Final Document
Technical Documentation
Editor

PG-SanDMAN

Alex Wiens Andreas Schultz
Christian Korfmacher Georg Stilow
Johannes Kampmeyer Jonas Manuel
Linghui Luo Manoj Keshavaprakash
Surendra Kulkarni Vipin Ravindran Vijayalakshmi
Waleed Bin Arshad

March 31, 2017

Contents

Chapter 1

Introduction

This is the technical documentation for the Editor's front- and backend.

The Editor is a web-based application whose frontend is running in the developer's browser. It is compatible with the SONATA project and makes use of other tools from the SONATA SDK. Its main purpose is to assist in the creation and editing of SONATA descriptor files. This is done by simplifying repetitive and complicated tasks e.g. the descriptors of NSs are visualized as a graph of VNFs and NSs. In contrast to the textual representation of the graphs, users get a better understanding of the nodes and the connections between them. It allows users to take a quick look at the graph layout and helps users to analyze the descriptor well. The SDK component could either run on the developer's local computer and can be configured locally or it could be hosted by a central SDK server as a part of the SONATA SDK e.g. for a company setup. It also handles workspace and project creation and executes other CLI tools (e.g. son-package, son-push) to handle descriptors and packages.

At first, we will give a short overview about the components of the system and how they relate to each other. The subsequent sections of the document highlight different functionalities provided by the NS Editor and will give more detailed information about the structure and technologies used by the two components. This is followed by some guides to install the components and a step by step user guide that explains how to use the Editor. In the Appendix we have also attached a description of the most important Application Programming Interface (API) calls, as well as the complete Sphinx documentation of the backend server.

1.1 Overview

The front- and backend can be hosted mainly independent of each other. This modular structure allowed us the concurrent development of both components and might allow creating different frontend implementations in the future.

The complete communication between front- and backend is done through a RESTful interface. The frontend will load the User Interface (UI) elements and JavaScript files from the frontend server and supply it with information from the backend using AJAX calls.

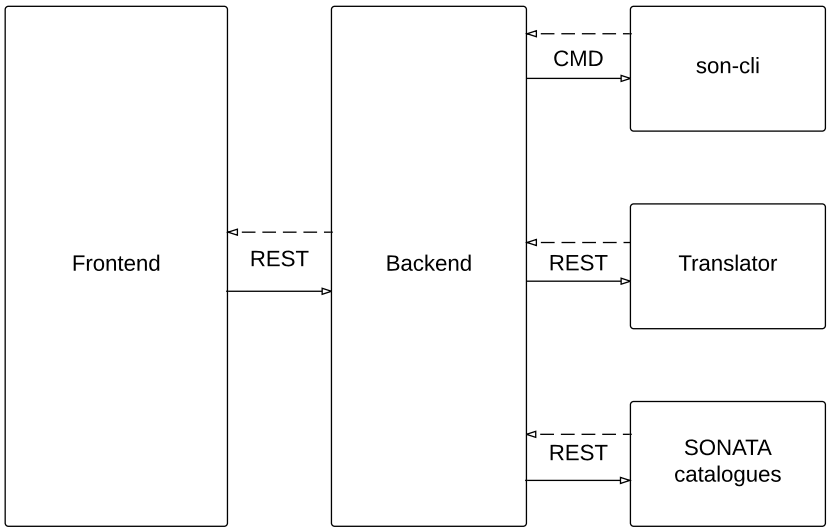


Figure 1.1: Editor front- and backend

Chapter 2

Frontend

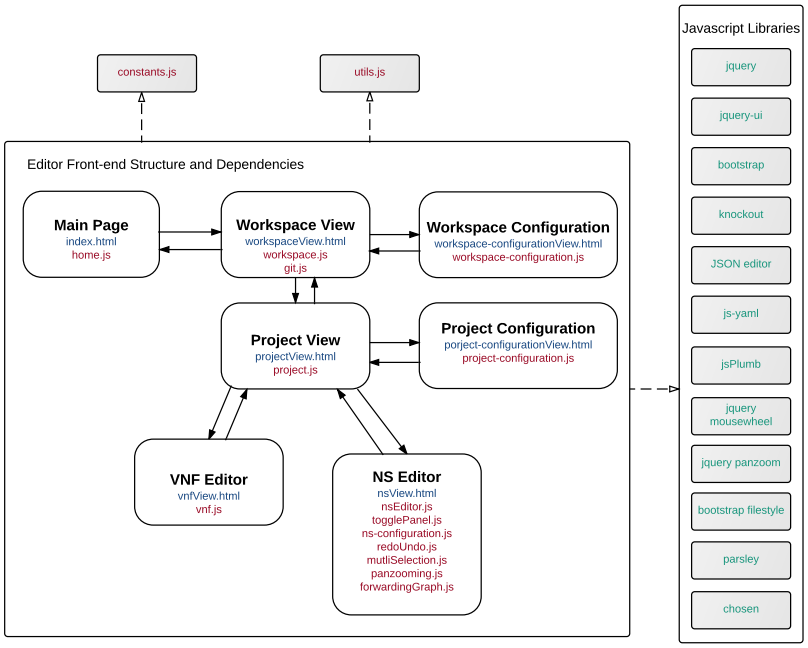


Figure 2.1: Editor frontend structure and dependencies

2.1 Core Functions

In this section we will highlight a few core functions that required the most effort to implement, to give an idea about the main problems we faced and our solutions to them.

2.1.1 NS Editor

Renaming and Deleting

Although seemingly simple on the surface, the feature to rename VNFs, NSs connection points and E-LANs took a lot of effort to get working flawlessly. The main reason for this are the many inner dependencies inside of the descriptor that need to be changed and checked for conflicts. When renaming the id of a VNF or NS several dependencies have to be observed:

- The id of a node must be unique across all nodes in the Editor.
- The id of a node is reflected in the naming of its connection points inside of the NS.
- The new connection point ids must be updated in every connection from or to these points.
- The id should be represented in the name of connections between nodes.
- References of the id in the service forwarding graph must be updated.

The other reason is the maintenance of connections between the visual model and the data it is representing.

- The change of a node's id must be announced to the jsPlumb instance.
- The node's endpoints have to be renamed, therefore they must be un- and reregistered with jsPlumb.
- Connections from and to the node's endpoints must be un- and reregistered with jsPlumb.

Most of the same dependencies and bindings have to be observed when deleting a node or a connection.

Computing Forwarding Graphs

Every time the connectivity of the network graph changes, our Editor executes an algorithm to compute a forwarding graph for every connected component of the NS. Each forwarding graph consists of all possible paths from an entry point to an exit point of the current NS. An entry point is characterized by only outgoing connections whereas an exit point has only incoming connections. The algorithm operates on the adjacency matrix of all connection points of the NS as well as all connection points of the constituent nodes. Because an inner node like a VNF can have multiple connection points, all of these connection points will be considered as connected. The algorithm then performs a depth first search from each entry point to compute all paths leading to endpoints.

Undo and Redo

The mechanism to undo and redo changes to the network descriptor works by pushing a copy of the latest descriptor to a stack on every update. In case an undo is requested, the mechanism pops the top element from the stack and pushes the current one into a redo stack. Then the state of the graphical Editor is reinitialized with the version from the stack. The reverse happens if a redo is triggered.

2.1.2 Form-based Editors

Initially we proposed to manually map all of the schema's properties to a form. In this approach we would have needed to generate editors for properties nested in arrays. Although a viable solution was found in using knockout (see 2.2.4), we finally decided against this approach for several reasons:

1. **Hard to read and maintain**
   Even though the needed code could be drastically reduced by creating a view model in knockout and binding it to the JavaScript Object Notation (JSON) structure of the descriptor, the remaining code still had the same complexity as the schema itself, which was not feasible in the long run. At several points during the development bugs were introduced by copying and pasting very similar code or not modifying it correctly.

2. **Frequent changes expected**
   Due to the unfinished nature of the SONATA project, it is under expectation that the schemas to be changed a few times in the upcoming future. This could have made the form based editors outdated quite frequently and resulted in wasted effort on behalf of maintainers.

For these reasons it was decided to change the approach and use an automatically generated editor to produce the form. By using the Jdorn JSON editor (see 2.2.5) changing the form is merely a matter of referencing the new schema in the backends configuration. By supplying the schema from the backend, it could also be ensured, that front- and backend always use the same schema to validate their descriptors against. Using the JSON editor we were able to reduce a several hundred lines of code down to only a few lines of configuration and data loading.

The only drawback we found using the JSON editor, is the sorting of properties inside the form. Because of the conversion from YAML Ain′t Markup Language (YAML) via a python dict to the JSON format, the original semantic ordering of the schemas properties are lost which results in an arbitrary sequence of form fields in the frontend. We deemed this drawback acceptable when compared to the huge maintenance costs of the manual approach.

2.1.3 Monitoring the current State of a Deployed NS

The Editor uses the HEAT API to get information of deployed stacks and the NOVA API to get the information of VNFs from the Emulator. After the Editor receives data from the Emulator, the JSON data is bound to the UI elements through 'ViewModels'. This data binding is handled by a library called 'Knockout.js'. In this case, every object element of the NOVA API response corresponds to a Node on UI. To collect the information of connections between source nodes and destination nodes, we use the 'Chaining APIs' and store each connection object in an array. After we have the information of nodes and connections, we feed this information to a library called as 'dagre.js' which calculates and renders a graph layout. The Editor polls the Emulator every 5 seconds and updates it's data in case of any change. For example, there can be changes in the nodes and connections in case of scale-out and scale-in. In this case, dagre updates the graph layout accordingly on the basis of the updated data.

2.2 Used Libraries

In this section we will give an overview of the used libraries in the frontend and their purpose for our Editor.

2.2.1 jQuery

jQuery is a JavaScript query library whose main purpose it is to select HTML-DOM elements through easy to read syntax and manipulate their attributes. It is a building block for many modern web applications and a dependency for a lot of other libraries, making it a de facto web standard. In our Editor it is used to select HTML elements, manipulate their css and other attributes as well as being a dependency for the other used libraries including jquery-ui, jquery mousewheel, jquery panzoom and jsPlumb.

2.2.2 jQuery-ui

jQuery-ui is an UI framework that offers UI widgets such as dialogs and auto-complete text fields and also has support for draggable elements. In the Editor it is used for the pop-up dialogs throughout the application, the search boxes in the workspace and project view and for dragging and dropping elements from the left hand toolbar in the NS Editor into the Editor's main area.

2.2.3 Bootstrap

Bootstrap is a styling and layout framework that offers modern looking style sheets and helps with making the web app more responsive for different browser sizes. In the Editor it is used for the overall structure of the web app's views as well as the styles for some of the buttons and input fields.

2.2.4 Knockout

Knockout is a data binding library that enables the developer to bind UI elements to a JavaScript object through so called view models. Instead of having to build HTML structures "by hand" in JavaScript, knockout handles changes to properties of the model by automatically manipulating the Document Object Model (DOM). It also makes it possible to bind events like clicks on the DOM elements to their respective data object which makes it easier to handle these events based on their attached data. In the Editor knockout is used in several places, for example to display lists of objects or to reflect changes to the name or id in the UI.

2.2.5 JSON Editor

The Jdorn JSON editor is a library that can create a form-based editor from a given JSON schema. If specified in the schema, all input fields can be validated and will be given a visible feedback if violated. In our Editor this library is used to generate HTML forms for the VNF Editor view and the NS configuration view of the NS Editor. Another use case is the display of the backend servers configuration in its respective view.

2.2.6 Js-yaml

Js-yaml is an utility that enables an easy conversion between the YAML format and JSON objects. In the Editor it is used to convert uploaded YAML files into JSON objects before populating the JSON editor and sending it to the backend server.

2.2.7 JsPlumb

JsPlumb is a library that allows the creation of interconnected components by dragging connectors from one node to another based on endpoints. It handles the drawing and routing of connections when nodes are dragged around and also offers support for dragging multiple nodes simultaneously by adding them into a selection. JsPlumb is the main component used in the graphical NS Editor.

2.2.8 jQuery Mousewheel

jQuery mousewheel is a jQuery plugin that simplifies registering functions to mousewheel events. It is used for the zooming functionality in the NS Editor.

2.2.9 jQuery Panzoom

jQuery panzoom is a jQuery plugin that helps to control the zooming and panning of an HTML element. It is used for the zooming and panning functionality in the NS Editor.

2.2.10 Bootstrap Filestyle

Bootstrap filestyle is a bootstrap plugin that extends the styles by offering a widget for file upload buttons. It is used for the upload buttons in the VNF and NS Editors.

2.2.11 Parsley

Parsley is a validation utility that enables validation based on regular expressions. It is used for a few input fields to ensure valid names and so on.

2.2.12 Chosen

Chosen is a jQuery plugin for displaying multiple selections. It is used in the project configuration view.

2.2.13 Dagre

Dagre is a JavaScript library that makes it easy to lay out directed graphs on the client-side. In the Editor, it is used to render the graph that visualizes the current state of the NS.

Chapter 3

Backend

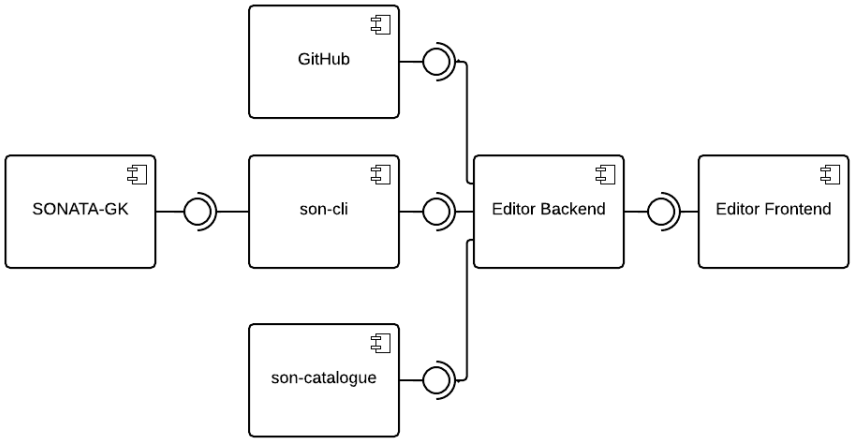


Figure 3.1: Global view on the backend environment

The backend provides a REST-ful interface for all functionalities that are required by an editor, for the creation, editing and organization of Virtual Network Functions and Network Services.

3.1 Project Structure

In this section we want to give an overview of the internal structure of the backend.

3.1.1 File Structure

The root folder contains set-up scripts and docker file configurations. Furthermore, the functionalities are split in different components which are reflected by the hierarchy of the project's file structure.

Overview

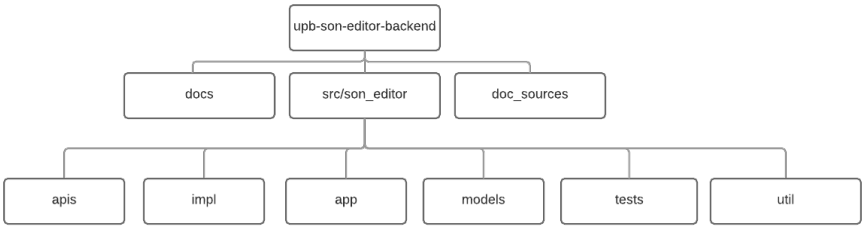


Figure 3.2: File structure of the source files

- *docs* contains documentation of functions, classes and methods.
- *doc_sources* contains scripts that generates the documentation.
- *src/son_editor* contains source files.

Source Structure

The source is split into the following sections:

- API Modules (apis)
  The API modules, implemented as Flask RESTPlus Resources that receive and interpret the requests from the frontend and relay those calls to the corresponding implementation modules. All API namespaces are registered on the main application from the module's initializer. A detailed documentation of backend's REST API can be seen in Appendix A.
- Implementation Modules (impl)
  The implementation modules handle the actual execution of the demands from the frontend and will Create, Read, Update and Delete (CRUD) the corresponding descriptors, workspaces and projects.
- App Module (app)
  The App module contains the entry point of our application that handles the initial start and setup of the server. It also defines the connection and the setup of the database, manages the access permissions of requests and handles any exceptions that may be thrown by the implementation modules.
- Database Models (models)
  The database models create a mapping between the structures saved to the file system and their corresponding entries in the database. They define the database schema depicted in Figure 3.3.

- Utility Functions (util)
  The Utility Functions are a collection of small tools that we use to load and write descriptors, communicate to the son-cli tools and formatting the responses that are returned to the frontend.

- Test Cases (tests)
  This module defines the unit tests we created to ensure stability during the development process. They are being executed on every build and determine if a new version is fit for release.

3.2 Data Management

The most important task of the backend server is storing and serving the VNF and NS descriptors on demand of the frontend. In this section we will explain how we use our database and the file system to manage and serve the descriptors.

3.2.1 Database

To make the descriptor accessible as fast as possible and to tie the data to a particular user, workspace and project we used a relational database represented by the Object Relational Mapper (ORM)-Models of SQLAlchemy (see 3.4.3). This way we could quickly find and serve the descriptors requested by the frontend.

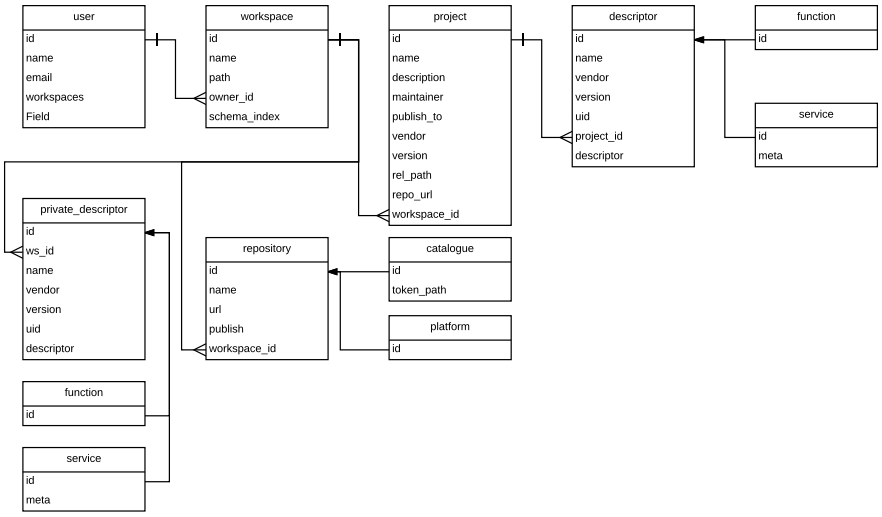


Figure 3.3: Backend entity relationships

3.2.2 Descriptor Files

All descriptors were not only written to the database but also to the file system. In this way the son-cli tools could be used easily by supplying the workspace path and letting the SDK tools discover the relevant descriptors that need to

be packaged. The other reason for not storing exclusively to the database was to have the ability to recreate the database from the file system.

