address': customer[4],
                    'city': customer[5],
                    'zip_code': customer[6],
                    'country': customer[7],
                    'nationality': customer[8],
                    'phone_number': customer[9],
                    'company_name': customer[10]
                }

                res_array.append(customer_obj)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-get_customers')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-get_customers')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Successful response', 'API-get_customers')
            return {'customers': res_array}, 200
```

5.1.122 services\backend_api\src\resources\resources_crm\get_all_services\get_all_services.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Get all services API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import read_from_database_all_rows
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import crm_namespace as api
from lib.auth_header import auth_header_parser
from models.models_crm.service_models.service_models import service_list_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


########################################################################################
##############################
# Resources definition for the get all services API call
########################################################################################
##############################

@api.header('Content-Type', 'application/json')
class GetAllServices(Resource):
    """ Class for handling the GET request

    This class defines the API call for the get all services script. The class consists of
one method which accepts a
    GET request. For the GET request no additional parameter are required.

    """


########################################################################################
##########################
    # Method for handling the GET request

########################################################################################
##########################

    @require_oauth(['admin', 'user'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Operation was successful', service_list_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self):
        """ GET definition for retrieving all services

        <p style="text-align: justify">This method defines the handler for the GET request
of the get all services
        script. It returns all service data stored in the database wrapped into a
dictionary defined by corresponding
        model.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify"></p>

        """

        db_query = """SELECT
                          service_id, service_name, service_comment, service_validity,
service_owner_geoville,
                          external, created_at
                      FROM
                          customer.services
                      WHERE
                          deleted_at IS NULL and visible_frontend = true
                   """

        try:
            service_data = read_from_database_all_rows(db_query, (), database_config_file,
database_config_section_api,
                                                       True)
            res_array = []

            for service in service_data:
                service_obj = {
                    'service_id': service[0],
                    'service_name': service[1],
                    'service_comment': service[2],
                    'service_validity': service[3],
                    'service_owner_geoville': service[4],
                    'external': service[5],
                    'date_of_creation': service[6].strftime("%Y-%m-%dT%H:%M:%S")
                }

                res_array.append(service_obj)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-get_services')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-get_services')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Successfully queried all services', 'API-
get_services')
            return {'services': res_array}, 200
```

5.1.123 services\backend_api\src\resources\resources_crm\get_all_tasks\get_all_tasks.py

```
#########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Get all tasks API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
#########################################################################################
##############################

from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import read_from_database_all_rows
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import crm_namespace as api
from lib.auth_header import auth_header_parser
from models.models_crm.manual_tasks_models.manual_tasks_models import task_list_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


#########################################################################################
##############################
# Resources definition for the get all tasks API call
#########################################################################################
##############################

@api.header('Content-Type', 'application/json')
class GetAllTasks(Resource):
    """ Class for handling the GET request

    This class defines the API call for the get all tasks script. The class consists of
one method which accepts a
    GET request. For the GET request no additional parameter are required.

    """


#########################################################################################
##########################
```

```
    # Method for handling the GET request
##########################################################################################
##########################

    @require_oauth(['admin', 'user'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Operation was successful', task_list_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self):
        """ GET definition for retrieving all tasks

        <p style="text-align: justify">This method defines the handler for the GET request
of the get all tasks
        script. It returns all task data stored in the database wrapped into a dictionary
defined by corresponding
        model.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify"></p>

        """

        db_query = """SELECT
                            task_id, task_name, task_comment, task_validity, task_owner,
                            external, created_at, order_id_not_required
                        FROM
                            customer.tasks
                        WHERE
                            deleted_at IS NULL
                     """

        try:
            task_data = read_from_database_all_rows(db_query, (), database_config_file,
database_config_section_api,
                                                             True)
            res_array = []

            for task in task_data:
                task_obj = {
                    'task_id': task[0],
                    'task_name': task[1],
                    'task_comment': task[2],
                    'task_validity': task[3],
                    'task_owner': task[4],
                    'external': task[5],
                    'date_of_creation': task[6].strftime("%Y-%m-%dT%H:%M:%S"),
                    'order_id_not_required': task[7]
                }

                res_array.append(task_obj)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_manual_tasks')
```

```
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_manual_tasks')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Successfully queried all tasks', 'API-
get_manual_tasks')
            return {'tasks': res_array}, 200
```

5.1.124 services\backend_api\src\resources\resources_crm\get_customers_by_filter\get_customers_by_filter.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Get customer by filter API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import read_from_database_all_rows
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import crm_namespace as api
from lib.auth_header import auth_header_parser
from lib.general_helper_methods import parameter_and_value_list_generation
from models.models_crm.customer_models.customer_models import customer_list_response_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback

##########################################################################################
##############################
# Query parameter definition of the GET request
##########################################################################################
##############################

query_param_parser = auth_header_parser.copy()
```

```
query_param_parser.add_argument('title', location='args', type=str, help='Title',
trim=True)
query_param_parser.add_argument('first_name', location='args', type=str, help='First
name')
query_param_parser.add_argument('last_name', location='args', type=str, help='Last name')
query_param_parser.add_argument('email', location='args', type=str, help='E-mail address')
query_param_parser.add_argument('address', location='args', type=str, help='Address 1')
query_param_parser.add_argument('zip_code', location='args', type=str, help='Zip Code')
query_param_parser.add_argument('city', location='args', type=str, help='City')
query_param_parser.add_argument('country', location='args', type=str, help='Country')
query_param_parser.add_argument('nationality', location='args', type=str,
help='Nationality')
query_param_parser.add_argument('phone', location='args', type=str, help='Phone number')
query_param_parser.add_argument('company_name', location='args', type=str, help='Company
name')


##########################################################################################
##############################
# Resource definition for the get customer API call
##########################################################################################
##############################

@api.header('Content-Type', 'application/json')
class GetCustomersByFilter(Resource):
    """ Class for handling the GET request

    This class defines the API call for the get customer by filter script. The class
consists of one method which
    accepts a GET request. For the GET request a JSON with several additional parameter is
required, defined in the
    corresponding model.

    """


##########################################################################################
##########################
    # Method for handling the POST request

##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(query_param_parser)
    @api.response(200, 'Operation successful', customer_list_response_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found Error', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self):
        """ GET definition for retrieving a customer by a filter

        <p style="text-align: justify">This method defines the handler for the GET request
of the get customer by filter
        script. It returns the customer data stored in the database wrapped into a
dictionary defined by corresponding
        model.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameter:</b>
        <ul>
        <li><p><i>client_id (str): </i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify"></p>

        """

        try:
            req_args = query_param_parser.parse_args()
            gemslog(LogLevel.INFO, f"Request payload: {req_args}", 'API-
get_customer_filter')

            param_list, val_list = parameter_and_value_list_generation(req_args)

            if not val_list and not param_list:
                db_query = """SELECT
                                  title, first_name, last_name, email, address, city,
zip_code, country, nationality,
                                  phone_number, company_name
                              FROM
                                  customer.customer
                              WHERE
                                  deleted_at IS NULL
                           """

            else:
                db_query = f"""SELECT
                                  title, first_name, last_name, email, address, city,
zip_code, country, nationality,
                                  phone_number, company_name
                              FROM
                                  customer.customer
                              WHERE
                                  {'AND '.join(param_list)} AND
                                  deleted_at IS NULL
                           """

            customer_data = read_from_database_all_rows(db_query, val_list,
database_config_file,
                                                                     database_config_section_api, True)

            if customer_data is None or customer_data is False:
                return {'customers': None}, 200

            customer_res = []

            for customer in customer_data:
                customer_obj = {
                    'title': customer[0],
                    'first_name': customer[1],
                    'last_name': customer[2],
                    'email': customer[3],
                    'address': customer[4],
                    'city': customer[5],
                    'zip_code': customer[6],
                    'country': customer[7],
                    'nationality': customer[8],
                    'phone_number': customer[9],
                    'company_name': customer[10]
                }

                customer_res.append(customer_obj)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
req_args, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
get_customer_filter')
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_customer_filter')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', req_args,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_customer_filter')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Successful response: {customer_res}', 'API-
get_customer_filter')
            return {'customers': customer_res}, 200
```

5.1.125 services\backend_api\src\resources\resources_crm\get_customer_by_id\get_customer_by_id.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Get customer by ID API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import read_from_database_one_row
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import crm_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_user_existence
from models.models_crm.customer_models.customer_models import customer_filter_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


##########################################################################################
##############################
# Resource definition for the get customer by ID API call
##########################################################################################
##############################

@api.header('Content-Type', 'application/json')
@api.param('user_id', 'User ID to be requested')
class GetCustomerById(Resource):
    """ Class for handling the GET request

    This class defines the API call for the get customer by ID script. The class consists
of one method which accepts a
    GET request. For the GET request one path variable is required and defined in the
corresponding class method.

    """


##########################################################################################
##########################
    # Method for handling the GET request

##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Operation was successful', customer_filter_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self, user_id):
        """ GET definition for retrieving a customer by a ID

        <p style="text-align: justify">This method defines the handler for the GET request
of the get customer by ID
        script. It returns the customer data stored in the database wrapped into a
dictionary defined by corresponding
        model.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameter:</b>
        <ul>
        <li><p><i>client_id (str): </i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify"></p>

        """
        try:
            gemslog(LogLevel.INFO, f'Request path parameter {user_id}', 'API-
get_customers_by_id')

            if not check_user_existence(user_id, database_config_file,
database_config_section_api):
                error = NotFoundError('User ID does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
get_customers_by_id')
                return {'message': error.to_dict()}, 404

            db_query = """SELECT
                                title, first_name, last_name, email, address, city,
zip_code, country, nationality,
                                phone_number, company_name
                            FROM
                                customer.customer
                            WHERE
                                customer_id = %s AND
                                deleted_at IS NULL
                        """

            customer = read_from_database_one_row(db_query, (user_id,),
database_config_file,
                                                    database_config_section_api, True)
            customer_data = {
                'title': customer[0],
                'first_name': customer[1],
                'last_name': customer[2],
                'email': customer[3],
                'address': customer[4],
                'city': customer[5],
                'zip_code': customer[6],
                'country': customer[7],
                'nationality': customer[8],
                'phone_number': customer[9],
                'company_name': customer[10]
            }

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
get_customers_by_id')
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_customers_by_id')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_customers_by_id')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Successful response: {customer_data}', 'API-
get_customers_by_id')
            return customer_data, 200
```

5.1.126 services\backend_api\src\resources\resources_crm\get_manual_tasks\get_manual_tasks.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Get all services API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from error_classes.http_error_422.http_error_422 import UnprocessableEntityError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import read_from_database_all_rows
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import crm_namespace as api
from lib.general_helper_methods import parameter_and_value_list_generation
from lib.auth_header import auth_header_parser
from lib.database_helper import check_service_name_existence,
check_processing_unit_exists, check_subproduction_unit_exists
from models.models_crm.manual_tasks_models.manual_tasks_models import
task_query_list_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_422 import error_422_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


########################################################################################
##############################
# Query parameter definition of the GET request
########################################################################################
##############################

query_param_parser = auth_header_parser.copy()
query_param_parser.add_argument('subproduction_unit', location='args', type=str,
required=False,
                                help='Sub-Production Unit', trim=True)
query_param_parser.add_argument('processing_unit', location='args', type=str,
required=False,
                                help='Processing Unit', trim=True)
query_param_parser.add_argument('service_name', location='args', type=str, required=False,
                                help='Name of the automatic service', trim=True)
query_param_parser.add_argument('order_status', location='args', type=str, required=False,
choices=('not_started', 'in_progress', 'failed', 'finished'),
                                help='Status of the service order', trim=True)
query_param_parser.add_argument('task_name', location='args', type=str, required=False,
                                help='Name of the manual task', trim=True)
```

```
query_param_parser.add_argument('task_status', location='args', type=str, required=False,
choices=('in_progress', 'failed', 'finished'),
                                help='Status of the manual task', trim=True)
##########################################################################################
##############################
# Resources definition for the get service orders API call
##########################################################################################
##############################

@api.header('Content-Type', 'application/json')
class GetManualTasks(Resource):
    """ Class for handling the GET request

    This class defines the API call for the get service orders script. The class consists
of one method which accepts a
    GET request. For the GET request no additional parameter are required.

    """


##########################################################################################
##########################
    # Method for handling the GET request

##########################################################################################
##########################

    @require_oauth(['admin', 'user'])
    @api.expect(query_param_parser)
    @api.response(200, 'Operation was successful', task_query_list_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self):
        """ GET definition for retrieving all services

        <p style="text-align: justify">This method defines the handler for the GET request
of the get service orders
        script. It returns all service data stored in the database wrapped into a
dictionary defined by corresponding
        model.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify"></p>

        """

        try:
            req_args = query_param_parser.parse_args()
            gemslog(LogLevel.INFO, 'Request path parameter: {}'.format(req_args),
                    'API-get_manual_tasks')

            if req_args['order_status'] == 'not_started':
                req_args.update({'order_status': 'RECEIVED OR order_status = QUEUED'})
            elif req_args['order_status'] == 'in_progress':
                req_args.update({'order_status': 'RUNNING'})
            elif req_args['order_status'] == 'failed':
                req_args.update({'order_status': 'FAILED'})
            elif req_args['order_status'] == 'finished':
                req_args.update({'order_status': 'SUCCESS'})

            if req_args['task_status'] == 'in_progress':
                req_args.update({'task_status': 'RUNNING'})
            elif req_args['task_status'] == 'failed':
                req_args.update({'task_status': 'FAILED'})
            elif req_args['task_status'] == 'finished':
                req_args.update({'task_status': 'SUCCESS'})

            if req_args['service_name'] and not
check_service_name_existence(req_args['service_name'], database_config_file,
database_config_section_api):
                error = BadRequestError('Service name does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
get_manual_tasks')
                return {'message': error.to_dict()}, 400

            # check if subproduction unit exists
            subproduction_unit_exists =
check_subproduction_unit_exists(req_args['subproduction_unit'], database_config_file,

database_config_section_api)
            if req_args['subproduction_unit'] and not subproduction_unit_exists:
                error = UnprocessableEntityError('Sub-Production unit ({0}) does not exist
in database'.format(
                    req_args['subproduction_unit']), '', '')
                return {'message': error.to_dict()}, 422

            # Check if processing unit exists
            processing_unit_exists =
check_processing_unit_exists(req_args['processing_unit'], database_config_file,

database_config_section_api)
            if req_args['processing_unit'] and not processing_unit_exists:
                error = UnprocessableEntityError('Processing unit ({0}) does not exist in
database'.format(
                    req_args['processing_unit']), '', '')
                return {'message': error.to_dict()}, 422

            param_list, val_list = parameter_and_value_list_generation(req_args)

            if not val_list and not param_list:
                db_query = """SELECT DISTINCT
                          ts.subproduction_unit,
                          ts.processing_unit,
                          ts.service_name,
                          ts.order_status,
                          ts.order_id,
                          ts.task_name,
                          ts.task_status,
                          ts.task_result
                      FROM
                          customer.tasks_and_services ts
                           """

            else:
                db_query = f"""SELECT DISTINCT
                          ts.subproduction_unit,
                          ts.processing_unit,
                          ts.service_name,
                          ts.order_status,
                          ts.order_id,
                          ts.task_name,
                          ts.task_status,
                          ts.task_result
                      FROM
                          customer.tasks_and_services ts
                               WHERE
```

```
                              {'AND '.join(param_list)}
                        """

            service_data = read_from_database_all_rows(db_query, val_list,
                                                       database_config_file,
database_config_section_api,
                                                       True)
            res_array = []

            for service in service_data:
                service_obj = {
                    'subproduction_unit': service[0],
                    'processing_unit': service[1],
                    'service_name': service[2],
                    'order_status': service[3],
                    'order_id': service[4],
                    'task_name': service[5],
                    'task_status': service[6],
                    'task_result': service[7]
                }

                res_array.append(service_obj)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-get_services')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-get_services')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Successfully queried all services', 'API-
get_services')
            return {'tasks': res_array}, 200
```

5.1.127 services\backend_api\src\resources\resources_crm\get_service_by_id\get_service_by_id.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Get service by ID API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import read_from_database_one_row
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import crm_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_service_existence
from models.models_crm.service_models.service_models import service_object_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


########################################################################################
##############################
# Resources definition for the get service by ID API call
########################################################################################
##############################

@api.header('Content-Type', 'application/json')
@api.param('service_id', 'Service ID to be requested')
class GetServiceByID(Resource):
    """ Class for handling the GET request

    This class defines the API call for the get service by ID script. The class consists
of one method which accepts a
    GET request. For the GET request an additional path parameter is required.

    """


########################################################################################
##########################
    # Method for handling the GET request

########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Operation was successful', service_object_model)
    @api.response(400, 'Bad Request', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self, service_id):
        """ GET definition for retrieving a service by ID

        <p style="text-align: justify">This method defines the handler for the GET request
of the get service by ID
        script. It returns all service data stored in the database wrapped into a
dictionary defined by corresponding
        model.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>
        <br><b>Path parameter:</b>
        <ul>
        <li><p><i>client_id (str): </i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify"></p>

        """

        db_query = """SELECT
                          service_id, service_name, service_comment, service_validity,
service_owner_geoville,
                          external, created_at
                      FROM
                          customer.services
                      WHERE
                          service_id = %s AND
                          deleted_at IS NULL
                   """

        try:
            gemslog(LogLevel.INFO, f'Request path parameter: {service_id}', 'API-
get_service_by_id')

            if not check_service_existence(service_id, database_config_file,
database_config_section_api):
                error = BadRequestError('Service ID does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
get_service_by_id')
                return {'message': error.to_dict()}, 400

            service_data = read_from_database_one_row(db_query, (service_id,),
database_config_file,
                                                       database_config_section_api, True)

            service_obj = {
                'service_id': service_data[0],
                'service_name': service_data[1],
                'service_comment': service_data[2],
                'service_validity': service_data[3],
                'service_owner_geoville': service_data[4],
                'external': service_data[5],
                'date_of_creation': service_data[6].strftime("%Y-%m-%dT%H:%M:%S")
            }

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_service_by_id')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_service_by_id')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Successful response', 'API-get_service_by_id')
            return service_obj, 200
```

5.1.128 services\backend_api\src\resources\resources_crm\get_service_by_name\get_service_by_name.py

```
########################################################################################################################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Get service by name API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################################################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import read_from_database_one_row
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import crm_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_service_name_existence
from models.models_crm.service_models.service_models import service_object_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


########################################################################################################################
# Resources definition for the get service by name API call
########################################################################################################################

@api.header('Content-Type', 'application/json')
@api.param('service_name', 'Service name to be requested')
class GetServiceByName(Resource):
    """ Class for handling the GET request

    This class defines the API call for the get service by name script. The class consists
of one method which accepts a
    GET request. For the GET request an additional path parameter is required.

    """

####################################################################################################################
    # Method for handling the GET request

####################################################################################################################
```

```
    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Operation was successful', service_object_model)
    @api.response(400, 'Bad Request', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self, service_name):
        """ GET definition for retrieving a service by name

        <p style="text-align: justify">This method defines the handler for the GET request
of the get service by name
        script. It returns all service data stored in the database wrapped into a
dictionary defined by corresponding
        model.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameter:</b>
        <ul>
        <li><p><i>client_id (str): </i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify"></p>

        """

        db_query = """SELECT
                          service_id, service_name, service_comment, service_validity,
service_owner_geoville,
                          external, created_at
                      FROM
                          customer.services
                      WHERE
                          service_name = %s AND
                          deleted_at IS NULL
                   """

        try:
            gemslog(LogLevel.INFO, f'Request path parameter: {service_name}', 'API-
get_service_by_name')

            if not check_service_name_existence(service_name, database_config_file,
database_config_section_api):
                error = BadRequestError('Service name does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
get_service_by_name')
                return {'message': error.to_dict()}, 400

            service_data = read_from_database_one_row(db_query, (service_name, ),
database_config_file,
                                                      database_config_section_api, True)

            service_obj = {
                'service_id': service_data[0],
                'service_name': service_data[1],
                'service_comment': service_data[2],
                'service_validity': service_data[3],
                'service_owner_geoville': service_data[4],
                'external': service_data[5],
                'date_of_creation': service_data[6].strftime("%Y-%m-%dT%H:%M:%S")
            }

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_service_by_name')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_service_by_name')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Successful response', 'API-get_service_by_name')
            return service_obj, 200
```

5.1.129 services\backend_api\src\resources\resources_crm\get_service_orders\get_service_orders.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Get all services API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from error_classes.http_error_422.http_error_422 import UnprocessableEntityError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import read_from_database_all_rows
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import crm_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_service_name_existence,
check_processing_unit_exists, check_subproduction_unit_exists
from lib.general_helper_methods import parameter_and_value_list_generation
from models.models_crm.service_models.service_models import query_list_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_422 import error_422_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


#########################################################################################
##############################
# Query parameter definition of the GET request
#########################################################################################
##############################

query_param_parser = auth_header_parser.copy()
query_param_parser.add_argument('subproduction_unit', location='args', type=str, 
required=False,
                                help='Sub-Production Unit', trim=True)
query_param_parser.add_argument('processing_unit', location='args', type=str, 
required=False,
                                help='Processing Unit', trim=True)
query_param_parser.add_argument('service_name', location='args', type=str, required=False,
                                help='Name of the automatic service', trim=True)
query_param_parser.add_argument('order_status', location='args', type=str, required=False,
choices=('Not started', 'In progress', 'Failed', 'Finished'),
                                help='Status of the service order', trim=True)
#########################################################################################
##############################
# Resources definition for the get service orders API call
#########################################################################################
##############################

@api.header('Content-Type', 'application/json')
class GetServiceOrders(Resource):
    """ Class for handling the GET request

    This class defines the API call for the get service orders script. The class consists 
of one method which accepts a
    GET request. For the GET request no additional parameter are required.

    """


#########################################################################################
##########################
    # Method for handling the GET request

#########################################################################################
##########################

    @require_oauth(['admin', 'user'])
    @api.expect(query_param_parser)
    @api.response(200, 'Operation was successful', query_list_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self):
        """ GET definition for retrieving all services

        <p style="text-align: justify">This method defines the handler for the GET request 
of the get service orders
        script. It returns all service data stored in the database wrapped into a 
dictionary defined by corresponding
        model.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>
        <br><b>Result:</b>
        <p style="text-align: justify"></p>

        """

        try:
            req_args = query_param_parser.parse_args()
            gemslog(LogLevel.INFO, 'Request path parameter: {}'.format(req_args),
                    'API-get_service_orders')

            if req_args['order_status'] == 'Not started':
                req_args['order_status'] = ('RECEIVED', 'QUEUED')
            elif req_args['order_status'] == 'In progress':
                req_args['order_status'] = 'RUNNING'
            elif req_args['order_status'] == 'Failed':
                req_args['order_status'] = 'FAILED'
            elif req_args['order_status'] == 'Finished':
                req_args['order_status'] = 'SUCCESS'

            if req_args['service_name'] and not
check_service_name_existence(req_args['service_name'], database_config_file,
database_config_section_api):
                error = BadRequestError('Service name does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
get_service_orders')
                return {'message': error.to_dict()}, 400

            # check if subproduction unit exists
            subproduction_unit_exists =
check_subproduction_unit_exists(req_args['subproduction_unit'], database_config_file,

database_config_section_api)
            if req_args['subproduction_unit'] and not subproduction_unit_exists:
                error = UnprocessableEntityError('Sub-Production unit ({0}) does not exist
in database'.format(
                    req_args['subproduction_unit']), '', '')
                return {'message': error.to_dict()}, 422

            # Check if processing unit exists
            processing_unit_exists =
check_processing_unit_exists(req_args['processing_unit'], database_config_file,

database_config_section_api)
            if req_args['processing_unit'] and not processing_unit_exists:
                error = UnprocessableEntityError('Processing unit ({0}) does not exist in
database'.format(
                    req_args['processing_unit']), '', '')
                return {'message': error.to_dict()}, 422

            param_list, val_list = parameter_and_value_list_generation(req_args)

            res_array = []
            for view in ["customer.tasks_and_services",
"customer.tasks_and_services_spu"]:
                if not val_list and not param_list:
                    db_query = """SELECT
                              ts.subproduction_unit,
                              ts.processing_unit,
                              ts.service_name,
                              ts.order_status,
                              ts.order_id,
                              ts.order_json,
                              ts.order_result
                          FROM
                              %s ts
                               """ % view
                else:
                    db_query = f"""SELECT
                              ts.subproduction_unit,
                              ts.processing_unit,
                              ts.service_name,
                              ts.order_status,
                              ts.order_id,
                              ts.order_json,
                              ts.order_result
                          FROM
                              {view} ts
                          WHERE
                              {'AND '.join(param_list)}
                                """

                gemslog(LogLevel.INFO, 'Execute Query {}'.format(db_query), 'API-
get_services')

                service_data = read_from_database_all_rows(db_query, val_list,
                                                           database_config_file,
database_config_section_api,
                                                           True)

                for service in service_data:
                    service_obj = {
                        'subproduction_unit': service[0],
                        'processing_unit': service[1],
                        'service_name': service[2],
                        'order_status': service[3],
                        'order_id': service[4],
                        'order_json': service[5],
                        'order_result': service[6]
                    }

                    res_array.append(service_obj)

            # remove duplicates
            for element_id, element in enumerate(res_array):
                for k, v in element.items():
                    if isinstance(v, dict):
                        res_array[element_id][k] = str(v)
            res_array = [dict(t) for t in {tuple(d.items()) for d in res_array}]

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-get_services')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-get_services')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Successfully queried all services', 'API-
get_services')
            return {'services': res_array}, 200
```

5.1.130 services\backend_api\src\resources\resources_crm\update_manual_task\update_manual_task.py

```
##########################################################################################
###############################
```

```
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Create customer API call
#
# Date created: 22.02.2021
# Date last modified: 04.03.2021
#
# __author__  = Patrick Wolf (wolf@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

import datetime
from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_422.http_error_422 import UnprocessableEntityError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import crm_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_user_existence
from models.models_crm.manual_tasks_models.manual_tasks_models import
manual_task_update_state_response_model
from models.models_crm.manual_tasks_models.manual_tasks_models import
manual_task_update_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_422 import error_422_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback
import datetime

##########################################################################################
##############################
# Resource definition for the update state API call
##########################################################################################
##############################

@api.expect(manual_task_update_model)
@api.header('Content-Type', 'application/json')
class UpdateManualTask(Resource):
    """ Class for handling the PUT request

    This class defines the API call for the update manual task script. The class consists
of one method which accepts a
    PUT request. For the PUT request a JSON with several parameters is required and
defined in the corresponding
    model.

    """


##########################################################################################
##########################
    # Method for handling the PUT request
```

```
##########################################################################################
##########################

    @require_oauth(['admin', 'user'])
    @api.expect(auth_header_parser)
    @api.response(201, 'Operation successful', manual_task_update_state_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Unprocessable Entity', error_422_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def put(self):
        """ PUT definition for update a manual task

        <p style="text-align: justify">This method defines the handler for the PUT request
of the update manual task
        (sate and result) script.</p>

        <br><b>Description:</b>
        <p style="text-align: justify">This service updates an existing manual task entry
in the database. The put
        request is used to update the <i>state</i> column in the database. If the results
value is <i>finished</i>,
        the <i>result</i> key is also required. As a result the service responses the HTTP
code 201. If an error occurs,
        the service returns one of the appropriate error status codes (400, 401, 403, 404,
500, 503)</p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>state (str): New state for the task (not_started, in_progress, failed,
finished)</i></p></li>
        <li><p><i>result (str): Result for the task (required when
state='finished')</i></p></li>
        <li><p><i>processing_unit (str): Unique processing unit identifier</i></p></li>
        <li><p><i>service_id (str): Unique service identifier</i></p></li>
        <li><p><i>task_id (str): Unique task identifier</i></p></li>
        <li><p><i>client_id (str): User client-id</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">SUCCESS: If the entry was updated successfully in
the database, the service
        returns the HTTP code 204 as well an a dictionary containing the key message.</p>
        <p style="text-align: justify">ERROR: In case of an error, the appropriate HTTP
error code is returned (400,
        401, 403, 404, 500, 503). Additionally, the service returns a dictionary with two
keys
        (message, error_definition)</p>

        """

        # Get request parameters and check if the payload parameter names are correct
        try:
            req_args = api.payload

            request_state = req_args['state']
            request_processing_unit = req_args['processing_unit']
            request_service_id = req_args['service_id']
            request_task_id = req_args['task_id']
            request_client_id = req_args['client_id']
            request_result = None if "result" not in req_args else req_args['result']
        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            return {'message': error.to_dict()}, 400

        try:
            # Validate user existence
            if not check_user_existence(request_client_id, database_config_file,
database_config_section_api):
                error = UnprocessableEntityError('User ID does not exist', '', '')
                return {'message': error.to_dict()}, 422

            # Check if status value is correct
            supported_state_values = {"not_started": None, "in_progress": "RUNNING",
"failed": "FAILED",
                                      "finished": "SUCCESS"}

            if request_state.lower() not in supported_state_values:
                error = UnprocessableEntityError('Status ({0}) is not
supported'.format(request_state), '', '')
                return {'message': error.to_dict()}, 422

            if request_state.lower() == "finished":
                if not request_result:
                    error = UnprocessableEntityError('Result missing - value for result
can not be NULL when state is '
                                                     'finished'.format(request_state), '',
'')
                    return {'message': error.to_dict()}, 422

                sql = """
                    UPDATE customer.manual_tasks
                    SET    status = %s,
                           "result" = %s,
                           task_stopped = %s
                    WHERE  ( cell_code = %s
                             AND service_id = %s
                             AND task_id = %s );
                """
                task_stopped = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                task_state = supported_state_values[request_state]

                parameter_tuple = (task_state, request_result, task_stopped,
request_processing_unit,
                                   request_service_id, request_task_id, )
                execute_database(sql, parameter_tuple, database_config_file,
database_config_section_api, True)

            else:
                sql = """
                    UPDATE customer.manual_tasks
                    SET    status = %s,
                           "result" = %s,
                           task_stopped = %s
                    WHERE  ( cell_code = %s
                             AND service_id = %s
                             AND task_id = %s );
                """

                task_state = supported_state_values[request_state]
                parameter_tuple = (task_state, None, None, request_processing_unit,
request_service_id, request_task_id,)
                execute_database(sql, parameter_tuple, database_config_file,
database_config_section_api, True)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            return {'message': error.to_dict()}, 500

        else:
            response = {
                'message': 'Your update has been successfully submitted',
            }

            return response, 200
```

5.1.131 services\backend_api\src\resources\resources_crm\update_manual_task_order_id\update_manual_task_order_id.py

```
################################################################################################################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Create customer API call
#
# Date created: 22.02.2021
# Date last modified: 04.03.2021
#
# __author__  = Patrick Wolf (wolf@geoville.com)
# __version__ = 21.03
#
################################################################################################################

import datetime
from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_422.http_error_422 import UnprocessableEntityError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import crm_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_user_existence
from models.models_crm.manual_tasks_models.manual_tasks_models import
manual_task_update_order_id_model
from models.models_crm.manual_tasks_models.manual_tasks_models import
manual_task__update_order_id_response_model
from lib.database_helper import check_order_id_exists
from lib.database_helper import check_state_is_success
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_422 import error_422_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback

###########################################################################################
##############################
# Resource definition for the update order_id API call
###########################################################################################
##############################

@api.expect(manual_task_update_order_id_model)
@api.header('Content-Type', 'application/json')
class UpdateManualTaskOrderID(Resource):
    """ Class for handling the PUT request

    This class defines the API call for the update manual task (order-id) script. The
class consists of one method
    which accepts a PUT request. For the PUT request a JSON with several parameters is
required and defined in the
    corresponding model.

