HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

COMPUTER SCIENCE

Term Project Thesis

Cisco DNA Center Multi Tenant Manager

University of Applied Science Rapperswil

Department of Computer Science

16.09.2019 - 20.12.2019

Authors	Aaron Meier	**Advisor**	Professor Laurent Metzger
	Dennis Ligtenberg	**External Co-Examiner**	Patrick Mosimann
		Internal Co-Examiner	Jessica Kalberer
Project Partner	Cisco Systems (Switzerland) GmbH		

Abstract

The Cisco Digital Network Architecture (DNA) offers a modern approach to campus networking by using an overlay based Software Defined Networking (SDN) approach. Thanks to an easy to use web platform tasks like network trouble shooting and edge device port configuration are massively simplified. A growing collection of API features (Intent API) opens the platform up for network automation and building customized tools additionally to the DNA Center Platform.

A feature often requested by Cisco's customers is the possibility to allow for multi tenancy in the host onboarding feature which offers configuration of edge/host ports. This way simple configurations can be made by less technically versed staff without having to call a technical specialist for every port that requires configuration. Adding multi tenancy can be interesting for organizations like universities that have multiple academic departments that inhabit and maintain separate spaces on the campus. To allow simple configuration currently full access to the DNA Center Platform would have to be granted to staff of each department.

In the scope of this Thesis a prove of concept has been built that allows access management to specific ports on the edge to user groups on a custom website using the DNA Center APIs. An administrator can not only grant read and write access to groups but also limit what configuration objects, like specific IP pools, can be used by a specific group.

Management Summary

Initial situation

The Cisco Digital Network Architecture (DNA) is a new approach to networking using Software Defined Networking. After initial setup many operational tasks are simplified thanks to a web interface that allows for configuration on all devices in the network. One example of this is the host onboarding feature. It allows network administrators to easily configure interfaces for end host usage. Neither physical access to a network nor command line knowledge is required to use this feature. This allows IT staff with less networking knowledge to complete this task.
Additionally to the many tools that can be used out of the box the platform is open for developers to build custom tools to fit any organizations needs.

A wish of many Cisco customers is to allow for multi tenancy on the host onboarding feature. This can be very useful for organizations like universities that have multiple academic departments that inhabit and maintain separate spaces on the campus. Currently there is no option to grant access to specific network interfaces to a specific user group. Using the open development interfaces DNA Center exposes to developers a third party tool can be built that allows for access management to settings on network device interfaces.

Process

The goal of the project was to build a prove of concept to show that using the open development interfaces allows for building a solution to add multi tenancy to the host onboarding process. The main focus is not on building an interface that can be used in production but to show the technical feasibility.
It was decided to use well known and open source tools the developers have some experience in to speed up development and improve the chance of delivering a working prototype. Additionally the application is made to be easily deployable so engineers can install it in lab environments for customer demos. This can be achieved by using modern containerization technologies.

Results

A web application was build that fully supports user and group management and a host onboarding feature that has the same capabilities as the feature in DNA Center but allows for multi tenancy. However some required functionality is not covered by the official development interfaces.

Using reverse engineering on the official web platform additional interfaces where discovered that allow for implementation of all desired features.

Outlook

While the developed application already offers a working platform that shows the capabilities an extensibility of DNA Center there is a variety of features that can be added to the existing application. One way to find useful features would be a design thinking session with Cisco networking specialists, that have productive experience in using DNA Center.

On top of new features the existing application can still be improved. The product is still a prove of concept and not yet a full productive application. By conducting usability tests to improve the frontend, improving code quality and testing, the application can be made suited to a productive environment.

Task description

Cisco DNA Center Multi-Tenant Manager

Introduction

Software-Defined Access (SD-Access) is the industry's first intent-based networking solution for the Enterprise built on the principles of Cisco's Digital Network Architecture (Cisco DNA). Cisco SD-Access provides automated end-to-end segmentation to separate user, device and application traffic without redesigning the network. Cisco SD-Access automates user access policy so that organizations can make sure the right policies are established for any user or device with any application across the network.