This is currently used in two cases. The first one is migrating the database to a new schema, which was often necessary during the development. The second one is importing previously created workspaces in bulk without having to upload everything through the frontend Editor. By simply copying the workspace into the users folder and restarting the backend server it will automatically scan the unknown files and import them to the database.

Later on in the project implementation phase, it also had the positive effect such that importing a project from GitHub was just a matter of scanning the cloned project folder and thereby creating the necessary database entries.

3.3 GitHub Integration

The backend also handles the main functionality of the GitHub Integration features.

3.3.1 GitHub OAuth Login

To avoid creating separate accounts for developing in the SONATA SDK we opted to use the GitHub OAuth API for our login process. This not only reduces the need for a separate account as all of the SONATA Development so far is OpenSource and is hosted on GitHub, but it also reduces the burden to create a secure place to store passwords as we can offload this part of the security to a trusted service provider.

In the OAuth process the developer first needs to register his application with GitHub and provide a callback Uniform Resource Locator (URL). When the user wants to login using GitHub, they are redirected to the Authorization page to confirm giving the application access to the requested data. GitHub then calls the callback URL with an authorization code, that can in turn be used to retrieve the user information. To identify the developers application they must provide a client ID with the authorization call and the client secret with the second call.

3.3.2 GitHub Sharing

Another feature we use GitHub for is our project sharing mechanism. Our GitHub integration enables the user to clone remote projects, initialize and publish local projects and use the git pull, commit and push implementations to keep the projects in sync between GitHub and the Editor. These features are explained in more details in the Usage chapter 6.3.

3.4 Used Libraries

3.4.1 Flask

Flask is a python framework for building python web applications. It makes it easy to set up URL routes to link an URL to the python method that will handle the request and offers an abstraction layer which deals with the encoding

and decoding of Hyper Text Transfer Protocol (HTTP) requests and responses. Flask is used as the basic infrastructure that we built the backend upon.

3.4.2 Flask-RESTPlus (+ Swagger)

Flask-RESTPlus is an extension for RESTful servers which is built on top of Flask. It organizes groups of URLs into so called Resources and handles the GET, POST, PUT and DELETE HTTP methods by implementing a python function for each of these messages. Another important feature is the support for documenting the REST endpoints as an interactive API documentation powered by Swagger. This works by simply annotating the Resources with descriptions and information about expected inputs and results. By using flask restplus for implementing and documenting the API endpoints, we got an easy way to browse and look at documentation that is hosted alongside the application at the root URL of the backend server.

3.4.3 SQLAlchemy

For accessing our database we used the SLQ-Abstraction library sql-alchemy as our ORM. By using sql-alchemy in our implementation modules instead of plain SLQ we get SQLAlchemy also removes the dependency on a particular type of database as it is able to connect to a variety of different architectures like SQLite, Postgresql, MySQL and more. This makes it easy to switch to another database in the future as it would be as simple as changing the initialization code without touching any of the code actually accessing the database.

3.4.4 Pyaml

Pyaml is a python library for reading and writing YAML files and converting them into Python dictionaries. It is used to load and store the SONATA descriptor files.

3.4.5 Requests

The requests library allows to make HTTP requests to query and load data from external sources. It is mainly used for communication with the GitHub API and the son-catalogue representational state transfer (REST) API.

3.4.6 Son-cli

The son-cli tools are part of the SONATA SDK and gives us access to functions like creating workspaces and projects as well as communication to the SONATA service platform. It also offers tools for extended validation and monitoring that are currently not used by the Editor but might be added in the future.

Chapter 4

Deployment and Continuous Integration

In this chapter we will describe the mechanisms we used during the development to provide a running system which can be used for testing at all times.

4.1 Dockerizing

From very early on in the project implementation phase we decided to use Docker as a deployment mechanism for the Editor. This had the advantage that installing dependencies, configuring the file system and using ports could be encapsulated inside a docker container. This also reduced the cost of having to make sure to install all dependencies and executing install scripts correctly by hand.

By using the extension docker-compose we were able to bundle the commands for deploying or tearing down an instance into a script that can be executed with a single command.

4.2 Automatic Deployment and Testing

To ease the frequent release of new versions while making sure of only deploying versions that passed our current tests, we used a combination of the GitHub webhooks, Travis CI (backend only) and a custom python script to automatically redeploy front- and backend, once a new version was pushed and successfully tested.

4.2.1 GitHub Webhooks

GitHub offers a mechanism which is called webhooks to support developers in automating their development cycle. In our case, once a new version is pushed to the GitHub repository, GitHub would trigger Travis CI to run our tests. Once Travis successfully returns, GitHub would issue a deployment and notify our deployment hook of the new version.

4.2.2 Travis CI

Travis CI is a web-based continuous integration suite for testing public GitHub projects and it is free to use. Once a project is pushed it will install, build and test the project based on a special script file located in the project. After finishing the test it will report back to GitHub with the status of the build allowing the developer to make an informed decision whether or not to merge a new pull request.

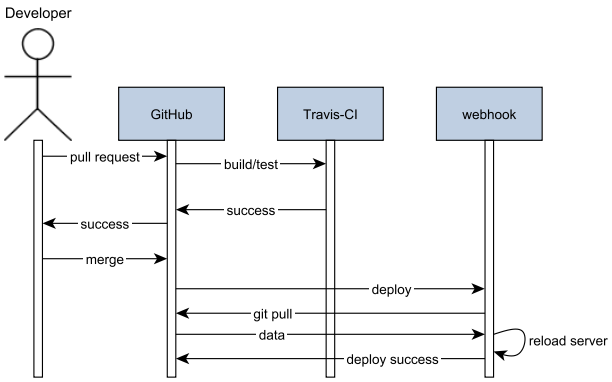


Figure 4.1: Successful CI process

4.2.3 Deployment Script

After the deployment hook is notified of a new version by GitHub, it will execute a deployment script, telling it to pull the latest changes from GitHub, rebuild the project (in case dependencies were changed) and then reload the code server to execute the new version.

4.2.4 Notifying the Developers

To notify the developers about the pull requests and the deployment status we also used the GitHub integration[1] for Slack[2].

[1] `https://github.com/integrations/slack`
[2] `https://slack.com/`

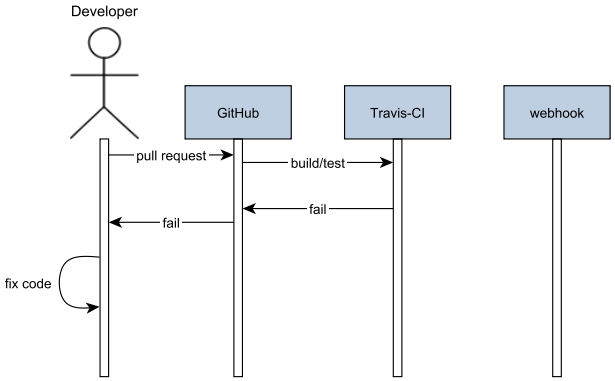


Figure 4.2: Failing CI process

Chapter 5

Installation

There are several methods to install the Editor on your server or development machine. In the following sections we will describe them from the easiest to the most configurable method. The most up-to-date instructions can also be found in the readme.md file in the GitHub repository[1]. However, to configure the installation, there are a few steps that need to be taken regardless of installation method.

5.1 Configuration

Firstly, the Editor backend must be configured with GitHub OAuth credentials. These are used for the OAuth based login and for the GitHub Integration features.

5.1.1 GitHub

First you need to register a new OAuth Application on GitHub:

1. Go to `https://github.com/settings/developers` and click on *Register a new application*
2. Fill in Application name, eg. SONATA Editor
3. Fill in Homepage URL, set to url where the Editor will be hosted
4. Fill in Authorization callback URL with *yourServerHost*/login e.g.: *https://fg-cn-sandman1.cs.upb.de:5000*/login
5. Click on *Register Application*
6. Note down the *Client ID* and the *Client Secret* that GitHub has created for you

To configure the Editor set the properties *ClientID* and *ClientSecret* in your

[1] `https://github.com/CN-UPB/upb-son-editor-backend`

config.yaml to the values from the previous step. The location of this file depends on the method of installation, in the first method however it should be located in the root of the project.

5.1.2 Sessionkey (Optional)

To secure the users session information, the application encrypts the session with a random session key that is generated when the application is started. To enable persisting the sessions across restarts of the server, you can set the session key under session¿secretKey to a random string. This key will override the randomly generated one and make old sessions readable after a restart.

5.1.3 Web Configuration (Optional)

To enable changing the configuration later remotely, you can add (uncomment) the following lines in your config.yaml:
config:
  pwd: passwordHere
  user: userNameHere

This will enable you to configure the server remotely through a few methods:

- Calling *yourFrontendURL*/serverConfigView.html and using the json-editor to change the configuration
- Using the backends API documentation from *yourBackendURL* and using the 'config' GET and POST UI
- Using a tool like CURL to GET and POST the config at *yourBackendURL*/config

The authentication method for these requests is a basic authentication header. Note: If the web configuration is deactivated all requests will fail with status 404.

5.2 Docker: One Container

The easiest way to install the Editor, is by using the repository found at `https://github.com/CN-UPB/upb-son-editor`. It will install the Editor front- and backend into a single container and requires only minimal configuration effort. To install, clone the repository to your server.

```
git clone https://github.com/CN-UPB/upb-son-editor
cd upb-son-editor
```

Edit the config.yaml file and add the configuration items described in 5.1.

5.2.1 Via Docker Compose

Docker Compose in version 1.8.0+ is required for installation in one step. Please refer to the installation guides and further instructions on the project's website[2].

[2] `https://docs.docker.com/compose/install/`

```
docker-compose up -d
```

If the Editor should run on another port from port 80, please change the line

```
- 80:80
to
- <yourPort>:80
```

in the docker-compose.yml.
If the Editor should not write any data to your systems hard drive (i.e. only save to the containers file system), please remove the following line

```
- ./workspaces:/root/son-editor/workspaces
```

from the docker-compose.yml file. If it should write to another folder than the *workspaces* folder in the project, please set another path in the docker-compose configuration.

5.2.2 Build the Container

Instead of using Docker Compose you can also build and run the container by calling

```
docker build -t son-editor .
docker run --name son-editor-instance -d \
-p 80:80 \
-v /path/to/upb-son-editor/workspaces:/root/son-editor/workspaces son-editor
```

5.3 Docker: Separate Containers

If you want to run the front- and backend as separate instances you can use the *upb-son-editor-fontend* and *upb-son-editor-backend* repositories.

5.3.1 Frontend

Clone the following repository:

```
git clone https://github.com/CN-UPB/upb-son-editor-frontend
cd upb-son-editor
```

5.3.2 Backend

Clone the following repository:

```
git clone https://github.com/CN-UPB/upb-son-editor-backend
cd upb-son-editor-backend
```

Edit the config.yaml located at *src/son_editor/config.yaml* according to the instructions from 5.1.
Start via Docker Compose:

```
docker-compose up -d
```

Build directly:

```
docker build -t son-editor-backend .
docker run --name son-editor-backend-instance -d \
-p 5000:5000 \
-v /path/to/upb-son-editor/workspaces:/root/son-editor/workspaces \
son-editor-backend
```

5.4 Build and Install manually

If Docker is not an option or you want to install the Editor in a development environment you can also build and run the Editor manually.

5.4.1 Backend

The following guide will use the venv[3] package of python which can be used to create a virtual environment. At first, create the virtual environment by using the following commands:

```
mkdir son-editor
python3 -m venv ~/son-editor/venv
```

Switch into the virtual environment.

```
source ~/son-editor/venv/bin/activate
```

Clone the backend of the Editor and install the required packages.

```
cd ~/son-editor
git clone https://github.com/CN-UPB/upb-son-editor-backend
cd upb-son-editor-backend
python3 setup.py install
```

Clone and install son-cli.

```
cd ~/son-editor
pip install git+https://github.com/sonata-nfv/son-cli@v2.0
```

Make sure to configure the backend as in 5.1.

```
vim ~/son-editor/upb-son-editor-backend/src/son_editor/config.yaml
```

Start the backend.

```
cd ~/son-editor/upb-son-editor-backend
python3 src/son_editor/app/__main__.py
```

You should now be able to access the swagger documentation on `http://localhost:5000`.

[3] `https://docs.python.org/3/library/venv.html`

5.4.2 Frontend

Clone the frontend and adjust the backend URL.

```
cd ~/son-editor
git clone https://github.com/CN-UPB/upb-son-editor-frontend
vim ~/son-editor/upb-son-editor-frontend/js/constants.js
```

Use a suitable HTTP server to host the Editor frontend with root directory

```
~/son-editor/upb-son-editor-frontend
```

5.5 Serving

While it is possible to run the Editor backend as a stand-alone application thanks to the integrated server of Flask (see 3.4.1), it is recommended to use a Web Server Gateway Interface (WSGI) server for hosting the application in production mode. In our docker based installations we use a combination of uwsgi[4] and nginx[5].

Since the frontend is written as a plain web app using HTML and JavaScript it can be served by many standard web servers. For our development phase we used a docker image[6] of the httpd[7] server.

5.5.1 Example Production Scenarios

In the following there are some production examples. The figures 5.1 and 5.2 show typical production scenarios, because the server allows the usage by multiple clients. Figure 5.3 shows a typical backend and frontend developer set-up

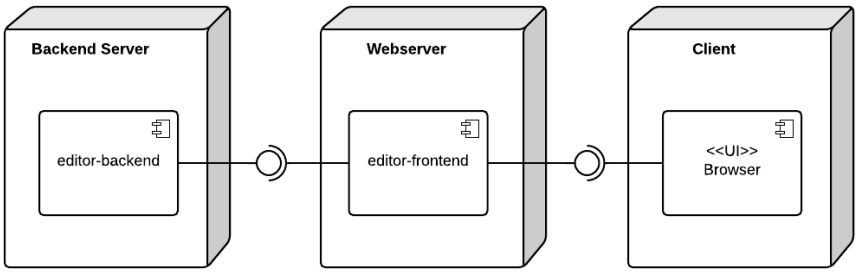


Figure 5.1: Each component is running on one host.

[4] https://uwsgi-docs.readthedocs.io/en/latest/
[5] http://nginx.org/
[6] https://hub.docker.com/_/httpd/
[7] https://httpd.apache.org/docs/current/programs/httpd.html

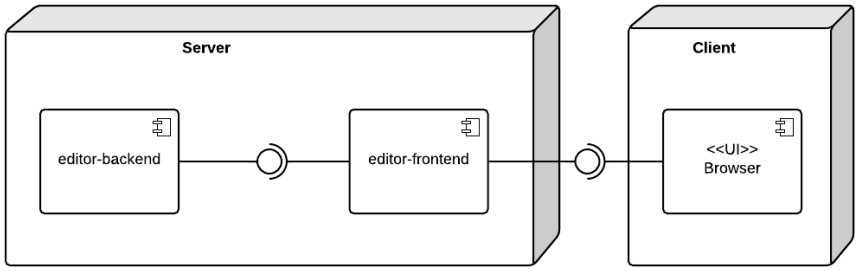


Figure 5.2: Backend and frontend is hosted by one server.

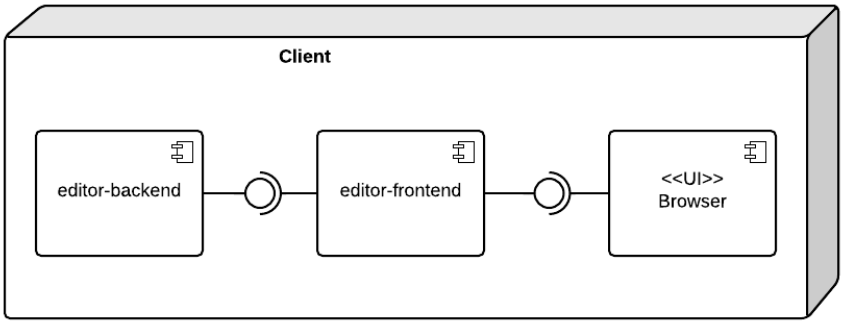


Figure 5.3: Backend, frontend and browser is hosted on a single machine. It is the most useful scenario to develop on the Editor backend and frontend.

Chapter 6

Usage

The Editor is a user-friendly tool to create virtual network functions and services. It is compatible with SONATA projects and makes use of other tools in the SONATA SDK. This chapter describes how to use the Editors features.

6.1 General Features

This section describes how to perform the basic tasks in the Editor.

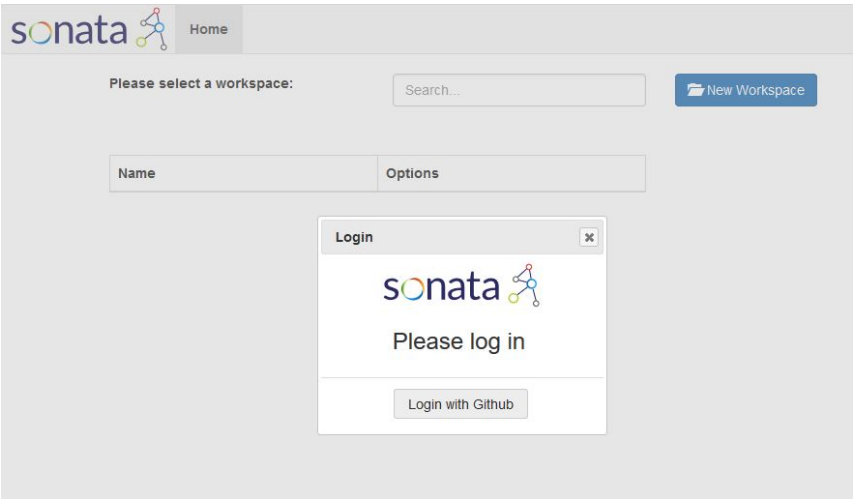
6.1.1 Access the Editor

You can access the Editor web interface from a web browser such as Google Chrome or Mozilla Firefox.

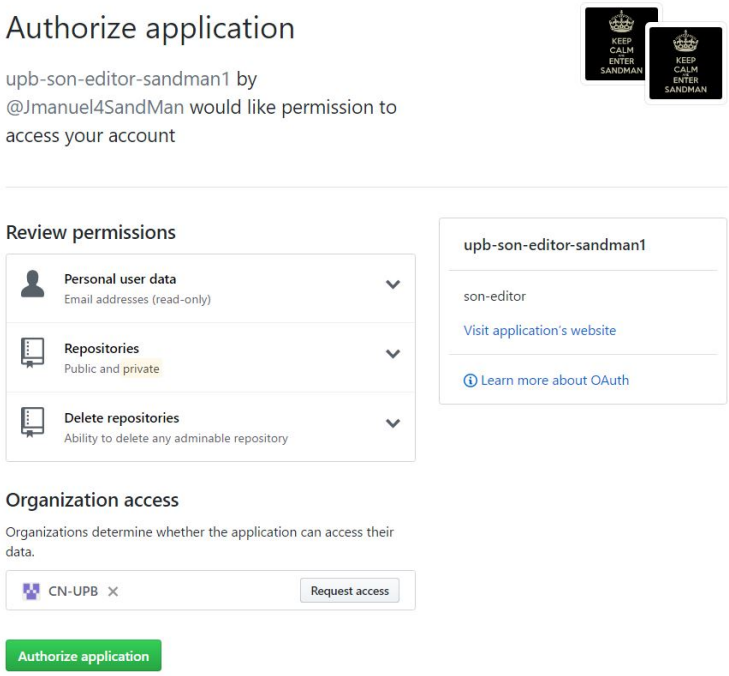
6.1.2 Log In

The Editor only supports to log in with a GitHub account. Therefore, the user must at first have a GitHub account in order to use the Editor.