    """


###########################################################################################
##########################
    # Method for handling the PUT request

###########################################################################################
##########################

    @require_oauth(['admin', 'user'])
    @api.expect(auth_header_parser)
    @api.response(201, 'Operation successful',
manual_task__update_order_id_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Unprocessable Entity', error_422_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def put(self):
        """ PUT definition for update a manual task

        <p style="text-align: justify">This method defines the handler for the PUT request
of the update manual task
        (order_id) script.</p>

        <br><b>Description:</b>
        <p style="text-align: justify">This service updates an existing manual task entry
in the database. The put
        request is used to update the <i>refers_to_order_id</i> column in the database. As
a result the service
        responses the HTTP code 201. If an error occurs, the service returns one of the
appropriate error status
        codes (400, 401, 403, 404, 500, 503)</p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>refers_to_order_id (str): New order-id value</i></p></li>
        <li><p><i>processing_unit (str): Unique processing unit identifier</i></p></li>
        <li><p><i>service_id (str): Unique service identifier</i></p></li>
        <li><p><i>task_id (str): Unique task identifier</i></p></li>
        <li><p><i>client_id (str): User client-id</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">SUCCESS: If the entry was updated successfully in
the database, the service
        returns the HTTP code 204 as well an a dictionary containing the key message.</p>
        <p style="text-align: justify">ERROR: In case of an error, the appropriate HTTP
error code is returned (400,
        401, 403, 404, 500, 503). Additionally, the service returns a dictionary with two
keys
        (message, error_definition)</p>

        """

        # Get request parameters and check if the payload parameter names are correct
        try:
            req_args = api.payload

            request_refers_to_order_id = req_args['refers_to_order_id']
            request_processing_unit = req_args['processing_unit']
            request_service_id = req_args['service_id']
            request_task_id = req_args['task_id']
            request_client_id = req_args['client_id']

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            return {'message': error.to_dict()}, 400

        try:
            # Validate user existence
            if not check_user_existence(request_client_id, database_config_file,
database_config_section_api):
                error = UnprocessableEntityError('User ID does not exist', '', '')
                return {'message': error.to_dict()}, 422

            # Check if order-id exists
            refers_to_order_id_exists = check_order_id_exists(request_refers_to_order_id,
database_config_file,
                                                                   database_config_section_api)
            if not refers_to_order_id_exists:
                error = UnprocessableEntityError('Order-ID ({0}) does not
exist'.format(request_refers_to_order_id), '', '')
                return {'message': error.to_dict()}, 422

            # Check if state is correct
            is_success = check_state_is_success(request_processing_unit,
request_service_id, request_task_id,
                                                 database_config_file,
database_config_section_api)
            if not is_success:
                error = UnprocessableEntityError('Could not update refers_to_order_id -
state is not SUCCESS or '
                                                  'database entity (task, service,
processing_unit) does not exist'.
                                                  format(request_refers_to_order_id), '',
'')
                return {'message': error.to_dict()}, 422

            # Update the refers_to_order_id column
            sql = """
                update
                    customer.manual_tasks
                set
                    refers_to_order_id = %s
                where
                    cell_code = %s
                    and service_id = %s
                    and task_id = %s ;
            """
            parameter_tuple = (request_refers_to_order_id, request_processing_unit,
request_service_id, request_task_id,)
            execute_database(sql, parameter_tuple, database_config_file,
database_config_section_api, True)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            return {'message': error.to_dict()}, 500

        else:
            response = {
                'message': 'Your update has been successfully submitted',
                'refers_to_order_id': request_refers_to_order_id
            }
            return response, 200
```

5.1.132 services\backend_api\src\resources\resources_crm\update_manual_task_spu\update_manual_task_spu.py

```
##########################################################################################
###############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Create customer API call
#
# Date created: 22.02.2021
# Date last modified: 04.03.2021
#
# __author__  = Patrick Wolf (wolf@geoville.com)
# __version__ = 21.02
#
##########################################################################################
###############################

import datetime
from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_422.http_error_422 import UnprocessableEntityError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database,
read_from_database_all_rows
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import crm_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_user_existence
from models.models_crm.manual_tasks_models.manual_tasks_models import
manual_task_update_state_response_model
from models.models_crm.manual_tasks_models.manual_tasks_models import
manual_task_update_spu_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_422 import error_422_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback
import datetime

###########################################################################################
##############################
# Resource definition for the update state API call
###########################################################################################
##############################

@api.expect(manual_task_update_spu_model)
@api.header('Content-Type', 'application/json')
class UpdateManualTaskSPU(Resource):
    """ Class for handling the PUT request

    This class defines the API call for the update manual task script. The class consists
of one method which accepts a
    PUT request. For the PUT request a JSON with several parameters is required and
defined in the corresponding
    model.

    """


###########################################################################################
##########################
    # Method for handling the PUT request

###########################################################################################
##########################

    @require_oauth(['admin', 'user'])
    @api.expect(auth_header_parser)
    @api.response(201, 'Operation successful', manual_task_update_state_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Unprocessable Entity', error_422_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def put(self):
        """ PUT definition for update a manual task

        <p style="text-align: justify">This method defines the handler for the PUT request
of the update manual task
        (sate and result) script.</p>

        <br><b>Description:</b>
        <p style="text-align: justify">This service updates an existing manual task entry
in the database. The put
        request is used to update the <i>state</i> column in the database. If the results
value is <i>finished</i>,
        the <i>result</i> key is also required. As a result the service responses the HTTP
code 201. If an error occurs,
        the service returns one of the appropriate error status codes (400, 401, 403, 404,
500, 503)</p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>state (str): New state for the task (not_started, in_progress, failed,
finished)</i></p></li>
        <li><p><i>result (str): Result for the task (required when
state='finished')</i></p></li>
        <li><p><i>subproduction_unit (str): Unique subproduction unit
identifier</i></p></li>
        <li><p><i>service_id (str): Unique service identifier</i></p></li>
        <li><p><i>task_id (str): Unique task identifier</i></p></li>
        <li><p><i>client_id (str): User client-id</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">SUCCESS: If the entry was updated successfully in
the database, the service
        returns the HTTP code 204 as well an a dictionary containing the key message.</p>
        <p style="text-align: justify">ERROR: In case of an error, the appropriate HTTP
error code is returned (400,
        401, 403, 404, 500, 503). Additionally, the service returns a dictionary with two
keys
        (message, error_definition)</p>

        """

        # Get request parameters and check if the payload parameter names are correct
        try:
            req_args = api.payload

            request_state = req_args['state']
            request_subproduction_unit = req_args['subproduction_unit']
            request_service_id = req_args['service_id']
            request_task_id = req_args['task_id']
            request_client_id = req_args['client_id']
            request_result = None if "result" not in req_args else req_args['result']

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            return {'message': error.to_dict()}, 400

        try:
            # Validate user existence
            if not check_user_existence(request_client_id, database_config_file,
database_config_section_api):
                error = UnprocessableEntityError('User ID does not exist', '', '')
                return {'message': error.to_dict()}, 422

            # Check if status value is correct
            supported_state_values = {"not_started": None, "in_progress": "RUNNING",
"failed": "FAILED",
                                      "finished": "SUCCESS"}

            if request_state.lower() not in supported_state_values:
                error = UnprocessableEntityError('Status ({0}) is not
supported'.format(request_state), '', '')
                return {'message': error.to_dict()}, 422

            sql = """
                UPDATE customer.manual_tasks mt
                SET    status = %s,
                       "result" = %s,
                       task_stopped = %s
                FROM grafana_monitoring.all_units au
                WHERE   (
                         mt.cell_code = au.cellcode
                         AND au.spu_id = %s
                         AND mt.service_id = %s
                         AND mt.task_id = %s );
            """

            sql_check = """
                SELECT
                    COUNT(*)
                FROM
                    grafana_monitoring.all_units au,
                    customer.manual_tasks mt
                WHERE   (
                         mt.cell_code = au.cellcode
                         AND au.spu_id = %s
                         AND mt.service_id = %s
                         AND mt.task_id = %s );
            """

            if request_state.lower() == "finished":
                if not request_result:
                    error = UnprocessableEntityError('Result missing - value for result
can not be NULL when state is '
                                                     'finished'.format(request_state), '',
'')
                    return {'message': error.to_dict()}, 422

                task_stopped = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                task_state = supported_state_values[request_state]

                parameter_tuple = (task_state, request_result, task_stopped,
request_subproduction_unit,
                                   request_service_id, request_task_id, )

                parameter_tuple_check = (request_subproduction_unit,
                                   request_service_id, request_task_id,)

                rows = read_from_database_all_rows(sql_check, parameter_tuple_check,
database_config_file, database_config_section_api, True)[0][0]
                if rows == 0:
                    error = UnprocessableEntityError(
                        'No entries for an update could be found', '', '')
                    return {'message': error.to_dict()}, 422
                execute_database(sql, parameter_tuple, database_config_file,
                                 database_config_section_api, True)

            else:
                task_state = supported_state_values[request_state]
                parameter_tuple = (task_state, None, None, request_subproduction_unit,
request_service_id, request_task_id,)
                parameter_tuple_check = (request_subproduction_unit,
                                   request_service_id, request_task_id,)
                rows = read_from_database_all_rows(sql_check, parameter_tuple_check,
database_config_file, database_config_section_api, True)[0][0]
                if rows == 0:
                    error = UnprocessableEntityError(
                        'No entries for an update could be found', '', '')
                    return {'message': error.to_dict()}, 422
                execute_database(sql, parameter_tuple, database_config_file,
                                 database_config_section_api, True)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            return {'message': error.to_dict()}, 500

        else:
            response = {
                'message': 'Your update has been successfully submitted',
            }

            return response, 200
```

5.1.133 services\backend_api\src\resources\resources_logging\log_error\log_error.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# API call for creating an error log
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import logging_namespace as api
from lib.auth_header import auth_header_parser
from lib. database_helper import check_service_name_similarity
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from models.models_logging.logging_models import logging_request_model
from oauth.oauth2 import require_oauth
import traceback


##########################################################################################
##############################
# Resources definition for creating an error log message via API call
##########################################################################################
##############################

@api.expect(logging_request_model)
@api.header('Content-Type', 'application/json')
```

```
class LogError(Resource):
    """ Class for handling the POST request

    This class defines the API call for creating an error log message. The class consists
of one method which accepts a
    POST request. For the POST request two additional parameter are required.

    """


###########################################################################################
##########################
    # Method for handling the POST request

###########################################################################################
##########################

    @require_oauth(['admin', 'user'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation was successful')
    @api.response(400, 'Bad request', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for creating an error log message

        <p style="text-align: justify">This method defines the handler for the POST
request of the create error log
        message script. It returns no message body and thus no contents. In contrast it
returns the HTTP status code
        204.</p>

        <br><b>Description:</b>
        <p style="text-align: justify">This service route enables the possibility to write
log messages into the central
        GEMS logging database without having implemented the corresponding Python module.
Therefore the service can be
        called from several different programming languages via a simple curL command. The
curL command can be retrieved
        by trying out the service.</p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>module_name (str): name of the module which triggered log
message</i></p></li>
        <li><p><i>log_message (str): actual log message</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the POST request does not contain any
object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any
other response status code
        than 204, indicates an error during the execution.</p>

        """

        try:
            req_args = api.payload

            if req_args['service_module_name'] == '':
                error = BadRequestError('Service name must be valid string', api.payload,
'')
                gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-log_error')
                return {'message': error.to_dict()}, 400

            if not check_service_name_similarity(req_args['service_module_name'],
database_config_file,
                                                 database_config_section_api):
                error = BadRequestError('Service name could not be found', api.payload,
'')
                gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-log_error')
                return {'message': error.to_dict()}, 400

            order_id = None if 'order_id' not in req_args else req_args['order_id']
            gemslog(LogLevel.ERROR, req_args['log_message'],
req_args['service_module_name'], order_id)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-log_error')
            return {'message': error.to_dict()}, 503

        except Exception as err:
            error = InternalServerErrorAPI(f'Unexpected error occurred: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-log_error')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Successfully stored log message', 'API-log_error')
            return '', 204
```

5.1.134 services\backend_api\src\resources\resources_logging\log_info\log_info.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# API call for creating an info log message
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import logging_namespace as api
from lib.auth_header import auth_header_parser
from lib. database_helper import check_service_name_similarity
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from models.models_logging.logging_models import logging_request_model
from oauth.oauth2 import require_oauth
import traceback


###########################################################################################
##############################
# Resources definition for creating an info log message via API call
###########################################################################################
##############################

@api.expect(logging_request_model)
@api.header('Content-Type', 'application/json')
class LogInfo(Resource):
    """ Class for handling the POST request

    This class defines the API call for creating an info log message. The class consists
of one method which accepts a
    POST request. For the POST request two additional parameter are required.

    """


###########################################################################################
##########################
    # Method for handling the POST request

###########################################################################################
##########################

    @require_oauth(['admin', 'user'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation was successful')
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for creating an info log message

        <p style="text-align: justify">This method defines the handler for the POST
request of the create info log
        message script. It returns no message body and thus no contents. In contrast it
returns the HTTP status code
        204.</p>

        <br><b>Description:</b>
        <p style="text-align: justify">This service route enables the possibility to write
log messages into the central
        GEMS logging database without having implemented the corresponding Python module.
Therefore the service can be
        called from several different programming languages via a simple curL command. The
curL command can be retrieved
        by trying out the service.</p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>module_name (str): name of the module which triggered log
message</i></p></li>
        <li><p><i>log_message (str): actual log message</i></p></li>
        </ul>
        <br><b>Result:</b>
        <p style="text-align: justify">The result of the POST request does not contain any 
object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any 
other response status code
        than 204, indicates an error during the execution.</p>

        """

        try:
            req_args = api.payload

            if req_args['service_module_name'] == '':
                error = BadRequestError('Service name must be valid string', api.payload, 
'')
                gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-log_info')
                return {'message': error.to_dict()}, 400

            if not check_service_name_similarity(req_args['service_module_name'], 
database_config_file,
                                                 database_config_section_api):
                error = BadRequestError('Service name could not be found', api.payload, 
'')
                gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-log_info')
                return {'message': error.to_dict()}, 400

            order_id = None if 'order_id' not in req_args else req_args['order_id']
            gemslog(LogLevel.INFO, req_args['log_message'], 
req_args['service_module_name'], order_id)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server', 
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-log_info')
            return {'message': error.to_dict()}, 503

        except Exception as err:
            error = InternalServerErrorAPI(f'Unexpected error occurred:{err}', 
api.payload, traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-log_info')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Successfully stored log message', 'API-log_info')
            return '', 204
```

5.1.135 services\backend_api\src\resources\resources_logging\log_warning\log_warning.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is 
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# API call for creating a warning log message
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
```

```
#
##########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import logging_namespace as api
from lib.auth_header import auth_header_parser
from lib. database_helper import check_service_name_similarity
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from models.models_logging.logging_models import logging_request_model
from oauth.oauth2 import require_oauth
import traceback


##########################################################################################
##############################
# Resources definition for creating a warning log message via API call
##########################################################################################
##############################

@api.expect(logging_request_model)
@api.header('Content-Type', 'application/json')
class LogWarning(Resource):
    """ Class for handling the POST request

    This class defines the API call for creating a warning log message. The class consists
of one method which accepts a
    POST request. For the POST request two additional parameter are required.

    """


##########################################################################################
##########################
    # Method for handling the POST request

##########################################################################################
##########################

    @require_oauth(['admin', 'user'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation was successful')
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for creating a warning log message

        <p style="text-align: justify">This method defines the handler for the POST
request of the create warning log
        message script. It returns no message body and thus no contents. In contrast it
returns the HTTP status code
        204.</p>

        <br><b>Description:</b>
        <p style="text-align: justify">This service route enables the possibility to write
log messages into the central
        GEMS logging database without having implemented the corresponding Python module.
Therefore the service can be
```

```
        called from several different programming languages via a simple curL command. The
curL command can be retrieved
        by trying out the service.</p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>module_name (str): name of the module which triggered log
message</i></p></li>
        <li><p><i>log_message (str): actual log message</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the POST request does not contain any
object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any
other response status code
        than 204, indicates an error during the execution.</p>

        """

        try:
            req_args = api.payload

            if req_args['service_module_name'] == '':
                error = BadRequestError('Service name must be valid string', api.payload,
'')
                gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
log_warning')
                return {'message': error.to_dict()}, 400

            if not check_service_name_similarity(req_args['service_module_name'],
database_config_file,
                                                 database_config_section_api):
                error = BadRequestError('Service name could not be found', api.payload,
'')
                gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
log_warning')
                return {'message': error.to_dict()}, 400

            order_id = None if 'order_id' not in req_args else req_args['order_id']
            gemslog(LogLevel.WARNING, req_args['log_message'],
req_args['service_module_name'], order_id)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-log_warning')
            return {'message': error.to_dict()}, 503

        except Exception as err:
            error = InternalServerErrorAPI(f'Unexpected error occurred: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-log_warning')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Successfully stored log message', 'API-log_warning')
            return '', 204
```

5.1.136 services\backend_api\src\resources\resources_products\get_national_product\get_national_product.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Get National Products API call
#
# Date created: 24.02.2022
# Date last modified: 24.02.2022
#
# __author__  = Johannes Schmid (schmid@geoville.com)
# __version__ = 22.02
#
########################################################################################
##############################

from check_message.check_message import check_message
from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from geoville_ms_orderid_generator.generator import generate_orderid
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import service_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_user_existence, get_service_id
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_408 import error_408_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from models.models_products.products_models import national_products_request_model,
products_success_response_model
from oauth.oauth2 import require_oauth
import json
import traceback


########################################################################################
##############################
# Resource definition for the get-national-products API call
########################################################################################
##############################

@api.expect(national_products_request_model)
@api.header('Content-Type', 'application/json')
class NationalProducts(Resource):
    """ Class for handling the POST request

    This class defines the API call for getting the specified product for a nation of
choice.
    The class consists of one method which accepts a GET request. For the GET request the
user ID is required,
    defined in the corresponding model.
    """


##########################################################################################
##########################
    # Method for handling the POST request

##########################################################################################
##########################

    @require_oauth(['admin', 'user', 'get_product'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Success', products_success_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(408, 'Request Timeout', error_408_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for requesting the specified product for a nation of choice.

        <p style="text-align: justify">This method defines the handler of the POST request
for getting the
        specified product for a nation of choice. It is a synchronous call and thus, it
returns the requested data
        immediately. To access the service it is necessary to generate a valid Bearer
        token with sufficient access rights, otherwise the request will return a HTTP
status code 401 or 403. In case of
        those errors, please contact the GeoVille service team for any support.</p>

        <br><b>Description:</b>
        <p style="text-align: justify">By providing a country name a specified product can
be retrieved.</p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>product (str): Name of the CLC+ Backbone product</i></p></li>
        <li><p><i>nation (str): Country name in English (e.g. Germany)</i></p></li>
        <li><p><i>user_id (str): User specific client ID</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">After the request was successful, a download link
will be returned which
        provides the ordered file.</p>
        """

        order_id = None

        try:
            req_args = api.payload

            payload_check = check_message(req_args)

            if not payload_check[0]:
                error = BadRequestError(f'Payload failed the GeoVille standards:
{payload_check[1]}', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-national-
products', order_id)
                return {'message': error.to_dict()}, 404
            if not check_user_existence(req_args['user_id'], database_config_file,
database_config_section_api):
                error = NotFoundError('User ID does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-national-
products', order_id)
                return {'message': error.to_dict()}, 404

            service_id = get_service_id("get_national_product", database_config_file,
database_config_section_api)
            order_id = generate_orderid(req_args['user_id'], service_id,
json.dumps(req_args))
            gemslog(LogLevel.INFO, f'Request payload: {req_args}', 'API-national-
products', order_id)

            # publish_to_queue("get_national_product", order_id, req_args)

            update_query = """UPDATE customer.service_orders
                                  set status = 'RECEIVED'
                              WHERE
                                  order_id = %s;
                           """
            execute_database(update_query, (order_id,), database_config_file,
database_config_section_api, True)

            national_info = {
                "ALBANIA": {"country_code": "AL", "epsg": "02462"},
                "AUSTRIA": {"country_code": "AT", "epsg": "31287"},
                "BOSNIA AND HERZEGOVINA": {"country_code": "BA", "epsg": "03908"},
                "BELGIUM": {"country_code": "BE", "epsg": "03812"},
                "BULGARIA": {"country_code": "BG", "epsg": "32635"},
                "SWITZERLAND": {"country_code": "CH", "epsg": "02056"},
                "CYPRUS": {"country_code": "CY", "epsg": "32636"},
                "CZECH REPUBLIC": {"country_code": "CZ", "epsg": "05514"},
                "GERMANY": {"country_code": "DE", "epsg": "32632"},
                "DENMARK": {"country_code": "DK", "epsg": "25832"},
                "ESTONIA": {"country_code": "EE", "epsg": "03301"},
                "SPAIN": {"country_code": "ES", "epsg": "25830"},
                "SPAIN (CANARIES)": {"country_code": "ESCanaries", "epsg": "32628"},
                "FINLAND": {"country_code": "FI", "epsg": "03067"},
                "FRANCE": {"country_code": "FR", "epsg": "02154"},
                "GREAT BRITAIN": {"country_code": "GB", "epsg": "27700"},
                "GUERNSEY (CHANNEL ISLANDS)": {"country_code": " British Crown
DepenGdBencies)", "epsg": "03108"},
                "GREECE": {"country_code": "GR", "epsg": "02100"},
                "CROATIA": {"country_code": "HR", "epsg": "03765"},
                "HUNGARY": {"country_code": "HU", "epsg": "23700"},
                "IRELAND": {"country_code": "IE", "epsg": "02157"},
                "ICELAND": {"country_code": "IS", "epsg": "05325"},
                "ITALY": {"country_code": "IT", "epsg": "32632"},
                "JERSEY (CHANNEL ISLANDS)": {"country_code": " British Crown
DependenGcBies)", "epsg": "03109"},
                "LIECHTENSTEIN": {"country_code": "LI", "epsg": "02056"},
                "LITHUANIA": {"country_code": "LT", "epsg": "03346"},
                "LUXEMBOURG": {"country_code": "LU", "epsg": "02169"},
                "LATVIA": {"country_code": "LV", "epsg": "03059"},
                "MONTENEGRO": {"country_code": "ME", "epsg": "25834"},
                "FYR OF MACEDONIA": {"country_code": "MK", "epsg": "06204"},
                "MALTA": {"country_code": "MT", "epsg": "23033"},
                "NORTHERN IRELAND": {"country_code": "NI", "epsg": "29903"},
                "NETHERLANDS": {"country_code": "NL", "epsg": "28992"},
                "NORWAY": {"country_code": "NO", "epsg": "25833"},
                "POLAND": {"country_code": "PL", "epsg": "02180"},
                "PORTUGAL": {"country_code": "PT", "epsg": "03763"},
                "PORTUGAL (AZORES CENTRAL AND EASTERN GROUP)": {"country_code":
"PTAzoresCentEast", "epsg": "05015"},
                "PORTUGAL (AZORES WESTERN GROUP)": {"country_code": "PTAzoresWest",
"epsg": "05014"},
                "PORTUGAL (MADEIRA)": {"country_code": "PTMadeira", "epsg": "05016"},
                "ROMANIA": {"country_code": "RO", "epsg": "03844"},
                "SERBIA": {"country_code": "RS", "epsg": "25834"},
                "SWEDEN": {"country_code": "SE", "epsg": "03006"},
                "SLOVENIA": {"country_code": "SI", "epsg": "03912"},
                "SLOVAKIA": {"country_code": "SK", "epsg": "05514"},
                "TURKEY": {"country_code": "TR", "epsg": "00000"},
                "KOSOVO UNDER UNSCR 1244/99": {"country_code": "XK", "epsg": "03909"},
            }

            country, epsg = national_info[req_args['nation'].upper()].values()
            file_name = f"CLMS_CLCplus_RASTER_2018_010m_{country.lower()}_{epsg}_V1_1.tif"
            dll = f"https://s3.waw2-
1.cloudferro.com/swift/v1/AUTH_b9657821e4364f88862ca20a180dc485/clcplus-public/" \
                  f"products/national/{file_name}"

            db_query = """UPDATE
                                  customer.service_orders
                              SET
                                  status = 'SUCCESS',
                                  result = %s,
                                  success = true,
                                  order_started = NOW(),
                                  order_received = NOW(),
                                  order_stopped = NOW()
                              WHERE order_id = %s
                           """
            execute_database(db_query, (dll, order_id), database_config_file,
database_config_section_api, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-national-
products', order_id)
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-national-
products', order_id)
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-national-
products', order_id)
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Request successful', 'API-national-products',
order_id)
            return {
                    'result': dll
                   }, 200
            # return {
            #             'message': 'Your order has been successfully submitted',
            #             'links': {
            #                 'href': f'/services/order_status/{order_id}',
            #                 'rel': 'services',
            #                 'type': 'GET'
            #             }
            #        }, 202
```

5.1.137 services\backend_api\src\resources\resources_products\get_product\get_product.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Get Products API call
#
# Date created: 23.07.2021
# Date last modified: 23.07.2021
#
# __author__  = Johannes Schmid (schmid@geoville.com)
# __version__ = 21.07
#
########################################################################################
##############################

from check_message.check_message import check_message
from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from geoville_ms_orderid_generator.generator import generate_orderid
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import service_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_user_existence, get_service_id
from lib.general_helper_methods import publish_to_queue
from models.general_models.general_models import service_success_response_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from models.models_products.products_models import products_request_model
from oauth.oauth2 import require_oauth
import json
import traceback
import pyproj
import shapely.wkt
from shapely.ops import transform

########################################################################################
##############################
# Resource definition for the get-products API call
########################################################################################
##############################

@api.expect(products_request_model)
@api.header('Content-Type', 'application/json')
class Products(Resource):
    """ Class for handling the POST request

    This class defines the API call for starting the get-products workflow. The class
consists of one methods which
    accepts a POST request. For the POST request a JSON with several parameters is
required, defined in the
    corresponding model.
    """


############################################################################################
##########################
    # Method for handling the POST request

############################################################################################
##########################

    @require_oauth(['admin', 'user', 'get_product'])
    @api.expect(auth_header_parser)
    @api.response(202, 'Order Received', service_success_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for requesting the get-products workflow

        <p style="text-align: justify">This method defines the handler of the POST request
for starting off the
        get-products processing chain within the GEMS service architecture. It is an
asynchronous call and thus, it does
        not return the requested data immediately but generates an order ID. After the
request has been submitted
        successfully, a message to a RabbitMQ queue will be send. A listener in the
backend triggers the further
        procedure, starting with the job scheduling of the order. The final result will be
stored in a database in form
        of a link and can be retrieved via the browser. To access the service it is
necessary to generate a valid Bearer
        token with sufficient access rights, otherwise the request will return a HTTP
status code 401 or 403. In case of
        those errors, please contact the GeoVille service team for any support.</p>

        <br><b>Description:</b>
        <p style="text-align: justify">By providing an Area of Interest (AOI) a specified
product can be ordered.</p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>product (str): Name of the CLC+ Backbone product</i></p></li>
        <li><p><i>aoi (str): Area of Interest as MultiPolygon (WKT) in WGS84
(EPSG:4326)</i></p></li>
        <li><p><i>user_id (str): User specific client ID</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">After the request was successfully received by the
GEMS API, an order ID will be
        created for the submitted job in the GEMS backend and stored in a database with
all necessary information for
        the consumer and the system itself. The initial status of the order in the
database will be set to 'received'.
        After that the order ID will be returned in form of a Hypermedia link which
enables the possibilities to
        programmatically check the status the order by a piece of code of a GEMS API
consumer. In the following the
        status of the order can be queried by using the GEMS service route listed
below:</p>
```

```
        <ul><li><p><i>/services/order_status/{order_id}</i></p></li></ul>

        <p style="text-align: justify">The status of the order will be updated whenever
the processing chain reaches the
        next step in its internal calculation. In case of success, the user will receive a
link, which provides the
        ordered file. Additionally an e-mail notification is enabled, which sends out an
e-mail in case of success, with
        the resulting link or in case of failure, with a possible error explanation.
        </p>

        """

        order_id = None

        try:
            req_args = api.payload

            payload_check = check_message(req_args)

            if not payload_check[0]:
                error = BadRequestError(f'Payload failed the GeoVille standards:
{payload_check[1]}', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-products',
order_id)
                return {'message': error.to_dict()}, 404

            if not check_user_existence(req_args['user_id'], database_config_file,
database_config_section_api):
                error = NotFoundError('User ID does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-products',
order_id)
                return {'message': error.to_dict()}, 404

            # get area of aoi
            reproject = pyproj.Transformer.from_crs(pyproj.CRS('EPSG:4326'),
pyproj.CRS('EPSG:3857'), always_xy=True).transform
            aoi_metric = transform(reproject, shapely.wkt.loads(req_args['aoi']))
            aoi_area = aoi_metric.area * 1.0E-6
            if aoi_area > 5000000:
                error = "The requested AOI is too big (> 5 Mio. kmÂ²). Please note that
countries and entire Europe " \
                        "can be requested with other endpoints."
                gemslog(LogLevel.WARNING, f"'message': {error}", 'API-products', order_id)
                return {'message': error}, 404



            service_id = get_service_id("get_product", database_config_file,
database_config_section_api)
            order_id = generate_orderid(req_args['user_id'], service_id,
json.dumps(req_args))
            gemslog(LogLevel.INFO, f'Request payload: {req_args}', 'API-products',
order_id)

            publish_to_queue("get_product", order_id, req_args)

            update_query = """UPDATE customer.service_orders
                                  set status = 'RECEIVED'
                              WHERE
                                  order_id = %s;
                           """
            execute_database(update_query, (order_id,), database_config_file,
database_config_section_api, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-products',
order_id)
```

```
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-products',
order_id)
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-products',
order_id)
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Request successful', 'API-products', order_id)
            return {
                       'message': 'Your order has been successfully submitted',
                       'links': {
                           'href': f'/services/order_status/{order_id}',
                           'area': '%.2f km2' % aoi_area,
                           'rel': 'services',
                           'type': 'GET'
                       }
                   }, 202
```

5.1.138 services\backend_api\src\resources\resources_products\get_product_europe\get_product_europe.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Get Products for Europe API call
#
# Date created: 24.02.2022
# Date last modified: 24.02.2022
#
# __author__  = Johannes Schmid (schmid@geoville.com)
# __version__ = 22.02
#
########################################################################################
##############################

from check_message.check_message import check_message
from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from geoville_ms_orderid_generator.generator import generate_orderid
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import service_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_user_existence, get_service_id
from models.general_models.general_models import service_success_response_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_408 import error_408_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from models.models_products.products_models import european_products_request_model,
products_success_response_model
from oauth.oauth2 import require_oauth
import json
import traceback


##########################################################################################
##############################
# Resource definition for the get-products-europe API call
##########################################################################################
##############################

@api.expect(european_products_request_model)
@api.header('Content-Type', 'application/json')
class ProductEurope(Resource):
    """ Class for handling the POST request

    This class defines the API call for getting the specified product for entire Europe.
    The class consists of one method which accepts a POST request. For the POST request
the user ID is required,
    defined in the corresponding model.