Idea

The DNA Center platform provides a web interface and multiple Application Programming Interfaces (APIs) to simplify and automate various network operation tasks. Currently, the platform's web interface and APIs do not offer true multi-tenancy. Access rights are only split between read and read/write authorization levels for all devices inside the DNA fabric. Organizations like universities might want to grant configuration access to specific network devices to an academic department (e.g. Computer Science, Electrical Engineering, Landscape Architecture, etc.).

Use Cases

The following Use Cases are going to be developed in the thesis:

- Network device list: A department IT manager views a list that includes an overview of all network appliances to which he has read access.
- Selection of a specific switch/port: Selection of a specific switch/port: A department IT manager selects a specific switch/port that needs to be configured.
- Port Configuration: A department IT Manager sets different preconfigured port setting like port type and authentication method.
- User and groups: A sysadmin creates, updates and deletes users and groups for the departments.
- View changelog: Any user views the history of changes made on a specific switch to which he/she has access.

Further Use Cases can be added while working on the thesis subject to prior consultation.

Task

In this thesis, a third-party app for the DNA Center shall be developed in order to give a department IT manager the possibility to configure defined ports for his own department.

Consumed APIs

Initially, the idea was to exclusively use DNA Center Intent APIs. Since it has been discovered that a lot of essential tasks cannot be achieved through pure Intent API calls, calls to unofficial endpoints are permitted in case there is no Intent alternative.

Functionality

The deliverable is a Proof of Concept (PoC) which shows that multi-tenancy can be achieved using the APIs. The focus should be set on the functionality, and not on the design of the user interface.

Cisco DNA Configuration

It is not part of the solution to add new devices or configure networks, VNs or any other object in Cisco DNA Center. These settings have to be configured by a professional directly in the DNA Center.

Contents

Part I

Technical report

Chapter 1

Introduction and overview

1.1 Problem definition

Cisco Digital Network Architecture (DNA) offers an overlay based Software defined networking (SDN) solution that can be accessed using the DNA Center, a platform based management tool[2]. The DNA Center platform provides a web interface and multiple Application Programming Interface (API) interfaces to simplify and automate various network operation tasks.

Currently the platforms web interface and APIs do not offer true multi tenancy. Access rights are only split between read and read/write authorization levels for all devices inside the DNA fabric. Organizations like universities might want to grant configuration access to specific network devices to a academic department (e.g. Computer Science, Microelectronics etc.). This would enable a layman to make simple changes to their devices.