1. Click on 'Login with GitHub' button in the login dialog box.



2. The browser will jump to the GitHub authorization page, click on the 'Authorize Application' button on GitHub.



3. Confirm GitHub user name and password to continue.

6.1.3 Log Out

Click on the 'Log Out' link on the top right corner of the Editor page to log out.

6.1.4 User Information

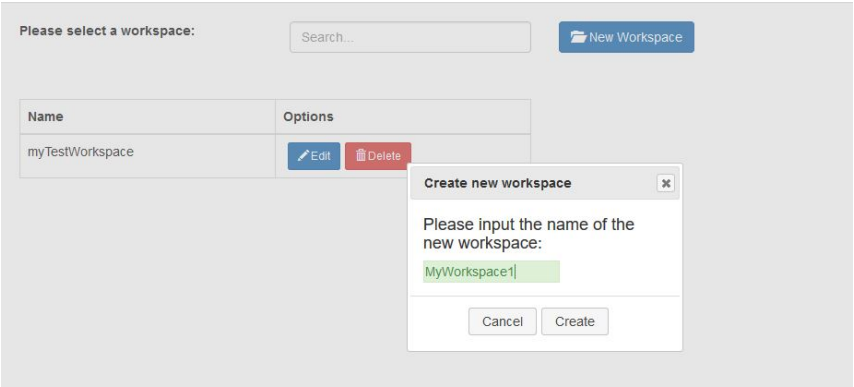
After Logging in with a GitHub account, the user name will be shown on the top right corner of the Editor page.

6.1.5 Home Page

The home page displays a list of available workspaces created by the user if exist. The user can create a new workspace from the home page.

6.1.6 Create New Workspace

1. Click on the 'New Workspace' button on the home page.



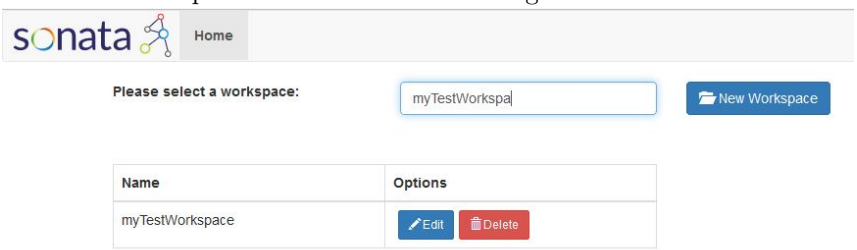
2. Input the name of the workspace and click on the 'Create' button.

3. Once a new workspace is created on the server, it will be displayed on the home page.

6.1.7 Search a Workspace

1. Start typing the workspace name in the search box provided on the home page.

2. Select the workspace name from the matching list.



3. Click on the selected workspace name to open the workspace.

6.1.8 Edit a Workspace

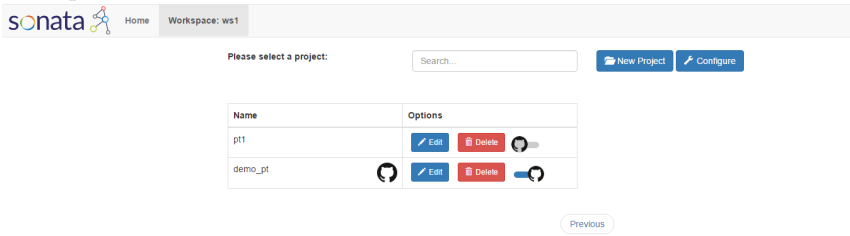
Click on the 'Edit' button for a respective workspace to view the workspace details. The browser will jump to workspace view which lists the projects belonging to that workspace.

6.1.9 Delete a Workspace

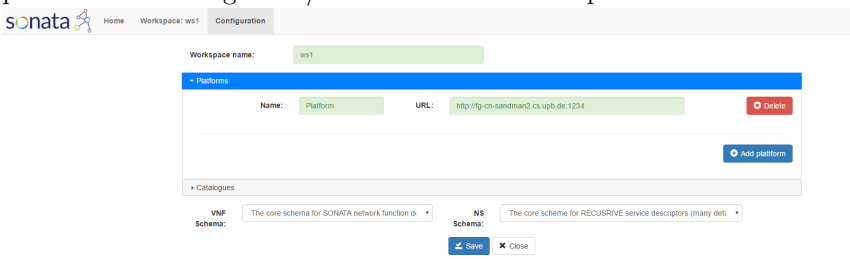
Click on the 'Delete' button for a respective workspace to delete it.

6.1.10 Configure a Workspace

1. Click on the 'Configure' button in the workspace view for a respective workspace.



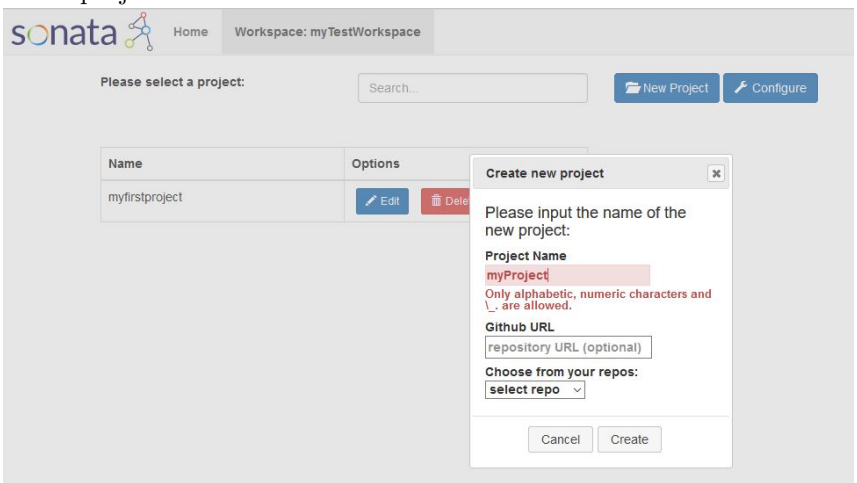
2. The browser will jump to the configuration view of the workspace. The user can change the workspace name, VNF schema, NS schema, add/delete platforms or catalogues to/from the current workspace.



6.1.11 Create New Project

1. Click on the 'New Project' button on workspace view page.

2. Input the name of the project and click on the 'Create' button to create a new project.[1]

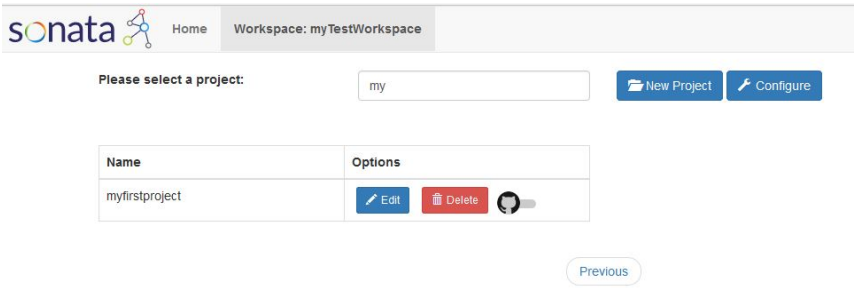


[1] for creating a project from GitHub please refer to 6.3

6.1.12 Search a Project

1. Start typing the project name in the search box provided on the workspace view page.

2. Select the searching project name from the matching list and click on the selected project name to edit the project.



6.1.13 Delete a Project

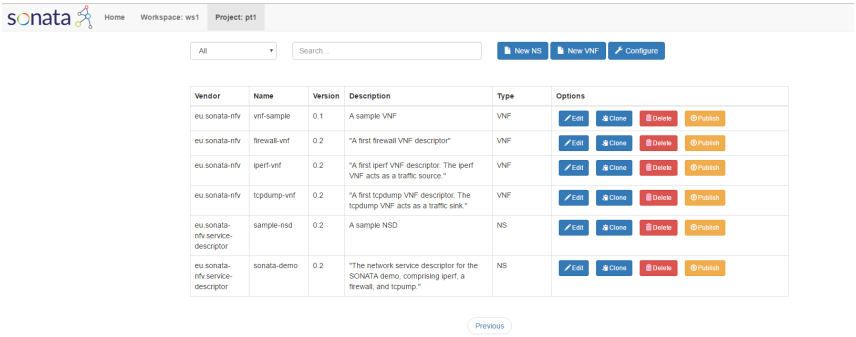
Click on the 'Delete' button for a respective project to delete it.

6.1.14 Edit a Project

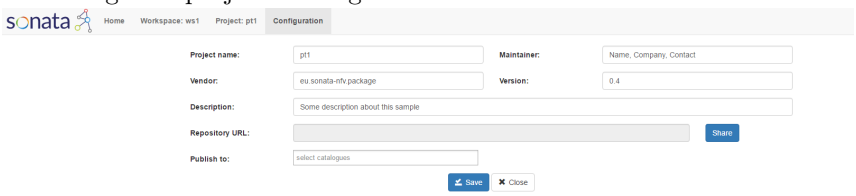
Click on the 'Edit' button for a respective project to view the project details. The browser will jump to the project view which lists the VNFs and NSs belonging to that project.

6.1.15 Configure a Project

1. Click on the 'Configure' button in the project view.



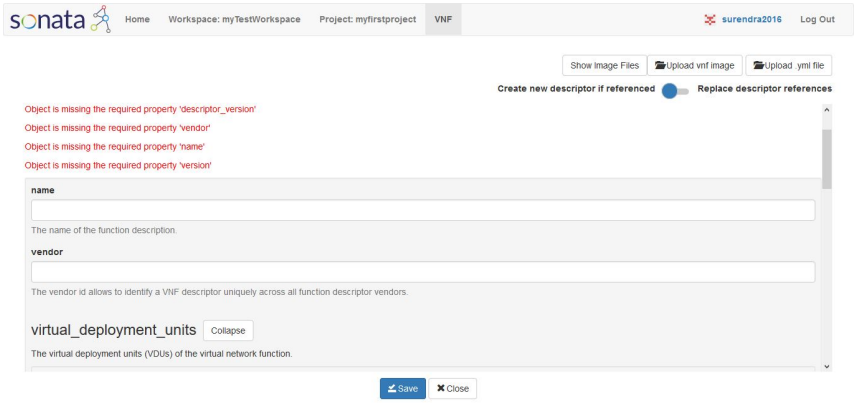
2. The browser will jump to the configuration view of the project. The user can change the project's configuration on this view.



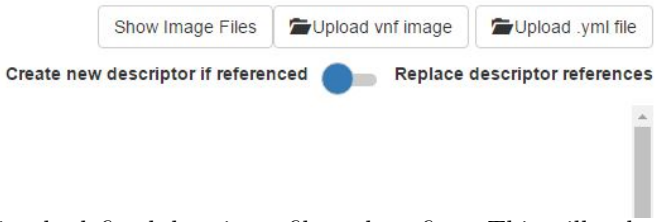
6.1.16 Create New VNF

There are two ways to create a new VNF in our Editor.

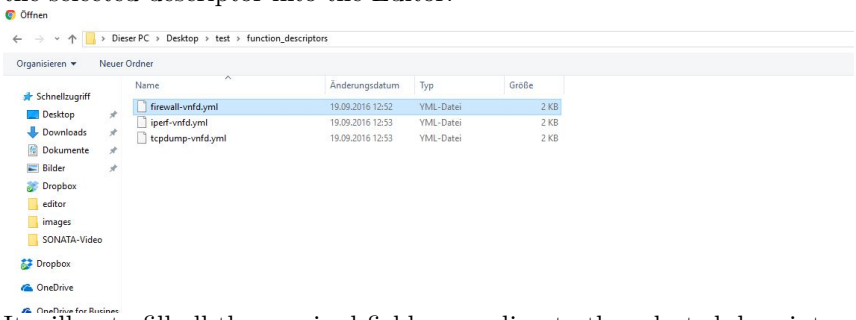
1. The first way is to create new VNF by filling all the required fields manually. To do this, click on the 'New VNF' button on project view page. Fill in the VNF details like name, vendor, version, Descriptor version, id etc. Click on the 'Save' button to save the VNF.



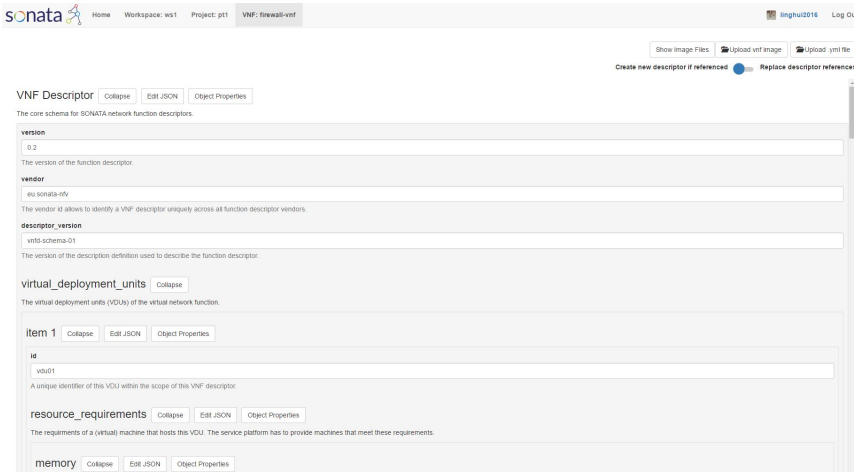
2. Another way to create new VNF is by loading a previously defined descriptor file which should be a .yml file. To do this, click on the 'New VNF' button on project view page. Click on the 'Upload .yml file' button.



Select the previously defined descriptor file and confirm. This will upload the selected descriptor into the Editor.



It will auto fill all the required fields according to the selected descriptor. Click on the 'Save' button to save the VNF.



6.1.17 Edit a VNF

Edit the VNF details by clicking on the 'Edit' button for that VNF.

6.1.18 Clone a VNF

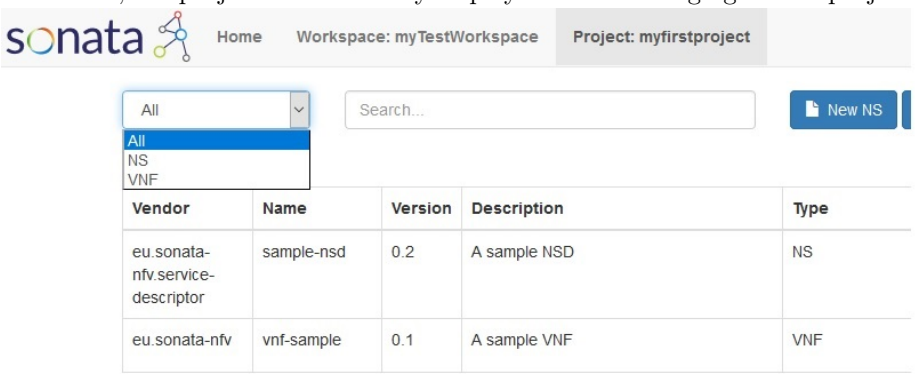
Create a copy of a respective VNF by clicking on the 'Clone' button for that VNF. The user must give a different name or version for the new VNF.

6.1.19 Delete a VNF

Delete the VNF by clicking on the 'Delete' button for that VNF.

6.1.20 Filter the VNFs and NSs

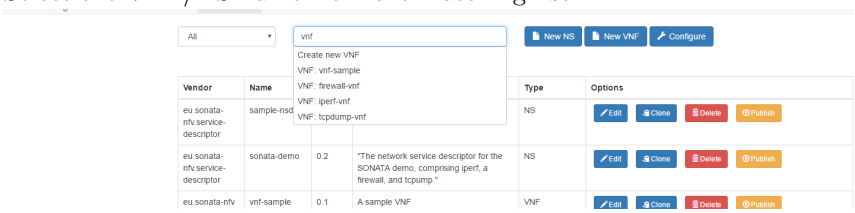
Select one of the menu options 'VNF', 'NS' or 'All' to see the filtered descriptors on project view. By default, 'All' is selected. When 'VNF' is selected, the project view will only display the VNFs belonging to this project. When 'NS' is selected, the project view will only display the NSs belonging to this project.



6.1.21 Search a VNF or NS

1. Start typing the VNF/NS name in the search box provided on the project view page.

2. Select the VNF/NS name from the matching list.



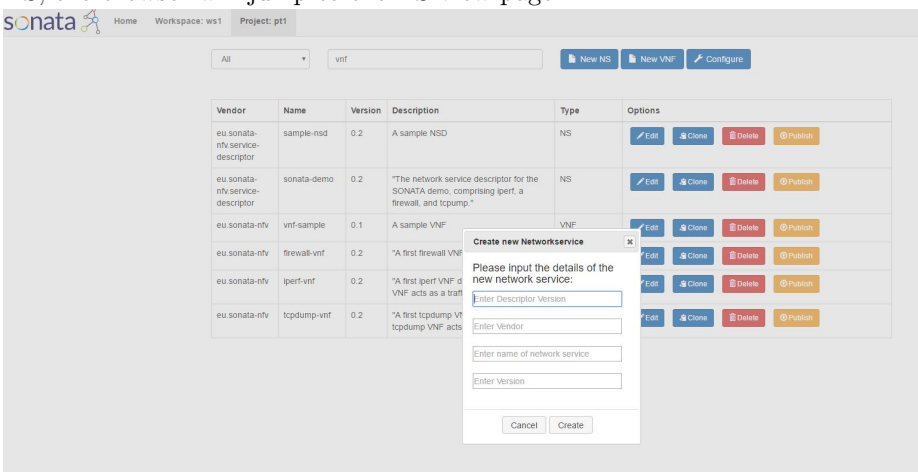
3. Click on the selected VNF/NS name to edit the VNF/NS.

6.2 NS Editor

This section describes the features provided by the NS Editor and how to use them.