    """


##########################################################################################
##########################
    # Method for handling the POST request

##########################################################################################
##########################

    @require_oauth(['admin', 'user', 'get_product'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Success', products_success_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(408, 'Request Timeout', error_408_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for requesting the specified product for entire Europe

        <p style="text-align: justify">This method defines the handler of the POST request
for getting the
        specified product for entire Europe. It is a synchronous call and thus, it returns
the requested data
        immediately. To access the service it is necessary to generate a valid Bearer
        token with sufficient access rights, otherwise the request will return a HTTP
status code 401 or 403. In case of
        those errors, please contact the GeoVille service team for any support.</p>

        <br><b>Description:</b>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>product (str): Name of the CLC+ Backbone product</i></p></li>
        <li><p><i>user_id (str): User specific client ID</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">After the request was successful, a download link
will be returned which
        provides the ordered file.</p>

        """

        order_id = None

        try:
            req_args = api.payload

            payload_check = check_message(req_args)

            if not payload_check[0]:
                error = BadRequestError(f'Payload failed the GeoVille standards:
{payload_check[1]}', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-products-
europe', order_id)
                return {'message': error.to_dict()}, 404

            if not check_user_existence(req_args['user_id'], database_config_file,
database_config_section_api):
                error = NotFoundError('User ID does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-products-
europe', order_id)
                return {'message': error.to_dict()}, 404

            service_id = get_service_id("get_product_europe", database_config_file,
database_config_section_api)
            order_id = generate_orderid(req_args['user_id'], service_id,
json.dumps(req_args))
            gemslog(LogLevel.INFO, f'Request payload: {req_args}', 'API-products-europe',
order_id)

            update_query = """UPDATE customer.service_orders
                                  set status = 'RECEIVED'
                              WHERE
                                  order_id = %s;
                           """
            execute_database(update_query, (order_id,), database_config_file,
database_config_section_api, True)

            dll = "https://s3.waw2-
1.cloudferro.com/swift/v1/AUTH_b9657821e4364f88862ca20a180dc485/clcplus-public/" \
                  "products/CLMS_CLCplus_RASTER_2018_010m_eu_03035_V1_1.tif"

            db_query = """UPDATE
                              customer.service_orders
                          SET
                              status = 'SUCCESS',
                              result = %s,
                              success = true,
                              order_started = NOW(),
                              order_received = NOW(),
                              order_stopped = NOW()
                          WHERE order_id = %s
                       """
            execute_database(db_query, (dll, order_id), database_config_file,
database_config_section_api, True)
```

```
        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-products-
europe', order_id)
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-products-
europe', order_id)
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-products-
europe', order_id)
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Request successful', 'API-products-europe', order_id)
            return {
                    'result': dll
                   }, 200
```

5.1.139 services\backend_api\src\resources\resources_products\nations\nations.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Get Nations API call
#
# Date created: 24.02.2022
# Date last modified: 24.02.2022
#
# __author__  = Johannes Schmid (schmid@geoville.com)
# __version__ = 22.02
#
##########################################################################################
##############################

from check_message.check_message import check_message
from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import service_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_user_existence
from models.general_models.general_models import service_success_response_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_408 import error_408_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from models.models_products.products_models import nations_request_model,
nations_success_response_model
from oauth.oauth2 import require_oauth
import traceback

##########################################################################################
##############################
# Resource definition for the nations API call
##########################################################################################
##############################

#@api.expect(nations_request_model)
@api.header('Content-Type', 'application/json')
class Nations(Resource):
    """ Class for handling the GET request

    This class defines the API call for getting the nation names for the
get_national_product endpoint.
    The class consists of one method which accepts a GET request. For the GET request the
user ID is required,
    defined in the corresponding model.
    """


##########################################################################################
##########################
    # Method for handling the GET request

##########################################################################################
##########################

    @require_oauth(['admin', 'user', 'get_product'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Success', nations_success_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(408, 'Request Timeout', error_408_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self):
        """ GET definition for requesting the nations

        <p style="text-align: justify">This method defines the handler of the GET request
for getting the nation names
        for the endpoint get_national_product. It is a synchronous call and thus, it
returns the requested data
        immediately. To access the service it is necessary to generate a valid Bearer
        token with sufficient access rights, otherwise the request will return a HTTP
status code 401 or 403. In case of
        those errors, please contact the GeoVille service team for any support.</p>

        <br><b>Description:</b>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>user_id (str): User specific client ID</i></p></li>
        </ul>
        <br><b>Result:</b>
        <p style="text-align: justify">After the request was successful, a download link
will be returned which
        provides the ordered file.
        </p>

        """
        order_id = None

        try:

            national_info = {
                "ALBANIA": {"country_code": "AL", "epsg": "02462"},
                "AUSTRIA": {"country_code": "AT", "epsg": "31287"},
                "BOSNIA AND HERZEGOVINA": {"country_code": "BA", "epsg": "03908"},
                "BELGIUM": {"country_code": "BE", "epsg": "03812"},
                "BULGARIA": {"country_code": "BG", "epsg": "32635"},
                "SWITZERLAND": {"country_code": "CH", "epsg": "02056"},
                "CYPRUS": {"country_code": "CY", "epsg": "32636"},
                "CZECH REPUBLIC": {"country_code": "CZ", "epsg": "05514"},
                "GERMANY": {"country_code": "DE", "epsg": "32632"},
                "DENMARK": {"country_code": "DK", "epsg": "25832"},
                "ESTONIA": {"country_code": "EE", "epsg": "03301"},
                "SPAIN": {"country_code": "ES", "epsg": "25830"},
                "SPAIN (CANARIES)": {"country_code": "ESCanaries", "epsg": "32628"},
                "FINLAND": {"country_code": "FI", "epsg": "03067"},
                "FRANCE": {"country_code": "FR", "epsg": "02154"},
                "GREAT BRITAIN": {"country_code": "GB", "epsg": "27700"},
                "GUERNSEY (CHANNEL ISLANDS)": {"country_code": " British Crown
DepenGdBencies)", "epsg": "03108"},
                "GREECE": {"country_code": "GR", "epsg": "02100"},
                "CROATIA": {"country_code": "HR", "epsg": "03765"},
                "HUNGARY": {"country_code": "HU", "epsg": "23700"},
                "IRELAND": {"country_code": "IE", "epsg": "02157"},
                "ICELAND": {"country_code": "IS", "epsg": "05325"},
                "ITALY": {"country_code": "IT", "epsg": "32632"},
                "JERSEY (CHANNEL ISLANDS)": {"country_code": " British Crown
DependenGcBies)", "epsg": "03109"},
                "LIECHTENSTEIN": {"country_code": "LI", "epsg": "02056"},
                "LITHUANIA": {"country_code": "LT", "epsg": "03346"},
                "LUXEMBOURG": {"country_code": "LU", "epsg": "02169"},
                "LATVIA": {"country_code": "LV", "epsg": "03059"},
                "MONTENEGRO": {"country_code": "ME", "epsg": "25834"},
                "FYR OF MACEDONIA": {"country_code": "MK", "epsg": "06204"},
                "MALTA": {"country_code": "MT", "epsg": "23033"},
                "NORTHERN IRELAND": {"country_code": "NI", "epsg": "29903"},
                "NETHERLANDS": {"country_code": "NL", "epsg": "28992"},
                "NORWAY": {"country_code": "NO", "epsg": "25833"},
                "POLAND": {"country_code": "PL", "epsg": "02180"},
                "PORTUGAL": {"country_code": "PT", "epsg": "03763"},
                "PORTUGAL (AZORES CENTRAL AND EASTERN GROUP)": {"country_code":
"PTAzoresCentEast", "epsg": "05015"},
                "PORTUGAL (AZORES WESTERN GROUP)": {"country_code": "PTAzoresWest",
"epsg": "05014"},
                "PORTUGAL (MADEIRA)": {"country_code": "PTMadeira", "epsg": "05016"},
                "ROMANIA": {"country_code": "RO", "epsg": "03844"},
                "SERBIA": {"country_code": "RS", "epsg": "25834"},
                "SWEDEN": {"country_code": "SE", "epsg": "03006"},
                "SLOVENIA": {"country_code": "SI", "epsg": "03912"},
                "SLOVAKIA": {"country_code": "SK", "epsg": "05514"},
                "TURKEY": {"country_code": "TR", "epsg": "00000"},
                "KOSOVO UNDER UNSCR 1244/99": {"country_code": "XK", "epsg": "03909"},
            }
            nations = [n.title() for n in national_info.keys()]

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'AAPI-nations',
order_id)
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'AAPI-nations',
order_id)
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'AAPI-nations',
order_id)
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Request successful', 'AAPI-nations', order_id)
            return {
                       'nations': nations
                   }, 200
```

5.1.140 services\backend_api\src\resources\resources_rabbitmq\delete_rabbitmq_queue\delete_rabbitmq_queue.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# RabbitMQ purge queue API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_401.http_error_401 import UnauthorizedError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import (rabbitmq_host, rabbitmq_management_port,
rabbitmq_password, rabbitmq_user,
                                     rabbitmq_virtual_host)
from init.namespace_constructor import rabbitmq_namespace as api
from lib.auth_header import auth_header_parser
from lib.rabbitmq_helper import list_queue_names, purge_queue
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
from pyrabbit.api import Client
from pyrabbit.http import HTTPError, NetworkError
import traceback


##########################################################################################
##############################
# Resource definition for the purge queue API call
##########################################################################################
##############################

@api.header('Content-Type', 'application/json')
@api.param('queue_name', 'Queue name to be deleted')
class DeleteRabbitMQQueue(Resource):
    """ Class for handling the DELETE request

    This class defines the API call for the RabbitMQ purge queue script. The class
consists of one method which accepts
    a DELETE request. For the DELETE request a JSON with 2 parameters is required, defined
in the corresponding model.

    """


##########################################################################################
##########################
    # Method for handling the DELETE request

##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation successful')
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def delete(self, queue_name):
        """ DELETE definition for removing a queue on the RabbitMQ instance

        <p style="text-align: justify">This method defines the handler for the DELETE
request of the RabbitMQ purge
        queue script. It returns no message body and thus no contents. In contrast it
returns the HTTP status code 204.
        </p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameter:</b>
        <ul>
        <li><p><i>client_id (str): </i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the DELETE request does not contain
any object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any
other response status code
        than 204, indicates an error during the execution.</p>

        """

        try:
            cl = Client(f'{rabbitmq_host}:{rabbitmq_management_port}', rabbitmq_user,
rabbitmq_password)

            if queue_name not in list_queue_names(cl):
                error = NotFoundError(f"Specified queue name does not exist:
{queue_name}", '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
delete_rabbitmq_queue')
                return {'message': error.to_dict()}, 404

            purge_result = purge_queue(cl, rabbitmq_virtual_host, queue_name)

            if purge_result[0] is False:
                error = InternalServerErrorAPI(f'Unexpected Error: {purge_result[1]}', '',
'')
                gemslog(LogLevel.ERROR, 'Successfully deleted queue', 'API-
delete_rabbitmq_queue')
                return {'message': error.to_dict()}, 500

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}', '', '')
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
delete_rabbitmq_queue')
            return {'message': error.to_dict()}, 400

        except HTTPError:
            error = UnauthorizedError('Submitted login credentials are incorrect', '', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_rabbitmq_queue')
            return {'message': error.to_dict()}, 401

        except NetworkError:
            error = ServiceUnavailableError('Could not connect to the specified RabbitMQ
service', '', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_rabbitmq_queue')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', '',
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_rabbitmq_queue')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, 'Successfully deleted queue', 'API-
delete_rabbitmq_queue')
            return '', 204
```

5.1.141 services\backend_api\src\resources\resources_rabbitmq\get_rabbitmq_message_count\get_rabbitmq_message_count.py

```
##########################################################################################
##############################
#
# Copyright (c) 2020, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# RabbitMQ message count API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
#########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_401.http_error_401 import UnauthorizedError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import (rabbitmq_host, rabbitmq_management_port,
rabbitmq_password, rabbitmq_user,
                                     rabbitmq_virtual_host)
from init.namespace_constructor import rabbitmq_namespace as api
from lib.auth_header import auth_header_parser
from lib.rabbitmq_helper import get_queue_message_count, list_queue_names
from models.models_rabbitmq.message_count.message_count_model import message_count_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
from pyrabbit.api import Client
from pyrabbit.http import HTTPError, NetworkError
import traceback


#########################################################################################
##############################
# Resource definition for the list queues API call
#########################################################################################
##############################

@api.header('Content-Type', 'application/json')
@api.param('queue_name', 'Queue name to count the messages of')
class RabbitMQMessageCount(Resource):
    """ Class for handling the GET request

    This class defines the API call for the RabbitMQ message count script. The class
consists of one method which accepts
    a GET request. For the GET request no additional parameters are required.

    """


#########################################################################################
##########################
    # Method for handling the POST request

#########################################################################################
##########################
```

```
    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Operation successful', message_count_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self, queue_name):
        """ GET definition for retrieving the message count of a queue in GEMS

        <p style="text-align: justify">This method defines the handler for the GET request
of the RabbitMQ message
        count script. It returns a dictionary with message count of the queue stored in
the path variable with the
        connection parameters of a RabbitMQ service stored in the environment
variables.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the </p>

        """

        try:
            cl = Client(f'{rabbitmq_host}:{rabbitmq_management_port}', rabbitmq_user,
rabbitmq_password)

            new_queue_name = None
            for filter_item in filter(lambda x: queue_name in x, list_queue_names(cl)):
                new_queue_name = filter_item

            if new_queue_name is None:
                error = NotFoundError(f'Specified queue name does not exist:
{queue_name}', '', '')
                gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
rabbitmq_message_count_gems')
                return {'message': error.to_dict()}, 404

            message_count = get_queue_message_count(cl, rabbitmq_virtual_host,
new_queue_name)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
rabbitmq_message_count')
            return {'message': error.to_dict()}, 400

        except HTTPError:
            error = UnauthorizedError('Submitted login credentials are incorrect', '', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
rabbitmq_message_count')
            return {'message': error.to_dict()}, 401

        except NetworkError:
            error = ServiceUnavailableError('Could not connect to the RabbitMQ service',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
rabbitmq_message_count')
            return {'message': error.to_dict()}, 503
        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
rabbitmq_message_count')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, 'Successful requested the RabbitMQ message count',
'API-rabbitmq_message_count')
            return {"virtual_host": rabbitmq_virtual_host,
                    "queue_name": queue_name,
                    "message_count": message_count}, 200
```

5.1.142 services\backend_api\src\resources\resources_rabbitmq\get_rabbitmq_queues\get_rabbitmq_queues.py

```
###########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# RabbitMQ list queues GEMS API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
###########################################################################################
##############################

from error_classes.http_error_401.http_error_401 import UnauthorizedError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import rabbitmq_host, rabbitmq_management_port,
rabbitmq_password, rabbitmq_user
from init.namespace_constructor import rabbitmq_namespace as api
from lib.auth_header import auth_header_parser
from models.models_rabbitmq.list_queues.list_queues_model import rabbitmq_response_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
from pyrabbit.api import Client
from pyrabbit.http import HTTPError, NetworkError
import traceback


###########################################################################################
##############################
# Resource definition for the list queues API call
###########################################################################################
##############################
```

```
@api.header('Content-Type', 'application/json')
class RabbitMQListQueues(Resource):
    """ Class for handling the GET request

    This class defines the API call for the RabbitMQ list queues script. The class
consists of one method which accepts
    a GET request. For the GET request no additional parameters are required.

    """


##########################################################################################
##########################
    # Method for handling the GET request

##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Operation successful', rabbitmq_response_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self):
        """ GET definition for retrieving the list of available queues in GEMS

        <p style="text-align: justify">This method defines the handler for the GET request
of the RabbitMQ list queues
        script. It returns a list of all available queues with the connection parameters
stored in the environment
        variables.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the </p>

        """


        try:
            cl = Client(f'{rabbitmq_host}:{rabbitmq_management_port}', rabbitmq_user,
rabbitmq_password)

        except HTTPError:
            error = UnauthorizedError('Login credentials derived from the environment
variables are incorrect', '', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
rabbitmq_queues')
            return {'messages': error.to_dict()}, 401

        except NetworkError:
            error = ServiceUnavailableError('Could not connect to the RabbitMQ service',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
rabbitmq_queues')
            return {'messages': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
```

```
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
rabbitmq_queues')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, 'Successfully retrieved the queue names', 'API-
rabbitmq_queues')
            return {'available_queues': [q['name'] for q in cl.get_queues()]}, 200
```

5.1.143 services\backend_api\src\resources\resources_rabbitmq\get_rabbitmq_server_status\get_rabbitmq_server_status.py

```
###########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# RabbitMQ server status GEMS API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
###########################################################################################
##############################

from error_classes.http_error_401.http_error_401 import UnauthorizedError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import rabbitmq_host, rabbitmq_management_port,
rabbitmq_password, rabbitmq_user
from init.namespace_constructor import rabbitmq_namespace as api
from lib.auth_header import auth_header_parser
from models.models_rabbitmq.server_status.server_status_model import
rabbitmq_response_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
from pyrabbit.api import Client
from pyrabbit.http import HTTPError, NetworkError
import traceback


###########################################################################################
##############################
# Resource definition for the server status API call
###########################################################################################
##############################

@api.header('Content-Type', 'application/json')
class RabbitMQServerStatus(Resource):
    """ Class for handling the GET requests
    This class defines the API call for the RabbitMQ server status script. The class
consists of one method which
    accepts a GET request. For the GET request no additional parameters are required.

    """


##########################################################################################
##########################
    # Method for handling the GET request
##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Success', rabbitmq_response_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    @api.marshal_with(rabbitmq_response_model)
    def get(self):
        """ GET definition for retrieving the server status of GEMS

        <p style="text-align: justify">This method defines the handler for the GET request
of the RabbitMQ server status
        script. It returns about the server status of GEMS RabbitMQ instance with the
connection parameters stored in
        the environment variables.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the </p>

        """

        stat_obj = {'host': rabbitmq_host}

        try:
            cl = Client(f'{rabbitmq_host}:{rabbitmq_management_port}', rabbitmq_user,
rabbitmq_password)
            server_stat = cl.is_alive()

            if server_stat:
                stat_obj['server_status'] = 'RabbitMQ service is up and running'

            else:
                stat_obj['server_status'] = 'The Virtual machine is up but the RabbitMQ
service is not available'

        except HTTPError:
            error = UnauthorizedError('Login credentials derived from the environment
variable are incorrect', '', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
rabbitmq_server_status')
            return {'message': error.to_dict()}, 401

        except NetworkError:
            error = ServiceUnavailableError('Could not connect to the RabbitMQ service',
'', '')
```

```
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
rabbitmq_server_status')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
rabbitmq_server_status')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, 'Successfully retrieved the server status', 'API-
rabbitmq_server_status')
            return stat_obj, 200
```

5.1.144 services\backend_api\src\resources\resources_rabbitmq\get_rabbitmq_users\get_rabbitmq_users.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# RabbitMQ list users GEMS API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_401.http_error_401 import UnauthorizedError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import rabbitmq_host, rabbitmq_management_port,
rabbitmq_password, rabbitmq_user
from init.namespace_constructor import rabbitmq_namespace as api
from lib.auth_header import auth_header_parser
from models.models_rabbitmq.list_users.list_users_model import users_response_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
from pyrabbit.api import Client
from pyrabbit.http import HTTPError, NetworkError
import traceback


########################################################################################
##############################
# Resource definition for the list users API call
########################################################################################
##############################
```

```
@api.header('Content-Type', 'application/json')
class RabbitMQUsers(Resource):
    """ Class for handling GET request

    This class defines the API call for the RabbitMQ list users script. The class consists
of one method which accepts
    a GET request. For the GET request no additional parameters are required.

    """


##########################################################################################
##########################
    # Method for handling the GET request

##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Operation successful', users_response_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self):
        """ GET definition for requesting the list of all available users in GEMS

        <p style="text-align: justify">This method defines the handler for the GET request
of the RabbitMQ list users
        script. It returns a list of all available user in GEMS with the connection
parameters stored in the environment
        variables.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the </p>

        """


        try:
            user_list = []
            cl = Client(f'{rabbitmq_host}:{rabbitmq_management_port}', rabbitmq_user,
rabbitmq_password)

            for user in cl.get_users():
                user_list.append({'name': user['name'], 'permission': user['tags']})

        except HTTPError:
            error = UnauthorizedError('Login credentials derived from the environment
variable are incorrect', '', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-rabbitmq_users')
            return {'message': error.to_dict()}, 401

        except NetworkError:
            error = ServiceUnavailableError('Could not connect to the specified RabbitMQ
service', '', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-rabbitmq_users')
            return {'message': error.to_dict()}, 503
        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-rabbitmq_users')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, 'Successfully retrieved the user names', 'API-
rabbitmq_users')
            return {'users': user_list}, 200
```

5.1.145 services\backend_api\src\resources\resources_rabbitmq\get_rabbitmq_vhosts\get_rabbitmq_vhosts.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# RabbitMQ list virtual hosts GEMS API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.http_error_401.http_error_401 import UnauthorizedError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import rabbitmq_host, rabbitmq_management_port,
rabbitmq_password, rabbitmq_user
from init.namespace_constructor import rabbitmq_namespace as api
from lib.auth_header import auth_header_parser
from models.models_rabbitmq.list_vhosts.list_vhosts_model import
virtual_host_response_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
from pyrabbit.api import Client
from pyrabbit.http import HTTPError, NetworkError
import traceback


##########################################################################################
##############################
# Resource definition for the list virtual hosts API call
##########################################################################################
##############################

@api.header('Content-Type', 'application/json')
class RabbitMQVHosts(Resource):
    """ Class for handling the GET requests
    This class defines the API call for the RabbitMQ list virtual hosts script. The class
consists of one method which
    accepts a GET request. For the GET request no additional parameters are required.

    """


###############################################################################################
###########################
    # Method for handling the GET request

###############################################################################################
###########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Operation successful', virtual_host_response_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self):
        """ GET definition for requesting the list of all available virtual hosts of the
RabbitMQ instance

        <p style="text-align: justify">This method defines the handler for the GET request
of the RabbitMQ list virtual
        hosts script. It returns a list of all available virtual hosts in GEMS with the
connection parameters stored in
        the environment variables.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the </p>

        """


        try:
            cl = Client(f'{rabbitmq_host}:{rabbitmq_management_port}', rabbitmq_user,
rabbitmq_password)

        except HTTPError:
            error = UnauthorizedError('Login credentials derived from the environment
variable are incorrect', '', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
rabbitmq_vhosts')
            return {'message': error.to_dict()}, 401

        except NetworkError:
            error = ServiceUnavailableError('Could not connect to the specified RabbitMQ
service', '', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
rabbitmq_vhosts')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
rabbitmq_vhosts')
            return {'message': error.to_dict()}, 500
        else:
            gemslog(LogLevel.INFO, 'Successfully retrieved the virtual hosts', 'API-
rabbitmq_vhosts')
            return {'virtual_hosts': cl.get_vhost_names()}, 200
```

5.1.146 services\backend_api\src\resources\resources_rois\create_roi\create_roi.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Create region of interest API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import rois_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_user_existence
from lib.general_helper_methods import validate_geojson
from lib.hashing_helper import generate_roi_id_hash
from models.models_rois.roi_models import roi_id_model, roi_request_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import json
import traceback


########################################################################################
##############################
# Resource definition for the create region of interest API call
########################################################################################
##############################

@api.expect(roi_request_model)
@api.header('Content-Type', 'application/json')
class CreateROI(Resource):
    """ Class for handling the POST request
    This class defines the API call for the create  region of interest script. The class
consists of one method which
    accepts a POST request. For the POST request a JSON with several parameters is
required and defined in the
    corresponding model.

    """


##########################################################################################
###########################
    # Method for handling the POST request

##########################################################################################
###########################

    @require_oauth(['admin', 'user'])
    @api.expect(auth_header_parser)
    @api.response(201, 'Operation successful', roi_id_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for creating a new region of interest

        <p style="text-align: justify">This method defines the handler for the POST
request of the create region of
        interest script. It returns a message wrapped into a dictionary about the status
of the insertion operation.</p>

        <br><b>Description:</b>
        <p style="text-align: justify">Most of the GEMS services work with an area of
interest as input parameter. For
        this, the GEMS API offers the consumer the possibility to create his own area of
interest. In the GEMS concept,
        it is called region of interest. Each API consumer can create as many region of
interests as necessary.</p>


        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>name (str): Name identifier for the region of interest</i></p></li>
        <li><p><i>description (str): Longer description for the region of interest but not
required</i></p></li>
        <li><p><i>user_id (str): User specific client ID to link the region of interst to
a user</i></p></li>
        <li><p><i>geoJSON (str): GeoJSON definition of the region of interest without any
additional attributes</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the GET request is a JSON which
contains an object with one key
        value pair. The region of interest ID is required as input parameter for several
GEMS services and should not be
        lost.</p>
        <ul>
        <li><p><i>roi_id: Unique identifier of a region of interest</i></p></li>
        </ul>

        """
        try:
            req_args = api.payload
            gemslog(LogLevel.INFO, f'Request payload: {req_args}', 'API-create_roi')

            if not check_user_existence(req_args['user_id'], database_config_file,
database_config_section_api):
                error = NotFoundError('User ID does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
create_roi')
                return {'message': error.to_dict()}, 404

            validation_res = validate_geojson(req_args['geoJSON'], database_config_file,
database_config_section_api)
            if False in validation_res:
                error = BadRequestError(f'GeoJSON is invalid: {validation_res[1]}',
api.payload, '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
create_roi')
                return {'message': error.to_dict()}, 400

            description = None if 'description' not in req_args else
req_args['description']
            roi_id = generate_roi_id_hash(req_args['user_id'], req_args['name'])

            db_query = """INSERT INTO customer.region_of_interests
                          (
                              roi_id, roi_name, description, customer_id, geom
                          )
                          VALUES
                          (
                              %s, %s, %s, %s,
ST_Force2D(ST_SetSRID(ST_GeomFromGeoJSON(%s), 4326))
                          )
                       """

            execute_database(db_query, (roi_id, req_args['name'], description,
req_args['user_id'],
                                        json.dumps(req_args['geoJSON'])),
database_config_file,
                             database_config_section_api, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-create_roi')
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-create_roi')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-create_roi')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Created ROI with ID: {roi_id}', 'API-create_roi')
            return {'roi_id': roi_id}, 201
```

5.1.147 services\backend_api\src\resources\resources_rois\delete_all_rois\delete_all_rois.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Delete all regions of interest API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import rois_namespace as api
from lib.auth_header import auth_header_parser
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


########################################################################################
##############################
# Resource definition for the delete all regions of interest API call
########################################################################################
##############################

@api.header('Content-Type', 'application/json')
class DeleteAllROIs(Resource):
    """ Class for handling the DELETE request

    This class defines the API call for the delete all regions of interest script. The
class consists of one method
    which accepts a DELETE request. For the DELETE request, no parameters are required.