DNA center overview

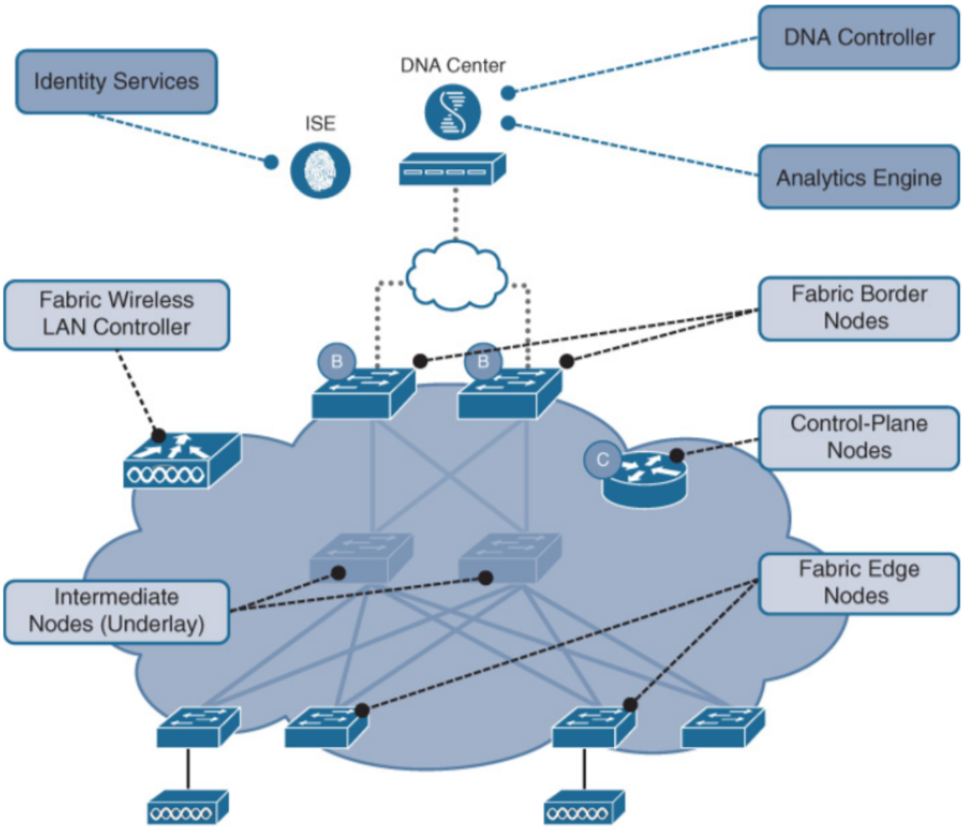


Figure 1.1: DNA center overview

DNA Centers virtual topology is based on VXLAN and includes the following components:

- **Cisco DNA Center**: Central component for designing, provisioning and monitoring/assure SD-Access deployments
- **Identity Services Engine**: Manages identities and policies. Is used for authorisation of fabrics enddevices. Possible connection to an Active Directory.
- **Control plane nodes**: Tracks the current location of fabrics users and devices (Based on LISP map). Can be used as single source of truth.

- **Border nodes**: One or more nodes, which connect the SD-Access fabric to the outside world (Uplink). Make sure that the fabric environment is reachable and translate different fabric types (e.g. virtual networks to SGTs).
- **Edge nodes**: Connects end devices (PCs, phones, cameras etc. with SD-Access fabric (First hop). Also called routing locators (RLOCs) in LISP
- **Fabric Wireless LAN Controller (WLC)**: Executes all Access Point relevant tasks: Controller for all APs which are connected to the network. Is responsible for fabric access trough radio frequencies.
- **Intermediate nodes**: All network devices, which are located between two fabric nodes (e.g. distributed or core switches)

DNA center has a minimum configuration of 3 devices: One Fabric Control Plane, one or more edges devices and one or more border devices.

1.2 Scope and limitations

Consumed APIs Initially the idea was to exclusively use DNA Center Intent APIs. Since it was discovered that a lot of essential tasks can not be achieved through pure Intent API calls, calls to unofficial endpoints are permitted if there is no Intent alternative. Further explanation and problems with the API are described in the project documentation,

Functionality The deliverable is a Proof of Concept (PoC) to show that multi tenancy can be achieved using the APIs. So the focus should be on functionality over the user interface and usability.

Cisco DNA Configuration It is not part of the solution to add new devices or configure networks, VNs or any other object in Cisco DNA Center. These settings are to be configured by a professional directly in the DNA Center. Generally speaking, we only support one-way, directional sync (Cisco DNA Center to Tenant Manager) where any settings on the DNA Center overwrite those on the Tenant Manager.

1.3 State of art

Currently there are no known software solutions that solve this exact problem. However there are already many well known user management and permission systems and the host onboarding is already present in DNA Center.

Chapter 2

Requirements

2.1 Use cases

In the following chapter Actors, Use cases in brief form are described to give an overview over the projects functionality.

2.1.1 Actors

Sysadmin

Sysadmins have full administrative power on the web interface, they manage all access permissions and typically have access to the DNA Center web panel. A sysadmin has advanced networking knowledge.

Department IT manager

Department IT managers belong to a sub organization (e.g. the Electronics department) that has access to basic configuration options for their networking infrastructure. To use the tool they only require basic networking knowledge.