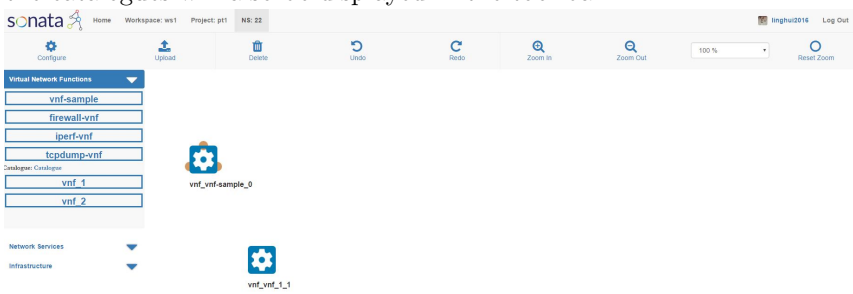
6.2.1 Create new NS

Click on the 'New NS' button on the project view page. Fill in the details like name, vendor, version and so on. Click on the 'Create' button. On creating the NS, the browser will jump to the NS view page.



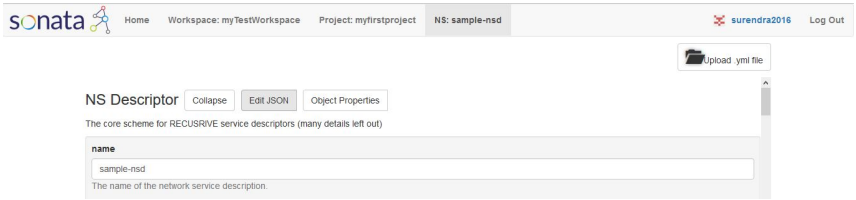
There are two ways to edit this new created NS in the NS Editor.

1. The user can add VNFs, NSs, connection points and E-LAN to the new NS manually by dragging and dropping the elements from the left-side tool bar. In the tool bar all VNFs and NSs belonging to the current project are displayed. In addition, if the user has added catalogues to the workspace in the workspace configuration view, all VNFs and NSs from the catalogues will also be displayed in the tool bar.

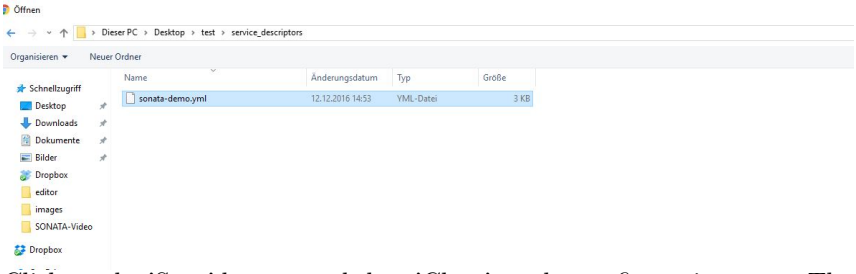


2. Another way to create a new NS is by uploading a previously defined service descriptor which should be a .yml file. To do this, click on the 'Configure' button on the tool bar of the NS view page.

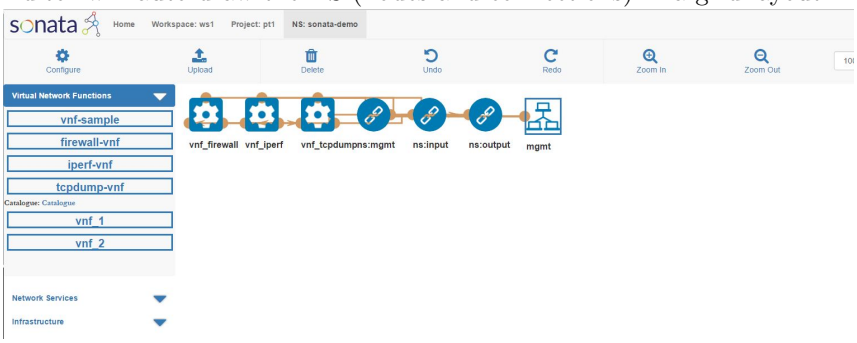
Click on the 'Upload .yml file' button



Select the previously defined service descriptor and confirm. This will upload the descriptor into the Editor.



Click on the 'Save' button and then 'Close' on the configuration page. The Editor will auto draw the NS (nodes and connections) in a grid layout.



6.2.2 Features of the NS Editor

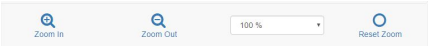
The NS Editor supports the following features.

1. Drag and drop the components: From the left-side navigation bar, user can drag VNFs, NSs and infrastructure components such as connection points (CP) or E-LAN to the editing area.

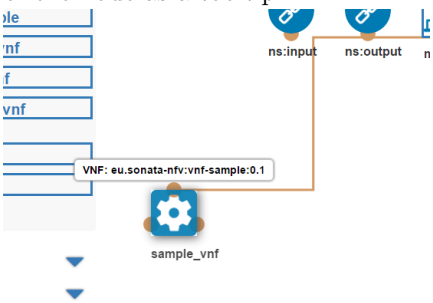
2. Rename a dropped node by clicking on the name displayed below that node.

3. Zoom and Pan: The user can perform zooming in and zooming out on the editing area using mouse scroll as well as 'Zoom-in', 'Zoom-out' buttons provided in the top menu. User can also set the zoom value using the values in drop down list. 'Pan' action is enabled by right pressing the mouse. Clicking on the 'Reset Zoom' button resets the zoom value to

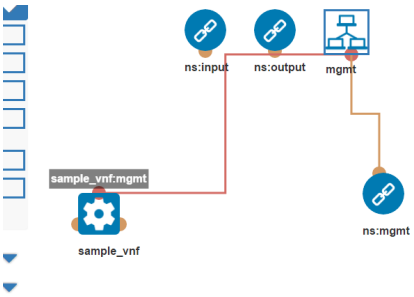
default.



4. Display the information of the dropped nodes:
   Hover the mouse over the node to display the 'type: name:vendor:version' of the node as a tooltip.

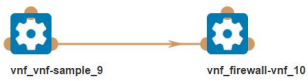


   Hover the mouse over the connection point of the dropped node to display the name of the connection point in form 'node name: connection point name'.

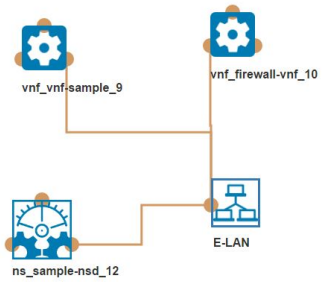


5. Create connections between dropped nodes:

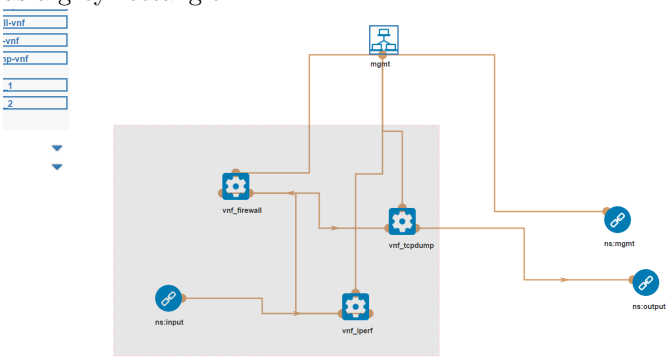
   (a) The user can connect the nodes by drawing a directed connection. A connection is drawn by dragging the mouse left pressed from a connection point (in orange) of one node to a connection point of another node. Such a connection is an E-LINE connection.



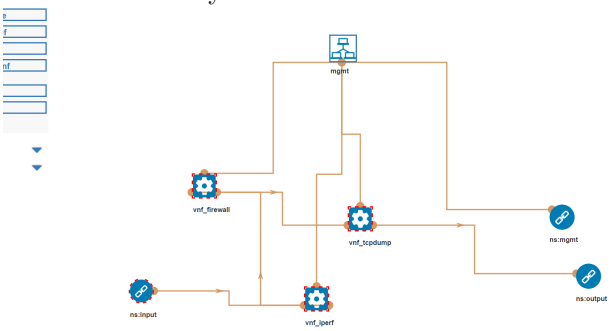
   (b) If the user needs to create an E-LAN connection, he first needs to drag and drop an E-LAN node from the left-side navigation bar to the editing area. Then draw connections from all ones in this E-LAN to the E-LAN node.



6. Multiselect nodes: The user can select multiple nodes in the editing area by left pressing and dragging the mouse. The selected area will be displayed as a grey rectangle.



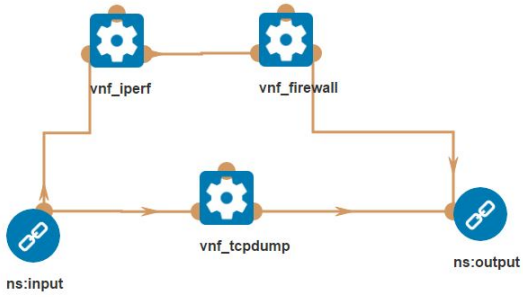
Selected nodes will be displayed with red dashed box around it and by dragging one of the selected nodes, all selected nodes can be dragged and deleted collectively.



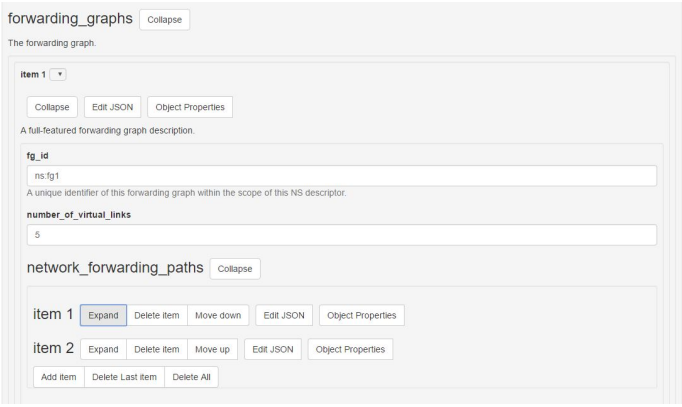
7. Delete the selected nodes: the user can delete multiple nodes by selecting them at once and clicking on the 'Delete' button on the tool bar or use the "delete" key on the keyboard.

8. Undo and Redo: On clicking on the 'Undo' and 'Redo' buttons, user can undo the last 10 actions or redo the earlier action.

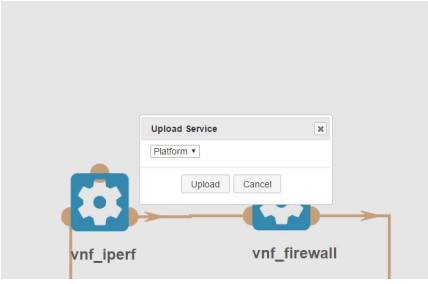
9. Compute the forwarding graph automatically: the following NS has a forwarding graph containing two forwarding paths: (ns:input → vnf_iperf → vnf_firewall → ns:ouput) and (ns:input → vnf_tcpdump → ns:ouput).



The Editor will calculate the forwarding graph automatically and writes it into the descriptor file and the user can check the forwarding graph in the configuration view under the section forwarding_graphs. The forwarding path contains all connection points of each node on the path.



10. Upload a NS to a plattform: click on the 'Upload' button and select a respective platform in the upload dialog box. Confirm 'Upload' to upload the service to the selected platform.



11. Save the positions of the dropped nodes: the positions of the nodes in the editing area will be automatically saved to the server and nodes will be painted on in the same positions once reloaded.

12. VNF Editor shortcut: If a single VNF from the project is selected, click

on the "Edit" button to edit the respective VNF descriptor.

6.2.3 Clone a NS

The user can create a copy of NS by clicking on the 'Clone' button for that NS in the project view. The user must give a different name or version for the new NS than the copied one.

6.2.4 Delete a NS

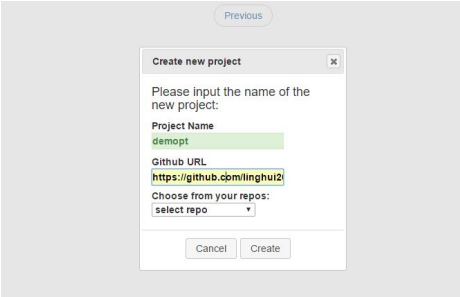
The user can delete a NS by clicking on the 'Delete' button for that NS in the project view.

6.3 Integration with GitHub

This section describes how the Editor is integrated with GitHub.

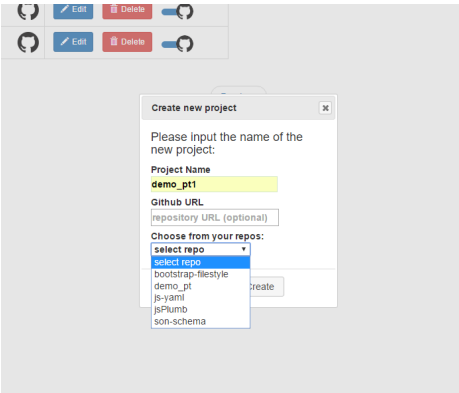
6.3.1 Create new Project: Clone from any GitHub repository

The user can create a project by cloning it from a Git repository. In the 'Create new project' dialog box the user can input any existing Git repository. Click on the 'Create' button. Then the project is copied from the Git repository.



6.3.2 Create new Project: Clone from own repository

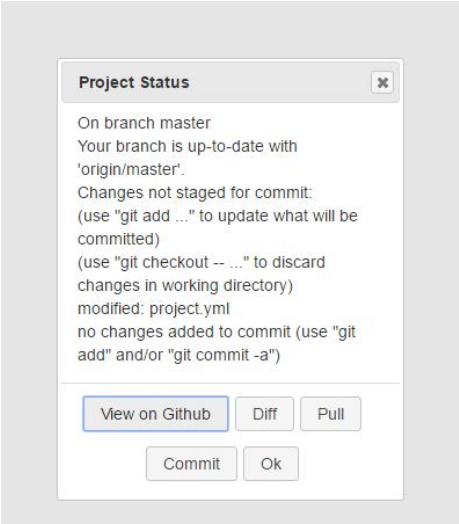
The user can create a project by cloning it from his own Git repository. In the 'Create new project' dialog box, the user can choose his repository from the drop down list. Click on the 'Create' button. Then the project is copied from the Git repository.



6.3.3 Check the Project Repository Status

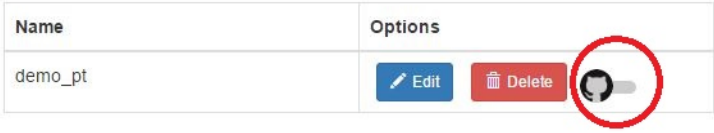
After creating a project, the user can check the git repository status of the project by clicking on the GitHub icon.

The user can take further action on the repository status, e.g. the user can commit the changes to GitHub, view differences and pull the changes from GitHub.

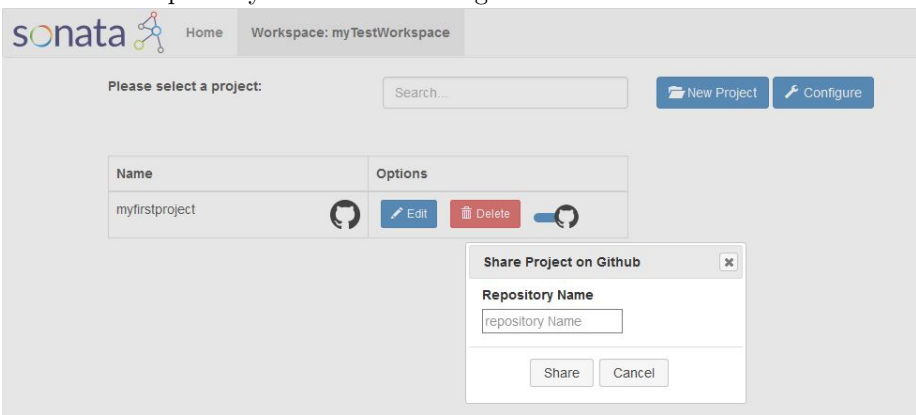


6.3.4 Share the Project on GitHub

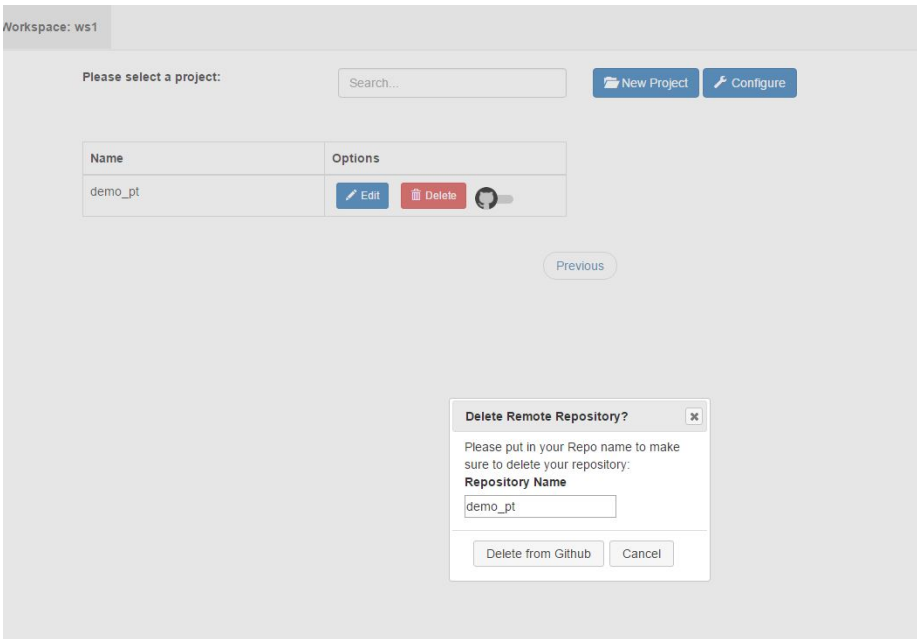
User can share the project created in the Editor on GitHub. To do so, click on the 'GitHub' slider button for that project in the 'Options' section and slide it to the right.



Provide the repository name in the dialog box and click on the 'Share' button.



If the project is shared on GitHub, the user can delete it from GitHub by sliding the 'GitHub' slider button to the left. The 'Delete Remote Repository' dialog box will be shown. Provide the repository name and click on the 'Delete from GitHub' button.



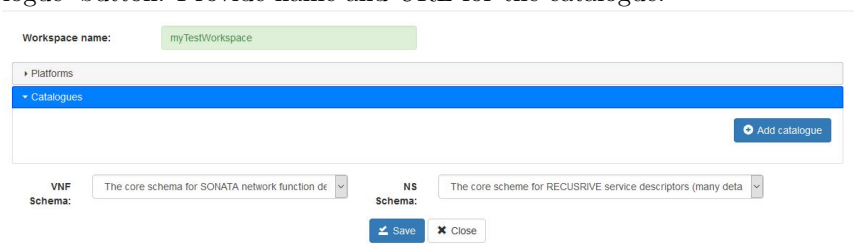
6.4 Platform and Catalogue Integration

6.4.1 Workspace Configuration: Configure platform and catalogue URLs

The user can provide platform and catalogue details in the workspace configuration. To add or view the workspace configuration details, click on the 'Configuration' button on the home page, it opens the workspace configuration page.

1. Add new platform: Click on the 'Add Platform' button. Provide name and URL for the platform.

2. Add new catalogue: Select 'Catalogues' tab and Click on the 'Add Catalogue' button. Provide name and URL for the catalogue.