    """


########################################################################################
##########################
    # Method for handling the DELETE request

########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation successful')
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def delete(self):
        """ DELETE definition for removing all regions of interest

        This method defines the handler for the DELETE request of the delete all regions
of interest script. It returns
        no message body and thus no contents. In contrast it returns the HTTP status code
204.

        <br><b>Description:</b>
        <p style="text-align: justify">This GEMS service can be used by a GEMS API
consumer in order to delete all
        stored region of interests belonging to GEMS customer by specifying the
corresponding customer ID. As common use
        in API design, all DELETE request will not provide any return message from
service. Only the HTTP status code
        should be checked for retrieving the result of the request.</p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the DELETE request does not contain
any object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any
other response status code
        than 204, indicated an error during the execution.</p>

        """

        db_query = "UPDATE customer.region_of_interests SET deleted_at = NOW()"

        try:
            execute_database(db_query, (), database_config_file,
database_config_section_api, True)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_all_rois')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_all_rois')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, 'Successfully deleted all ROIs', 'API-delete_all_rois')
            return '', 204
```

5.1.148 services\backend_api\src\resources\resources_rois\delete_roi_by_id\delete_roi_by_id.py

```
##########################################################################################
################################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
```

```
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Delete region of interest by ID API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import rois_namespace as api
from lib.auth_header import auth_header_parser
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


##########################################################################################
##############################
# Resource definition for the delete region of interest by ID API call
##########################################################################################
##############################

@api.header('Content-Type', 'application/json')
@api.param('roi_id', 'ROI ID to be deleted')
class DeleteROIByID(Resource):
    """ Class for handling the DELETE request

    This class defines the API call for the delete region of interest by ID script. The
class consists of one method
    which accepts a DELETE request. For the DELETE request a JSON with one parameter is
required and defined in the
    corresponding model.

    """


##########################################################################################
##########################
    # Method for handling the DELETE request

##########################################################################################
##########################

    @require_oauth(['admin', 'user'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation successful')
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def delete(self, roi_id):
        """ DELETE definition for removing a region of interest by ID

        <p style="text-align: justify">This method defines the handler for the DELETE
request for the delete region of
        interest by ID script. It returns no message body and thus no contents. In
contrast it returns the HTTP status
        code 204.</p>

        <br><b>Description:</b>
        <p style="text-align: justify">This GEMS service can be used by a GEMS API
consumer in order to delete a stored
        region of interest by specifying the region of interest ID. As common use in API
design, all DELETE request will
        not provide any return message from service. Only the HTTP status code should be
checked for retrieving the
        result of the request.</p>


        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameters:</b>
        <ul>
        <li><p><i>roi_id (str): Unique identifier for the region of interest</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the DELETE request does not contain
any object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any
other response status code
        than 204, indicated an error during the execution.</p>

        """

        try:
            db_query = "UPDATE customer.region_of_interests SET deleted_at = NOW() WHERE
roi_id = %s"
            execute_database(db_query, (roi_id,), database_config_file,
database_config_section_api, True)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_roi_by_id')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_roi_by_id')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Deleted ROI with ID {roi_id}', 'API-
delete_roi_by_id')
            return '', 204
```

5.1.149 services\backend_api\src\resources\resources_rois\delete_roi_by_user_id\delete_roi_by_user_id.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Delete region of interest by ID API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import rois_namespace as api
from lib.auth_header import auth_header_parser
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


##########################################################################################
##############################
# Resource definition for the delete region of interest by ID API call
##########################################################################################
##############################

@api.header('Content-Type', 'application/json')
@api.param('user_id', 'User ID to be deleted')
class DeleteROIByUserID(Resource):
    """ Class for handling the DELETE request

    This class defines the API call for the delete region of interest by ID script. The
class consists of one method
    which accepts a DELETE request. For the DELETE request a JSON with one parameter is
required and defined in the
    corresponding model.

    """


##########################################################################################
##########################
    # Method for handling the DELETE request

##########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation successful')
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def delete(self, user_id):
        """ DELETE definition for removing a region of interest by a customer ID

        <p style="text-align: justify">This method defines the handler for the DELETE 
request of the delete region of
        interests by customer ID script. It returns no message body and thus no contents. 
In contrast it returns the
        HTTP status code 204.</p>

        <br><b>Description:</b>
        <p style="text-align: justify">This GEMS service can be used by a GEMS API 
consumer in order to delete all
        stored region of interests belonging to GEMS customer by specifying the 
corresponding customer ID. As common use
        in API design, all DELETE request will not provide any return message from 
service. Only the HTTP status code
        should be checked for retrieving the result of the request.</p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameters:</b>
        <ul>
        <li><p><i>customer_id (str): Unique identifier for a GEMS customer</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the DELETE request does not contain 
any object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any 
other response status code
        than 204, indicated an error during the execution.</p>

        """

        try:
            db_query = "UPDATE customer.region_of_interests SET deleted_at = NOW() WHERE 
roi_id = %s"
            execute_database(db_query, (user_id,), database_config_file, 
database_config_section_api, True)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server', 
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_roi_by_user')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload, 
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
delete_roi_by_user')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Deleted ROI with user ID {user_id}', 'API-
delete_roi_by_user')
            return '', 204
```

5.1.150 services\backend_api\src\resources\resources_rois\get_all_rois\get_all_rois.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Get all regions of interest API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import read_from_database_all_rows
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import rois_namespace as api
from lib.auth_header import auth_header_parser
from models.models_rois.roi_models import several_roi_response_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import json
import traceback


########################################################################################
##############################
# Resource definition for the get all regions of interest API call
########################################################################################
##############################

@api.header('Content-Type', 'application/json')
class GetAllROIs(Resource):
    """ Class for handling the GET request

    This class defines the API call for the get all regions of interest script. The class
consists of one method which
    accepts a GET request. For the GET request no parameters are required.

    """


########################################################################################
##########################
    # Method for handling the GET request

########################################################################################
##########################

    @require_oauth(['admin'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Operation successful', several_roi_response_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self):
        """ GET definition for retrieving all regions of interest

        <p style="text-align: justify">This method defines the handler for the GET request
of the get all regions of
        interest script. It returns a list of regions of interest of all data sets in the
database if the required table
        is not empty.</p>

        <br><b>Description:</b>
        <p style="text-align: justify">This GEMS service provides an overview of all
stored region of interests in the
        database belonging to a GEMS customer. By specifying the customer ID all the
available region of interests will
        be returned and listed in an detailed manner with all the parameters submitted in
the creation process.</p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the GET request is a JSON </p>

        """

        db_query = """SELECT
                            roi_id, roi_name, description, customer_id, ST_AsGeoJSON(geom),
created_at
                        FROM
                            customer.region_of_interests
                        WHERE
                            deleted_at IS NULL;
                      """

        try:
            roi_data = read_from_database_all_rows(db_query, (), database_config_file,
database_config_section_api, True)
            res_array = []

            for roi in roi_data:
                roi_obj = {
                    'roi_id': roi[0],
                    'roi_name': roi[1],
                    'description': roi[2],
                    'customer_id': roi[3],
                    'geoJSON': json.loads(roi[4]),
                    'creation_date': roi[5].strftime("%Y-%m-%dT%H:%M:%S")
                }

                res_array.append(roi_obj)

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-get_all_rois')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-get_all_rois')
            return {'message': error.to_dict()}, 500
        else:
            gemslog(LogLevel.INFO, f'Successful response: {res_array}', 'API-
get_all_rois')
            return {'rois': res_array}, 200
```

5.1.151 services\backend_api\src\resources\resources_rois\get_roi_by_id\get_roi_by_id.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Get region of interest by ID API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import read_from_database_one_row
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import rois_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_roi_existence
from models.models_rois.roi_models import single_roi_response_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import json
import traceback


########################################################################################
##############################
# Resource definition for the get region of interest by ID API call
########################################################################################
##############################

@api.header('Content-Type', 'application/json')
@api.param('roi_id', 'ROI ID to be requested')
class GetROIByID(Resource):
    """ Class for handling the GET request

    This class defines the API call for the get region of interest by ID script. The class
consists of one method which
```

```
    accepts a GET request. For the GET request a JSON with one parameter is required and
defined in the corresponding
    model.

    """


##########################################################################################
##########################
    # Method for handling the GET request

##########################################################################################
##########################

    @require_oauth(['admin', 'user'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Operation successful', single_roi_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found Error', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self, roi_id):
        """ GET definition for retrieving a region of interest by ID

        <p style="text-align: justify">This method defines the handler for the GET request
of the get region of interest
        by ID script. It returns the region of interest for the given ID if it exists in
the database.</p>

        <br><b>Description:</b>
        <p style="text-align: justify">This GEMS service provides an overview of all
stored region of interests in the
        database belonging to a GEMS customer. By specifying the customer ID all the
available region of interests will
        be returned and listed in an detailed manner with all the parameters submitted in
the creation process.</p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameters:</b>
        <ul>
        <li><p><i>roi_id (str): Unique identifier for the region of interest</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the GET request is a JSON</p>

        """

        db_query = """SELECT
                          roi_id, roi_name, description, customer_id, ST_AsGeoJSON(geom),
created_at
                      FROM
                          customer.region_of_interests
                      WHERE
                          roi_id = %s AND
                          deleted_at IS NULL;
                   """

        try:
            gemslog(LogLevel.INFO, f'Request path parameter: {roi_id}', 'API-
get_roi_by_id')
```

```
            if not check_roi_existence(roi_id, database_config_file,
database_config_section_api):
                error = NotFoundError('ROI ID does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
get_roi_by_id')
                return {'message': error.to_dict()}, 404

            roi_data = read_from_database_one_row(db_query, (roi_id,),
database_config_file,
                                                  database_config_section_api, True)

            roi_obj = {
                'roi_id': roi_data[0],
                'roi_name': roi_data[1],
                'description': roi_data[2],
                'customer_id': roi_data[3],
                'geoJSON': json.loads(roi_data[4]),
                'creation_date': str(roi_data[5])
            }

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
get_roi_by_id')
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-get_roi_by_id')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-get_roi_by_id')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Successful response: {roi_obj}', 'API-get_roi_by_id')
            return roi_obj, 200
```

5.1.152 services\backend_api\src\resources\resources_rois\get_roi_by_user_id\get_roi_by_user_id.py

```
###########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Get region of interest by ID API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
```

```
##########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import read_from_database_all_rows
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import rois_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_user_existence
from models.models_rois.roi_models import several_roi_response_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import json
import traceback


##########################################################################################
##############################
# Resource definition for the get region of interest by ID API call
##########################################################################################
##############################

@api.header('Content-Type', 'application/json')
@api.param('user_id', 'User ID to be deleted')
class GetROIByUserID(Resource):
    """ Class for handling the GET request

    This class defines the API call for the get region of interest by ID script. The class
consists of one method which
    accepts a GET request. For the GET request a JSON with one parameter is required
    and defined in the corresponding model.

    """


##########################################################################################
##########################
    # Method for handling the GET request

##########################################################################################
##########################

    @require_oauth(['admin', 'user'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Operation successful', several_roi_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found Error', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self, user_id):
        """ GET definition for retrieving a region of interest by user ID

        <p style="text-align: justify">This method defines the handler for the GET request
of the get region of interest
        by user ID script. It returns the region of interest for the given ID if it exists
in the database.</p>
```

```
        <br><b>Description:</b>
        <p style="text-align: justify">This GEMS service provides an overview of all
stored region of interests in the
        database belonging to a GEMS customer. By specifying the customer ID all the
available region of interests will
        be returned and listed in an detailed manner with all the parameters submitted in
the creation process.</p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameters:</b>
        <ul>
        <li><p><i>customer_id (str): Unique identifier for a GEMS customer</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the GET request is a JSON which
contains a list of objects with
        four key value pairs. Those pairs are conform with the parameters submitted during
the region of interest
        creation</p>
        <ul>
        <li><p><i>name</i></p></li>
        <li><p><i>description</i></p></li>
        <li><p><i>customer_id</i></p></li>
        <li><p><i>geoJSON</i></p></li>
        </ul>

        """

        db_query = """SELECT
                            roi_id, roi_name, description, customer_id, ST_AsGeoJSON(geom),
created_at
                        FROM
                            customer.region_of_interests
                        WHERE
                            customer_id = %s AND
                            deleted_at IS NULL;
                    """

        try:
            gemslog(LogLevel.INFO, f'Request path parameter: {user_id}', 'API-
get_roi_by_user')

            if not check_user_existence(user_id, database_config_file,
database_config_section_api):
                error = NotFoundError('User ID does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
get_roi_by_user')
                return {'message': error.to_dict()}, 404

            roi_data = read_from_database_all_rows(db_query, (user_id,),
database_config_file,
                                                   database_config_section_api, True)

            res_array = []

            for roi in roi_data:
                roi_obj = {
                    'roi_id': roi[0],
                    'roi_name': roi[1],
                    'description': roi[2],
                    'customer_id': roi[3],
                    'geoJSON': json.loads(roi[4]),
                    'creation_date': str(roi[5])
                }

                res_array.append(roi_obj)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
get_roi_by_user')
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_roi_by_user')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
get_roi_by_user')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Successful response: {res_array}', 'API-
get_roi_by_user')
            return {'rois': res_array}, 200
```

5.1.153 services\backend_api\src\resources\resources_rois\set_roi_attributes_by_id\set_roi_attributes_by_id.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Updates particular attributes of a region of interest
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
##########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import rois_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_roi_existence, check_user_existence
from lib.general_helper_methods import parameter_and_value_list_generation,
validate_geojson
from models.models_rois.roi_models import roi_attributes_request
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import traceback


##########################################################################################
##############################
# Resource definition for the create customer API call
##########################################################################################
##############################

@api.expect(roi_attributes_request)
@api.header('Content-Type', 'application/json')
class UpdateROIAttributes(Resource):
    """ Class for handling the PATCH request

    This class defines the API call for the update region of interest script. The class
consists of one method which
    accepts a PATCH request. For the PATCH request a JSON with several parameters is
required and defined in the
    corresponding model.

    """


##########################################################################################
##########################
    # Method for handling the PATCH request

##########################################################################################
##########################

    @require_oauth(['admin', 'user'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation successful')
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def patch(self):
        """ PATCH definition for updating particular attributes of a region of interest

        <p style="text-align: justify">This method defines the handler for the PATCH
request of the set region of
        interest attribute script. Since this request call</p>

        <br><b>Description:</b>
        <p style="text-align: justify">This GEMS service was designed to quickly update an
entire region of interest
        entity. Thus all required attributes must be submitted during the request call. As
common use in API design, all
        PATCH request will not provide any return message from service. Only the HTTP
status code should be checked for
        retrieving the result of the request.</p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>roi_id (str): Unique identifier for a region of interest</i></p></li>
        <li><p><i>name (str) (optional): Name identifier for the region of
interest</i></p></li>
        <li><p><i>description (str) (optional): Longer description for the region of
interest but not required</i></p></li>
        <li><p><i>customer_id (str) (optional): User specific client ID to link the region
of interst to a user</i></p></li>
        <li><p><i>geoJSON (str) (optional): GeoJSON definition of the region of interest
without any additional attributes</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the PATCH request does not contain
any object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any
other response status code
        than 204, indicated an error during the execution.</p>

        """

        try:
            req_args = api.payload
            gemslog(LogLevel.INFO, f'Request payload: {req_args}', 'API-
update_roi_entity_by_id')
            param_list, val_list = parameter_and_value_list_generation(req_args)

            if not param_list and not val_list:
                gemslog(LogLevel.INFO, 'No ROI update necessary', 'API-
update_roi_entity_by_id')
                return '', 204

            if not check_roi_existence(req_args['roi_id'], database_config_file,
database_config_section_api):
                error = NotFoundError('The ROI ID does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
update_roi_entity_by_id')
                return {'message': error.to_dict()}, 404

            if 'customer_id' in req_args and not check_user_existence(req_args['user_id'],
database_config_file,

database_config_section_api):
                error = NotFoundError('Customer ID does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
update_roi_entity_by_id')
                return {'message': error.to_dict()}, 404

            if 'geoJSON' in req_args:
                validation_res = validate_geojson(req_args['geoJSON'],
database_config_file, database_config_section_api)
                if False in validation_res:
                    error = BadRequestError(f'GeoJSON is invalid: {validation_res[1]}',
api.payload, '')
                    gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
update_roi_entity_by_id')
                    return {'message': error.to_dict()}, 400

            db_query = f"""UPDATE
                               customer.region_of_interests
                           SET
                               {', '.join(param_list)}
                           WHERE
                               roi_id = %s;
                        """
            val_list.append(req_args['roi_id'])
            execute_database(db_query, val_list, database_config_file,
database_config_section_api, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
update_roi_entity_by_id')
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
update_roi_entity_by_id')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
update_roi_entity_by_id')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, 'Updated ROI successfully', 'API-
update_roi_entity_by_id')
            return '', 204
```

5.1.154 services\backend_api\src\resources\resources_rois\update_roi_entity_by_id\update_roi_entity_by_id.py

```
###########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Update region of interest API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
###########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import rois_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_roi_existence, check_user_existence
from lib.general_helper_methods import validate_geojson
from models.models_rois.roi_models import roi_entity_request
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from oauth.oauth2 import require_oauth
import json
import traceback


###########################################################################################
##############################
# Resource definition for the create customer API call
###########################################################################################
##############################

@api.expect(roi_entity_request)
@api.header('Content-Type', 'application/json')
class UpdateROIEntity(Resource):
    """ Class for handling the PUT request

    This class defines the API call for the update region of interest script. The class
consists of one method which
    accepts a PUT request. For the PUT request a JSON with several parameters is required
and defined in the
    corresponding model.

    """


###########################################################################################
##########################
    # Method for handling the PUT request

###########################################################################################
##########################

    @require_oauth(['admin', 'user'])
    @api.expect(auth_header_parser)
    @api.response(204, 'Operation successful')
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def put(self):
        """ PUT definition for updating a complete region of interest entity

        <p style="text-align: justify">This method defines the handler for the PUT request
of the update region of
        interest script. It returns a message wrapped into a dictionary about the status
of the update operation.</p>

        <br><b>Description:</b>
        <p style="text-align: justify">This GEMS service was designed to quickly update a
single or a couple of
        attributes of a region of interest without submitting attributes which should not
be changed, that's why some
        attributes can be submitted as optional parameters. As common use in API design,
all PATCH request will not
        provide any return message from service. Only the HTTP status code should be
checked for retrieving the result
        of the request.</p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>roi_id (str): Unique identifier for a region of interest</i></p></li>
        <li><p><i>name (str): Name identifier for the region of interest</i></p></li>
        <li><p><i>description (str): Longer description for the region of interest but not 
required</i></p></li>
        <li><p><i>customer_id (str): User specific client ID to link the region of interst 
to a user</i></p></li>
        <li><p><i>geoJSON (str): GeoJSON definition of the region of interest without any 
additional attributes</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the PUT request does not contain any 
object or message in the
        response body. The HTTP status signalise the result of the submitted request. Any 
other response status code
        than 204, indicated an error during the execution.</p>

        """

        try:
            req_args = api.payload
            gemslog(LogLevel.INFO, f'Request payload: {req_args}', 'API-
update_roi_entity_by_id')

            if not check_roi_existence(req_args['roi_id'], database_config_file, 
database_config_section_api):
                error = NotFoundError('ROI ID does not exist', '', '')
                gemslog(LogLevel.WARNING, 'ROI ID does not exist', 'API-
update_roi_entity_by_id')
                return {'message': error.to_dict()}, 404

            if not check_user_existence(req_args['user_id'], database_config_file, 
database_config_section_api):
                error = NotFoundError('User ID does not exist', '', '')
                gemslog(LogLevel.WARNING, 'Customer ID does not exist', 'API-
update_roi_entity_by_id')
                return {'message': error.to_dict()}, 404

            validation_res = validate_geojson(req_args['geoJSON'], database_config_file, 
database_config_section_api)
            if False in validation_res:
                error = BadRequestError(f'GeoJSON is invalid: {validation_res[1]}', 
api.payload, '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
update_roi_entity_by_id')
                return {'message': error.to_dict()}, 400

            description = None if 'description' not in req_args else 
req_args['description']

            db_query = """UPDATE
                            customer.region_of_interests
                        SET
                            roi_name = %s,
                            description = %s,
                            customer_id = %s,
                            geom = ST_Force2D(ST_SetSRID(ST_GeomFromGeoJSON(%s), 4326))
                        WHERE
                            roi_id = %s;
                       """

            execute_database(db_query, (req_args['name'], description, 
req_args['user_id'],
                                                json.dumps(req_args['geoJSON']),
req_args['roi_id']), database_config_file,
                                   database_config_section_api, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
update_roi_entity_by_id')
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
update_roi_entity_by_id')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
update_roi_entity_by_id')
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, 'Updated ROI successfully', 'API-
update_roi_entity_by_id')
            return '', 204
```

5.1.155 services\backend_api\src\resources\resources_services\batch_classification_production \batch_classification_production.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Batch classification production API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from geoville_ms_orderid_generator.generator import generate_orderid
from init.init_env_variables import database_config_file, database_config_section_api,
database_config_section_oauth
from init.namespace_constructor import service_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_service_name_existence, get_service_id,
query_user_id
from lib.general_helper_methods import publish_to_queue
from models.general_models.general_models import service_success_response_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from models.models_services.batch_classification.batch_classification_model import
batch_classification_production_model
from oauth.oauth2 import require_oauth
import flask
import json
import traceback


########################################################################################
##############################
# Resource definition for the batch classification production API call
########################################################################################
##############################

@api.expect(batch_classification_production_model)
@api.header('Content-Type', 'application/json')
class BatchClassificationProduction(Resource):
    """ Class for handling the POST request

    This class defines the API call for starting the batch classification production
workflow. The class consists of one
    methods which accepts a POST request. For the POST request a JSON with several
parameters is required, defined in
    the corresponding model.

    """


########################################################################################
##########################
    # Method for handling the POST request

########################################################################################
##########################

    @require_oauth(['admin', 'batch_classification', 'gaf'])
    @api.expect(auth_header_parser)
    @api.response(202, 'Order Received', service_success_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for triggering the batch classification production workflow

        <p style="text-align: justify">This method defines the handler of the POST request
for starting off the batch
        classification production processing chain within the GEMS service architecture.
It is an asynchronous call and
        thus, it does not return the requested data immediately but generates an order ID.
After the request has been
        submitted successfully, a message to a RabbitMQ queue will be send. A listener in
the backend triggers the
        further procedure, starting with the job scheduling of the order. The final result
will be stored in a database
        in form of a link and can be retrieved via the browser. To access the service it
is necessary to generate a
```

```
        valid Bearer token with sufficient access rights, otherwise the request will
return a HTTP status code 401 or
        403. In case of those errors, please contact the GeoVille service team for any
support.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>params (dict): Dictionary with a set of parameters</i></p></li>
        <li><p><i>user_id (str): User specific client ID</i></p></li>
        <li><p><i>service_name (str): Unique name of the service to be called. Name should
not be changed</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">After the request was successfully received by the
GEMS API, an order ID will be
        created for the submitted job in the GEMS backend and stored in a database with
all necessary information for
        the consumer and the system itself. The initial status of the order in the
database will be set to 'received'.
        After that the order ID will be returned in form of a Hypermedia link which
enables the possibilities to
        programmatically check the status the order by a piece of code of a GEMS API
consumer. In the following the
        status of the order can be queried by using the GEMS service route listed
below:</p>

        <ul><li><p><i>/services/order_status/{order_id}</i></p></li></ul>

        <p style="text-align: justify">The status of the order will be updated whenever
the processing chain reaches the
        next step in its internal calculation. In case of success, the user will receive a
link, which provides the
        ordered file. Additionally an e-mail notification is enabled, which sends out an
e-mail in case of success, with
        the resulting link or in case of failure, with a possible error explanation.
        </p>

        """

        order_id = None

        try:
            req_args = api.payload
            access_token = flask.request.headers.get('Authorization').split("
")[1].strip()

            user_id = query_user_id(access_token, database_config_file,
database_config_section_oauth)

            if not check_service_name_existence(req_args['service_name'],
database_config_file,
                                                database_config_section_api):
                error = NotFoundError('Service name does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
batch_classification_production')
                return {'message': error.to_dict()}, 404

            service_id = get_service_id(req_args['service_name'], database_config_file,
database_config_section_api)
            order_id = generate_orderid(user_id, service_id, json.dumps(req_args))
            publish_to_queue(req_args['service_name'], order_id, req_args)

            update_query = """UPDATE customer.service_orders
                                  set status = 'RECEIVED'
                              WHERE
                                  order_id = %s;
                           """
            execute_database(update_query, (order_id,), database_config_file,
database_config_section_api, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
batch_classification_production', order_id)
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
batch_classification_production', order_id)
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
batch_classification_production', order_id)
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Request successful with ID {order_id}', 'API-
batch_classification_production', order_id)
            return {
                       'message': 'Your order has been successfully submitted',
                       'links': {
                           'href': f'/services/order_status/{order_id}',
                           'rel': 'services',
                           'type': 'GET'
                       }
                   }, 202
```

5.1.156 services\backend_api\src\resources\resources_services\batch_classification_staging\batch_classification_staging.py

```
###########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Batch classification staging API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
###########################################################################################
##############################
```

```
from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from geoville_ms_orderid_generator.generator import generate_orderid
from init.init_env_variables import database_config_file, database_config_section_api,
database_config_section_oauth
from init.namespace_constructor import service_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_service_name_existence, get_service_id,
query_user_id
from lib.general_helper_methods import publish_to_queue
from models.general_models.general_models import service_success_response_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from models.models_services.batch_classification.batch_classification_model import
batch_classification_staging_model
from oauth.oauth2 import require_oauth
import flask
import json
import traceback


###########################################################################################
##############################
# Resource definition for the batch classification staging API call
###########################################################################################
##############################

@api.expect(batch_classification_staging_model)
@api.header('Content-Type', 'application/json')
class BatchClassificationStaging(Resource):
    """ Class for handling the POST request

    This class defines the API call for starting the batch classification staging
workflow. The class consists of one
    methods which accepts a POST request. For the POST request a JSON with several
parameters is required, defined in
    the corresponding model.

    """


###########################################################################################
##########################
    # Method for handling the POST request

###########################################################################################
##########################

    @require_oauth(['admin', 'batch_classification', 'gaf'])
    @api.expect(auth_header_parser)
    @api.response(202, 'Order Received', service_success_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for triggering the batch classification staging workflow

        <p style="text-align: justify">This method defines the handler of the POST request 
for starting off the batch
        classification staging processing chain within the GEMS service architecture. It 
is an asynchronous call and
        thus, it does not return the requested data immediately but generates an order ID. 
After the request has been
        submitted successfully, a message to a RabbitMQ queue will be send. A listener in 
the backend triggers the
        further procedure, starting with the job scheduling of the order. The final result 
will be stored in a database
        in form of a link and can be retrieved via the browser. To access the service it 
is necessary to generate a
        valid Bearer token with sufficient access rights, otherwise the request will 
return a HTTP status code 401 or
        403. In case of those errors, please contact the GeoVille service team for any 
support.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>params (dict): Dictionary with a set of parameters</i></p></li>
        <li><p><i>user_id (str): User specific client ID</i></p></li>
        <li><p><i>service_name (str): Unique name of the service to be called. Name should 
not be changed</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">After the request was successfully received by the 
GEMS API, an order ID will be
        created for the submitted job in the GEMS backend and stored in a database with 
all necessary information for
        the consumer and the system itself. The initial status of the order in the 
database will be set to 'received'.
        After that the order ID will be returned in form of a Hypermedia link which 
enables the possibilities to
        programmatically check the status the order by a piece of code of a GEMS API 
consumer. In the following the
        status of the order can be queried by using the GEMS service route listed 
below:</p>

        <ul><li><p><i>/services/order_status/{order_id}</i></p></li></ul>

        <p style="text-align: justify">The status of the order will be updated whenever 
the processing chain reaches the
        next step in its internal calculation. In case of success, the user will receive a 
link, which provides the
        ordered file. Additionally an e-mail notification is enabled, which sends out an 
e-mail in case of success, with
        the resulting link or in case of failure, with a possible error explanation.
        </p>

        """

        order_id = None

        try:
            req_args = api.payload
            access_token = flask.request.headers.get('Authorization').split(" 
")[1].strip()
```

```
            user_id = query_user_id(access_token, database_config_file,
database_config_section_oauth)

            if not check_service_name_existence(req_args['service_name'],
database_config_file,
                                                database_config_section_api):
                error = NotFoundError('Service name does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
batch_classification_staging')
                return {'message': error.to_dict()}, 404

            service_id = get_service_id(req_args['service_name'], database_config_file,
database_config_section_api)
            order_id = generate_orderid(user_id, service_id, json.dumps(req_args))

            publish_to_queue(req_args['service_name'], order_id, req_args)

            update_query = """UPDATE customer.service_orders
                                  set status = 'RECEIVED'
                              WHERE
                                  order_id = %s;
                           """
            execute_database(update_query, (order_id,), database_config_file,
database_config_section_api, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
batch_classification_staging', order_id)
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
batch_classification_staging', order_id)
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
batch_classification_staging', order_id)
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Request successful with order ID {order_id}', 'API-
batch_classification_staging', order_id)
            return {
                       'message': 'Your order has been successfully submitted',
                       'links': {
                           'href': f'/services/order_status/{order_id}',
                           'rel': 'services',
                           'type': 'GET'
                       }
                   }, 202
```

5.1.157 services\backend_api\src\resources\resources_services\batch_classification_test\batch_classification_test.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
```

```
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Batch classification test API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from geoville_ms_orderid_generator.generator import generate_orderid
from init.init_env_variables import database_config_file, database_config_section_api,
database_config_section_oauth
from init.namespace_constructor import service_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_service_name_existence, get_service_id,
query_user_id
from lib.general_helper_methods import publish_to_queue
from models.general_models.general_models import service_success_response_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from models.models_services.batch_classification.batch_classification_model import
batch_classification_model
from oauth.oauth2 import require_oauth
import flask
import json
import traceback


########################################################################################
##############################
# Resource definition for the batch classification test API call
########################################################################################
##############################

@api.expect(batch_classification_model)
@api.header('Content-Type', 'application/json')
class BatchClassificationTest(Resource):
    """ Class for handling the POST request

    This class defines the API call for starting the batch classification test workflow.
The class consists of one
    methods which accepts a POST request. For the POST request a JSON with several
parameters is required, defined in
    the corresponding model.