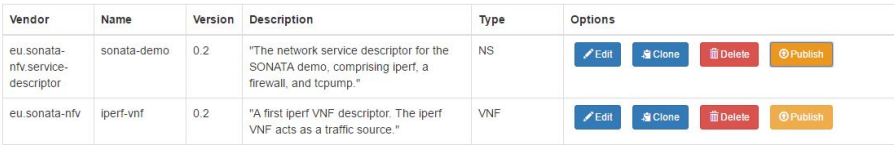
3. Click on the 'Save' button to save the configuration.

6.4.2 Use Descriptors from Catalogues in NS Editor

After the workspace configuration is done and platforms, catalogues are added, the user can use the descriptors from catalogues in the NS Editor to create NSs. To use it, navigate to the NS Editor. Observe the left-side navigation bar showing available VNFs and NSs. Along with the created VNFs and NS, it now shows descriptor definitions from catalogues.

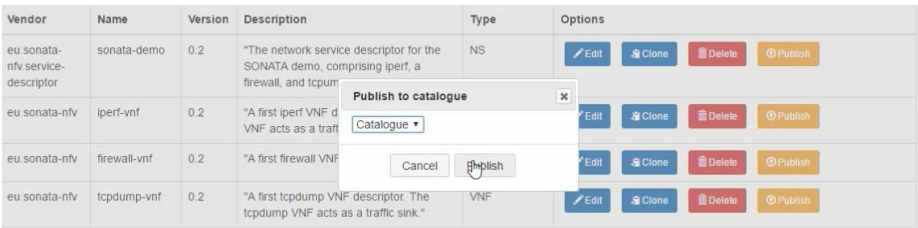
6.4.3 Publish a VNF/NS to a Catalogue

The user can publish VNFs and NSs to a catalogue. To do so, navigate to the project view page. On the project view page, a list of VNFs and NSs is shown. Click on the 'Publish' button for the VNF/NS you want to publish.



Vendor	Name	Version	Description	Type	Options
eu.sonata-nfv.service-descriptor	sonata-demo	0.2	"The network service descriptor for the SONATA demo, comprising iperf, a firewall, and tcpump."	NS	Edit Clone Delete Publish
eu.sonata-nfv	iperf-vnf	0.2	"A first iperf VNF descriptor. The iperf VNF acts as a traffic source."	VNF	Edit Clone Delete Publish

In the dialog box, select the catalogue name from the drop down list and Click on the 'Publish' button.



6.4.4 Upload a NS to a Platform

Once a NS is well-defined, the user can upload the NS to a platform. To do so, click on the 'Upload' button on NS view page. It opens a dialog box asking for the platform details. Select the platform name from the drop down list and click on the 'Upload' button.

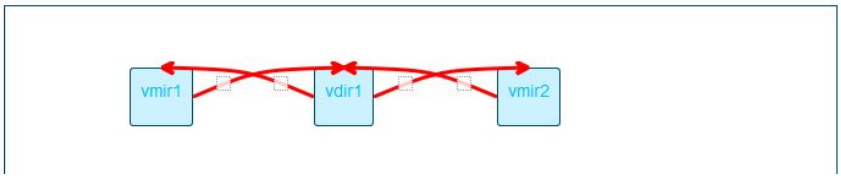
6.5 Monitor the current State of a Deployed NS

After a NS is created from the Editor and is deployed by the service platform (translator), the user can monitor the current state of that service in the form of a Service Graph by connecting to the OpenStack server (emulator). To see the Service Graph for currently deployed services, navigate to the project view page in the Editor. Click on the 'Monitor NS' button. The resulting page consists of two parts. The left part of the page shows the list of deployed NSs and on the right side of the page, the service graph for the NS is shown.

1. List of currently deployed NSs.

Name of The Server	Status	Options
4527724-vmb	UPDATE_COMPLETE	Show Graph Clear

2. Clicking on the 'Show Graph' button shows current state of the NSe. It is a graph that represents the servers and their connections from the deployed NS. The graph refreshes after every five seconds and displays the updates in graph in real time.



Chapter 7

Abbreviations

API Application Programming Interface

CI Continous Integration

CLI Command Line Interface

CRUD Create, Read, Update and Delete

DOM Document Object Model

JSON JavaScript Object Notation

NS Network Service

ORM Object Relational Mapper

HTTP Hyper Text Transfer Protocol

HTML Hyper Text Markup Language

REST representational state transfer

SDK Software Development Kit

UI User Interface

URL Uniform Resource Locator

VNF Virtual Network Function

WSGI Web Server Gateway Interface

YAML YAML Ain′t Markup Language

Appendix A

Backend Rest API

As mentioned before, the frontend communicates with the backend via its REST API. The format for any requests and answers is JSON. This chapter gives an excerpt of the most important API functions of the backend server. We will give some examples for interacting with the descriptors and the workspace structure.

A.1 General URL Structure

The Uniform Resource Locator for accessing the contents of a workspace can be constructed by traversing the following hierarchy:

```
|/workspaces/
|---|<wsID>
    |---|/projects/
        |---|<pjID>
            |---|/functions/
            |   |---<vnfID>
            |---|/services/
                |---<servID>
```

For each category there is a main resource that can be queried with a GET request for a list of its direct children. To create a resource a POST request can be made to its parent-to-be and will receive a new ID as part of the result. For accessing a specific child the respective ID has to be added to the end of the URL-path. Then a GET request can be made to load the resource, PUT requests to change the resource and DELETE requests to remove it.

A.2 Workspaces

A.2.1 List

To list all workspaces for the currently logged in user make a GET request to

`http://localhost:5000/workspaces/`

Answer

```
STATUS-CODE: 200

[
  {
    "id": 1,
    "platforms": [],
    "catalogues": [],
    "path": "/root/son-editor/workspaces/UserName/workspace_name",
    "schema_index": 0,
    "name": "workspace_name"
  }
]
```

A.2.2 Create

To create a new workspace make a POST request to

`http://localhost:5000/workspaces/`

With data like

```
{
  "name": "example_name"
}
```

The answer would be

```
STATUS-CODE: 201

{
  "id": 2,
  "platforms": [],
  "catalogues": [],
  "path": "/root/son-editor/workspaces/UserName/example_name",
  "schema_index": 0,
  "name": "example_name"
}
```

A.2.3 Read

To read a workspace make a GET request to

`http://localhost:5000/workspaces/2`

to get

```
STATUS-CODE: 200

{
  "id": 2,
  "platforms": [],
  "catalogues": [],
```

```
  "path": "/root/son-editor/workspaces/UserName/example_name",
  "schema_index": 0,
  "name": "example_name"
}
```

A.2.4 Update

To update a workspace make a PUT request with data

```
{
  "id": 2,
  "platforms": [
    {
      "url": "http://platform1.example.com:1234",
      "name": "Platform1"
    }
  ],
  "catalogues": [],
  "path": "/root/son-editor/workspaces/UserName/example_name",
  "schema_index": 0,
  "name": "new_name"
}
```

to

```
http://localhost:5000/workspaces/2
```

to get

```
STATUS-CODE: 200

{
  "id": 2,
  "platforms": [
    {
      "url": "http://platform1.example.com:1234",
      "name": "Platform1"
    }
  ],
  "catalogues": [],
  "path": "/root/son-editor/workspaces/UserName/new_name",
  "schema_index": 0,
  "name": "new_name"
}
```

Note: Changes to properties ”id” and ”path” will be ignored

A.2.5 Delete

To delete a workspace, make a DELETE request to

```
http://localhost:5000/workspaces/2
```

to get the answer

```
STATUS-CODE: 200

{
  "id": 2,
  "platforms": [
    {
      "url": "http://platform1.example.com:1234",
      "name": "Platform1"
    }
  ],
  "catalogues": [],
  "path": "/root/son-editor/workspaces/UserName/new_name",
  "schema_index": 0,
  "name": "new_name"
}
```

A.3 Projects

A.3.1 List

To list all projects for the workspace with ID 2, make a GET request to

`http://localhost:5000/workspaces/2/projects/`

Answer

```
STATUS-CODE: 200

[
  {
    "workspace_id": 2
    "id": 1,
    "repo_url": "",
    "publish_to": [ "personal"],
    "vendor": "eu.sonata-nfv.package",
    "rel_path": "project_name",
    "description": "Some description about this sample",
    "name": "project_name",
    "version": "0.4",
    "maintainer": "Name, Company, Contact"
  }
]
```

A.3.2 Create

To create a new project make a POST request to

`http://localhost:5000/workspaces/2/projects/`

With data like

```
{
  "name": "example_project"
}
```

The answer would be

```
STATUS-CODE: 201

[
  {
    "workspace_id": 2
    "id": 2,
    "repo_url": "",
    "publish_to": [ "personal"],
    "vendor": "eu.sonata-nfv.package",
    "rel_path": "example_project",
    "description": "Some description about this sample",
    "name": "example_project",
    "version": "0.4",
    "maintainer": "Name, Company, Contact"
  }
]
```

A.3.3 Read

To read a workspace make a GET request to

```
http://localhost:5000/workspaces/2
```

to get

```
STATUS-CODE: 200

[
  {
    "workspace_id": 2
    "id": 2,
    "repo_url": "",
    "publish_to": [ "personal"],
    "vendor": "eu.sonata-nfv.package",
    "rel_path": "example_project",
    "description": "Some description about this sample",
    "name": "example_project",
    "version": "0.4",
    "maintainer": "Name, Company, Contact"
  }
]
```

A.3.4 Update

To update a project make a PUT request with data

```
{
  "workspace_id": 2
  "id": 2,
  "repo_url": "",
  "publish_to": [ "personal"],
  "vendor": "eu.sonata-nfv.package",
  "rel_path": "example_project",
  "description": "A better description",
  "name": "new_project",
  "version": "0.4",
  "maintainer": "Your Name, Your Company,Your Contact"
}
```

to

```
http://localhost:5000/workspaces/2/projects/2
```

to get

```
STATUS-CODE: 200

{
  "workspace_id": 2
  "id": 2,
  "repo_url": "",
  "publish_to": [ "personal"],
  "vendor": "eu.sonata-nfv.package",
  "rel_path": "new_project",
  "description": "A better description",
  "name": "new_project",
  "version": "0.4",
  "maintainer": "Your Name, Your Company,Your Contact"
}
```

Note: Changes to properties "id", "rel_path" and "version" will be ignored

A.3.5 Delete

To delete a workspace, make a DELETE request to

```
http://localhost:5000/workspaces/2/projects/2
```

to get the answer

```
STATUS-CODE: 200

{
  "workspace_id": 2
  "id": 2,
  "repo_url": "",
  "publish_to": [ "personal"],
  "vendor": "eu.sonata-nfv.package",
  "rel_path": "new_project",
```

```
  "description": "A better description",
  "name": "new_project",
  "version": "0.4",
  "maintainer": "Your Name, Your Company,Your Contact"
}
```


A.4 Functions and Services

A.4.1 List

To list all functions and services for the project with ID 2, make GET requests to

`http://localhost:5000/workspaces/2/projects/2/functions`

and

`http://localhost:5000/workspaces/2/projects/2/services`

Answers


```
STATUS-CODE: 200

[
  {
    "descriptor": {
      "virtual_deployment_units": [
        {
          "resource_requirements": {
            "memory": {
              "size": 1
            },
            "cpu": {
              "vcpus": 1
            }
          },
          "id": "vdu1"
        }
      ],
      "vendor": "sandman",
      "name": "vnf_name",
      "descriptor_version": "vnfd_schema_01",
      "version": "0.1"
    },
    "id": 1,
    "uid": "sandman:vnf_name:0.1"
  }
]
```


and

```
STATUS-CODE: 200

[
  {
    "descriptor": {
      "vendor": "sandman",
      "forwarding_graphs": [],
      "name": "service_name",
      "version": "0.1",
      "descriptor_version": "nsd_schema_01"
    },
    "id": 1,
    "meta": {
      "positions": {},
      "adjacency_matrix": {},
      "counter": 0
    },
    "uid": "sandman:service_name:0.1"
  }
]
```

A.4.2 Create

To create a new function or service make POST requests to

`http://localhost:5000/workspaces/2/projects/2/functions/`

and

`http://localhost:5000/workspaces/2/projects/2/services/`

respectively.
With function data like

```
{
  "descriptor": {
    "virtual_deployment_units": [
      {
        "resource_requirements": {
        "memory": {
          "size": 1
        },
        "cpu": {
            "vcpus": 1
          }
        },
        "id": "vdu1"
      }
    ],
    "vendor": "sandman",
    "name": "example_vnf",
    "descriptor_version": "vnfd_schema_01",
```

```
    "version": "0.1"
  }
}
```


The answer would be

```
STATUS-CODE: 201
```


```
{
  "descriptor": {
    "virtual_deployment_units": [
      {
        "resource_requirements": {
        "memory": {
          "size": 1
        },
        "cpu": {
            "vcpus": 1
          }
        },
        "id": "vdu1"
      }
    ],
    "vendor": "sandman",
    "name": "example_vnf",
    "descriptor_version": "vnfd_schema_01",
    "version": "0.1"
  }
  "id": 2,
  "uid": "sandman:example_vnf:0.1"
}
```


With service data like


```
{
  "descriptor": {
    "vendor": "sandman",
    "name": "example_service",
    "version": "0.1",
    "descriptor_version": "nsd_schema_01"
  }
}
```


The answer would be

```
STATUS-CODE: 201
```


```
{
  "descriptor": {
    "vendor": "sandman",
    "name": "example_service",
    "version": "0.1",
```

```
    "descriptor_version": "nsd_schema_01"
  },
  "id": 2,
  "meta": {
  },
  "uid": "sandman:example_service:0.1"
}
```


Note: The "meta" object can be arbitrary data that the Editor frontend needs to store about a service. It is NOT part of the descriptor written to disk and might therefore be discarded if the Database is deleted or a project is transferred via their descriptor files.

The "descriptor" objects should adhere to the configured schemas for VNFs and network services.

A.4.3 Read

To read a function or serve make a GET requests to

`http://localhost:5000/workspaces/2/projects/2/functions/2`

and

`http://localhost:5000/workspaces/2/projects/2/services/2`

to get


```
STATUS-CODE: 200
{
  "descriptor": {
    "virtual_deployment_units": [
      {
        "resource_requirements": {
        "memory": {
          "size": 1
        },
        "cpu": {
            "vcpus": 1
          }
        },
        "id": "vdu1"
      }
    ],
    "vendor": "sandman",
    "name": "example_vnf",
    "descriptor_version": "vnfd_schema_01",
    "version": "0.1"
  }
  "id": 2,
  "uid": "sandman:example_vnf:0.1"
}
```

and

```
STATUS-CODE: 200

{
  "descriptor": {
    "vendor": "sandman",
    "forwarding_graphs": [],
    "name": "example_service",
    "version": "0.1",
    "descriptor_version": "nsd_schema_01"
  },
  "id": 2,
  "meta": {
  },
  "uid": "sandman:example_service:0.1"
}
```

respectively.

A.4.4 Update

To update a function make a PUT request with data

```
{
  "descriptor": {
    "virtual_deployment_units": [
      {
        "resource_requirements": {
        "memory": {
          "size": 1
        },
        "cpu": {
            "vcpus": 1
          }
        },
        "id": "vdu1"
      }
    ],
    "vendor": "sandman",
    "name": "new_vnf",
    "descriptor_version": "vnfd_schema_01",
    "version": "0.1",
    "description": "A really good description looks different!"
  }
  "id": 2,
  "uid": "sandman:example_vnf:0.1"
}
```

to

```
http://localhost:5000/workspaces/2/projects/2/functions/2
```

to get

```
STATUS-CODE: 200
```


```
{
  "descriptor": {
    "virtual_deployment_units": [
      {
        "resource_requirements": {
        "memory": {
          "size": 1
        },
        "cpu": {
            "vcpus": 1
          }
        },
        "id": "vdu1"
      }
    ],
    "vendor": "sandman",
    "name": "new_vnf",
    "descriptor_version": "vnfd_schema_01",
    "version": "0.1",
    "description": "A really good description looks different!"
  }
  "id": 2,
  "uid": "sandman:new_vnf:0.1"
}
```


To update a service make a PUT request with data


```
{
  "descriptor": {
    "vendor": "sandman",
    "forwarding_graphs": [],
    "name": "new_service",
    "version": "0.1",
    "descriptor_version": "nsd_schema_01"
  },
  "id": 2,
  "meta": {
    "positions": [1337, 42],
    "custom_property": "Enter SanDMAN!"
  },
  "uid": "sandman:example_service:0.1"
}
```


to

```
http://localhost:5000/workspaces/2/projects/2/functions/2
```

to get

```
STATUS-CODE: 200

{
  "descriptor": {
    "vendor": "sandman",
    "forwarding_graphs": [],
    "name": "new_service",
    "version": "0.1",
    "descriptor_version": "nsd_schema_01"
  },
  "id": 2,
  "meta": {
    "positions": [1337, 42],
    "custom_property": "Enter SanDMAN!"
  },
  "uid": "sandman:new_service:0.1"
}
```

Note: Changes to properties "id" and "uid" will be ignored

A.4.5 Delete

To delete a workspace, make DELETE requests to

`http://localhost:5000/workspaces/2/projects/2/functions/2`

or

`http://localhost:5000/workspaces/2/projects/2/services/2`

to get the answers

```
STATUS-CODE: 200

{
  "descriptor": {
    "virtual_deployment_units": [
      {
        "resource_requirements": {
        "memory": {
          "size": 1
        },
        "cpu": {
            "vcpus": 1
          }
        },
        "id": "vdu1"
      }
    ],
    "vendor": "sandman",
    "name": "new_vnf",
    "descriptor_version": "vnfd_schema_01",
```

```
    "version": "0.1",
    "description": "A really good description looks different!"
  }
  "id": 2,
  "uid": "sandman:new_vnf:0.1"
}
```


and


```
STATUS-CODE: 200

{
  "descriptor": {
    "vendor": "sandman",
    "forwarding_graphs": [],
    "name": "new_service",
    "version": "0.1",
    "descriptor_version": "nsd_schema_01"
  },
  "id": 2,
  "meta": {
    "positions": [1337, 42],
    "custom_property": "Enter SanDMAN!"
  },
  "uid": "sandman:new_service:0.1"
}
```


A.5 Git Life-cycle

In this section we will give an example how to use the GitHub Integration of the backend.

A.5.1 List user repositories

To access a list of all user repositories, so the user can choose which to clone, you can call

```
http://localhost:5000/workspaces/2/git/list
```

as a GET request. The answer will be a list of all user repositories.


```
[
 {
    "description": "Student project group's network service editor backend.",
    "name": "upb-son-editor-backend",
    "html_url": "https://github.com/Jmanuel4SandMan/upb-son-editor-backend",
    [....]
  }
]
```

A.5.2 Cloning

There are 2 ways to clone a project from GitHub it can either be done by supplying a "repo_url" along with a newly created project or by using the git API of the backend directly.

In this example we will show using the git API.

To clone a project make a POST request to

`http://localhost:5000/workspaces/2/git/clone`

with the payload

```
{
  "url": https://github.com/pgsandmanbot/son-repo"
}
```

This will download the project into the workspace and return a JSON object containing the ID of the newly created project.

```
{
  "id": 3
}
```

This id can then be used as the project ID to query the project as usual.

A.5.3 Status and Diff

The status and diff calls are to determine the differences between the latest commit and the changes made (diff) as well as the differences between the remote repository and the local one (status). To call the methods make a POST request containing a project ID under source control to

`http://localhost:5000/workspaces/2/git/diff`

and

`http://localhost:5000/workspaces/2/git/status`

respectively.

```
{
  "project_id": 3
}
```

The response will be JSON object with a message that contains the command line output of the respective 'git' command.

```
{
  "message": "On branch master\nYour branch is up-to-date with 'origin/master'.
      Changes not staged for commit:
      (use \"git add <file>...\" to update what will be committed)
      (use \"git checkout -- <file>...\" to discard changes in working directory)

          modified:   project.yml

      no changes added to commit (use \"git add\" and/or \"git commit -a\")
      "
}
```

A.5.4 Init and Create

To upload a project to GitHub two methods need to be called consecutively. First the local repository will be initialized by a POST request to

```
http://localhost:5000/workspaces/2/git/init
```

with

```
{
  "project_id": 3
}
```

Wich will be answered with a response code of 200 and a message from the server.

```
{
  "message": "Initialized empty Git repository in
        /root/son-editor/workspaces/UserName/example_workspace/projects/example_proje
       "
}
```

After this step, the project can be pushed to GitHub by calling

```
http://localhost:5000/workspaces/2/git/create
```

with

```
{
  "project_id": 3,
  "repo_name": "example_project"
}
```

and will be answered with the output of the 'git' command line:

```
{
  "message": "[master (root-commit) 123456]  'Initial commit'
     4 files changed, 21 insertions(+)
     create mode 100644 project.yml
     create mode 100644 sources/pattern/sample/patterm.yml
     create mode 100644 sources/ssm/sample/ssm.yml
     create mode 100644 sources/vnf/vnf_name/sandman-vnf_name-0.1.yml
  "
}
```

A.5.5 Commit and Push

To upload changes to GitHub, the backend also offers to commit and push in one step. By calling

```
http://localhost:5000/workspaces/2/git/commit
```

with

```
{
  "project_id": 3,
  "commit_message": "The commit message"
}
```

All changes will be committed locally and directly pushed to the remote if successful.

```
{
  "message": "[master 54321]  'The commit message'
         1 file changed, 2 insertions(+), 1 deletion(-)
        "
}
```

A.5.6 pull

To pull any remote changes into the local project the pull command can be executed

`http://localhost:5000/workspaces/2/git/pull`

with

```
{
  "project_id": 3
}
```

The result will show the output of the 'git' command line:

```
{
  "message": "Updating 88d8c6a..74681e2
        Fast-forward
         project.yml | 3 ++-
         1 file changed, 2 insertions(+), 1 deletion(-)
        "
}
```

A.5.7 delete

Finally to delete the remote repository make a DELETE request to

`http://localhost:5000/workspaces/2/git/delete`

with data

```
{
  "project_id": 3,
  "repo_name": "example_project"
}
```

Where the "repo_name" serves as a confirmation from the user to delete the repository. It must be set to the remote repositories name.

Answer:

```
{
  "message": "Successfully deleted"
}
```

A.6 Service Platform

To package and upload a service to the SONATA Service Platform, you can send a POST request to

`http://localhost:5000/workspaces/2/platforms/1/services/`

with data

```
{
  "id": 2
}
```

This will create a network service package via son-package and then upload it to the service platform via son-access. Answer:

```
{
  "message": "Answer from Service Platform"
}
```

A.7 Full API

The full API documentation can be viewed by running the backend server and opening the root URL (e.g. `http://localhost:5000/`) with a browser. This will bring up the API documentation powered by Swagger and can be used to test interactions with the backend before implementing them from a frontend.

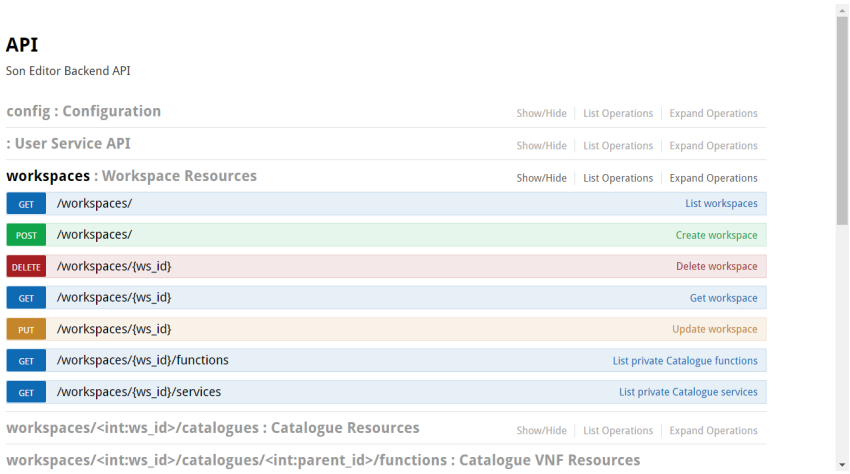


Figure A.1: An example of the backends interactive api documentation

Appendix B

Source Documentation

The backends source code was documented using python doc strings and compiled into this document using Sphinx. It can also be accessed by downloading the doc directory and opening the *index.html* file in a browser, or visiting the online version hosted through GitHub pages at `https://cn-upb.github.io/upb-son-editor-backend/`.

Sonata Editor Backend Documentation

Release 0.9

PG SanDMAN

Mar 28, 2017

CONTENTS:

CHAPTER

ONE

SRC

1.1 son_editor package

1.1.1 Subpackages

1.1.1.1 son_editor.apis package

1.1.1.1.1 son_editor.apis.catalogue_functionsapi module

Created on 22.07.2016

@author: Jonas

class `son_editor.apis.catalogue_functionsapi.`**`Function`**(*api=None*, **args*, ***kwargs*)

Bases: `flask_restplus.resource.Resource`

Resource methods for specific function descriptors

`delete`(*ws_id*, *catalogue_id*, *vnf_id*)

Deletes a function

Deletes a function in the project or catalogue by its id

`get`(*ws_id*, *catalogue_id*, *vnf_id*)

Get a specific function

Gets a specific function information by its id

`methods` **= ['DELETE', 'GET', 'PUT']**

`put`(*ws_id*, *catalogue_id*, *vnf_id*)

Updates a function

Updates a function in the project or catalogue by its id

class `son_editor.apis.catalogue_functionsapi.`**`Functions`**(*api=None*, **args*, ***kwargs*)

Bases: `flask_restplus.resource.Resource`

Resource methods for all function descriptors of this directory

`get`(*ws_id*, *catalogue_id*)

List all functions

Lists all available functions in the given project or catalogue.

`methods` **= ['GET', 'POST']**

post (*ws_id*, *catalogue_id*)
Creates a new function

Creates a new function in the project or catalogue

1.1.1.1.2 son_editor.apis.catalogue_servicesapi module

Created on 22.07.2016

@author: Jonas

class `son_editor.apis.catalogue_servicesapi.`**`Service`**(*api=None*, **args*, ***kwargs*)
Bases: `flask_restplus.resource.Resource`

delete (*ws_id*, *catalogue_id*, *service_id*)
Delete the Service

Deletes the service from the Project or Catalogue

get (*ws_id*, *catalogue_id*, *service_id*)
Return a specific Service

Returns the referenced service from the Project or catalogue

methods = ['DELETE', 'GET', 'PUT']

put (*ws_id*, *catalogue_id*, *service_id*)
Update the service

Updates the referenced service in the catalogue

class `son_editor.apis.catalogue_servicesapi.`**`Services`**(*api=None*, **args*, ***kwargs*)
Bases: `flask_restplus.resource.Resource`

Api Methods for all services in this resource

get (*ws_id*, *catalogue_id*)
Get a list of all Services Returns a list of all services available in this resource

methods = ['GET', 'POST']

post (*ws_id*, *catalogue_id*)
Create a new Service

Publishes a new Service in the catalogue

1.1.1.1.3 son_editor.apis.cataloguesapi module

Created on 18.07.2016

@author: Jonas

class `son_editor.apis.cataloguesapi.`**`Catalogue`**(*api=None*, **args*, ***kwargs*)
Bases: `flask_restplus.resource.Resource`

delete (*ws_id*, *catalogue_id*)
Deletes a specific catalogue

Deletes a catalogue by its id

get (*ws_id*, *catalogue_id*)
Gets a specific catalogue

Gets a specifc catalogue by its id

methods = ['DELETE', 'GET', 'PUT']

put (*ws_id*, *catalogue_id*)
Updates a specific catalogue

Updates a specific catalogue by its id

class `son_editor.apis.cataloguesapi.`**`Catalogues`**(*api=None*, **args*, ***kwargs*)
Bases: `flask_restplus.resource.Resource`

get (*ws_id*)
Lists catalogues

Lists catalogues in a specific workspace

methods = ['GET', 'POST']

post (*ws_id*)
Creates a new service catalogue

Creates a new service catalogue in the specific workspace

1.1.1.1.4 son_editor.apis.configapi module

class `son_editor.apis.configapi.`**`Configuration`**(*api=None*, **args*, ***kwargs*)
Bases: `flask_restplus.resource.Resource`

Web configuration

get (**args*, ***kwargs*)

methods = ['GET', 'POST']

post (**args*, ***kwargs*)

`son_editor.apis.configapi.`**`authenticate`**()
Sends a 401 response that enables basic auth

`son_editor.apis.configapi.`**`check_auth`**(*username*, *password*)
This function is called to check if a username / password combination is valid.

`son_editor.apis.configapi.`**`requires_auth`**(*f*)

1.1.1.1.5 son_editor.apis.gitapi module

class `son_editor.apis.gitapi.`**`GitClone`**(*api=None*, **args*, ***kwargs*)
Bases: `flask_restplus.resource.Resource`

methods = ['POST']

post (*ws_id*)
Clones projects into the workspace

class `son_editor.apis.gitapi.`**`GitCommit`**(*api=None*, **args*, ***kwargs*)
Bases: `flask_restplus.resource.Resource`

methods = ['POST']

post (*ws_id*)
Commits and pushes changes

class son_editor.apis.gitapi.**GitCreate** (*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

methods = ['POST']

post (*ws_id*)
Creates a remote repository and pushes a project for it

class son_editor.apis.gitapi.**GitDelete** (*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

delete (*ws_id*)
Deletes a remote repository

methods = ['DELETE']

class son_editor.apis.gitapi.**GitDiff** (*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

methods = ['POST']

post (*ws_id*)
Retrieves the current diff of the project directory

class son_editor.apis.gitapi.**GitInit** (*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

methods = ['POST']

post (*ws_id*)
Initializes a repository in the given project

class son_editor.apis.gitapi.**GitList** (*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

get (*ws_id*)
Lists remote repository information

methods = ['GET']

class son_editor.apis.gitapi.**GitPull** (*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

methods = ['POST']

post (*ws_id*)
Pulls updates from a project

class son_editor.apis.gitapi.**GitStatus** (*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

methods = ['POST']

post (*ws_id*)
Retrieves the current status of the project directory

1.1.1.1.6 son_editor.apis.misc module

class son_editor.apis.misc.**Log** (*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

get ()
Return the logfile as string

methods = ['GET']

1.1.1.1.7 son_editor.apis.nsfslookup module

Created on 22.07.2016

@author: Jonas

class son_editor.apis.nsfslookup.**FunctionLookup**(*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

get(*ws_id*, *project_id*, *vendor*, *name*, *version*)
Retrieves a virtual network function by vendor name version

Finds a specific virtual network with given vendor / name / version

methods = ['GET']

class son_editor.apis.nsfslookup.**ServiceLookup**(*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

get(*ws_id*, *project_id*, *vendor*, *name*, *version*)
Retrieves a network service by vendor name version

Finds a specific network service with given vendor / name / version

methods = ['GET']

1.1.1.1.8 son_editor.apis.platformsapi module

Created on 18.07.2016

@author: Jonas

class son_editor.apis.platformsapi.**Platform**(*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

delete(*ws_id*, *platform_id*)
Delete Platform

Deletes a service platform by its id

get(*ws_id*, *platform_id*)
Get service Platform

Retrieves a service platform by its id

methods = ['DELETE', 'GET', 'PUT']

put(*ws_id*, *platform_id*)
Update platform

Updates a service platform by its id

class son_editor.apis.platformsapi.**PlatformServices**(*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

methods = ['POST']

post (*ws_id*, *platform_id*)
Upload a service package

Packages and uploads a network service to the platform

class son_editor.apis.platformsapi.**Platforms** (*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

get (*ws_id*)
List platforms

Lists all service platforms in the given workspace

methods = ['GET', 'POST']

post (*ws_id*)
Create a new service platform

Creates a new service platform in the given workspace

1.1.1.1.9 son_editor.apis.project_functionsapi module

Created on 22.07.2016

@author: Jonas

class son_editor.apis.project_functionsapi.**Function** (*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

Resource methods for specific function descriptors

delete (*ws_id*, *project_id*, *vnf_id*)
Deletes a function

Deletes a function in the project or catalogue by its id

get (*ws_id*, *project_id*, *vnf_id*)
Get a specific function

Gets a specific function information by its id

methods = ['DELETE', 'GET', 'PUT']

put (*ws_id*, *project_id*, *vnf_id*)
Updates a function

Updates a function in the project or catalogue by its id

class son_editor.apis.project_functionsapi.**FunctionUpload** (*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

static delete (*ws_id*, *project_id*, *vnf_id*, *filename*)
Delete VNF Image

Delete the VNF Image by file name

Parameters

- **ws_id** – The workspace ID
- **project_id** – The project ID
- **vnf_id** – The VNF ID
- **filename** – The filename to delete

Returns

methods = ['DELETE']

class son_editor.apis.project_functionsapi.**Functions**(*api=None*, **args*, ***kwargs*)

Bases: flask_restplus.resource.Resource

Resource methods for all function descriptors of this directory

get(*ws_id*, *project_id*)

List all functions

Lists all available functions in the given project or catalogue.

methods = ['GET', 'POST']

post(*ws_id*, *project_id*)

Creates a new function

Creates a new function in the project or catalogue

class son_editor.apis.project_functionsapi.**PrivateFunction**(*api=None*, **args*, ***kwargs*)

Bases: flask_restplus.resource.Resource

get(*ws_id*, *project_id*, *vnf_id*)

Publish function to private

Publishes the function to the workspace wide catalogue

Parameters

- **ws_id** – The Workspace ID
- **project_id** – The Project ID
- **vnf_id** – The VNF ID

Returns dict with "message" property

methods = ['GET']

1.1.1.1.10 son_editor.apis.project_servicesapi module

Created on 22.07.2016

@author: Jonas

class son_editor.apis.project_servicesapi.**PrivateService**(*api=None*, **args*, ***kwargs*)

Bases: flask_restplus.resource.Resource

Private service publishing method

get(*ws_id*, *project_id*, *service_id*)

Publish service to private

Publishes the service to the workspace wide catalogue

Parameters

- **ws_id** – The Workspace ID
- **project_id** – The Project ID
- **service_id** – The Service ID

Returns A dict with a "message" property.

methods = ['GET']

class son_editor.apis.project_servicesapi.**Service**(*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

delete(*ws_id*, *project_id*, *service_id*)
Delete the Service

Deletes the service from the Project or Catalogue

get(*ws_id*, *project_id*, *service_id*)
Return a specific Service

Returns the referenced service from the Project or catalogue

methods = ['DELETE', 'GET', 'PUT']

put(*ws_id*, *project_id*, *service_id*)
Update the service

Updates the referenced service in the project or in the catalogue or platform

class son_editor.apis.project_servicesapi.**Services**(*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

Api Methods for all services in this resource

get(*ws_id*, *project_id*)
Get a list of all Services Returns a list of all services available in this resource

methods = ['GET', 'POST']

post(*ws_id*, *project_id*)
Create a new Service

Creates a new Service in this project or publishes it in the catalogue or platform

1.1.1.1.11 son_editor.apis.projectsapi module

Created on 18.07.2016

@author: Jonas

class son_editor.apis.projectsapi.**Project**(*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

delete(*ws_id*, *project_id*)
Delete project

Deletes the project by its id

get(*ws_id*, *project_id*)
Retrieves project

Gets information of a given project

methods = ['DELETE', 'GET', 'PUT']

put(*ws_id*, *project_id*)
Updates a project

Updates the project by its id

class son_editor.apis.projectsapi.**Projects**(*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

get(*ws_id*)
Lists projects

Lists projects in the given workspace

methods = ['GET', 'POST']

post(*ws_id*)
Creates a new project

Creates a new project in the given workspace

1.1.1.1.12 son_editor.apis.schemaapi module

class son_editor.apis.schemaapi.**Schema**(*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

Single schema retrieval

get(*ws_id*, *schema_id*)
Get schema

Returns the requested schema from the schema_master at schema_index from this workspace

Parameters

- **ws_id** – The workspace ID
- **schema_id** – Either "ns" or "vnf"

Returns The requested schema

methods = ['GET']

class son_editor.apis.schemaapi.**Schemas**(*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

Get all schemas for this server

get(*ws_id*)
List Schemas

Returns a list of all schemas configured for this server

Parameters **ws_id** – The workspace ID

Returns A list of schemas

methods = ['GET']

1.1.1.1.13 son_editor.apis.userserviceapi module

class son_editor.apis.userserviceapi.**Information**(*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

User information

static get ()
User Information

Returns github information about the current user

methods = ['GET']

class son_editor.apis.userserviceapi.**Login** (*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

static get ()
Login User

Login the User with a referral code from the github oauth process

methods = ['GET']

class son_editor.apis.userserviceapi.**Logout** (*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

Logs out the current user

static get ()
Logout

Logs out the current user

Returns "Logged out"

methods = ['GET']

1.1.1.1.14 son_editor.apis.workspacesapi module

Created on 18.07.2016

@author: Jonas

class son_editor.apis.workspacesapi.**PrivateFunctions** (*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

List functions of private Catalogue

get (*ws_id*)
List private Catalogue functions

Lists all functions in the Private workspace wide catalogue

methods = ['GET']

class son_editor.apis.workspacesapi.**PrivateServices** (*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

List Services of private Catalogue

get (*ws_id*)
List private Catalogue services

Lists all services in the Private workspace wide catalogue

methods = ['GET']

class son_editor.apis.workspacesapi.**Workspace** (*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