    """


########################################################################################
##########################
```

```
    # Method for handling the POST request
##########################################################################################
##########################

    @require_oauth(['admin', 'batch_classification', 'gaf'])
    @api.expect(auth_header_parser)
    @api.response(202, 'Order Received', service_success_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for triggering the batch classification test workflow

        <p style="text-align: justify">This method defines the handler of the POST request
for starting off the batch
        classification processing chain within the GEMS service architecture. It is an
asynchronous call and thus, it
        does not return the requested data immediately but generates an order ID. After
the request has been submitted
        successfully, a message to a RabbitMQ queue will be send. A listener in the
backend triggers the further
        procedure, starting with the job scheduling of the order. The final result will be
stored in a database in form
        of a link and can be retrieved via the browser. To access the service it is
necessary to generate a valid Bearer
        token with sufficient access rights, otherwise the request will return a HTTP
status code 401 or 403. In case of
        those errors, please contact the GeoVille service team for any support.</p>

        <br><b>Description:</b>
        <p style="text-align: justify"></p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>params (dict): Dictionary with a set of parameters</i></p></li>
        <li><p><i>user_id (str): User specific client ID</i></p></li>
        <li><p><i>service_name (str): Unique name of the service to be called. Name should
not be changed</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">After the request was successfully received by the
GEMS API, an order ID will be
        created for the submitted job in the GEMS backend and stored in a database with
all necessary information for
        the consumer and the system itself. The initial status of the order in the
database will be set to 'received'.
        After that the order ID will be returned in form of a Hypermedia link which
enables the possibilities to
        programmatically check the status the order by a piece of code of a GEMS API
consumer. In the following the
        status of the order can be queried by using the GEMS service route listed
below:</p>

        <ul><li><p><i>/services/order_status/{order_id}</i></p></li></ul>

        <p style="text-align: justify">The status of the order will be updated whenever
the processing chain reaches the
        next step in its internal calculation. In case of success, the user will receive a
link, which provides the
```

```
        ordered file. Additionally an e-mail notification is enabled, which sends out an
e-mail in case of success, with
        the resulting link or in case of failure, with a possible error explanation.
        </p>

        """

        order_id = None

        try:
            req_args = api.payload
            access_token = flask.request.headers.get('Authorization').split("
")[1].strip()

            user_id = query_user_id(access_token, database_config_file,
database_config_section_oauth)

            if not check_service_name_existence(req_args['service_name'],
database_config_file,
                                                database_config_section_api):
                error = NotFoundError('Service name does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
batch_classification_test')
                return {'message': error.to_dict()}, 404

            service_id = get_service_id(req_args['service_name'], database_config_file,
database_config_section_api)
            order_id = generate_orderid(user_id, service_id, json.dumps(req_args))

            publish_to_queue(req_args['service_name'], order_id, req_args)

            update_query = """UPDATE customer.service_orders
                              set status = 'RECEIVED'
                            WHERE
                                order_id = %s;
                        """
            execute_database(update_query, (order_id,), database_config_file,
database_config_section_api, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
batch_classification_test', order_id)
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
batch_classification_test', order_id)
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
batch_classification_test', order_id)
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Request successful with order ID {order_id}', 'API-
batch_classification_test', order_id)
            return {
                        'message': 'Your order has been successfully submitted',
                        'links': {
                            'href': f'/services/order_status/{order_id}',
                            'rel': 'services',
                            'type': 'GET'
                        }
                    }, 202
```

5.1.158 services\backend_api\src\resources\resources_services\harmonics\harmonics.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# harmonics API call
#
# Date created: 01.06.2020
# Date last modified: 15.04.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.04
#
########################################################################################
##############################

from check_message.check_message import check_message
from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from geoville_ms_orderid_generator.generator import generate_orderid
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import service_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_service_name_existence, check_user_existence,
get_service_id
from lib.general_helper_methods import publish_to_queue
from models.general_models.general_models import service_success_response_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from models.models_services.harmonics.harmonics_model import harmonics_request_model
from oauth.oauth2 import require_oauth
import json
import traceback


########################################################################################
##############################
# Resource definition for the harmonics API call
########################################################################################
##############################

@api.expect(harmonics_request_model)
@api.header('Content-Type', 'application/json')
class Harmonics(Resource):
    """ Class for handling the POST request
    This class defines the API call for starting the harmonics workflow. The class
consists of one methods which
    accepts a POST request. For the POST request a JSON with several parameters is
required, defined in the
    corresponding model.

    """


##########################################################################################
##########################
    # Method for handling the POST request

##########################################################################################
##########################

    @require_oauth(['admin', 'harmonics'])
    @api.expect(auth_header_parser)
    @api.response(202, 'Order Received', service_success_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for requesting the harmonics workflow

        <p style="text-align: justify">This method defines the handler of the POST request
for starting off the
        harmonics processing chain within the GEMS service architecture. It is an
asynchronous call and thus, it does
        not return the requested data immediately but generates an order ID. After the
request has been submitted
        successfully, a message to a RabbitMQ queue will be send. A listener in the
backend triggers the further
        procedure, starting with the job scheduling of the order. The final result will be
stored in a database in form
        of a link and can be retrieved via the browser. To access the service it is
necessary to generate a valid Bearer
        token with sufficient access rights, otherwise the request will return a HTTP
status code 401 or 403. In case of
        those errors, please contact the GeoVille service team for any support.</p>

        <br><b>Description:</b>
        <p style="text-align: justify">From a time-series of satellite imagery, a single
band is taken and a combination
        of an n-th order harmonic function and a k-th order polynomial is fitted to the
data. The "seasonality-layer" is
        an example of a 1st order harmonic (n=1) in combination with a linear trend (k=1).
The results can be used as an
        input for classification or as a fast smoothig algorithm. Weights can be supplied,
e.g. for cloud-masking.</p>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>tile_id (str): Sentinel-2 tile ID</i></p></li>
        <li><p><i>start_date (str): Start date in the format YYYY-MM-DD</i></p></li>
        <li><p><i>end_date (str): End date in the format YYYY-MM-DD</i></p></li>
        <li><p><i>band (str): Available raster bands</i></p></li>
        <li><p><i>resolution (int): Raster band resolution (>=10)</i></p></li>
        <li><p><i>ndi_band (str): NDI band</i></p></li>
        <li><p><i>user_id (str): User specific client ID</i></p></li>
        <li><p><i>service_name (str): Unique name of the service to be called. Name should
not be changed</i></p></li>
```

```
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">After the request was successfully received by the
GEMS API, an order ID will be
        created for the submitted job in the GEMS backend and stored in a database with
all necessary information for
        the consumer and the system itself. The initial status of the order in the
database will be set to 'received'.
        After that the order ID will be returned in form of a Hypermedia link which
enables the possibilities to
        programmatically check the status the order by a piece of code of a GEMS API
consumer. In the following the
        status of the order can be queried by using the GEMS service route listed
below:</p>

        <ul><li><p><i>/services/order_status/{order_id}</i></p></li></ul>

        <p style="text-align: justify">The status of the order will be updated whenever
the processing chain reaches the
        next step in its internal calculation. In case of success, the user will receive a
link, which provides the
        ordered file. Additionally an e-mail notification is enabled, which sends out an
e-mail in case of success, with
        the resulting link or in case of failure, with a possible error explanation.
        </p>

        """

        order_id = None

        try:
            req_args = api.payload

            payload_check = check_message(req_args)

            if not payload_check[0]:
                error = BadRequestError(f'Payload failed the GeoVille standards:
{payload_check[1]}', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
harmonics', order_id)
                return {'message': error.to_dict()}, 404

            if req_args['start_date'] > req_args['end_date']:
                error = BadRequestError('Start date is greater than end date', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
harmonics', order_id)
                return {'message': error.to_dict()}, 400

            if not check_user_existence(req_args['user_id'], database_config_file,
database_config_section_api):
                error = NotFoundError('User ID does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
harmonics', order_id)
                return {'message': error.to_dict()}, 404

            if not check_service_name_existence(req_args['service_name'],
database_config_file,
                                                    database_config_section_api):
                error = NotFoundError('Service name does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
harmonics', order_id)
                return {'message': error.to_dict()}, 404

            service_id = get_service_id(req_args['service_name'], database_config_file,
database_config_section_api)
            order_id = generate_orderid(req_args['user_id'], service_id,
json.dumps(req_args))
            gemslog(LogLevel.INFO, f'Request payload: {req_args}', 'API-harmonics',
order_id)
```

```
            publish_to_queue(req_args['service_name'], order_id, req_args)

            update_query = """UPDATE customer.service_orders
                                  set status = 'RECEIVED'
                              WHERE
                                  order_id = %s;
                           """
            execute_database(update_query, (order_id,), database_config_file,
database_config_section_api, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-harmonics',
order_id)
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-harmonics',
order_id)
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', api.payload,
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-harmonics',
order_id)
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Request successful', 'API-harmonics', order_id)
            return {
                       'message': 'Your order has been successfully submitted',
                       'links': {
                           'href': f'/services/order_status/{order_id}',
                           'rel': 'services',
                           'type': 'GET'
                       }
                   }, 202
```

5.1.159 services\backend_api\src\resources\resources_services\retransformation\retransformation.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Retransformation API call
#
# Date created: 07.07.2021
# Date last modified: 07.07.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.07
#
```

```
##########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from geoville_ms_orderid_generator.generator import generate_orderid
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import service_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_service_name_existence, check_user_existence,
get_service_id
from lib.general_helper_methods import publish_to_queue
from models.general_models.general_models import service_success_response_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from models.models_services.retransformation.retransformation_model import
retransformation_request_model
from check_message.check_message import check_message
from oauth.oauth2 import require_oauth
import json
import traceback


##########################################################################################
##############################
# Resource definition for the retransformation API call
##########################################################################################
##############################

@api.expect(retransformation_request_model)
@api.header('Content-Type', 'application/json')
class Retransformation(Resource):
    """ Class for handling the POST request

    This class defines the API call for starting the retransformation workflow. The class
consists of one methods which
    accepts a POST request. For the POST request a JSON with several parameters is
required, defined in the
    corresponding model.

    """


##########################################################################################
##########################
    # Method for handling the POST request

##########################################################################################
##########################

    @require_oauth(['admin', 'user', 'retransformation'])
    @api.expect(auth_header_parser)
    @api.response(202, 'Order Received', service_success_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for the retransformation

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>subproduction_unit_name  (str): Name of the SPU</i></p></li>
        <li><p><i>user_id (str): User specific client ID</i></p></li>
        <li><p><i>service_name (str): Unique name of the service to be called. Name should
not be changed</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">After the request was successfully received by the
GEMS API, an order ID will be
        created for the submitted job in the GEMS backend and stored in a database with
all necessary information for
        the consumer and the system itself. The initial status of the order in the
database will be set to 'received'.
        After that the order ID will be returned in form of a Hypermedia link which
enables the possibilities to
        programmatically check the status the order by a piece of code of a GEMS API
consumer. In the following the
        status of the order can be queried by using the GEMS service route listed
below:</p>

        <ul><li><p><i>/services/order_status/{order_id}</i></p></li></ul>

        <p style="text-align: justify">The status of the order will be updated whenever
the processing chain reaches the
        next step in its internal calculation. In case of success, the user will receive a
link, which provides the
        ordered file. Additionally an e-mail notification is enabled, which sends out an
e-mail in case of success, with
        the resulting link or in case of failure, with a possible error explanation.
        </p>

        """

        order_id = None

        try:
            req_args = api.payload
            payload_check = check_message(req_args)

            if not payload_check[0]:
                error = BadRequestError(f'Payload failed the GeoVille standards:
{payload_check[1]}', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
retransformation', order_id)
                return {'message': error.to_dict()}, 404

            if not check_user_existence(req_args['user_id'], database_config_file,
database_config_section_api):
                error = NotFoundError('User ID does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
retransformation', order_id)
                return {'message': error.to_dict()}, 404

            if not check_service_name_existence(req_args['service_name'],
database_config_file,
                                                database_config_section_api):
                error = NotFoundError('Service name does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
retransformation', order_id)
```

```
                return {'message': error.to_dict()}, 404

            service_id = get_service_id(req_args['service_name'], database_config_file,
database_config_section_api)
            order_id = generate_orderid(req_args['user_id'], service_id,
json.dumps(req_args))
            gemslog(LogLevel.INFO, f'Request payload: {req_args}', 'API-retransformation',
order_id)

            publish_to_queue(req_args['service_name'], order_id, req_args)

            update_query = """UPDATE customer.service_orders
                                  set status = 'RECEIVED'
                              WHERE
                                  order_id = %s;
                           """
            execute_database(update_query, (order_id,), database_config_file,
database_config_section_api, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
retransformation', order_id)
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
retransformation', order_id)
            return {'message': error.to_dict()}, 503

        except Exception as err:
            error = InternalServerErrorAPI(f'Unexpected error occurred: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
retransformation', order_id)
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Request successful', 'API-retransformation',
order_id)
            return {
                       'message': 'Your order has been successfully submitted',
                       'links': {
                           'href': f'/services/order_status/{order_id}',
                           'rel': 'services',
                           'type': 'GET'
                       }
                   }, 202
```

5.1.160 services\backend_api\src\resources\resources_services\service_order_status\order_status.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Status order API call
```

```
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
###########################################################################################
##############################

from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import service_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import query_order_status
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from models.models_services.service_order_status.order_status_model import
order_status_response_model
from oauth.oauth2 import require_oauth
import traceback


###########################################################################################
##############################
# Resource definition for the status order API call
###########################################################################################
##############################

@api.header('Content-Type', 'application/json')
class OrderStatus(Resource):
    """ Class for handling the GET request

    This class defines the API call for receiving the status of a client's order. The
class consists of one methods
    which accepts a GET request. For the POST request a JSON with one parameters is
required, defined in the
    corresponding model.

    """


###########################################################################################
##########################
    # Method for handling the GET request

###########################################################################################
##########################

    @require_oauth(['admin', 'user'])
    @api.expect(auth_header_parser)
    @api.response(200, 'Operation successful', order_status_response_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found Error', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def get(self, order_id):
        """ GET definition for retrieving the order status
        <p style="text-align: justify">This method defines the handler for the GET request
for receiving the status of a
        GEMS customers order. The service receives the current status from the database
for the specified order ID and
        returns the link to a possible result file.</p>

        <br><b>Description:</b>
        <p style="text-align: justify">The GEMS service route 'order status' was designed
to query the status of an
        asynchronous GEMS service what enables the possibility to programmatically check
the result of a submitted
        order and further process the expected result file. An order can have the
following states:</p>
        <ul>
        <li><p><i>FAILED â€“ An unexpected error occurred during the execution of the
service</i></p></li>
        <li><p><i>SUCCESS â€“ Service calculation was successful</i></p></li>
        <li><p><i>QUEUED â€“ Submitted request is in the waiting list</i></p></li>
        <li><p><i>RECEIVED â€“ API received the service request and created an order
ID</i></p></li>
        <li><p><i>RUNNING â€“ Service is being calculated at the moment</i></p></li>
        <li><p><i>INVALID â€“ It indicates that there are no available satellite image for
the date and tile, the consumer
        requested</i></p></li>
        </ul>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Path parameter:</b>
        <ul>
        <li><p><i>order_id: Order ID received from a triggered asynchronous GEMS
service</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">The result of the GET request is a JSON which
contains an object with three key
        value pairs.</p>
        <ul>
        <li><p><i>order_id: Order ID received from the triggered asynchronous GEMS
service</i></p></li>
        <li><p><i>status: Status of the submitted service</i></p></li>
        <li><p><i>result: link to the final result file or NULL if the process is still
running or aborted</i></p></li>
        </ul>

        """

        try:
            order_res = query_order_status(order_id, database_config_file,
database_config_section_api)

            if order_res is None or order_res is False:
                error = NotFoundError(f'No database entry found for order ID: {order_id}',
'', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
order_status')
                return {'message': error.to_dict()}, 404

            if order_res[2]:
                return {
                    'order_id': order_id,
                    'status': order_res[0],
                    'result': order_res[1]
                }
            else:
                return {
                    'order_id': order_id,
                    'status': order_res[0],
                    'result': None
                }

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-order_status')
            return {'message': error.to_dict()}, 503

        except Exception:
            error = InternalServerErrorAPI('Unexpected error occurred', '',
traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-order_status')
            return {'message': error.to_dict()}, 500
```

5.1.161 services\backend_api\src\resources\resources_services\task_1_batch_classification\task_1_batch_classification.py

```
########################################################################################################################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Batch classification API call
#
# Date created: 19.04.2021
# Date last modified: 19.04.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.04
#
########################################################################################################################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from geoville_ms_orderid_generator.generator import generate_orderid
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import service_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_service_name_existence, check_user_existence,
get_service_id
from lib.general_helper_methods import publish_to_queue
from models.general_models.general_models import service_success_response_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from models.models_services.task_1_batch_classification.task_1_batch_classification_model
import t1_batch_classification_request_model
from check_message.check_message import check_message
from oauth.oauth2 import require_oauth
import json
import traceback


#########################################################################################
##############################
# Resource definition for the batch classification (Task 1) API call
#########################################################################################
##############################

@api.expect(t1_batch_classification_request_model)
@api.header('Content-Type', 'application/json')
class Task1BatchClassification(Resource):
    """ Class for handling the POST request

    This class defines the API call for starting the batch classification (Task 1)
workflow. The class consists of one
    methods which accepts a POST request. For the POST request a JSON with several
parameters is required, defined in
    the corresponding model.

    """


#########################################################################################
##########################
    # Method for handling the POST request

#########################################################################################
##########################

    @require_oauth(['admin', 'batch_classification_task_1'])
    @api.expect(auth_header_parser)
    @api.response(202, 'Order Received', service_success_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for the batch classification in task 1

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>start_date (str): Oldest acquisition date to consider in the format
YYYY-MM-DD</i></p></li>
        <li><p><i>end_date (str): Newest acquisition date to consider in the format YYYY-
MM-DD</i></p></li>
        <li><p><i>processing_unit_name (str): Name of the processing unit</i></p></li>
        <li><p><i>cloud_cover (int): Maximum cloud cover (in %) to consider</i></p></li>
        <li><p><i>interval_size (int): Time difference in days for temporal
interpolation</i></p></li>
        <li><p><i>s1_bands (list): List of names of the required Sentinel-1 bands or
indices:
                                    ['ASC_DVVVH', 'ASC_NDVVVH', 'ASC_RVVVH', 'ASC_VV',
'ASC_VH', 'DSC_DVVVH',
                                    'DSC_NDVVVH', 'DSC_RVVVH', 'DSC_VV',
'DSC_VH']</i></p></li>
```

```
        <li><p><i>s2_bands (list): list of names of the required Sentinel-2 bands or
indices:
                                    ['B01', 'B02', 'B03', 'B04', 'B05', 'B06', 'B07',
'B08', 'B09', 'B10', 'B11', 'B12',
                                    'B8A', 'BRGHT', 'IRECI', 'NBR', 'NDVI', 'NDWI',
'NDWIGREEN', 'NGDR',
                                    'RENDVI']</i></p></li>
        <li><p><i>precalculated_features (list): Names of the required auxiliary features:
                                                 ['geomorpho90', 'distance',
'dem']</i></p></li>
        <li><p><i>use_cache (bool): Use cache results</i></p></li>
        <li><p><i>aoi_coverage (int): Minimum coverage in percent for one scene of an
AOI</i></p></li>
        <li><p><i>user_id (str): User specific client ID</i></p></li>
        <li><p><i>service_name (str): Unique name of the service to be called. Name should
not be changed</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">After the request was successfully received by the
GEMS API, an order ID will be
        created for the submitted job in the GEMS backend and stored in a database with
all necessary information for
        the consumer and the system itself. The initial status of the order in the
database will be set to 'received'.
        After that the order ID will be returned in form of a Hypermedia link which
enables the possibilities to
        programmatically check the status the order by a piece of code of a GEMS API
consumer. In the following the
        status of the order can be queried by using the GEMS service route listed
below:</p>

        <ul><li><p><i>/services/order_status/{order_id}</i></p></li></ul>

        <p style="text-align: justify">The status of the order will be updated whenever
the processing chain reaches the
        next step in its internal calculation. In case of success, the user will receive a
link, which provides the
        ordered file. Additionally an e-mail notification is enabled, which sends out an
e-mail in case of success, with
        the resulting link or in case of failure, with a possible error explanation.
        </p>

        """

        order_id = None

        try:
            req_args = api.payload

            payload_check = check_message(req_args)

            if not payload_check[0]:
                error = BadRequestError(f'Payload failed the GeoVille standards:
{payload_check[1]}', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
batch_classification', order_id)
                return {'message': error.to_dict()}, 404

            if not check_user_existence(req_args['user_id'], database_config_file,
database_config_section_api):
                error = NotFoundError('User ID does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
batch_classification', order_id)
                return {'message': error.to_dict()}, 404

            if not check_service_name_existence(req_args['service_name'],
database_config_file,
                                                database_config_section_api):
                error = NotFoundError('Service name does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
batch_classification', order_id)
                return {'message': error.to_dict()}, 404

            if req_args['start_date'] > req_args['end_date']:
                error = BadRequestError('Start date is greater than end date', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
batch_classification', order_id)
                return {'message': error.to_dict()}, 400

            service_id = get_service_id(req_args['service_name'], database_config_file,
database_config_section_api)
            order_id = generate_orderid(req_args['user_id'], service_id,
json.dumps(req_args))
            gemslog(LogLevel.INFO, f'Request payload: {req_args}', 'API-
batch_classification', order_id)

            publish_to_queue(req_args['service_name'], order_id, req_args)

            update_query = """UPDATE customer.service_orders
                                  set status = 'RECEIVED'
                              WHERE
                                  order_id = %s;
                           """
            execute_database(update_query, (order_id,), database_config_file,
database_config_section_api, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
batch_classification', order_id)
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
batch_classification', order_id)
            return {'message': error.to_dict()}, 503

        except Exception as err:
            error = InternalServerErrorAPI(f'Unexpected error occurred: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
batch_classification', order_id)
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Request successful', 'API-batch_classification',
order_id)
            return {
                       'message': 'Your order has been successfully submitted',
                       'links': {
                           'href': f'/services/order_status/{order_id}',
                           'rel': 'services',
                           'type': 'GET'
                       }
                   }, 202
```

5.1.162 services\backend_api\src\resources\resources_services\task_1_feature_classification\task_1_feature_classification.py

```
###########################################################################################
##############################
#
```

```
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Task 2 feature calculation API call
#
# Date created: 19.04.2021
# Date last modified: 19.04.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.04
#
#########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from geoville_ms_orderid_generator.generator import generate_orderid
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import service_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_service_name_existence, check_user_existence,
get_service_id
from lib.general_helper_methods import publish_to_queue
from models.general_models.general_models import service_success_response_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from models.models_services.task_1_feature_calculation.task_1_feature_calculation_model
import t1_feature_calculation_request_model
from check_message.check_message import check_message
from oauth.oauth2 import require_oauth
import json
import traceback


#########################################################################################
##############################
# Resource definition for the feature calculation (Task 2) API call
#########################################################################################
##############################

@api.expect(t1_feature_calculation_request_model)
@api.header('Content-Type', 'application/json')
class Task1FeatureCalculation(Resource):
    """ Class for handling the POST request

    This class defines the API call for starting the feature calculation (Task 2)
workflow. The class consists of one
    methods which accepts a POST request. For the POST request a JSON with several
parameters is required, defined in
    the corresponding model.

    """
```

```
##########################################################################################
##########################
    # Method for handling the POST request

##########################################################################################
##########################

    @require_oauth(['admin', 'feature_calculation_task_1'])
    @api.expect(auth_header_parser)
    @api.response(202, 'Order Received', service_success_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for calculating interpolation features for CLC+ task 1

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>start_date (str): Oldest acquisition date to consider in the format
YYYY-MM-DD</i></p></li>
        <li><p><i>end_date (str): Newest acquisition date to consider in the format YYYY-
MM-DD</i></p></li>
        <li><p><i>processing_unit_name (str): Name of the processing unit</i></p></li>
        <li><p><i>cloud_cover (int): Maximum cloud cover (in %) to consider</i></p></li>
        <li><p><i>interval_size (int): Time difference in days for temporal
interpolation</i></p></li>
        <li><p><i>s1_bands (list): List of names of the required Sentinel-1 bands or
indices:
                                   ['ASC_DVVVH', 'ASC_NDVVVH', 'ASC_RVVVH', 'ASC_VV',
'ASC_VH', 'DSC_DVVVH',
                                   'DSC_NDVVVH', 'DSC_RVVVH', 'DSC_VV',
'DSC_VH']</i></p></li>
        <li><p><i>s2_bands (list): list of names of the required Sentinel-2 bands or
indices:
                                    ['B01', 'B02', 'B03', 'B04', 'B05', 'B06', 'B07',
'B08', 'B09', 'B10', 'B11', 'B12',
                                    'B8A', 'BRGHT', 'IRECI', 'NBR', 'NDVI', 'NDWI',
'NDWIGREEN', 'NGDR',
                                    'RENDVI']</i></p></li>
        <li><p><i>precalculated_features (list): Names of the required auxiliary features:
                                                 ['geomorpho90', 'distance',
'dem']</i></p></li>
        <li><p><i>use_cache (bool): Use cache results</i></p></li>
        <li><p><i>aoi_coverage (int): Minimum coverage in percent for one scene of an
AOI</i></p></li>
        <li><p><i>user_id (str): User specific client ID</i></p></li>
        <li><p><i>service_name (str): Unique name of the service to be called. Name should
not be changed</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">After the request was successfully received by the
GEMS API, an order ID will be
        created for the submitted job in the GEMS backend and stored in a database with
all necessary information for
        the consumer and the system itself. The initial status of the order in the
database will be set to 'received'.
        After that the order ID will be returned in form of a Hypermedia link which
enables the possibilities to
        programmatically check the status the order by a piece of code of a GEMS API
consumer. In the following the
```

```
        status of the order can be queried by using the GEMS service route listed
below:</p>

        <ul><li><p><i>/services/order_status/{order_id}</i></p></li></ul>

        <p style="text-align: justify">The status of the order will be updated whenever
the processing chain reaches the
        next step in its internal calculation. In case of success, the user will receive a
link, which provides the
        ordered file. Additionally an e-mail notification is enabled, which sends out an
e-mail in case of success, with
        the resulting link or in case of failure, with a possible error explanation.
        </p>

        """

        order_id = None

        try:
            req_args = api.payload

            payload_check = check_message(req_args)

            if not payload_check[0]:
                error = BadRequestError(f'Payload failed the GeoVille standards:
{payload_check[1]}', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
feature_calculation_t1', order_id)
                return {'message': error.to_dict()}, 404

            if not check_user_existence(req_args['user_id'], database_config_file,
database_config_section_api):
                error = NotFoundError('User ID does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
feature_calculation_t1', order_id)
                return {'message': error.to_dict()}, 404

            if not check_service_name_existence(req_args['service_name'],
database_config_file,
                                                database_config_section_api):
                error = NotFoundError('Service name does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
feature_calculation_t1', order_id)
                return {'message': error.to_dict()}, 404

            if req_args['start_date'] > req_args['end_date']:
                error = BadRequestError('Start date is greater than end date', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
feature_calculation_t1', order_id)
                return {'message': error.to_dict()}, 400

            service_id = get_service_id(req_args['service_name'], database_config_file,
database_config_section_api)
            order_id = generate_orderid(req_args['user_id'], service_id,
json.dumps(req_args))
            gemslog(LogLevel.INFO, f'Request payload: {req_args}', 'API-
feature_calculation_t1', order_id)

            publish_to_queue(req_args['service_name'], order_id, req_args)

            update_query = """UPDATE customer.service_orders
                                  set status = 'RECEIVED'
                              WHERE
                                  order_id = %s;
                           """
            execute_database(update_query, (order_id,), database_config_file,
database_config_section_api, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
feature_calculation_t1', order_id)
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
feature_calculation_t1', order_id)
            return {'message': error.to_dict()}, 503

        except Exception as err:
            error = InternalServerErrorAPI(f'Unexpected error occurred: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
feature_calculation_t1', order_id)
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Request successful', 'API-feature_calculation_t1',
order_id)
            return {
                       'message': 'Your order has been successfully submitted',
                       'links': {
                           'href': f'/services/order_status/{order_id}',
                           'rel': 'services',
                           'type': 'GET'
                       }
                   }, 202
```

5.1.163 services\backend_api\src\resources\resources_services\task_1_reprocessing\task_1_reprocessing.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Reprocessing API call
#
# Date created: 19.04.2021
# Date last modified: 19.04.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.04
#
##########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from geoville_ms_orderid_generator.generator import generate_orderid
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import service_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_service_name_existence, check_user_existence,
get_service_id
from lib.general_helper_methods import publish_to_queue
from models.general_models.general_models import service_success_response_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from models.models_services.task_1_reprocessing.task_1_reprocessing_model import
task_1_reprocessing_request_model
from check_message.check_message import check_message
from oauth.oauth2 import require_oauth
import json
import traceback


###########################################################################################
##############################
# Resource definition for the reprocessing (Task 1) API call
###########################################################################################
##############################

@api.expect(task_1_reprocessing_request_model)
@api.header('Content-Type', 'application/json')
class Task1Reprocessing(Resource):
    """ Class for handling the POST request

    This class defines the API call for starting the reprocessing (Task 1) workflow. The
class consists of one
    methods which accepts a POST request. For the POST request a JSON with several
parameters is required, defined in
    the corresponding model.