Methods for a single workspace resource

delete(*ws_id*)
Delete workspace

Deletes a specific workspace by its id

get(*ws_id*)
Get workspace

Gets information about a specific workspace

methods = ['DELETE', 'GET', 'PUT']

put(*ws_id*)
Update workspace

Updates a specific workspace by its id

class son_editor.apis.workspacesapi.**Workspaces**(*api=None*, **args*, ***kwargs*)
Bases: flask_restplus.resource.Resource

Methods for the workspace resource directory

get()
List workspaces

Gets all available workspaces

methods = ['GET', 'POST']

post()
Create workspace

Creates a new workspace

1.1.1.2 son_editor.app package

1.1.1.2.1 son_editor.app.database module

son_editor.app.database.**init_db**()
Import model modules

Import all modules here that might define models so that they will be registered properly on the metadata. Otherwise you will have to import them first before calling init_db()

son_editor.app.database.**reset_db**()
Resets the database

only used in tests

son_editor.app.database.**scan_project_dir**(*project_path*, *pj*)
Scan project dir

Scans the project dir for any new functions and services via _scan_for_functions and _scan_for_services

Parameters

- **project_path** – The path of the project to scan
- **pj** – The project from the database to attach the descriptors to

Returns

son_editor.app.database.**scan_workspaces_dir**()
Scan workpaces directory

Scans the workpaces directory for any users. Will visit every one to add new workspaces etc via _scan_user_dir

1.1.1.2.2 son_editor.app.exceptions module

exception son_editor.app.exceptions.**ExtNotReachable**(*msg*)
Bases: Exception

Thrown whenever an external host cannot be contacted, be it because of a wrong url or network problems

exception son_editor.app.exceptions.**InvalidArgument**(*msg: str*)
Bases: Exception

Thrown whenever an argument is missing data or is supplying the wrong data

exception son_editor.app.exceptions.**NameConflict**(*msg: str*)
Bases: Exception

Thrown whenever a name conflict arises for names that must be unique

exception son_editor.app.exceptions.**NotFound**(*msg: str*)
Bases: Exception

Thrown whenever a resource cannot be located, or the user has no access to it

exception son_editor.app.exceptions.**PackException**(*msg: str*)
Bases: Exception

Thrown whenever a service cannot be packaged by son-package, wraps the cli tools error message

exception son_editor.app.exceptions.**StillReferenced**(*msg: str*)
Bases: Exception

Thrown whenever an action is taken towards an element that is still referenced by another that would compromise the relationship like deleting the referenced object

exception son_editor.app.exceptions.**UnauthorizedException**(*msg: str*)
Bases: Exception

Thrown whenever an unauthorized access is detected, e.g. because the user has not yet logged in

1.1.1.2.3 son_editor.app.securityservice module

son_editor.app.securityservice.**check_access**(*request*)
checks if the current user is allowed to access a given resource. Session will be invalidated if the login information cannot be found

Parameters **request** – The http request made to the server

Returns nothing if access granted

Raises *UnauthorizedException* – if user not logged in

1.1.1.3 son_editor.impl package

1.1.1.3.1 son_editor.impl.catalogue_servicesimpl module

son_editor.impl.catalogue_servicesimpl.**build_URL**(*is_vnf*, *name*, *vendor*, *version*)
builds the url from the given name vendor and version

Parameters

- **is_vnf** –
- **name** –
- **vendor** –
- **version** –

Returns

son_editor.impl.catalogue_servicesimpl.**create_id**(*e: dict*)
Expects a dict type with name, vendor, version :param e: dict with name, vendor, version :return: the id of the descriptor

son_editor.impl.catalogue_servicesimpl.**create_in_catalogue**(*catalogue_id*, *function_id*, *is_vnf*)
Creates a function on the catalogue

Parameters

- **catalogue_id** –
- **function_id** –
- **is_vnf** –

Returns

son_editor.impl.catalogue_servicesimpl.**decode_id**(*id*) → tuple
Returns the parts of a given id :param id: :return: tuple of vendor, name, version

son_editor.impl.catalogue_servicesimpl.**delete_service_catalogue**(*ws_id*, *catalogue_id*, *descriptor_uid*, *is_vnf*)
Delete the descriptor in the catalogue

Parameters

- **ws_id** – The workspace ID
- **catalogue_id** – The catalogue ID
- **descriptor_uid** – The descriptors uid (vendor:name:version)
- **is_vnf** – If the descriptor is a VNF or NS

Returns

son_editor.impl.catalogue_servicesimpl.**getType**(*is_vnf: bool*) → str
Returns the vnf / ns prefix

Parameters **is_vnf** –

Returns the prefix for the type

son_editor.impl.catalogue_servicesimpl.**get_all_in_catalogue**(*ws_id*, *catalogue_id*, *is_vnf*)

Retrieves a list of catalogue functions

Parameters

- **ws_id** –
- **catalogue_id** –
- **is_vnf** –

Returns

son_editor.impl.catalogue_servicesimpl.**get_catalogue**(*catalogue_id*)

Retrieves a catalogue by given id

Parameters **catalogue_id** – int

Returns

son_editor.impl.catalogue_servicesimpl.**get_function**(*function_id*)

Returns a function with the given function id

son_editor.impl.catalogue_servicesimpl.**get_in_catalogue**(*ws_id*, *catalogue_id*, *function_id*, *is_vnf*)

Gets a specific function

Parameters

- **ws_id** –
- **catalogue_id** –
- **function_id** –
- **is_vnf** –

Returns

son_editor.impl.catalogue_servicesimpl.**get_service**(*service_id*)

Returns a service with the given service id

son_editor.impl.catalogue_servicesimpl.**update_service_catalogue**(*ws_id*, *catalogue_id*, *descriptor_uid*, *descriptor_data*, *is_vnf*)

Update the descriptor in the catalogue

Parameters

- **ws_id** – The workspace ID
- **catalogue_id** – The catalogue id
- **descriptor_uid** – the descriptors uid (vendor:name:version)
- **descriptor_data** – The descriptor data as a dict containing a "descriptor" property
- **is_vnf** – if the descriptor is a VNF or NS

1.1.1.3.2 son_editor.impl.cataloguesimpl module

son_editor.impl.cataloguesimpl.**create_catalogue**(*workspace_id: int*, *catalogue_data*)

Creates a catalgoue in the given workspace. A catalogue is defined by its name and url. These are given as json data

Parameters **workspace_id** – Workspace ID of the target workspace, where the catalogue should get created.

Returns Catalogue descriptor

son_editor.impl.cataloguesimpl.**delete**(*workspace_id*, *catalogue_id*)

Deletes a catalogue by its id

Parameters

- **workspace_id** – The workspace ID
- **catalogue_id** – The Catalogue ID

Returns The deleted catalogue descriptor

son_editor.impl.cataloguesimpl.**get_catalogue**(*catalogue_id*)

Retrieves a catalogue by its id

Parameters **catalogue_id** – The catalogues ID

Returns

son_editor.impl.cataloguesimpl.**get_catalogues**(*workspace_id*)

Retrieves all catalogues of a specific workspace

Parameters **workspace_id** – the workspace id

Returns list of catalogue descriptors of this workspace

son_editor.impl.cataloguesimpl.**update_catalogue**(*workspace_id*, *catalogue_id*, *catalogue_data*)

Updates a specific catalogue by its id. The catalogue applies the given name and url, that are in the json parameter. :param workspace_id: The Workspace ID :param catalogue_id: The Catalogue ID :return: The updated Catalogue descriptor

1.1.1.3.3 son_editor.impl.functionsimpl module

son_editor.impl.functionsimpl.**create_function**(*ws_id: int*, *project_id: int*, *function_data: dict*) → dict

Creates a new vnf in the project

Parameters

- **ws_id** – The workspace ID
- **project_id** – The Project ID
- **function_data** – The function data to create

Returns The created function as a dict

son_editor.impl.functionsimpl.**delete_function**(*ws_id: int*, *project_id: int*, *function_id: int*) → dict

Deletes the function

Parameters

- **ws_id** – The workspace ID
- **project_id** – The project ID
- **function_id** – The function ID

Returns the deleted function

son_editor.impl.functionsimpl.**delete_image_file**(*ws_id*, *project_id*, *vnf_id*, *filename*)
Deletes the image file with the given name

Parameters

- **ws_id** – The workspace ID
- **project_id** – The project ID
- **vnf_id** – The VNF ID
- **filename** – The name of the file to delete

Returns A success message

Raises *NotFound* – if the image file could not be located

son_editor.impl.functionsimpl.**get_function_project**(*ws_id: int*, *project_id: int*, *vnf_id: int*) → dict
Get a single function from the specified project

Parameters

- **ws_id** – The Workspace ID
- **project_id** – The project ID
- **vnf_id** – The VNF ID

Returns The requested function descriptor

son_editor.impl.functionsimpl.**get_functions**(*ws_id: int*, *project_id: int*) → list
Get a list of all functions

Parameters

- **ws_id** – The workspace ID
- **project_id** – The project id

Returns

son_editor.impl.functionsimpl.**get_image_files**(*ws_id*, *project_id*, *function_id*)
Returns a list of image file names located in the vnf folder

Parameters

- **ws_id** – The Workspace ID
- **project_id** – The project ID
- **function_id** – The function ID

Returns A List of image file names for this VNF

son_editor.impl.functionsimpl.**get_references**(*function*, *session*)
Search for references to the function in the database

Parameters

- **function** – The referenced function

- **session** – The database session

Returns A list of services referencing the given function.

son_editor.impl.functionsimpl.**get_uid**(*vendor*, *name*, *version*)

son_editor.impl.functionsimpl.**replace_function_refs**(*refs*, *vendor*, *name*, *version*, *new_vendor*, *new_name*, *new_version*)

Replaces every reference in refs to the function with (name:vendor:version) with the new_(name:vendor:version)

Parameters

- **refs** – The services referencing the function descriptors
- **vendor** – The old vendor
- **name** – The old name
- **version** – The old version
- **new_vendor** – The new vendor
- **new_name** – The new name
- **new_version** – The new version

son_editor.impl.functionsimpl.**save_image_file**(*ws_id*, *project_id*, *function_id*, *file*)

Saves the vnf image file into the vnfs folder

Parameters

- **ws_id** – The workspace ID
- **project_id** – The project ID
- **function_id** – The function ID
- **file** – The image file

Returns A success message

son_editor.impl.functionsimpl.**update_function**(*ws_id: int*, *prj_id: int*, *func_id: int*, *func_data: dict*) → dict

Update the function descriptor

Parameters

- **ws_id** – The Workspace ID
- **prj_id** – The Project ID
- **func_id** – The function ID
- **func_data** – The funtion Data for updating

Returns The updated function descriptor

son_editor.impl.functionsimpl.**validate_vnf**(*schema_index: int*, *descriptor: dict*) → None

Validates the VNF against the VNF schema

Parameters

- **schema_index** – The index of the schema repository
- **descriptor** – The descriptor to validate

Returns Nothing if descriptor id valid

Raises *InvalidArgument* – if the schema is not Valid

1.1.1.3.4 son_editor.impl.gitimpl module

son_editor.impl.gitimpl.**build_github_delete**(*owner: str*, *repo_name: str*) → str

Builds relative github api url to delete a repository

Parameters

- **owner** – Owner of the github repository
- **repo_name** – Repository name

Returns the relative GitHub api url

son_editor.impl.gitimpl.**check_son_validity**(*project_path: str*)

Checks if the given project path is a valid son project, otherwise it raises an exception. Valid means, it has a consistent son file structure, so no semantics will be tested.

Parameters **project_path** – the path of the cloned project

son_editor.impl.gitimpl.**clone**(*ws_id: int*, *url: str*, *name: str = None*)

Clones a repository by url into given workspace

Parameters

- **name** – Optional name of the local repository name, otherwise the remote name is taken
- **user_data** – Session data to get access token for GitHub
- **ws_id** – Destination workspace to clone
- **url** – URL of the source repository

Returns True if successful, otherwise NameConflict is thrown

son_editor.impl.gitimpl.**commit_and_push**(*ws_id: int*, *project_id: int*, *commit_message: str*)

Commits and then pushes changes.

Parameters

- **ws_id** – The workspace ID
- **project_id** – The project ID
- **commit_message** – The commit message

Returns a dictionary containing the result of the operation

son_editor.impl.gitimpl.**create_commit_and_push**(*ws_id: int*, *project_id: int*, *remote_repo_name: str*)

Creates a remote GitHub repository named remote_repo_name and pushes given git project into it.

Parameters

- **ws_id** – Workspace ID
- **project_id** – Project ID to create and push it
- **remote_repo_name** – Remote repository name

Returns a dictionary containing the result of the operation

son_editor.impl.gitimpl.**create_info_dict**(*out: str = None*, *err: str = None*, *exitcode: int = 0*) → dict

Creates a dict that holds process information

Parameters

- **out** – Out bytes
- **err** – Err bytes
- **exitcode** – exitcode

Returns Dict with packed information.

son_editor.impl.gitimpl.**create_oauth_header**() → dict

Creates oauth header by providing the access token in the header.

Returns Header as dict

son_editor.impl.gitimpl.**delete**(*ws_id: int*, *project_id: int*, *remote_repo_name: str*, *organization_name: str = None*)

Deletes given project on remote repository

Parameters

- **project_id** –
- **ws_id** – Workspace of the project
- **remote_repo_name** – Remote repository name
- **organization_name** – Optional parameter to specify the organization / login

Returns a dictionary containing the result of the operation

son_editor.impl.gitimpl.**diff**(*ws_id: int*, *pj_id: int*)

Shows the local changes of the given project.

Parameters

- **ws_id** – Workspace of the project.
- **pj_id** – Given project to show from.

Returns a dictionary containing the result of the operation

son_editor.impl.gitimpl.**get_project**(*ws_id*, *pj_id: int*, *session=<sqlalchemy.orm.session.Session object at 0x000002BDC27C7748>*) → son_editor.models.project.Project

Returns a project and raises 404, when project not found.

Parameters

- **ws_id** – Workspace id
- **pj_id** – Project id
- **session** – db session

Returns Project model

son_editor.impl.gitimpl.**get_workspace**(*ws_id: int*) → son_editor.models.workspace.Workspace

Returns the workspace model of the given workspace

Parameters **ws_id** – The workspace ID

Returns The corresponding workspace model

son_editor.impl.gitimpl.**git_command**(*git_args: list*, *cwd: str = None*)

Calls the git command with given args and returns out, err and exitcode

Parameters

- **git_args** – Arguments for git
- **cwd** – Optional current working directory

Returns out, error, exitcode

son_editor.impl.gitimpl.**init**(*ws_id: int*, *project_id: int*)
Initializes a git repository in the given project

Parameters

- **ws_id** – The workpace ID
- **project_id** – The project ID to initialize

Returns a dictionary containing the result of the operation

son_editor.impl.gitimpl.**is_github**(*netloc*)
Checks if the given url is on github

Parameters **netloc** – http url

Returns True if on github, False else

son_editor.impl.gitimpl.**list**()
Lists the available remote repositories.

Parameters **ws_id** – The workspace ID

Returns https://developer.github.com/v3/repos/#response

son_editor.impl.gitimpl.**pull**(*ws_id: int*, *project_id: int*)
Pulls data from the given project_id. :param ws_id: Workspace of the project :param project_id: Project to pull. :return: a dictionary containing the result of the operation

son_editor.impl.gitimpl.**setup_git_user_email**(*project_full_path: str*)
Setting up the git user in the local git config to be able to make commits and push

Parameters **project_full_path** – The absolute project path

son_editor.impl.gitimpl.**status**(*ws_id: int*, *pj_id: int*)
Shows the git status of the repository

Parameters

- **ws_id** – The workspace ID
- **pj_id** – The project ID

Returns a dictionary containing the result of the operation

1.1.1.3.5 son_editor.impl.nsfslookupimpl module

son_editor.impl.nsfslookupimpl.**find_by_priority**(*user_data*, *ws_id*, *project_id*, *vendor*, *name*, *version*, *is_vnf*)

Tries to find vnf / network services by descending priority

1. project
2. private catalogue
3. public catalogues.

Parameters

- **user_data** – Information about the current user
- **ws_id** – The Workspace ID
- **project_id** – The project ID
- **vendor** – The descriptors vendor
- **name** – The descriptors name
- **version** – The descriptors versions
- **is_vnf** – if the descriptor is a VNF

Returns The descriptor if found

son_editor.impl.nsfslookupimpl.**find_network_service**(*user_data*, *ws_id*, *project_id*, *vendor*, *name*, *version*)

Tries to find a network service by descending priority 1. project 2. private catalogue 3. public catalogues.

Parameters

- **user_data** – Information about the current user
- **ws_id** – The Workspace ID
- **project_id** – The project ID
- **vendor** – Vendor name of the network service
- **name** – Name of the network service
- **version** – The version of the network service

Returns If found, it returns the network service

son_editor.impl.nsfslookupimpl.**find_vnf**(*user_data*, *ws_id*, *project_id*, *vendor*, *name*, *version*)

Tries to find a vnf by descending priority 1. project 2. private catalogue 3. public catalogues.