    """


###########################################################################################
##########################
    # Method for handling the POST request

###########################################################################################
##########################

    @require_oauth(['admin', 'reprocessing_task_1', 'gaf'])
    @api.expect(auth_header_parser)
    @api.response(202, 'Order Received', service_success_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for the reprocessing in task 1

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>start_date (str): Oldest acquisition date to consider in the format
YYYY-MM-DD</i></p></li>
```

```
        <li><p><i>end_date (str): Newest acquisition date to consider in the format YYYY-
MM-DD</i></p></li>
        <li><p><i>processing_unit_name (str): Name of the processing unit</i></p></li>
        <li><p><i>cloud_cover (int): Maximum cloud cover (in %) to consider</i></p></li>
        <li><p><i>interval_size (int): Time difference in days for temporal
interpolation</i></p></li>
        <li><p><i>s1_bands (list): List of names of the required Sentinel-1 bands or
indices:
                                    ['ASC_DVVVH', 'ASC_NDVVVH', 'ASC_RVVVH', 'ASC_VV',
'ASC_VH', 'DSC_DVVVH',
                                    'DSC_NDVVVH', 'DSC_RVVVH', 'DSC_VV',
'DSC_VH']</i></p></li>
        <li><p><i>s2_bands (list): list of names of the required Sentinel-2 bands or
indices:
                                     ['B01', 'B02', 'B03', 'B04', 'B05', 'B06', 'B07',
'B08', 'B09', 'B10', 'B11', 'B12',
                                     'B8A', 'BRGHT', 'IRECI', 'NBR', 'NDVI', 'NDWI',
'NDWIGREEN', 'NGDR',
                                     'RENDVI']</i></p></li>
        <li><p><i>precalculated_features (list): Names of the required auxiliary features:
                                                  ['geomorpho90', 'distance',
'dem']</i></p></li>
        <li><p><i>use_cache (bool): Use cache results</i></p></li>
        <li><p><i>aoi_coverage (int): Minimum coverage in percent for one scene of an
AOI</i></p></li>
        <li><p><i>user_id (str): User specific client ID</i></p></li>
        <li><p><i>service_name (str): Unique name of the service to be called. Name should
not be changed</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">After the request was successfully received by the
GEMS API, an order ID will be
        created for the submitted job in the GEMS backend and stored in a database with
all necessary information for
        the consumer and the system itself. The initial status of the order in the
database will be set to 'received'.
        After that the order ID will be returned in form of a Hypermedia link which
enables the possibilities to
        programmatically check the status the order by a piece of code of a GEMS API
consumer. In the following the
        status of the order can be queried by using the GEMS service route listed
below:</p>

        <ul><li><p><i>/services/order_status/{order_id}</i></p></li></ul>

        <p style="text-align: justify">The status of the order will be updated whenever
the processing chain reaches the
        next step in its internal calculation. In case of success, the user will receive a
link, which provides the
        ordered file. Additionally an e-mail notification is enabled, which sends out an
e-mail in case of success, with
        the resulting link or in case of failure, with a possible error explanation.
        </p>

        """

        order_id = None

        try:
            req_args = api.payload

            payload_check = check_message(req_args)

            if not payload_check[0]:
                error = BadRequestError(f'Payload failed the GeoVille standards:
{payload_check[1]}', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
reprocessing', order_id)
                return {'message': error.to_dict()}, 404
            if not check_user_existence(req_args['user_id'], database_config_file,
database_config_section_api):
                error = NotFoundError('User ID does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
reprocessing', order_id)
                return {'message': error.to_dict()}, 404

            if not check_service_name_existence(req_args['service_name'],
database_config_file,
                                                database_config_section_api):
                error = NotFoundError('Service name does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
reprocessing', order_id)
                return {'message': error.to_dict()}, 404

            if req_args['start_date'] > req_args['end_date']:
                error = BadRequestError('Start date is greater than end date', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
reprocessing', order_id)
                return {'message': error.to_dict()}, 400

            service_id = get_service_id(req_args['service_name'], database_config_file,
database_config_section_api)
            order_id = generate_orderid(req_args['user_id'], service_id,
json.dumps(req_args))
            gemslog(LogLevel.INFO, f'Request payload: {req_args}', 'API-reprocessing',
order_id)

            publish_to_queue(req_args['service_name'], order_id, req_args)

            update_query = """UPDATE customer.service_orders
                                  set status = 'RECEIVED'
                              WHERE
                                  order_id = %s;
                           """
            execute_database(update_query, (order_id,), database_config_file,
database_config_section_api, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-reprocessing',
order_id)
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-reprocessing',
order_id)
            return {'message': error.to_dict()}, 503

        except Exception as err:
            error = InternalServerErrorAPI(f'Unexpected error occurred: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-reprocessing',
order_id)
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Request successful', 'API-reprocessing', order_id)
            return {
                       'message': 'Your order has been successfully submitted',
                       'links': {
                           'href': f'/services/order_status/{order_id}',
                           'rel': 'services',
                           'type': 'GET'
                       }
                   }, 202
```

5.1.164 services\backend_api\src\resources\resources_services\task_1_reprocessing_test\task_1_reprocessing_test.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Reprocessing API call
#
# Date created: 19.04.2021
# Date last modified: 19.04.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.04
#
########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from geoville_ms_orderid_generator.generator import generate_orderid
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import service_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_service_name_existence, check_user_existence,
get_service_id
from lib.general_helper_methods import publish_to_queue
from models.general_models.general_models import service_success_response_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from models.models_services.task_1_reprocessing_test.task_1_reprocessing_test_model import
task_1_reprocessing_test_request_model
from check_message.check_message import check_message
from oauth.oauth2 import require_oauth
import json
import traceback


########################################################################################
##############################
# Resource definition for the reprocessing (Task 1) API call
########################################################################################
##############################

@api.expect(task_1_reprocessing_test_request_model)
@api.header('Content-Type', 'application/json')
class Task1ReprocessingTest(Resource):
    """ Class for handling the POST request
    This class defines the API call for starting the reprocessing (Task 1) workflow. The
class consists of one
    methods which accepts a POST request. For the POST request a JSON with several
parameters is required, defined in
    the corresponding model.

    """


##########################################################################################
##########################
    # Method for handling the POST request

##########################################################################################
##########################

    @require_oauth(['admin', 'reprocessing_task_1', 'gaf'])
    @api.expect(auth_header_parser)
    @api.response(202, 'Order Received', service_success_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for the reprocessing in task 1

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>start_date (str): Oldest acquisition date to consider in the format
YYYY-MM-DD</i></p></li>
        <li><p><i>end_date (str): Newest acquisition date to consider in the format YYYY-
MM-DD</i></p></li>
        <li><p><i>processing_unit_name (str): Name of the processing unit</i></p></li>
        <li><p><i>cloud_cover (int): Maximum cloud cover (in %) to consider</i></p></li>
        <li><p><i>interval_size (int): Time difference in days for temporal
interpolation</i></p></li>
        <li><p><i>s1_bands (list): List of names of the required Sentinel-1 bands or
indices:
                                   ['ASC_DVVVH', 'ASC_NDVVVH', 'ASC_RVVVH', 'ASC_VV',
'ASC_VH', 'DSC_DVVVH',
                                   'DSC_NDVVVH', 'DSC_RVVVH', 'DSC_VV',
'DSC_VH']</i></p></li>
        <li><p><i>s2_bands (list): list of names of the required Sentinel-2 bands or
indices:
                                    ['B01', 'B02', 'B03', 'B04', 'B05', 'B06', 'B07',
'B08', 'B09', 'B10', 'B11', 'B12',
                                    'B8A', 'BRGHT', 'IRECI', 'NBR', 'NDVI', 'NDWI',
'NDWIGREEN', 'NGDR',
                                    'RENDVI']</i></p></li>
        <li><p><i>precalculated_features (list): Names of the required auxiliary features:
                                                 ['geomorpho90', 'distance',
'dem']</i></p></li>
        <li><p><i>use_cache (bool): Use cache results</i></p></li>
        <li><p><i>aoi_coverage (int): Minimum coverage in percent for one scene of an
AOI</i></p></li>
        <li><p><i>user_id (str): User specific client ID</i></p></li>
        <li><p><i>service_name (str): Unique name of the service to be called. Name should
not be changed</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">After the request was successfully received by the
GEMS API, an order ID will be
```

```
        created for the submitted job in the GEMS backend and stored in a database with
all necessary information for
        the consumer and the system itself. The initial status of the order in the
database will be set to 'received'.
        After that the order ID will be returned in form of a Hypermedia link which
enables the possibilities to
        programmatically check the status the order by a piece of code of a GEMS API
consumer. In the following the
        status of the order can be queried by using the GEMS service route listed
below:</p>

        <ul><li><p><i>/services/order_status/{order_id}</i></p></li></ul>

        <p style="text-align: justify">The status of the order will be updated whenever
the processing chain reaches the
        next step in its internal calculation. In case of success, the user will receive a
link, which provides the
        ordered file. Additionally an e-mail notification is enabled, which sends out an
e-mail in case of success, with
        the resulting link or in case of failure, with a possible error explanation.
        </p>

        """

        order_id = None

        try:
            req_args = api.payload

            # payload_check = check_message(req_args)
            #
            # if not payload_check[0]:
            #     error = BadRequestError(f'Payload failed the GeoVille standards:
{payload_check[1]}', '', '')
            #     gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
reprocessing', order_id)
            #     return {'message': error.to_dict()}, 404

            if not check_user_existence(req_args['user_id'], database_config_file,
database_config_section_api):
                error = NotFoundError('User ID does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
reprocessing', order_id)
                return {'message': error.to_dict()}, 404

            if not check_service_name_existence(req_args['service_name'],
database_config_file,
                                                        database_config_section_api):
                error = NotFoundError('Service name does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
reprocessing', order_id)
                return {'message': error.to_dict()}, 404

            if req_args['start_date'] > req_args['end_date']:
                error = BadRequestError('Start date is greater than end date', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
reprocessing', order_id)
                return {'message': error.to_dict()}, 400

            service_id = get_service_id(req_args['service_name'], database_config_file,
database_config_section_api)
            order_id = generate_orderid(req_args['user_id'], service_id,
json.dumps(req_args))
            gemslog(LogLevel.INFO, f'Request payload: {req_args}', 'API-reprocessing',
order_id)

            publish_to_queue(req_args['service_name'], order_id, req_args)

            update_query = """UPDATE customer.service_orders
                                      set status = 'RECEIVED'
```

```
                              WHERE
                                  order_id = %s;
                           """
            execute_database(update_query, (order_id,), database_config_file,
database_config_section_api, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-reprocessing',
order_id)
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-reprocessing',
order_id)
            return {'message': error.to_dict()}, 503

        except Exception as err:
            error = InternalServerErrorAPI(f'Unexpected error occurred: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-reprocessing',
order_id)
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Request successful', 'API-reprocessing', order_id)
            return {
                       'message': 'Your order has been successfully submitted',
                       'links': {
                           'href': f'/services/order_status/{order_id}',
                           'rel': 'services',
                           'type': 'GET'
                       }
                   }, 202
```

5.1.165 services\backend_api\src\resources\resources_services\task_1_stitching\task_1_stitching.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Stitching API call
#
# Date created: 02.08.2021
# Date last modified: 02.08.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.08
#
##########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from geoville_ms_orderid_generator.generator import generate_orderid
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import service_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_service_name_existence, check_user_existence,
get_service_id
from lib.general_helper_methods import publish_to_queue
from models.general_models.general_models import service_success_response_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from models.models_services.task_1_stitching.task_1_stitching_model import
task_1_stitching_request_model
from check_message.check_message import check_message
from oauth.oauth2 import require_oauth
import json
import traceback


########################################################################################
##############################
# Resource definition for the stitching (Task 1) API call
########################################################################################
##############################

@api.expect(task_1_stitching_request_model)
@api.header('Content-Type', 'application/json')
class Task1Stitching(Resource):
    """ Class for handling the POST request

    This class defines the API call for starting the stitching (Task 1) workflow. The
class consists of one
    methods which accepts a POST request. For the POST request a JSON with several
parameters is required, defined in
    the corresponding model.

    """


########################################################################################
##########################
    # Method for handling the POST request

########################################################################################
##########################

    @require_oauth(['admin', 'stitching_task_1', 'gaf', 'task_1'])
    @api.expect(auth_header_parser)
    @api.response(202, 'Order Received', service_success_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for the stitching process in task 1

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>processing_unit_name (str): Name of the input PU</i></p></li>
        <li><p><i>surrounding_pus (str): Name of the surrounding PU's</i></p></li>
        <li><p><i>user_id (str): User specific client ID</i></p></li>
        <li><p><i>service_name (str): Unique name of the service to be called. Name should
not be changed</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">After the request was successfully received by the
GEMS API, an order ID will be
        created for the submitted job in the GEMS backend and stored in a database with
all necessary information for
        the consumer and the system itself. The initial status of the order in the
database will be set to 'received'.
        After that the order ID will be returned in form of a Hypermedia link which
enables the possibilities to
        programmatically check the status the order by a piece of code of a GEMS API
consumer. In the following the
        status of the order can be queried by using the GEMS service route listed
below:</p>

        <ul><li><p><i>/services/order_status/{order_id}</i></p></li></ul>

        <p style="text-align: justify">The status of the order will be updated whenever
the processing chain reaches the
        next step in its internal calculation. In case of success, the user will receive a
link, which provides the
        ordered file. Additionally an e-mail notification is enabled, which sends out an
e-mail in case of success, with
        the resulting link or in case of failure, with a possible error explanation.
        </p>

        """

        order_id = None

        try:
            req_args = api.payload
            payload_check = check_message(req_args)

            if not payload_check[0]:
                error = BadRequestError(f'Payload failed the GeoVille standards:
{payload_check[1]}', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
task1_stitching', order_id)
                return {'message': error.to_dict()}, 404

            if not check_user_existence(req_args['user_id'], database_config_file,
database_config_section_api):
                error = NotFoundError('User ID does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
task1_stitching', order_id)
                return {'message': error.to_dict()}, 404

            if not check_service_name_existence(req_args['service_name'],
database_config_file,
                                                            database_config_section_api):
                error = NotFoundError('Service name does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
task1_stitching', order_id)
                return {'message': error.to_dict()}, 404

            service_id = get_service_id(req_args['service_name'], database_config_file,
database_config_section_api)
            order_id = generate_orderid(req_args['user_id'], service_id,
json.dumps(req_args))
```

```
            gemslog(LogLevel.INFO, f'Request payload: {req_args}', 'API-task1_stitching', 
order_id)

            publish_to_queue(req_args['service_name'], order_id, req_args)

            update_query = """UPDATE customer.service_orders
                                  set status = 'RECEIVED'
                              WHERE
                                  order_id = %s;
                           """
            execute_database(update_query, (order_id,), database_config_file, 
database_config_section_api, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}', 
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
task1_stitching', order_id)
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server', 
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
task1_stitching', order_id)
            return {'message': error.to_dict()}, 503

        except Exception as err:
            error = InternalServerErrorAPI(f'Unexpected error occurred: {err}', 
api.payload, traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
task1_stitching', order_id)
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Request successful', 'API-task1_stitching', order_id)
            return {
                       'message': 'Your order has been successfully submitted',
                       'links': {
                           'href': f'/services/order_status/{order_id}',
                           'rel': 'services',
                           'type': 'GET'
                       }
                   }, 202
```

5.1.166 services\backend_api\src\resources\resources_services\task_2_apply_model\task_2_apply_model.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is 
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Task 2 apply model API call
#
# Date created: 01.06.2020
# Date last modified: 15.04.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.04
```

```
#
##########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from geoville_ms_orderid_generator.generator import generate_orderid
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import service_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_service_name_existence, check_user_existence,
get_service_id
from lib.general_helper_methods import publish_to_queue
from models.general_models.general_models import service_success_response_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from models.models_services.task_2_apply_model.task_2_apply_model_model import
t2_apply_model_request_model
from check_message.check_message import check_message
from oauth.oauth2 import require_oauth
import json
import traceback


##########################################################################################
##############################
# Resource definition for the apply model (Task 2) API call
##########################################################################################
##############################

@api.expect(t2_apply_model_request_model)
@api.header('Content-Type', 'application/json')
class Task2ApplyModel(Resource):
    """ Class for handling the POST request

    This class defines the API call for starting the apply model (Task 2) workflow. The
class consists of one methods
    which accepts a POST request. For the POST request a JSON with several parameters is
required, defined in the
    corresponding model.

    """


##########################################################################################
##########################
    # Method for handling the POST request

##########################################################################################
##########################

    @require_oauth(['admin', 'apply_model_task_2'])
    @api.expect(auth_header_parser)
    @api.response(202, 'Order Received', service_success_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for calculating features and applying models for CLC+ task 2

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>model_path (str): parameter for saving data</i></p></li>
        <li><p><i>start_date (str): Oldest acquisition date to consider in the format
YYYY-MM-DD</i></p></li>
        <li><p><i>end_date (str): Newest acquisition date to consider in the format YYYY-
MM-DD</i></p></li>
        <li><p><i>processing_unit_name (str): Name of the processing unit</i></p></li>
        <li><p><i>cloud_cover (int): Maximum cloud cover (in %) to consider</i></p></li>
        <li><p><i>interval_size (int): Time difference in days for temporal
interpolation</i></p></li>
        <li><p><i>s1_bands (list): List of names of the required Sentinel-1 bands or
indices:
                                    ['ASC_DVVVH', 'ASC_NDVVVH', 'ASC_RVVVH', 'ASC_VV',
'ASC_VH', 'DSC_DVVVH',
                                    'DSC_NDVVVH', 'DSC_RVVVH', 'DSC_VV',
'DSC_VH']</i></p></li>
        <li><p><i>s2_bands (list): list of names of the required Sentinel-2 bands or
indices:
                                     ['B01', 'B02', 'B03', 'B04', 'B05', 'B06', 'B07',
'B08', 'B09', 'B10', 'B11', 'B12',
                                     'B8A', 'BRGHT', 'IRECI', 'NBR', 'NDVI', 'NDWI',
'NDWIGREEN', 'NGDR',
                                     'RENDVI']</i></p></li>
        <li><p><i>precalculated_features (list): Names of the required auxiliary features:
                                                  ['geomorpho90', 'distance',
'dem']</i></p></li>
        <li><p><i>use_cache (bool): Use cache results</i></p></li>
        <li><p><i>aoi_coverage (int): Minimum coverage in percent for one scene of an
AOI</i></p></li>
        <li><p><i>user_id (str): User specific client ID</i></p></li>
        <li><p><i>service_name (str): Unique name of the service to be called. Name should
not be changed</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">After the request was successfully received by the
GEMS API, an order ID will be
        created for the submitted job in the GEMS backend and stored in a database with
all necessary information for
        the consumer and the system itself. The initial status of the order in the
database will be set to 'received'.
        After that the order ID will be returned in form of a Hypermedia link which
enables the possibilities to
        programmatically check the status the order by a piece of code of a GEMS API
consumer. In the following the
        status of the order can be queried by using the GEMS service route listed
below:</p>

        <ul><li><p><i>/services/order_status/{order_id}</i></p></li></ul>

        <p style="text-align: justify">The status of the order will be updated whenever
the processing chain reaches the
        next step in its internal calculation. In case of success, the user will receive a
link, which provides the
        ordered file. Additionally an e-mail notification is enabled, which sends out an
e-mail in case of success, with
        the resulting link or in case of failure, with a possible error explanation.
        </p>

        """
        order_id = None

        try:
            req_args = api.payload

            payload_check = check_message(req_args)

            if not payload_check[0]:
                error = BadRequestError(f'Payload failed the GeoVille standards:
{payload_check[1]}', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
apply_model', order_id)
                return {'message': error.to_dict()}, 404

            if not check_user_existence(req_args['user_id'], database_config_file,
database_config_section_api):
                error = NotFoundError('User ID does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
apply_model', order_id)
                return {'message': error.to_dict()}, 404

            if not check_service_name_existence(req_args['service_name'],
database_config_file,
                                                        database_config_section_api):
                error = NotFoundError('Service name does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
apply_model', order_id)
                return {'message': error.to_dict()}, 404

            if req_args['start_date'] > req_args['end_date']:
                error = BadRequestError('Start date is greater than end date', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
apply_model', order_id)
                return {'message': error.to_dict()}, 400

            service_id = get_service_id(req_args['service_name'], database_config_file,
database_config_section_api)
            order_id = generate_orderid(req_args['user_id'], service_id,
json.dumps(req_args))
            gemslog(LogLevel.INFO, f'Request payload: {req_args}', 'API-apply_model',
order_id)

            publish_to_queue(req_args['service_name'], order_id, req_args)

            update_query = """UPDATE customer.service_orders
                                  set status = 'RECEIVED'
                              WHERE
                                  order_id = %s;
                           """
            execute_database(update_query, (order_id,), database_config_file,
database_config_section_api, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-apply_model',
order_id)
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-apply_model',
order_id)
            return {'message': error.to_dict()}, 503

        except Exception as err:
            error = InternalServerErrorAPI(f'Unexpected error occurred: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-apply_model',
order_id)
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Request successful', 'API-apply_model', order_id)
            return {
                       'message': 'Your order has been successfully submitted',
                       'links': {
                           'href': f'/services/order_status/{order_id}',
                           'rel': 'services',
                           'type': 'GET'
                       }
                   }, 202
```

5.1.167 services\backend_api\src\resources\resources_services\task_2_apply_model_fast_lane\task_2_apply_model_fast_lane.py

```
##########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Task 2 apply model fast lane API call
#
# Date created: 01.06.2020
# Date last modified: 15.04.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.04
#
##########################################################################################
##############################

from check_message.check_message import check_message
from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from geoville_ms_orderid_generator.generator import generate_orderid
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import service_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_service_name_existence, check_user_existence,
get_service_id
from lib.general_helper_methods import publish_to_queue
from models.general_models.general_models import service_success_response_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
```

```
from
models.models_services.task_2_apply_model_fast_lane.task_2_apply_model_fast_lane_model
import t2_apply_model_fast_lane_request_model
from oauth.oauth2 import require_oauth
import json
import traceback


##########################################################################################
##############################
# Resource definition for the apply model fast lane(Task 2) API call
##########################################################################################
##############################

@api.expect(t2_apply_model_fast_lane_request_model)
@api.header('Content-Type', 'application/json')
class Task2ApplyModelFastLane(Resource):
    """ Class for handling the POST request

    This class defines the API call for starting the apply model fast lane (Task 2)
workflow. The class consists of one
    methods which accepts a POST request. For the POST request a JSON with several
parameters is required, defined in
    the corresponding model.

    """


##########################################################################################
##########################
    # Method for handling the POST request

##########################################################################################
##########################

    @require_oauth(['admin', 'apply_model_task_2'])
    @api.expect(auth_header_parser)
    @api.response(202, 'Order Received', service_success_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for calculating features and applying models for CLC+ task 2
(fast lane)

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>model_path (str): parameter for saving data</i></p></li>
        <li><p><i>start_date (str): Oldest acquisition date to consider in the format
YYYY-MM-DD</i></p></li>
        <li><p><i>end_date (str): Newest acquisition date to consider in the format YYYY-
MM-DD</i></p></li>
        <li><p><i>processing_unit_name (str): Name of the processing unit</i></p></li>
        <li><p><i>cloud_cover (int): Maximum cloud cover (in %) to consider</i></p></li>
        <li><p><i>interval_size (int): Time difference in days for temporal
interpolation</i></p></li>
        <li><p><i>s1_bands (list): List of names of the required Sentinel-1 bands or
indices:
                                   ['ASC_DVVVH', 'ASC_NDVVVH', 'ASC_RVVVH', 'ASC_VV',
'ASC_VH', 'DSC_DVVVH',
                                   'DSC_NDVVVH', 'DSC_RVVVH', 'DSC_VV',
'DSC_VH']</i></p></li>
```

```
        <li><p><i>s2_bands (list): list of names of the required Sentinel-2 bands or
indices:
                                    ['B01', 'B02', 'B03', 'B04', 'B05', 'B06', 'B07',
'B08', 'B09', 'B10', 'B11', 'B12',
                                    'B8A', 'BRGHT', 'IRECI', 'NBR', 'NDVI', 'NDWI',
'NDWIGREEN', 'NGDR',
                                    'RENDVI']</i></p></li>
        <li><p><i>precalculated_features (list): Names of the required auxiliary features:
                                                 ['geomorpho90', 'distance',
'dem']</i></p></li>
        <li><p><i>use_cache (bool): Use cache results</i></p></li>
        <li><p><i>aoi_coverage (int): Minimum coverage in percent for one scene of an
AOI</i></p></li>
        <li><p><i>user_id (str): User specific client ID</i></p></li>
        <li><p><i>service_name (str): Unique name of the service to be called. Name should
not be changed</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">After the request was successfully received by the
GEMS API, an order ID will be
        created for the submitted job in the GEMS backend and stored in a database with
all necessary information for
        the consumer and the system itself. The initial status of the order in the
database will be set to 'received'.
        After that the order ID will be returned in form of a Hypermedia link which
enables the possibilities to
        programmatically check the status the order by a piece of code of a GEMS API
consumer. In the following the
        status of the order can be queried by using the GEMS service route listed
below:</p>

        <ul><li><p><i>/services/order_status/{order_id}</i></p></li></ul>

        <p style="text-align: justify">The status of the order will be updated whenever
the processing chain reaches the
        next step in its internal calculation. In case of success, the user will receive a
link, which provides the
        ordered file. Additionally an e-mail notification is enabled, which sends out an
e-mail in case of success, with
        the resulting link or in case of failure, with a possible error explanation.
        </p>

        """

        order_id = None

        try:
            req_args = api.payload

            payload_check = check_message(req_args)

            if not payload_check[0]:
                error = BadRequestError(f'Payload failed the GeoVille standards:
{payload_check[1]}', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
apply_model', order_id)
                return {'message': error.to_dict()}, 404

            if not check_user_existence(req_args['user_id'], database_config_file,
database_config_section_api):
                error = NotFoundError('User ID does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
apply_model', order_id)
                return {'message': error.to_dict()}, 404

            if not check_service_name_existence(req_args['service_name'],
database_config_file,
                                                database_config_section_api):
                error = NotFoundError('Service name does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
apply_model', order_id)
                return {'message': error.to_dict()}, 404

            if req_args['start_date'] > req_args['end_date']:
                error = BadRequestError('Start date is greater than end date', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
apply_model', order_id)
                return {'message': error.to_dict()}, 400

            service_id = get_service_id(req_args['service_name'], database_config_file,
database_config_section_api)
            order_id = generate_orderid(req_args['user_id'], service_id,
json.dumps(req_args))
            gemslog(LogLevel.INFO, f'Request payload: {req_args}', 'API-apply_model',
order_id)

            publish_to_queue(req_args['service_name'], order_id, req_args)

            update_query = """UPDATE customer.service_orders
                                  set status = 'RECEIVED'
                              WHERE
                                  order_id = %s;
                           """
            execute_database(update_query, (order_id,), database_config_file,
database_config_section_api, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-apply_model',
order_id)
            return {'message': error.to_dict()}, 400

        except AttributeError as err:
            error = ServiceUnavailableError(f'Could not connect to the database server:
{err}', '', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-apply_model',
order_id)
            return {'message': error.to_dict()}, 503

        except Exception as err:
            error = InternalServerErrorAPI(f'Unexpected error occurred: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-apply_model',
order_id)
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Request successful', 'API-apply_model', order_id)
            return {
                       'message': 'Your order has been successfully submitted',
                       'links': {
                           'href': f'/services/order_status/{order_id}',
                           'rel': 'services',
                           'type': 'GET'
                       }
                   }, 202
```

5.1.168 services\backend_api\src\resources\resources_services\task_2_feature_calculation\task_2_feature_calculation.py

```
###########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Task 2 feature calculation API call
#
# Date created: 01.06.2020
# Date last modified: 10.02.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.02
#
########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from geoville_ms_orderid_generator.generator import generate_orderid
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import service_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_service_name_existence, check_user_existence,
get_service_id
from lib.general_helper_methods import publish_to_queue
from models.general_models.general_models import service_success_response_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from models.models_services.task_2_feature_calculation.task_2_feature_calculation_model
import t2_feature_calculation_request_model
from check_message.check_message import check_message
from oauth.oauth2 import require_oauth
import json
import traceback


########################################################################################
##############################
# Resource definition for the feature calculation (Task 2) API call
########################################################################################
##############################

@api.expect(t2_feature_calculation_request_model)
@api.header('Content-Type', 'application/json')
class Task2FeatureCalculation(Resource):
    """ Class for handling the POST request

    This class defines the API call for starting the feature calculation (Task 2)
workflow. The class consists of one
    methods which accepts a POST request. For the POST request a JSON with several
parameters is required, defined in
    the corresponding model.

    """


########################################################################################
##########################
```

```
    # Method for handling the POST request
##########################################################################################
##########################

    @require_oauth(['admin', 'feature_calculation_task_2'])
    @api.expect(auth_header_parser)
    @api.response(202, 'Order Received', service_success_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for calculating interpolation features for CLC+ task 2

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>start_date (str): Oldest acquisition date to consider in the format
YYYY-MM-DD</i></p></li>
        <li><p><i>end_date (str): Newest acquisition date to consider in the format YYYY-
MM-DD</i></p></li>
        <li><p><i>processing_unit_name (str): Name of the processing unit</i></p></li>
        <li><p><i>cloud_cover (int): Maximum cloud cover (in %) to consider</i></p></li>
        <li><p><i>interval_size (int): Time difference in days for temporal
interpolation</i></p></li>
        <li><p><i>s1_bands (list): List of names of the required Sentinel-1 bands or
indices:
                                   ['ASC_DVVVH', 'ASC_NDVVVH', 'ASC_RVVVH', 'ASC_VV',
'ASC_VH', 'DSC_DVVVH',
                                   'DSC_NDVVVH', 'DSC_RVVVH', 'DSC_VV',
'DSC_VH']</i></p></li>
        <li><p><i>s2_bands (list): list of names of the required Sentinel-2 bands or
indices:
                                    ['B01', 'B02', 'B03', 'B04', 'B05', 'B06', 'B07',
'B08', 'B09', 'B10', 'B11', 'B12',
                                    'B8A', 'BRGHT', 'IRECI', 'NBR', 'NDVI', 'NDWI',
'NDWIGREEN', 'NGDR',
                                    'RENDVI']</i></p></li>
        <li><p><i>precalculated_features (list): Names of the required auxiliary features:
                                                 ['geomorpho90', 'distance',
'dem']</i></p></li>
        <li><p><i>use_cache (bool): Use cache results</i></p></li>
        <li><p><i>aoi_coverage (int): Minimum coverage in percent for one scene of an
AOI</i></p></li>
        <li><p><i>user_id (str): User specific client ID</i></p></li>
        <li><p><i>service_name (str): Unique name of the service to be called. Name should
not be changed</i></p></li>
        </ul>

        <br><b>Result:</b>
        <p style="text-align: justify">After the request was successfully received by the
GEMS API, an order ID will be
        created for the submitted job in the GEMS backend and stored in a database with
all necessary information for
        the consumer and the system itself. The initial status of the order in the
database will be set to 'received'.
        After that the order ID will be returned in form of a Hypermedia link which
enables the possibilities to
        programmatically check the status the order by a piece of code of a GEMS API
consumer. In the following the
        status of the order can be queried by using the GEMS service route listed
below:</p>
```

```
        <ul><li><p><i>/services/order_status/{order_id}</i></p></li></ul>

        <p style="text-align: justify">The status of the order will be updated whenever
the processing chain reaches the
        next step in its internal calculation. In case of success, the user will receive a
link, which provides the
        ordered file. Additionally an e-mail notification is enabled, which sends out an
e-mail in case of success, with
        the resulting link or in case of failure, with a possible error explanation.
        </p>

        """

        order_id = None

        try:
            req_args = api.payload

            payload_check = check_message(req_args)

            if not payload_check[0]:
                error = BadRequestError(f'Payload failed the GeoVille standards:
{payload_check[1]}', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
feature_calculation_t2', order_id)
                return {'message': error.to_dict()}, 404

            if not check_user_existence(req_args['user_id'], database_config_file,
database_config_section_api):
                error = NotFoundError('User ID does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
feature_calculation_t2', order_id)
                return {'message': error.to_dict()}, 404

            if not check_service_name_existence(req_args['service_name'],
database_config_file,
                                                database_config_section_api):
                error = NotFoundError('Service name does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
feature_calculation_t2', order_id)
                return {'message': error.to_dict()}, 404

            if req_args['start_date'] > req_args['end_date']:
                error = BadRequestError('Start date is greater than end date', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
feature_calculation_t2', order_id)
                return {'message': error.to_dict()}, 400

            service_id = get_service_id(req_args['service_name'], database_config_file,
database_config_section_api)
            order_id = generate_orderid(req_args['user_id'], service_id,
json.dumps(req_args))
            gemslog(LogLevel.INFO, f'Request payload: {req_args}', 'API-
feature_calculation_t2', order_id)

            publish_to_queue(req_args['service_name'], order_id, req_args)

            update_query = """UPDATE customer.service_orders
                                  set status = 'RECEIVED'
                              WHERE
                                  order_id = %s;
                           """
            execute_database(update_query, (order_id,), database_config_file,
database_config_section_api, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
```

```
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
feature_calculation_t2', order_id)
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
feature_calculation_t2', order_id)
            return {'message': error.to_dict()}, 503

        except Exception as err:
            error = InternalServerErrorAPI(f'Unexpected error occurred: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
feature_calculation_t2', order_id)
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Request successful', 'API-feature_calculation_t2',
order_id)
            return {
                       'message': 'Your order has been successfully submitted',
                       'links': {
                           'href': f'/services/order_status/{order_id}',
                           'rel': 'services',
                           'type': 'GET'
                       }
                   }, 202
```

5.1.169 services\backend_api\src\resources\resources_services\vector_class_attribution\vector_class_attribution.py

```
########################################################################################
##############################
#
# Copyright (c) 2021, GeoVille Information Systems GmbH
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without modification, is
prohibited for all commercial
# applications without licensing by GeoVille GmbH.
#
# Vector class attribution API call
#
# Date created: 15.04.2021
# Date last modified: 15.04.2021
#
# __author__  = Michel Schwandner (schwandner@geoville.com)
# __version__ = 21.04
#
########################################################################################
##############################

from error_classes.http_error_400.http_error_400 import BadRequestError
from error_classes.http_error_404.http_error_404 import NotFoundError
from error_classes.http_error_500.http_error_500 import InternalServerErrorAPI
from error_classes.http_error_503.http_error_503 import ServiceUnavailableError
from flask_restx import Resource
from geoville_ms_database.geoville_ms_database import execute_database
from geoville_ms_logging.geoville_ms_logging import gemslog, LogLevel
from geoville_ms_orderid_generator.generator import generate_orderid
from init.init_env_variables import database_config_file, database_config_section_api
from init.namespace_constructor import service_namespace as api
from lib.auth_header import auth_header_parser
from lib.database_helper import check_service_name_existence, check_user_existence,
get_service_id
from lib.general_helper_methods import publish_to_queue
from models.general_models.general_models import service_success_response_model
from models.models_error.http_error_400 import error_400_model
from models.models_error.http_error_401 import error_401_model
from models.models_error.http_error_403 import error_403_model
from models.models_error.http_error_404 import error_404_model
from models.models_error.http_error_500 import error_500_model
from models.models_error.http_error_503 import error_503_model
from models.models_services.vector_class_attribution.vector_class_attribution_model import
\
    vector_class_attribution_request_model
from oauth.oauth2 import require_oauth
import json
import traceback


################################################################################################
##############################
# Resource definition for the vector class attribution API call
################################################################################################
##############################

@api.expect(vector_class_attribution_request_model)
@api.header('Content-Type', 'application/json')
class VectorClassAttribution(Resource):
    """ Class for handling the POST request

    This class defines the API call for starting the vector class attribution workflow.
The class consists of one
    methods which accepts a POST request. For the POST request a JSON with several
parameters is required, defined in
    the corresponding model.

    """


################################################################################################
##########################
    # Method for handling the POST request

################################################################################################
##########################

    @require_oauth(['admin', 'vector_class_attribution'])
    @api.expect(auth_header_parser)
    @api.response(202, 'Order Received', service_success_response_model)
    @api.response(400, 'Validation Error', error_400_model)
    @api.response(401, 'Unauthorized', error_401_model)
    @api.response(403, 'Forbidden', error_403_model)
    @api.response(404, 'Not Found', error_404_model)
    @api.response(500, 'Internal Server Error', error_500_model)
    @api.response(503, 'Service Unavailable', error_503_model)
    def post(self):
        """ POST definition for starting off the vector class attribution workflow

        <p style="text-align: justify">This method defines the handler of the POST request
for starting off the vector
        class attribution processing chain for CLC+ Task 2.2 and Task 3. It is an
asynchronous call and thus, it does
        not return the requested data immediately but generates an order ID. After the
request has been submitted
        successfully, a message to a RabbitMQ queue will be send. A listener in the
backend triggers the further
        procedure, starting with the job scheduling of the order. The final result will be
stored in a database in form
        of a link and can be retrieved via the browser. To access the service it is
necessary to generate a valid Bearer
```

```
        token with sufficient access rights, otherwise the request will return a HTTP
status code 401 or 403. In case of
        those errors, please contact the GeoVille service team for any support.</p>

        <br><b>Description:</b>
        <p style="text-align: justify">This method can be used for both Task 2.2. and Task
3 for the Production of the
        CLC+ Backbone. The basic steps of this methods are:</p>
        <ul>
        <li><p><i>Preprocessing: Mosaics and clips input rasters to target vector file
extent</i></p></li>
        <li><p><i>Extraction: Extracts values for each polygon and applies a function to
these values.
        The result will be store in a csv file)</i></p></li>
        <li><p><i>Postprocessing / QC: Checks if any errors occurred during the extraction
and prepares the csv files
        for the subsequent insertion into the HDF5 cubes</p></li>
        <li><p><i>Upload to S3</i></p></li>
        </ul>

        <br><b>Request headers:</b>
        <ul>
        <li><p><i>Authorization: Bearer token in the format "Bearer XXXX"</i></p></li>
        </ul>

        <br><b>Request payload:</b>
        <ul>
        <li><p><i>vector (str): Path to Vector File from Task 1. Can be any format such as
gdb, shp, gpkg, etc...
         (e.g. "/vsis3/task22/tests/in/test1.shp")</i></p></li>
        <li><p><i>raster (str): List of Input Rasters that will be used for the extraction
         (e.g. "/vsis3/task22/tests/in/test.tif /vsis3/task22/tests/in/test2.tif") -
separated with a whitespace</i></p></li>
        <li><p><i>subproduction_unit_name (int): ID of the Subproduction Unit (e.g.
214)</i></p></li>
        <li><p><i>id_column (str): Name of the ID column of the provided Vector File.
(e.g. "id")</i></p></li>
        <li><p><i>na_value (int): NA Value that should be ignored during extraction (e.g.
255)</i></p></li>
        <li><p><i>method (str): Name of the aggregation method for the extracted values.
To call Task 2.2. specify
        "clcp_vector_class", for Task 3 currently the following methods are implemented:
"mean", "sd", "statistics"
         (Combination of mean and sd), "relative_count" (Relative occurrence of a specified
class per polygon. To call
        the relative_count method "method_params" have to be specified indicating which
value should be regarded),
        "min", "max", "quantile", "ffi" and "masl".
        </i></p></li>
        <li><p><i>method_params (str): Some methods expect parameters. (e.g. method
relative_count: "0 2" In this case
        the relative occurrence inside each polygon for the classes 0 and 2 will be
extracted)</i></p></li>
        <li><p><i>col_names (str): List of column names, same number as output parameters
for a specific method
         (e.g. 2 for statistics "IMD_mean IMD_sd")</i></p></li>
        <li><p><i>reference_year (str): Year that will be used during post processing to
create a "reference_year" column.
        The expected format is "YYYY"</i></p></li>
        <li><p><i>bucket_path (str): Path including the S3 bucket name where the extracted
data and QC reports should
        be stored. Please note that the folder on S3 doesn't have to exist, since it will
be automatically created
        during this process. However pay close attention to the file path naming
convention:
        "bucketname/folder/subfolder/". Note that there is no leading "/" before the
bucketname and a mandatory "/"
        after the folder name!</i></p></li>
           <li><p><i>config (str): S3 / GDAL config parameters for reading and writing
from / to S3.
```

```
           (e.g. "AWS_SECRET_ACCESS_KEY 123abc AWS_S3_ENDPOINT cf2.cloudferro.com:8080
AWS_VIRTUAL_HOSTING FALSE
           AWS_ACCESS_KEY_ID abc123")</i></p></li>
        <li><p><i>user_id (str): User specific client ID</i></p></li>
        <li><p><i>service_name (str): Unique name of the service to be called. Name should
not be changed</i></p></li>
        </ul>



        <br><b>Result:</b>
        <p style="text-align: justify">After the request was successfully received by the
CLC+ API, an order ID will be
        created for the submitted job in the GEMS backend and stored in a database with
all necessary information for
        the consumer and the system itself. The initial status of the order in the
database will be set to 'received'.
        After that the order ID will be returned in form of a Hypermedia link which
enables the possibilities to
        programmatically check the status the order by a piece of code of a CLC+ API
consumer. In the following the
        status of the order can be queried by using the GEMS service route listed
below:</p>

        <ul><li><p><i>/services/order_status/{order_id}</i></p></li></ul>

        <p style="text-align: justify">The status of the order will be updated whenever
the processing chain reaches
        the next step in its internal calculation. In case of success, the user will
receive a link, which provides the
        ordered file. Additionally an e-mail notification is enabled, which sends out an
e-mail in case of success,
        with the resulting link or in case of failure, with a possible error explanation.
        </p>

        """

        order_id = None

        try:
            req_args = api.payload

            if not check_user_existence(req_args['user_id'], database_config_file,
database_config_section_api):
                error = NotFoundError('User ID does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
vector_class_attribution', order_id)
                return {'message': error.to_dict()}, 404

            if not check_service_name_existence(req_args['service_name'],
database_config_file,
                                                   database_config_section_api):
                error = NotFoundError('Service name does not exist', '', '')
                gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
vector_class_attribution', order_id)
                return {'message': error.to_dict()}, 404

            service_id = get_service_id(req_args['service_name'], database_config_file,
database_config_section_api)
            order_id = generate_orderid(req_args['user_id'], service_id,
json.dumps(req_args))
            gemslog(LogLevel.INFO, f'Request payload: {req_args}', 'API-
vector_class_attribution', order_id)

            publish_to_queue(req_args['service_name'], order_id, req_args)

            update_query = """UPDATE customer.service_orders
                                 set status = 'RECEIVED'
                             WHERE
                                 order_id = %s;
```

```
                              """
            execute_database(update_query, (order_id,), database_config_file,
database_config_section_api, True)

        except KeyError as err:
            error = BadRequestError(f'Key error resulted in a BadRequest: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.WARNING, f"'message': {error.to_dict()}", 'API-
vector_class_attribution', order_id)
            return {'message': error.to_dict()}, 400

        except AttributeError:
            error = ServiceUnavailableError('Could not connect to the database server',
'', '')
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
vector_class_attribution', order_id)
            return {'message': error.to_dict()}, 503

        except Exception as err:
            error = InternalServerErrorAPI(f'Unexpected error occurred: {err}',
api.payload, traceback.format_exc())
            gemslog(LogLevel.ERROR, f"'message': {error.to_dict()}", 'API-
vector_class_attribution', order_id)
            return {'message': error.to_dict()}, 500

        else:
            gemslog(LogLevel.INFO, f'Request successful', 'API-vector_class_attribution',
order_id)
            return {
                       'message': 'Your order has been successfully submitted',
                       'links': {
                           'href': f'/services/order_status/{order_id}',
                           'rel': 'services',
                           'type': 'GET'
                       }
                   }, 202
```

5.1.170 services\database\create_certificates.sh

```
#!/bin/bash

# Script variables
DIR=ca_certificates
CERT_DURATION=3650

# Creates the directory if not exists already
if [[ ! -e $DIR ]]; then
    mkdir $DIR
fi

# Generates a private key without passphrase
openssl genrsa -out ./ca_certificates/server.key 2048

# Generates the server certificate
openssl req -new -key ./ca_certificates/server.key \
        -days $CERT_DURATION \
        -out ./ca_certificates/server.crt \
        -x509 \
        -subj '/C=AT/ST=Tyrol/L=Innsbruck/O=GeoVille Information Systems and Data
Processing GmbH/CN=api.clcplusbackbone.geoville.com/emailAddress=IT-Services@geoville.com'

# Generates the server certificate
cp ./ca_certificates/server.crt ./ca_certificates/root.crt
```

5.1.171 services\database\Dockerfile

```
FROM postgis/postgis:12-master

RUN mkdir ca_certificates
COPY /ca_certificates/server.crt /ca_certificates
COPY /ca_certificates/server.key /ca_certificates
COPY /ca_certificates/root.crt /ca_certificates

RUN chmod 400 /ca_certificates/*
RUN chown postgres:postgres /ca_certificates/*
```