Parameters

- **user_data** – Information about the current user
- **ws_id** – The Workspace ID
- **project_id** – The project ID
- **vendor** – Vendor name of the function
- **name** – Name of the function
- **version** – The version of the function

Returns If found, it returns the function

son_editor.impl.nsfslookupimpl.**get_function**(*functions*, *vendor*, *name*, *version*)

Finds a function in the given set of functions which matches vendor,name,version

Parameters

- **functions** – Set of functions to look for the specific one
- **vendor** – Vendor name of the function

- **name** – Name of the function
- **version** – The version of the function

Returns If found, it returns the function. Otherwise it returns None

son_editor.impl.nsfslookupimpl.**get_project**(*project_id*)
Retrieves the project which matches the given project id. Otherwise it raises NotFound Exception

Parameters **project_id** – The project ID

Returns The project model

1.1.1.3.6 son_editor.impl.platform_connector module

son_editor.impl.platform_connector.**create_service_on_platform**(*ws_id*, *platform_id*, *service_data*)
Deploys the service on the referenced Platform

Parameters

- **ws_id** – The workspace ID
- **platform_id** – The platform ID
- **service_data** – The service descriptor data

Returns A message if the function was deployed successfully

son_editor.impl.platform_connector.**publish_referenced_functions**(*ws_id*, *proj_id*, *descriptor*)
Publishes the referenced functions to the private cataloge after packaging

Parameters

- **ws_id** – The workspace ID
- **proj_id** – The project ID
- **descriptor** – The service descriptor

1.1.1.3.7 son_editor.impl.platformsimpl module

son_editor.impl.platformsimpl.**create_platform**(*workspace_id: int*, *platform_data*) → dict
Create a new platform entry

Parameters

- **workspace_id** – The workspace ID
- **platform_data** – The platform info

Returns The newly created platform descriptor

son_editor.impl.platformsimpl.**delete**(*workspace_id: int*, *platform_id: int*) → dict
Deletes the platform from the workspace

Parameters

- **workspace_id** – The workspace ID
- **platform_id** – The platform ID

Returns the deleted platform description

son_editor.impl.platformsimpl.**get_platform**(*platform_id: int*) → dict
Get the platform definition

Parameters **platform_id** – The platform ID

Returns The platform information

son_editor.impl.platformsimpl.**get_platforms**(*workspace_id: int*) → list
Get a list of platforms for this workspace

Parameters **workspace_id** – The workspace ID

Returns A list of all platforms defined for this workspace

son_editor.impl.platformsimpl.**update_platform**(*workspace_id: int*, *platform_id: int*, *platform_data*) → dict
Update the platform entry

Parameters

- **workspace_id** – The workspace ID
- **platform_id** – The platform ID

Returns The updated platform definition

1.1.1.3.8 son_editor.impl.private_catalogue_impl module

son_editor.impl.private_catalogue_impl.**get_private_nsfs_list**(*ws_id*, *is_vnf*)
Get a list of all private services or functions

Parameters

- **ws_id** – the Workspace ID
- **is_vnf** – if vnf or services should be queried

Returns List of all private services or functions

son_editor.impl.private_catalogue_impl.**publish_private_nsfs**(*ws_id: int*, *descriptor: dict*, *is_vnf: bool*)
Publishes a function or service to the private catalogue repository

Parameters

- **ws_id** – The Workspace ID
- **descriptor** – The descriptor to publish

son_editor.impl.private_catalogue_impl.**query_private_nsfs**(*ws_id*, *vendor*, *name*, *version*, *is_vnf*)
Finds a function in the private catalogue

Parameters

- **ws_id** – The workspace ID
- **is_vnf** – if descriptor is a VNF
- **vendor** – the descriptors vendor
- **name** – the descriptors name
- **version** – the descriptors version

Returns The requested descriptor if found, None if nothing found

1.1.1.3.9 son_editor.impl.projectsimpl module

Created on 05.08.2016

@author: Jonas

son_editor.impl.projectsimpl.**create_project**(*ws_id: int*, *project_data: dict*) → dict
Create a new Project in this workspace

Parameters

- **ws_id** – The workpace ID
- **project_data** – The project data to create

Returns The new project descriptor as a dict

son_editor.impl.projectsimpl.**delete_project**(*project_id: int*) → dict
Deletes the project from the database and from the Disk

Parameters **project_id** – The id of the project to be deleted

Returns The deleted project descriptor

son_editor.impl.projectsimpl.**get_project**(*ws_id*, *pj_id*)
Get a specific project :param ws_id: The workspace ID :param pj_id: The project ID :return: The project descriptor

son_editor.impl.projectsimpl.**get_project_path**(*workspace_path: str*, *rel_path: str*) → str
Helper method to resolve the project path on disk for the given project

Parameters

- **workspace_path** – the path to the workspace
- **rel_path** – the relative path of the project

Returns The absolute project path

son_editor.impl.projectsimpl.**get_projects**(*ws_id: int*) → list
Get a list of projects in this workspace

Parameters **ws_id** – The workspace ID

Returns List of all projects

son_editor.impl.projectsimpl.**on_rm_error**(*func*, *path*, *exc_info*)
Gets called if rm_tree gets an error, happens especially if trying to remove .git files on windows

son_editor.impl.projectsimpl.**set_data**(*project: son_editor.models.project.Project*, *project_data: dict*) → None
Extracts the data from the dictionary and sets it on the database model

Parameters

- **project** – The project database model
- **project_data** – The project data dictionary from the frontend

son_editor.impl.projectsimpl.**update_project**(*project_data*, *project_id*)
Update the Project

Parameters

- **project_data** – The project Data
- **project_id** – The project ID to update

Returns The updated project descriptor

1.1.1.3.10 son_editor.impl.servicesimpl module

son_editor.impl.servicesimpl.**create_service**(*ws_id: int*, *project_id: int*, *service_data: dict*) → dict

Creates a service in the given project

Parameters

- **ws_id** – The Workspace of the project
- **project_id** – The Project of the Service
- **service_data** – the service descriptor

Returns The created service descriptor

son_editor.impl.servicesimpl.**delete_service**(*project_id: int*, *service_id: int*) → dict

Deletes the service from the Database and from the disk

Parameters

- **project_id** – The Projects ID
- **service_id** – The Services ID

Returns The descriptor of the deleted service

son_editor.impl.servicesimpl.**get_references**(*service*, *session*)

Searches for references to the service.

Parameters

- **service** – The service
- **session** – the db_session

Returns a list of services referencing the given service

son_editor.impl.servicesimpl.**get_service**(*ws_id*, *parent_id*, *service_id*)

Get the service by ID

Parameters

- **ws_id** – The workspace ID of the Project
- **parent_id** – The project ID
- **service_id** – the Service ID

Returns The requested service data

son_editor.impl.servicesimpl.**get_services**(*ws_id: int*, *project_id: int*) → list

Get a list of all services in this Project

Parameters

- **ws_id** – The workspace ID
- **project_id** – The project ID

Returns A list of service descriptors as dicts

son_editor.impl.servicesimpl.**get_uid**(*vendor*, *name*, *version*)

son_editor.impl.servicesimpl.**replace_service_refs**(*refs*, *vendor*, *name*, *version*, *new_vendor*, *new_name*, *new_version*)

son_editor.impl.servicesimpl.**update_service**(*ws_id*, *project_id*, *service_id*, *service_data*)

Update the service using the service data from the request

Will also check for references by other services and create a copy if so

Parameters

- **ws_id** – The Workspace ID
- **project_id** – The project ID
- **service_id** – The service ID
- **service_data** – The service data containing the "descriptor" and optionally some "meta" data

Returns The updated service data

son_editor.impl.servicesimpl.**validate_service_descriptor**(*schema_index: int*, *descriptor: dict*) → None

Validates the given descriptor with the schema loaded from the configuration

Parameters

- **schema_index** – the workspace
- **descriptor** – the service descriptor

Raises InvalidArgument: if the validation fails

1.1.1.3.11 son_editor.impl.usermanagement module

Created on 05.08.2016

@author: Jonas

son_editor.impl.usermanagement.**get_user**(*login: str*)

Gets the user from the Database if it exists or creates a new user in the Database using the login data from the session. If the database does not yet have the full user Data it is queried from Github using the access Token

Returns The database user model

1.1.1.3.12 son_editor.impl.userserviceimpl module

son_editor.impl.userserviceimpl.**get_user_info**() → dict

Returns current user information

son_editor.impl.userserviceimpl.**login**()

Login the User with a referral code from the github oauth process

son_editor.impl.userserviceimpl.**logout**()

Logs out the current user and removes all session related stuff

Returns Redirect

son_editor.impl.userserviceimpl.**origin_from_referrer**(*referrer*)

1.1.1.3.13 son_editor.impl.workspaceimpl module

Created on 25.07.2016

@author: Jonas

son_editor.impl.workspaceimpl.**create_token_file**(*ws_path*, *token*)

son_editor.impl.workspaceimpl.**create_workspace**(*login: str*, *workspace_data: dict*) → dict

Creates a workspace (on disk and in the database) from the given workspace data

Parameters **workspace_data** – The workspace configuration data

Returns The created workspace

son_editor.impl.workspaceimpl.**delete_workspace**(*wsid*)

Deletes the workspace from the database and from disk

Parameters **wsid** – The workspace ID

Returns The deleted workspace

son_editor.impl.workspaceimpl.**get_workspace**(*ws_id: int*) → dict

Get a workspace by ID

Parameters **ws_id** – The workspace ID

Returns A dictionary wich contains the Workspace configuration

son_editor.impl.workspaceimpl.**get_workspaces**(*login: str*) → list

Get all workspaces for the current user

Returns A list wof workspace dictionaries

son_editor.impl.workspaceimpl.**on_rm_error**(*func*, *path*, *exc_info*)

Gets called if rm_tree gets an error, happens especially if trying to remove .git files on windows

son_editor.impl.workspaceimpl.**test_url**(*name*, *url*)

Tests the url for reachability :param name: the server name :param url: the server url :raises ExtNotReachable: if the external server could not be contacted

son_editor.impl.workspaceimpl.**update_workspace**(*workspace_data*, *wsid*)

Updates the workspace with the given workspace data

Parameters

- **workspace_data** – The new workspace configuration
- **wsid** – the workspace ID

Returns The updated workspace

1.1.1.4 son_editor.models package

1.1.1.4.1 son_editor.models.descriptor module

class son_editor.models.descriptor.**Descriptor**(*name=None*, *version=None*, *vendor=None*, *descriptor=None*)

Bases: sqlalchemy.ext.declarative.api.Base

The Base class for storing the function and the service descriptors. Both descriptors share the properties name, vendor and version and are constrained to have a unique uid inside of a project

as_dict()

descriptor

id

name

project_id

uid

vendor

version

class son_editor.models.descriptor.**Function**(*name=None*, *version=None*, *vendor=None*, *descriptor=None*, *project=None*)

Bases: *son_editor.models.descriptor.Descriptor*

The Model for the function Descriptor

descriptor

id

name

project

project_id

uid

vendor

version

class son_editor.models.descriptor.**Service**(*name=None*, *version=None*, *vendor=None*, *descriptor=None*, *project=None*, *meta='{}'*)

Bases: *son_editor.models.descriptor.Descriptor*

The Model for the service descriptor additionally contains a meta property that can hold arbitrary meta-data about the service

as_dict()

descriptor

id

meta

name

project

project_id

uid

vendor

version

1.1.1.4.2 son_editor.models.private_descriptor module

class son_editor.models.private_descriptor.**PrivateDescriptor**(*ws_id=None, name=None, version=None, vendor=None, descriptor=None*)

Bases: sqlalchemy.ext.declarative.api.Base

The private descriptor is the model for the service and function descriptors in the private catalogue

as_dict()

descriptor

id

name

uid

vendor

version

ws_id

class son_editor.models.private_descriptor.**PrivateFunction**(*ws_id=None, name=None, version=None, vendor=None, descriptor=None*)

Bases: *son_editor.models.private_descriptor.PrivateDescriptor*

The Private function is the Model for the Function descriptors in the private catalogue

descriptor

id

name

uid

vendor

version

workspace

ws_id

class son_editor.models.private_descriptor.**PrivateService**(*ws_id=None, name=None, version=None, vendor=None, descriptor=None*)

Bases: *son_editor.models.private_descriptor.PrivateDescriptor*

The Private service is the Model for the Service descriptors in the private catalogue

descriptor

id

name

uid

vendor

version

workspace

ws_id

1.1.1.4.3 son_editor.models.project module

class `son_editor.models.project.`**`Project`**(*name=None*, *rel_path=None*, *workspace=None*)

Bases: `sqlalchemy.ext.declarative.api.Base`

The Project model corresponds to the project folders in the workpace. It stores references to the services and functions of one project in the database. If the project was shared via GitHub the repo_url points to the respective repository.

as_dict()

description

functions

id

maintainer

name

publish_to

rel_path

repo_url

services

vendor

version

workspace

workspace_id

1.1.1.4.4 son_editor.models.repository module

class `son_editor.models.repository.`**`Catalogue`**(*name=None*, *url=None*, *publish=None*, *workspace=None*)

Bases: *son_editor.models.repository.Repository*

id

name

publish

url

workspace

workspace_id

class son_editor.models.repository.**Platform**(*name=None*, *url=None*, *publish=None*, *workspace=None*, *token_path='empty'*)

Bases: *son_editor.models.repository.Repository*

id

name

publish

token_path

url

workspace

workspace_id

class son_editor.models.repository.**Repository**(*name=None*, *url=None*, *publish=False*)

Bases: sqlalchemy.ext.declarative.api.Base

The repository model is the base class for the external repositories that can be contacted through the given url

as_dict()

id

name

publish

url

workspace_id

1.1.1.4.5 son_editor.models.user module

class son_editor.models.user.**User**(*name=None*, *email=None*)

Bases: sqlalchemy.ext.declarative.api.Base

The user model stores the username and his email that was registered at github. It is the root object for all data belonging to a user

email

id

name

workspaces

1.1.1.4.6 son_editor.models.workspace module

class son_editor.models.workspace.**Workspace**(*name=None*, *path=None*, *owner=None*, *schema_index=0*)

Bases: sqlalchemy.ext.declarative.api.Base

The workspace model stores information about the workspace and has references to its children like the catalogues, platform and projects The schema_index corresponds to the index of the schema_remote_masters that are configured in the server configuration

as_dict()

catalogues

id

name

owner

owner_id

path

platforms

priv_functions

priv_services

projects

schema_index

1.1.1.5 son_editor.util package

1.1.1.5.1 son_editor.util.constants module

Created on 22.07.2016

@author: Jonas

son_editor.util.constants.**CATALOGUES** = **'catalogues'**
The network service url constant

son_editor.util.constants.**GIT** = **'git'**
The project path relative to the workspace

class son_editor.util.constants.**Github**
Bases: object

Holds GitHub API relevant strings.

API_CREATE_REPO_REL = **'/user/repos'**

1.Argument is owner of the repos to delete

2.Argument is the name of the remote repository

API_DELETE_REPO = **'/repos/{}/{}'**

1.Argument is Username

API_LIST_REPOS = **'/users/{}/repos'**

API_URL = **'https://api.github.com'**

DOMAINS = **['github.com', 'www.github.com']**

son_editor.util.constants.**NSFS** = **'nsfs'**
The git api url constant

son_editor.util.constants.**PLATFORMS** = **'platforms'**
the platform url constant

son_editor.util.constants.**PROJECTS** = **'projects'**
The platform url constant

son_editor.util.constants.**SERVICES** = **'services'**
The virtual network function url constant

son_editor.util.constants.**VNFS** = **'functions'**
The network services and funtions url constant

son_editor.util.constants.**WORKSPACES** = **'workspaces'**
The project url constant

1.1.1.5.2 son_editor.util.context module

son_editor.util.context.**init_test_context**()

Initializes a test-case context and cleans up workspace location beforehand

Returns The test client

1.1.1.5.3 son_editor.util.descriptorutil module

son_editor.util.descriptorutil.**get_file_name**(*model*) → str
Get the standard file name for a descriptor

Parameters **model** – The database model of the descriptor

Returns The standard descriptor file name, computed from the models vendor name and version

son_editor.util.descriptorutil.**get_file_path**(*folder: str*, *model*) → str
Returns the filepath to the descriptor computed from the models vendor name and version

Parameters

- **folder** – the folder to write to, either "vnf" or "nsd" to specify if a vnf or network service needs to be saved
- **model** – The database model of the descriptor

Returns None

son_editor.util.descriptorutil.**get_schema**(*schema_index*, *schema_id: str*) → dict
Get the requested schema :param schema_index: The schema index referring to the "schema"-index in the configuration file :param schema_id: either "vnf" or "ns" :return: The requested schema

son_editor.util.descriptorutil.**get_schemas**()
Get the schemas

Will load the schemas if still empty

son_editor.util.descriptorutil.**load_ns_vnf_from_disk**(*file: str*, *model*)
Loads a vnf or network service descriptor from disk and initializes the given model

Parameters

- **file** – the file path of the descriptor
- **model** – The database model of the descriptor

Returns the given updated model

son_editor.util.descriptorutil.**load_project_descriptor**(*project*) → dict
Loads the project descriptor from disk

son_editor.util.descriptorutil.**load_schemas**()
Loads the schemas congigured under "schemas" from the schema remotes

son_editor.util.descriptorutil.**load_workspace_descriptor**(*workspace*) → None
Loads the workspace descriptor from disk and updates the database model

Parameters **workspace** – The workspace database model

son_editor.util.descriptorutil.**sync_project_descriptor**(*project*) → None
Updates the project model with data from the project descriptor and vice versa

Parameters **project** – The projects database model

son_editor.util.descriptorutil.**update_workspace_descriptor**(*workspace*) → None
Updates the workspace descriptor with data from the workspace model

Parameters **workspace** – The workspace model

son_editor.util.descriptorutil.**write_ns_vnf_to_disk**(*folder: str*, *model*) → None
Saves the given model to disk as a yml file

Parameters

- **folder** – the folder to write to, either "vnf" or "nsd" to specify if a vnf or network service needs to be saved
- **model** – The database model of the descriptor

Returns None

son_editor.util.descriptorutil.**write_private_descriptor**(*workspace_path: str*, *is_vnf: bool*, *descriptor: dict*)
Write the private descriptor into the private cataloge folder on disk

Parameters

- **workspace_path** – The workspace path
- **is_vnf** – If the descriptor is a vnf
- **descriptor** – the descriptor data

Returns

son_editor.util.descriptorutil.**write_project_descriptor**(*project*, *project_descriptor*)
Writes the project database model to disk

1.1.1.5.4 son_editor.util.publishutil module

son_editor.util.publishutil.**pack_project**(*project: son_editor.models.project.Project*) → str
Calls the son-package cli tool to pack the project and prepare it for deployment

Parameters **project** – The project to pack

Returns The file name of the package file

son_editor.util.publishutil.**push_to_platform**(*package_path: str*, *ws: son_editor.models.workspace.Workspace*) → str
Pushes the package located at the package_path to the specified Platform

Parameters

- **package_path** – the location of package to be pushed on disk
- **platform** – The platform to upload to

Returns The uuid returned by the platform

1.1.1.5.5 son_editor.util.requestutil module

Created on 28.07.2016

@author: Jonas

`son_editor.util.requestutil.`**`get_config`**`()`
Returns the current configuration

`son_editor.util.requestutil.`**`get_json`**`(`*request: flask.wrappers.Request*`)` → dict
Helper function to get a json dict out of a request

Parameters **`request`** – Request to get the json data from

Returns json data as dict

`son_editor.util.requestutil.`**`prepare_error`**`(`*data=None*, *code=500*`)` → tuple
Prepares the error response and returns it as a tuple to accommodate for flask_restplus's way to deal with errors

Parameters

- **`data`** – The error message
- **`code`** – the http error code, 500 by default

Returns A tuple of the data, the status code and the headers

`son_editor.util.requestutil.`**`prepare_response`**`(`*data=None*, *code=200*`)` → flask.wrappers.Response
Sets the necessary headers and status code on the response

Parameters

- **`data`** – The data to be returned to the client
- **`code`** – the status code. 200 by default

Returns The Response object with the headers set according to the input data

`son_editor.util.requestutil.`**`rreplace`**`(`*s*, *old*, *new*, *occurrence*`)`
Replaces 'occurences' occurences of string 'old' in the given string 's' from right by 'new'

Parameters

- **`s`** – String that contains the replacing string
- **`old`** – String that gets replaced
- **`new`** – New string that replaces the old string
- **`occurrence`** – How many occurences get replaced

Returns String that has replaced strings

`son_editor.util.requestutil.`**`update_config`**`(`*config*`)`
Update the configuratiion file

Parameters **`config`** – The new configuration

Returns Message if successful