5.1.172 services\database\postgresql.conf

```
# -----------------------------
# PostgreSQL configuration file
# -----------------------------
#
# This file consists of lines of the form:
#
#   name = value
#
# (The "=" is optional.)  Whitespace may be used.  Comments are introduced with
# "#" anywhere on a line.  The complete list of parameter names and allowed
# values can be found in the PostgreSQL documentation.
#
# The commented-out settings shown in this file represent the default values.
# Re-commenting a setting is NOT sufficient to revert it to the default value;
# you need to reload the server.
#
# This file is read on server startup and when the server receives a SIGHUP
# signal.  If you edit the file on a running system, you have to SIGHUP the
# server for the changes to take effect, run "pg_ctl reload", or execute
# "SELECT pg_reload_conf()".  Some parameters, which are marked below,
# require a server shutdown and restart to take effect.
#
# Any parameter can also be given as a command-line option to the server, e.g.,
# "postgres -c log_connections=on".  Some parameters can be changed at run time
# with the "SET" SQL command.
#
# Memory units:  kB = kilobytes        Time units:  ms  = milliseconds
#                MB = megabytes                     s   = seconds
#                GB = gigabytes                     min = minutes
#                TB = terabytes                     h   = hours
#                                                   d   = days


#------------------------------------------------------------------------------
# FILE LOCATIONS
#------------------------------------------------------------------------------

# The default values of these variables are driven from the -D command-line
# option or PGDATA environment variable, represented here as ConfigDir.

#data_directory = 'ConfigDir'           # use data in another directory
                                 # (change requires restart)
#hba_file = 'ConfigDir/pg_hba.conf'     # host-based authentication file
                                 # (change requires restart)
#ident_file = 'ConfigDir/pg_ident.conf' # ident configuration file
                                 # (change requires restart)

# If external_pid_file is not explicitly set, no extra PID file is written.
#external_pid_file = ''                 # write an extra PID file
                                 # (change requires restart)
```

```
#------------------------------------------------------------------------------
# CONNECTIONS AND AUTHENTICATION
#------------------------------------------------------------------------------

# - Connection Settings -

listen_addresses = '*'
                                 # comma-separated list of addresses;
                                 # defaults to 'localhost'; use '*' for all
                                 # (change requires restart)
#port = 5432                     # (change requires restart)
max_connections = 200                   # (change requires restart)
#superuser_reserved_connections = 3     # (change requires restart)
#unix_socket_directories = '/tmp' # comma-separated list of directories
                                 # (change requires restart)
#unix_socket_group = ''                 # (change requires restart)
#unix_socket_permissions = 0777         # begin with 0 to use octal notation
                                 # (change requires restart)
#bonjour = off                          # advertise server via Bonjour
                                 # (change requires restart)
#bonjour_name = ''               # defaults to the computer name
                                 # (change requires restart)

# - TCP settings -
# see "man 7 tcp" for details

#tcp_keepalives_idle = 0         # TCP_KEEPIDLE, in seconds;
                                 # 0 selects the system default
#tcp_keepalives_interval = 0            # TCP_KEEPINTVL, in seconds;
                                 # 0 selects the system default
#tcp_keepalives_count = 0        # TCP_KEEPCNT;
                                 # 0 selects the system default
#tcp_user_timeout = 0                   # TCP_USER_TIMEOUT, in milliseconds;
                                 # 0 selects the system default

# - Authentication -

#authentication_timeout = 1min          # 1s-600s
#password_encryption = md5       # md5 or scram-sha-256
#db_user_namespace = off

# GSSAPI using Kerberos
#krb_server_keyfile = ''
#krb_caseins_users = off


# - SSL -

ssl = on
ssl_ca_file = '/ca_certificates/root.crt'
ssl_cert_file = '/ca_certificates/server.crt'
#ssl_crl_file = ''
ssl_key_file = '/ca_certificates/server.key'
#ssl_ciphers = 'HIGH:MEDIUM:+3DES:!aNULL' # allowed SSL ciphers
ssl_prefer_server_ciphers = on
#ssl_ecdh_curve = 'prime256v1'
#ssl_min_protocol_version = 'TLSv1'
#ssl_max_protocol_version = ''
#ssl_dh_params_file = ''
#ssl_passphrase_command = ''
#ssl_passphrase_command_supports_reload = off


#------------------------------------------------------------------------------
# RESOURCE USAGE (except WAL)
```

```
#------------------------------------------------------------------------------

# - Memory -

shared_buffers = 4GB                    # min 128kB
                                # (change requires restart)
#huge_pages = try               # on, off, or try
                                # (change requires restart)
#temp_buffers = 8MB             # min 800kB
#max_prepared_transactions = 0          # zero disables the feature
                                # (change requires restart)
# Caution: it is not advisable to set max_prepared_transactions nonzero unless
# you actively intend to use prepared transactions.
work_mem = 20971kB  # min 64kB
maintenance_work_mem = 1GB      # min 1MB
#autovacuum_work_mem = -1       # min 1MB, or -1 to use maintenance_work_mem
#max_stack_depth = 2MB                  # min 100kB
#shared_memory_type = mmap      # the default is the first option
                                # supported by the operating system:
                                #   mmap
                                #   sysv
                                #   windows
                                # (change requires restart)
#dynamic_shared_memory_type = posix     # the default is the first option
                                # supported by the operating system:
                                #   posix
                                #   sysv
                                #   windows
                                #   mmap
                                # (change requires restart)

# - Disk -

#temp_file_limit = -1                   # limits per-process temp file space
                                # in kB, or -1 for no limit

# - Kernel Resources -

#max_files_per_process = 1000           # min 25
                                # (change requires restart)

# - Cost-Based Vacuum Delay -

#vacuum_cost_delay = 0                  # 0-100 milliseconds (0 disables)
#vacuum_cost_page_hit = 1       # 0-10000 credits
#vacuum_cost_page_miss = 10             # 0-10000 credits
#vacuum_cost_page_dirty = 20            # 0-10000 credits
#vacuum_cost_limit = 200        # 1-10000 credits

# - Background Writer -

#bgwriter_delay = 200ms                 # 10-10000ms between rounds
#bgwriter_lru_maxpages = 100            # max buffers written/round, 0 disables
#bgwriter_lru_multiplier = 2.0          # 0-10.0 multiplier on buffers scanned/round
#bgwriter_flush_after = 0       # measured in pages, 0 disables

# - Asynchronous Behavior -

effective_io_concurrency = 2            # 1-1000; 0 disables prefetching
max_worker_processes = 2        # (change requires restart)
max_parallel_maintenance_workers = 1    # taken from max_parallel_workers
max_parallel_workers_per_gather =1      # taken from max_parallel_workers
#parallel_leader_participation = on
max_parallel_workers = 2        # maximum number of max_worker_processes that
                                # can be used in parallel operations
#old_snapshot_threshold = -1            # 1min-60d; -1 disables; 0 is immediate
```

```
					# (change requires restart)
#backend_flush_after = 0		# measured in pages, 0 disables


#------------------------------------------------------------------------------
# WRITE-AHEAD LOG
#------------------------------------------------------------------------------

# - Settings -

#wal_level = replica			# minimal, replica, or logical
					# (change requires restart)
#fsync = on				# flush data to disk for crash safety
					# (turning this off can cause
					# unrecoverable data corruption)
#synchronous_commit = on		# synchronization level;
					# off, local, remote_write, remote_apply, or on
#wal_sync_method = fsync		# the default is the first option
					# supported by the operating system:
					#   open_datasync
					#   fdatasync (default on Linux)
					#   fsync
					#   fsync_writethrough
					#   open_sync
#full_page_writes = on			# recover from partial page writes
#wal_compression = off			# enable compression of full-page writes
#wal_log_hints = off			# also do full page writes of non-critical updates
					# (change requires restart)
#wal_init_zero = on			# zero-fill new WAL files
#wal_recycle = on			# recycle WAL files
wal_buffers = 16MB			# min 32kB, -1 sets based on shared_buffers
					# (change requires restart)
#wal_writer_delay = 200ms		# 1-10000 milliseconds
#wal_writer_flush_after = 1MB		# measured in pages, 0 disables

#commit_delay = 0			# range 0-100000, in microseconds
#commit_siblings = 5			# range 1-1000

# - Checkpoints -

#checkpoint_timeout = 5min		# range 30s-1d
max_wal_size = 4GB
min_wal_size = 1GB
checkpoint_completion_target = 0.7	# checkpoint target duration, 0.0 - 1.0
#checkpoint_flush_after = 0		# measured in pages, 0 disables
#checkpoint_warning = 30s		# 0 disables

# - Archiving -

#archive_mode = off		# enables archiving; off, on, or always
				# (change requires restart)
#archive_command = ''		# command to use to archive a logfile segment
				# placeholders: %p = path of file to archive
				#               %f = file name only
				# e.g. 'test ! -f /mnt/server/archivedir/%f && cp %p
/mnt/server/archivedir/%f'
#archive_timeout = 0		# force a logfile segment switch after this
				# number of seconds; 0 disables

# - Archive Recovery -

# These are only used in recovery mode.

#restore_command = ''		# command to use to restore an archived logfile segment
				# placeholders: %p = path of file to restore
				#               %f = file name only
				# e.g. 'cp /mnt/server/archivedir/%f %p'
```

```
                          # (change requires restart)
#archive_cleanup_command = ''     # command to execute at every restartpoint
#recovery_end_command = '' # command to execute at completion of recovery

# - Recovery Target -

# Set these only when performing a targeted recovery.

#recovery_target = ''             # 'immediate' to end recovery as soon as a
                                # consistent state is reached
                          # (change requires restart)
#recovery_target_name = '' # the named restore point to which recovery will proceed
                          # (change requires restart)
#recovery_target_time = '' # the time stamp up to which recovery will proceed
                          # (change requires restart)
#recovery_target_xid = ''  # the transaction ID up to which recovery will proceed
                          # (change requires restart)
#recovery_target_lsn = ''  # the WAL LSN up to which recovery will proceed
                          # (change requires restart)
#recovery_target_inclusive = on # Specifies whether to stop:
                          # just after the specified recovery target (on)
                          # just before the recovery target (off)
                          # (change requires restart)
#recovery_target_timeline = 'latest'    # 'current', 'latest', or timeline ID
                          # (change requires restart)
#recovery_target_action = 'pause' # 'pause', 'promote', 'shutdown'
                          # (change requires restart)


#------------------------------------------------------------------------------
# REPLICATION
#------------------------------------------------------------------------------

# - Sending Servers -

# Set these on the master and on any standby that will send replication data.

#max_wal_senders = 10             # max number of walsender processes
                          # (change requires restart)
#wal_keep_segments = 0            # in logfile segments; 0 disables
#wal_sender_timeout = 60s  # in milliseconds; 0 disables

#max_replication_slots = 10       # max number of replication slots
                          # (change requires restart)
#track_commit_timestamp = off     # collect timestamp of transaction commit
                          # (change requires restart)

# - Master Server -

# These settings are ignored on a standby server.

#synchronous_standby_names = ''   # standby servers that provide sync rep
                          # method to choose sync standbys, number of sync standbys,
                          # and comma-separated list of application_name
                          # from standby(s); '*' = all
#vacuum_defer_cleanup_age = 0     # number of xacts by which cleanup is delayed

# - Standby Servers -

# These settings are ignored on a master server.

#primary_conninfo = ''                   # connection string to sending server
                                  # (change requires restart)
#primary_slot_name = ''                  # replication slot on sending server
                                  # (change requires restart)
#promote_trigger_file = ''        # file name whose presence ends recovery
```

```
#hot_standby = on                  # "off" disallows queries during recovery
                                   # (change requires restart)
#max_standby_archive_delay = 30s  # max delay before canceling queries
                                   # when reading WAL from archive;
                                   # -1 allows indefinite delay
#max_standby_streaming_delay = 30s      # max delay before canceling queries
                                   # when reading streaming WAL;
                                   # -1 allows indefinite delay
#wal_receiver_status_interval = 10s     # send replies at least this often
                                   # 0 disables
#hot_standby_feedback = off             # send info from standby to prevent
                                   # query conflicts
#wal_receiver_timeout = 60s             # time that receiver waits for
                                   # communication from master
                                   # in milliseconds; 0 disables
#wal_retrieve_retry_interval = 5s # time to wait before retrying to
                                   # retrieve WAL after a failed attempt
#recovery_min_apply_delay = 0           # minimum delay for applying changes during
recovery

# - Subscribers -

# These settings are ignored on a publisher.

#max_logical_replication_workers = 4    # taken from max_worker_processes
                                   # (change requires restart)
#max_sync_workers_per_subscription = 2  # taken from max_logical_replication_workers


#------------------------------------------------------------------------------
# QUERY TUNING
#------------------------------------------------------------------------------

# - Planner Method Configuration -

#enable_bitmapscan = on
#enable_hashagg = on
#enable_hashjoin = on
#enable_indexscan = on
#enable_indexonlyscan = on
#enable_material = on
#enable_mergejoin = on
#enable_nestloop = on
#enable_parallel_append = on
#enable_seqscan = on
#enable_sort = on
#enable_tidscan = on
#enable_partitionwise_join = off
#enable_partitionwise_aggregate = off
#enable_parallel_hash = on
#enable_partition_pruning = on

# - Planner Cost Constants -

#seq_page_cost = 1.0                    # measured on an arbitrary scale
random_page_cost = 4                    # same scale as above
#cpu_tuple_cost = 0.01                  # same scale as above
#cpu_index_tuple_cost = 0.005           # same scale as above
#cpu_operator_cost = 0.0025             # same scale as above
#parallel_tuple_cost = 0.1         # same scale as above
#parallel_setup_cost = 1000.0      # same scale as above

#jit_above_cost = 100000           # perform JIT compilation if available
                                   # and query more expensive than this;
                                   # -1 disables
#jit_inline_above_cost = 500000         # inline small functions if query is
                                   # more expensive than this; -1 disables
```

```
#jit_optimize_above_cost = 500000# use expensive JIT optimizations if
                                 # query is more expensive than this;
                                 # -1 disables

#min_parallel_table_scan_size = 8MB
#min_parallel_index_scan_size = 512kB
effective_cache_size = 12GB

# - Genetic Query Optimizer -

#geqo = on
#geqo_threshold = 12
#geqo_effort = 5                 # range 1-10
#geqo_pool_size = 0              # selects default based on effort
#geqo_generations = 0                  # selects default based on effort
#geqo_selection_bias = 2.0       # range 1.5-2.0
#geqo_seed = 0.0                 # range 0.0-1.0

# - Other Planner Options -

default_statistics_target = 100  # range 1-10000
#constraint_exclusion = partition# on, off, or partition
#cursor_tuple_fraction = 0.1           # range 0.0-1.0
#from_collapse_limit = 8
#join_collapse_limit = 8         # 1 disables collapsing of explicit
                                 # JOIN clauses
#force_parallel_mode = off
#jit = on                        # allow JIT compilation
#plan_cache_mode = auto                # auto, force_generic_plan or
                                 # force_custom_plan


#------------------------------------------------------------------------------
# REPORTING AND LOGGING
#------------------------------------------------------------------------------

# - Where to Log -

#log_destination = 'stderr'            # Valid values are combinations of
                                 # stderr, csvlog, syslog, and eventlog,
                                 # depending on platform.  csvlog
                                 # requires logging_collector to be on.

# This is used when logging to stderr:
#logging_collector = off         # Enable capturing of stderr and csvlog
                                 # into log files. Required to be on for
                                 # csvlogs.
                                 # (change requires restart)

# These are only used if logging_collector is on:
#log_directory = 'log'                 # directory where log files are written,
                                 # can be absolute or relative to PGDATA
#log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'      # log file name pattern,
                                 # can include strftime() escapes
#log_file_mode = 0600                  # creation mode for log files,
                                 # begin with 0 to use octal notation
#log_truncate_on_rotation = off        # If on, an existing log file with the
                                 # same name as the new log file will be
                                 # truncated rather than appended to.
                                 # But such truncation only occurs on
                                 # time-driven rotation, not on restarts
                                 # or size-driven rotation.  Default is
                                 # off, meaning append to existing files
                                 # in all cases.
#log_rotation_age = 1d                 # Automatic rotation of logfiles will
                                 # happen after that time.  0 disables.
```

```
#log_rotation_size = 10MB         # Automatic rotation of logfiles will
                                  # happen after that much log output.
                                  # 0 disables.

# These are relevant when logging to syslog:
#syslog_facility = 'LOCAL0'
#syslog_ident = 'postgres'
#syslog_sequence_numbers = on
#syslog_split_messages = on

# This is only relevant when logging to eventlog (win32):
# (change requires restart)
#event_source = 'PostgreSQL'

# - When to Log -

#log_min_messages = warning              # values in order of decreasing detail:
                                  #   debug5
                                  #   debug4
                                  #   debug3
                                  #   debug2
                                  #   debug1
                                  #   info
                                  #   notice
                                  #   warning
                                  #   error
                                  #   log
                                  #   fatal
                                  #   panic

#log_min_error_statement = error  # values in order of decreasing detail:
                                  #   debug5
                                  #   debug4
                                  #   debug3
                                  #   debug2
                                  #   debug1
                                  #   info
                                  #   notice
                                  #   warning
                                  #   error
                                  #   log
                                  #   fatal
                                  #   panic (effectively off)

#log_min_duration_statement = -1  # -1 is disabled, 0 logs all statements
                                  # and their durations, > 0 logs only
                                  # statements running at least this number
                                  # of milliseconds

#log_transaction_sample_rate = 0.0       # Fraction of transactions whose statements
                                  # are logged regardless of their duration. 1.0 logs all
                                  # statements from all transactions, 0.0 never logs.

# - What to Log -

#debug_print_parse = off
#debug_print_rewritten = off
#debug_print_plan = off
#debug_pretty_print = on
#log_checkpoints = off
#log_connections = off
#log_disconnections = off
#log_duration = off
#log_error_verbosity = default           # terse, default, or verbose messages
#log_hostname = off
#log_line_prefix = '%m [%p] '            # special values:
```

```
                                 #   %a = application name
                                 #   %u = user name
                                 #   %d = database name
                                 #   %r = remote host and port
                                 #   %h = remote host
                                 #   %p = process ID
                                 #   %t = timestamp without milliseconds
                                 #   %m = timestamp with milliseconds
                                 #   %n = timestamp with milliseconds (as a Unix epoch)
                                 #   %i = command tag
                                 #   %e = SQL state
                                 #   %c = session ID
                                 #   %l = session line number
                                 #   %s = session start timestamp
                                 #   %v = virtual transaction ID
                                 #   %x = transaction ID (0 if none)
                                 #   %q = stop here in non-session
                                 #        processes
                                 #   %% = '%'
                                 # e.g. '<%u%%%d> '
#log_lock_waits = off                   # log lock waits >= deadlock_timeout
#log_statement = 'none'                 # none, ddl, mod, all
#log_replication_commands = off
#log_temp_files = -1                    # log temporary files equal or larger
                                 # than the specified size in kilobytes;
                                 # -1 disables, 0 logs all temp files
#log_timezone = 'GMT'

#------------------------------------------------------------------------------
# PROCESS TITLE
#------------------------------------------------------------------------------

#cluster_name = ''               # added to process titles if nonempty
                                 # (change requires restart)
#update_process_title = on


#------------------------------------------------------------------------------
# STATISTICS
#------------------------------------------------------------------------------

# - Query and Index Statistics Collector -

track_activities = on
track_counts = on
#track_io_timing = off
#track_functions = none                 # none, pl, all
#track_activity_query_size = 1024 # (change requires restart)
#stats_temp_directory = 'pg_stat_tmp'


# - Monitoring -

#log_parser_stats = off
#log_planner_stats = off
#log_executor_stats = off
#log_statement_stats = off


#------------------------------------------------------------------------------
# AUTOVACUUM
#------------------------------------------------------------------------------

autovacuum = on                  # Enable autovacuum subprocess?  'on'
                                 # requires track_counts to also be on.
log_autovacuum_min_duration = 3600      # -1 disables, 0 logs all actions and
```

```
                                # their durations, > 0 logs only
                                # actions running at least this number
                                # of milliseconds.
autovacuum_max_workers = 3      # max number of autovacuum subprocesses
                                # (change requires restart)
#autovacuum_naptime = 1min      # time between autovacuum runs
autovacuum_vacuum_threshold = 50  # min number of row updates before vacuum
autovacuum_analyze_threshold = 50 # min number of row updates before analyze
autovacuum_vacuum_scale_factor = 0.2    # fraction of table size before vacuum
autovacuum_analyze_scale_factor = 0.1   # fraction of table size before analyze
#autovacuum_freeze_max_age = 200000000  # maximum XID age before forced vacuum
                                # (change requires restart)
#autovacuum_multixact_freeze_max_age = 400000000      # maximum multixact age
                                # before forced vacuum
                                # (change requires restart)
#autovacuum_vacuum_cost_delay = 2ms     # default vacuum cost delay for
                                # autovacuum, in milliseconds;
                                # -1 means use vacuum_cost_delay
#autovacuum_vacuum_cost_limit = -1      # default vacuum cost limit for
                                # autovacuum, -1 means use
                                # vacuum_cost_limit


#------------------------------------------------------------------------------
# CLIENT CONNECTION DEFAULTS
#------------------------------------------------------------------------------

# - Statement Behavior -

#client_min_messages = notice           # values in order of decreasing detail:
                                #   debug5
                                #   debug4
                                #   debug3
                                #   debug2
                                #   debug1
                                #   log
                                #   notice
                                #   warning
                                #   error
#search_path = '"$user", public'  # schema names
#row_security = on
#default_tablespace = ''        # a tablespace name, '' uses the default
#temp_tablespaces = ''                  # a list of tablespace names, '' uses
                                # only default tablespace
#default_table_access_method = 'heap'
#check_function_bodies = on
#default_transaction_isolation = 'read committed'
#default_transaction_read_only = off
#default_transaction_deferrable = off
#session_replication_role = 'origin'
#statement_timeout = 0                  # in milliseconds, 0 is disabled
#lock_timeout = 0               # in milliseconds, 0 is disabled
#idle_in_transaction_session_timeout = 0        # in milliseconds, 0 is disabled
#vacuum_freeze_min_age = 50000000
#vacuum_freeze_table_age = 150000000
#vacuum_multixact_freeze_min_age = 5000000
#vacuum_multixact_freeze_table_age = 150000000
#vacuum_cleanup_index_scale_factor = 0.1        # fraction of total number of tuples
                                        # before index cleanup, 0 always performs
                                        # index cleanup
#bytea_output = 'hex'                   # hex, escape
#xmlbinary = 'base64'
#xmloption = 'content'
#gin_fuzzy_search_limit = 0
#gin_pending_list_limit = 4MB
```

```
# - Locale and Formatting -

#datestyle = 'iso, mdy'
#intervalstyle = 'postgres'
#timezone = 'GMT'
#timezone_abbreviations = 'Default'     # Select the set of available time zone
                                 # abbreviations.  Currently, there are
                                 #   Default
                                 #   Australia (historical usage)
                                 #   India
                                 # You can create your own file in
                                 # share/timezonesets/.
#extra_float_digits = 1                 # min -15, max 3; any value >0 actually
                                 # selects precise output mode
#client_encoding = sql_ascii            # actually, defaults to database
                                 # encoding

# These settings are initialized by initdb, but they can be changed.
#lc_messages = 'C'                # locale for system error message
                                 # strings
#lc_monetary = 'C'                # locale for monetary formatting
#lc_numeric = 'C'                 # locale for number formatting
#lc_time = 'C'                          # locale for time formatting

# default configuration for text search
#default_text_search_config = 'pg_catalog.simple'

# - Shared Library Preloading -

#shared_preload_libraries = ''    # (change requires restart)
#local_preload_libraries = ''
#session_preload_libraries = ''
#jit_provider = 'llvmjit'         # JIT library to use

# - Other Defaults -

#dynamic_library_path = '$libdir'


#------------------------------------------------------------------------------
# LOCK MANAGEMENT
#------------------------------------------------------------------------------

#deadlock_timeout = 1s
#max_locks_per_transaction = 64         # min 10
                                 # (change requires restart)
#max_pred_locks_per_transaction = 64    # min 10
                                 # (change requires restart)
#max_pred_locks_per_relation = -2 # negative values mean
                                 # (max_pred_locks_per_transaction
                                 #  / -max_pred_locks_per_relation) - 1
#max_pred_locks_per_page = 2            # min 0


#------------------------------------------------------------------------------
# VERSION AND PLATFORM COMPATIBILITY
#------------------------------------------------------------------------------

# - Previous PostgreSQL Versions -

#array_nulls = on
#backslash_quote = safe_encoding # on, off, or safe_encoding
#escape_string_warning = on
#lo_compat_privileges = off
#operator_precedence_warning = off
#quote_all_identifiers = off
```

```
#standard_conforming_strings = on
#synchronize_seqscans = on

# - Other Platforms and Clients -

#transform_null_equals = off


#------------------------------------------------------------------------------
# ERROR HANDLING
#------------------------------------------------------------------------------

#exit_on_error = off                    # terminate session on any error?
#restart_after_crash = on         # reinitialize after backend crash?
#data_sync_retry = off                  # retry or panic on failure to fsync
                                  # data?
                                  # (change requires restart)


#------------------------------------------------------------------------------
# CONFIG FILE INCLUDES
#------------------------------------------------------------------------------

# These options allow settings to be loaded from files other than the
# default postgresql.conf.  Note that these are directives, not variable
# assignments, so they can usefully be given more than once.

#include_dir = '...'                    # include files ending in '.conf' from
                                  # a directory, e.g., 'nginx.conf'
#include_if_exists = '...'        # include file only if it exists
#include = '...'                  # include file


#------------------------------------------------------------------------------
# CUSTOMIZED OPTIONS
#------------------------------------------------------------------------------

# Add settings for extensions here
```

5.1.173 services\database\db_init_script\01_init_database.sh

```
#!/bin/bash

# Immediately exits if any error occurs during the script execution. If not set, an error
could occur and the script
# would continue its execution.
set -o errexit

# Creating an array that defines the environment variables that must be set. This can be
consumed later via arrray
# variable expansion ${REQUIRED_ENV_VARS[@]}.
readonly REQUIRED_ENV_VARS=(
  "DB_DATABASE_NAME"
)

# Checks if all of the required environment variables are set. If one of them isn't,
echoes a text explaining which one
# isn't and the name of the ones that need to be
for required_env_var in ${REQUIRED_ENV_VARS[@]}; do
  if [[ -z "${!required_env_var}" ]]; then
    echo "Error:
          Environment variable '$required_env_var' not set.
          Make sure you have the following environment variables set:
          - ${REQUIRED_ENV_VARS[@]}
          Aborting."
    exit 1
  fi
done

# Performs the initialization in the already-started PostgreSQL using the preconfigured
POSTGRE_USER user.
psql -v ON_ERROR_STOP=1 --username "$POSTGRES_USER" --dbname "$POSTGRES_USER" <<-EOSQL

   CREATE DATABASE $DB_DATABASE_NAME;

EOSQL
```

5.1.174 services\database\db_init_script\02_init_database.sql

```
-- Switch connection to clcplus_backend database
\c clcplus_backend;

-------------------------------------------------------------------------------------------
------------------------------
-- Creates the PostGIS extension to the public schema
-------------------------------------------------------------------------------------------
------------------------------
CREATE EXTENSION postgis SCHEMA "public";

-------------------------------------------------------------------------------------------
------------------------------
-- Creates the necessary schemas for the API
-------------------------------------------------------------------------------------------
------------------------------
CREATE SCHEMA customer AUTHORIZATION postgres;
CREATE SCHEMA msgeovilleconfig AUTHORIZATION postgres;
CREATE SCHEMA logging AUTHORIZATION postgres;

-------------------------------------------------------------------------------------------
------------------------------
-- Creates the table that holds the customer data
-------------------------------------------------------------------------------------------
------------------------------
CREATE TABLE customer.customer (
       customer_id varchar(128) NOT NULL,
       title varchar(3) NOT NULL,
       first_name varchar(64) NOT NULL,
       last_name varchar(64) NOT NULL,
       email varchar(128) NOT NULL,
       password varchar(128 NOT NULL),
       address varchar(128) NOT NULL,
       city varchar(64) NOT NULL,
       zip_code varchar(16) NOT NULL,
       country varchar(64) NOT NULL,
       nationality varchar(128) NULL,
       phone_number varchar(64) NOT NULL,
       company_name varchar(1000) NULL,
       active bool NULL DEFAULT true,
       created_at timestamp NOT NULL DEFAULT NOW(),
       updated_at timestamp NOT NULL DEFAULT NOW(),
       deleted_at timestamp NULL,
       CONSTRAINT customer_pk PRIMARY KEY (customer_id)
);

-------------------------------------------------------------------------------------------
------------------------------
-- Creates the table that holds the service data
-------------------------------------------------------------------------------------------
------------------------------
CREATE TABLE customer.services (
       service_id varchar(64) NOT NULL,
        service_name varchar(500) NULL,
        service_comment varchar(10000) NULL,
        service_validity bool NULL,
        service_owner_geoville varchar(500) NULL,
        external bool NOT NULL DEFAULT true,
        created_at timestamp NOT NULL DEFAULT NOW(),
        updated_at timestamp NOT NULL DEFAULT NOW(),
        deleted_at timestamp NULL,
        CONSTRAINT services_pk PRIMARY KEY (service_id)
);

INSERT INTO customer.services (service_id, service_name, service_comment,
service_validity, service_owner_geoville, created_at, external_service)
VALUES('5439922d772e8361d5aa6bb40180f7a8150757f39616a0be7ed246749fefde5e', 'logger', 'The
internal service that logs', TRUE, 'GeoVille', now(), FALSE);

------------------------------------------------------------------------------------------
------------------------------
-- Creates the table that holds the region of interest data
------------------------------------------------------------------------------------------
------------------------------
CREATE TABLE customer.region_of_interests (
        roi_id varchar(64) NOT NULL,
        roi_name varchar(64) NOT NULL,
        description text NULL,
        customer_id varchar(128) NOT NULL,
        geom geometry(MULTIPOLYGON) NULL,
        created_at timestamp NOT NULL DEFAULT NOW(),
        updated_at timestamp NOT NULL DEFAULT NOW(),
        deleted_at timestamp NULL,
        CONSTRAINT region_of_interests_pk PRIMARY KEY (roi_id),
        CONSTRAINT region_of_interests_fk_customer_id FOREIGN KEY (customer_id) REFERENCES
customer.customer(customer_id) ON UPDATE CASCADE ON DELETE CASCADE
);

------------------------------------------------------------------------------------------
------------------------------
-- Creates the table that holds the ROI to service mapping data
------------------------------------------------------------------------------------------
------------------------------
CREATE TABLE customer.roi_service_mapping (
        roi_id varchar(64) NOT NULL,
        service_id varchar(64) NOT NULL,
        created_at timestamp NOT NULL DEFAULT NOW(),
        updated_at timestamp NOT NULL DEFAULT NOW(),
        deleted_at timestamp NULL,
        CONSTRAINT roi_service_mapping_pk PRIMARY KEY (roi_id, service_id),
        CONSTRAINT roi_service_mapping_fk_roi_id FOREIGN KEY (roi_id) REFERENCES
customer.region_of_interests(roi_id) ON UPDATE CASCADE ON DELETE CASCADE,
        CONSTRAINT roi_service_mapping_fk_service_id FOREIGN KEY (service_id) REFERENCES
customer.services(service_id) ON UPDATE CASCADE ON DELETE CASCADE
);

------------------------------------------------------------------------------------------
------------------------------
-- Creates the table that holds the customer to service mapping data
------------------------------------------------------------------------------------------
------------------------------
CREATE TABLE customer.service_customer_mapping (
        customer_id varchar(128) NOT NULL,
        service_id varchar(64) NOT NULL,
        usage_validity bool NULL,
        usage_start timestamp NOT NULL,
        usage_stop timestamp NULL,
        usage_limit numeric(15) NULL,
        usage_interval varchar(64) NULL,
        created_at timestamp NOT NULL DEFAULT NOW(),
        updated_at timestamp NOT NULL DEFAULT NOW(),
        deleted_at timestamp NULL,
       CONSTRAINT service_customer_mapping_pk PRIMARY KEY (customer_id, service_id, 
usage_start),
       CONSTRAINT service_customer_mapping_fk_customer_id FOREIGN KEY (customer_id) 
REFERENCES customer.customer(customer_id) ON UPDATE CASCADE ON DELETE CASCADE,
       CONSTRAINT service_customer_mapping_fk_service_id FOREIGN KEY (service_id) 
REFERENCES customer.services(service_id) ON UPDATE CASCADE ON DELETE CASCADE
);

-----------------------------------------------------------------------------------------
------------------------------
-- Creates the table that holds the service order data
-----------------------------------------------------------------------------------------
------------------------------
CREATE TABLE customer.service_orders (
       customer_id varchar(128) NOT NULL,
       service_id varchar(128) NOT NULL,
       order_id varchar(64) NOT NULL,
       order_received timestamptz NOT NULL,
       order_started timestamptz NULL,
       order_stopped timestamptz NULL,
       cancelled_by_user bool NULL,
       cancelled_by_system bool NULL,
       status varchar(64) NULL,
       success bool NULL,
       "result" varchar(512) NULL,
       order_json jsonb NULL,
       created_at timestamp NOT NULL DEFAULT NOW(),
       updated_at timestamp NOT NULL DEFAULT NOW(),
       deleted_at timestamp NULL,
       CONSTRAINT service_orders_pk PRIMARY KEY (order_id),
       CONSTRAINT service_orders_fk_customer FOREIGN KEY (customer_id) REFERENCES 
customer.customer(customer_id) ON UPDATE CASCADE ON DELETE CASCADE,
       CONSTRAINT service_orders_fk_services FOREIGN KEY (service_id) REFERENCES 
customer.services(service_id) ON UPDATE CASCADE ON DELETE CASCADE
);

-----------------------------------------------------------------------------------------
------------------------------
-- Creates the table that holds the Airflow configuration data
-----------------------------------------------------------------------------------------
------------------------------
CREATE TABLE msgeovilleconfig.airflow_config (
       service_name varchar(500) NOT NULL,
       command varchar(1000) NOT NULL,
       description varchar(1000) NULL,
       created_at timestamp NOT NULL DEFAULT NOW(),
       updated_at timestamp NOT NULL DEFAULT NOW(),
       deleted_at timestamp NULL,
       CONSTRAINT airflow_config_pk PRIMARY KEY (service_name, command)
);

-----------------------------------------------------------------------------------------
------------------------------
-- Creates the table that holds the logging service configuration data
-----------------------------------------------------------------------------------------
------------------------------
CREATE TABLE msgeovilleconfig.logger_saver_config (
       "key" varchar(50) NOT NULL,
       value varchar(500) NOT NULL,
       CONSTRAINT logger_saver_config_pk PRIMARY KEY (key, value)
);

INSERT INTO msgeovilleconfig.logger_saver_config ("key", value) VALUES('duration_in_sec', 
'60');

-----------------------------------------------------------------------------------------
------------------------------
-- Creates the table that holds the message checker configuration data
```

```
-----------------------------------------------------------------------------------------
------------------------------
CREATE TABLE msgeovilleconfig.message_checker (
       "instance" varchar(64) NOT NULL,
       "key" varchar(64) NOT NULL,
       value varchar(500) NOT NULL,
       created_at timestamp NOT NULL DEFAULT NOW(),
       updated_at timestamp NOT NULL DEFAULT NOW(),
       deleted_at timestamp NULL,
       CONSTRAINT message_checker_pl PRIMARY KEY (instance, key, value)
);

-----------------------------------------------------------------------------------------
------------------------------
-- Creates the table that holds the message key data for de- and encryption of RabbitMQ
messages
-----------------------------------------------------------------------------------------
------------------------------
CREATE TABLE msgeovilleconfig.message_key (
       "name" varchar(500) NOT NULL,
       "key" varchar(50) NOT NULL,
       CONSTRAINT message_key_pk PRIMARY KEY (name, key)
);

INSERT INTO msgeovilleconfig.message_key ("name", "key") VALUES('message_key',
'GqcOMRSp6sOm33fyCsN2KxGh6Z-Vi2oLhYHikJ7UM1I=');

-----------------------------------------------------------------------------------------
------------------------------
-- Creates the table that holds the RabbitMQ queue configuration data
-----------------------------------------------------------------------------------------
------------------------------
CREATE TABLE msgeovilleconfig.message_queue_config (
       service_id varchar(64) NOT NULL,
       queue_name varchar(500) NOT NULL,
       host varchar(500) NOT NULL,
       port varchar(50) NOT NULL,
       created_at timestamp NOT NULL DEFAULT NOW(),
       updated_at timestamp NOT NULL DEFAULT NOW(),
       deleted_at timestamp NULL,
       CONSTRAINT message_queue_config_pk PRIMARY KEY (service_id, queue_name)
);

ALTER TABLE msgeovilleconfig.message_queue_config ADD CONSTRAINT constraint_fk FOREIGN KEY
(service_id) REFERENCES customer.services(service_id) ON DELETE CASCADE;
INSERT INTO msgeovilleconfig.message_queue_config (service_id, queue_name, host, port)
VALUES('5439922d772e8361d5aa6bb40180f7a8150757f39616a0be7ed246749fefde5e', 'tools_logger',
'api.clcplusbackbone.geoville.com', '5672');


-----------------------------------------------------------------------------------------
------------------------------
-- Creates the table that holds the logging entries
-----------------------------------------------------------------------------------------
------------------------------
CREATE TABLE logging.logging (
       id serial NOT NULL,
       service_name varchar(50) NULL,
       order_id varchar(64) NULL,
       log_level varchar(15) NOT NULL,
       log_message text NOT NULL,
       time_stamp timestamp NOT NULL DEFAULT NOW(),
       deleted_at timestamp NULL,
       CONSTRAINT logger_pkey PRIMARY KEY (id)
);
```

5.1.175 services\proxy_server\traefik\dynamic_config\config.yml

```
tls:
  options:
    # Set default for a minimum of TLS v1.2 with the most secure ciphers; the browser can
negotiate to TLS v1.3 if preferred.
    default:
      minVersion: VersionTLS12
      cipherSuites:
        # Recommended ciphers for TLSv1.2
        - TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
        - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
        - TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305
        # Recommended ciphers for TLSv1.3
        - TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
        - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
        - TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305

    # Custom option for only TLS v1.3; if browser does not support TLS v1.3, the page will
fail to load.
    tlsv13only:
      minVersion: VersionTLS13

http:
  middlewares:
    # Enabled Secure headers
    secure-headers:
      headers:
        sslRedirect: true
        frameDeny: true
        stsIncludeSubdomains: true
        stsPreload: true
        stsSeconds: 63072000
        contentTypeNosniff: true
        accessControlAllowMethods:
          - GET
          - POST
        accessControlMaxAge: 100
        addVaryheader: true
        contentSecurityPolicy: script-src 'self'
        referrerPolicy: origin-when-cross-origin

    # Semi Secure Headers to allow custom contentSecurityPolicy
    semi-secure-headers:
      headers:
        sslRedirect: true
        frameDeny: true
        stsIncludeSubdomains: true
        stsPreload: true
        stsSeconds: 63072000
        contentTypeNosniff: true
        accessControlAllowMethods:
          - GET
          - POST
        accessControlMaxAge: 100
        addVaryheader: true
        referrerPolicy: origin-when-cross-origin

    # Allow compressed content
    compress-content:
      compress: {}
```

5.1.176 services\proxy_server\traefik\static_config\traefik.yml

```
log:
  level: "INFO"

entryPoints:
  web_http:
    address: ":80"
    http:
      redirections:
        entryPoint:
          to: web_secure_https
          scheme: https
          permanent: true
  web_secure_https:
    address: ":443"

api:
  dashboard: true

providers:
  docker:
    endpoint: "unix:///var/run/docker.sock"
    exposedByDefault: false
  file:
    directory: /configs

certificatesResolvers:
  myresolver:
    acme:
      email: IT-Services@geoville.com
      storage: /letsencrypt/acme.json
      httpChallenge:
        entryPoint: web_http
```

5.1.177 services\rabbitmq\create_certificates.sh

```
#!/bin/bash

# Script variables
DIR=ca_certificates
CERT_DURATION=3650

# Creates the directory if not exists already
if [[ ! -e $DIR ]]; then
    mkdir $DIR
fi

# Generates a private key without passphrase
openssl genrsa -out ./ca_certificates/server.key 2048

# Generates the server certificate
openssl req -new -key ./ca_certificates/server.key \
        -days $CERT_DURATION \
        -out ./ca_certificates/server.crt \
        -x509 \
        -subj '/C=AT/ST=Tyrol/L=Innsbruck/O=GeoVille Information Systems and Data
Processing GmbH/CN=api.clcplusbackbone.geoville.com/emailAddress=IT-Services@geoville.com'

# Generates the server certificate
cp ./ca_certificates/server.crt ./ca_certificates/root.crt
```

5.1.178 services\rabbitmq\Dockerfile

```
FROM rabbitmq
```

```
RUN mkdir ca_certificates

COPY /ca_certificates/server.crt /ca_certificates
COPY /ca_certificates/server.key /ca_certificates
COPY /ca_certificates/root.crt /ca_certificates

RUN rabbitmq-plugins enable --offline rabbitmq_management
```

5.1.179 services\rabbitmq\config_files\enabled_plugins

```
[rabbitmq_management].
```

5.1.180 services\rabbitmq\config_files\rabbitmq.conf

```
#rabbitmq configuration file

default_user = geoville
default_pass = nEy0W5gFikZ6gslE

listeners.ssl.default = 5671
management.tcp.port = 15672

ssl_options.cacertfile = /ca_certificates/root.crt
ssl_options.certfile   = /ca_certificates/server.crt
ssl_options.keyfile    = /ca_certificates/server.crt
ssl_options.verify     = verify_peer
ssl_options.fail_if_no_peer_cert = true
```