2.1.2 Use case diagram

Since all actions in the use case diagram require authentication it is left out in this diagram. To make the diagram more compact and readable optional use cases are left out.

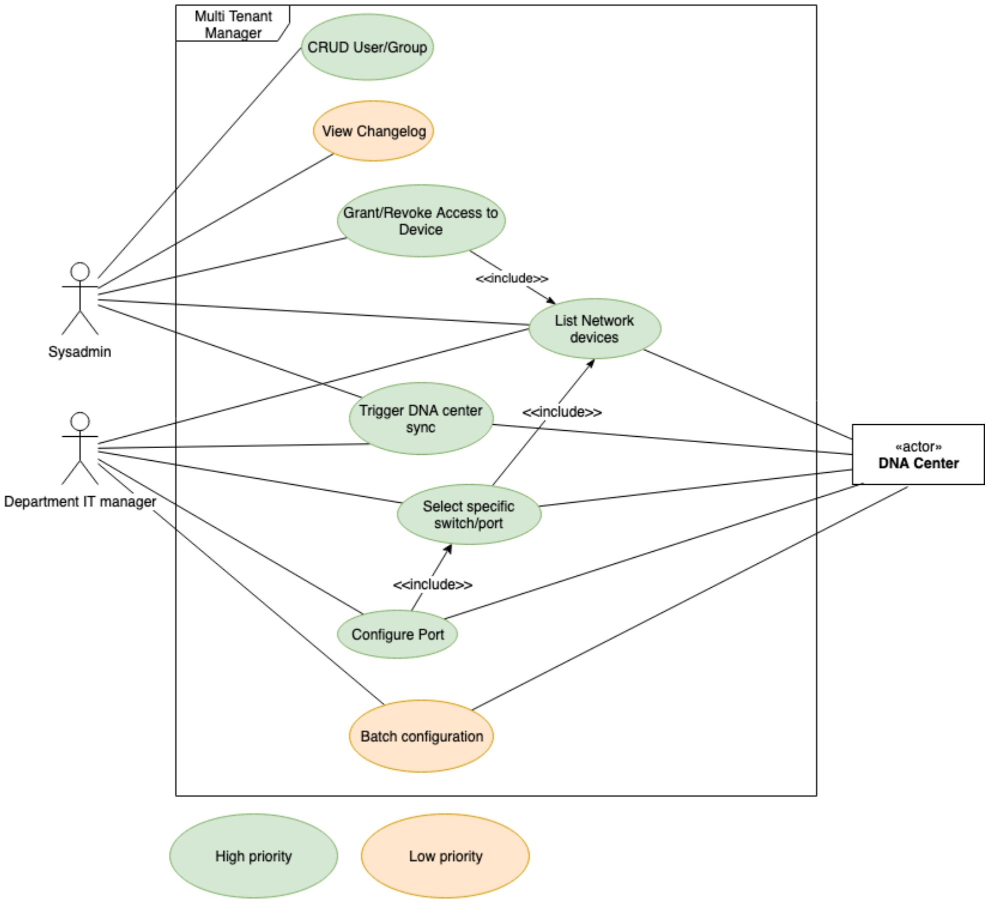


Figure 2.1: Use case diagram

2.1.3 Use cases brief

UC01 - List network devices

A department IT manager views a list with an overview of of all network appliances they have read access to.

UC02 - Grant and revoke access

A sysadmin grants or revokes a specific group read/write access to a specific switch/port.

UC03 - CRUD users and groups

A sysadmin creates/updates/deletes users and groups and adds users to groups.

UC04 - View changelog

Any user views a history of changes made on a specific switch they have access too.

UC5 - Select specific switch/port

A department IT manager selects a specific switch/port to configure.

UC6 - Configure port

A department IT manager sets different preconfigured port settings like port type and authentication method.

UC7 - Batch configuration(optional)

A department IT manager applies the same configuration to multiple ports at a time using a CSV file.

UC8 - Trigger DNA center sync

Any authenticated user manually triggers a sync to update all data of the application with the current state from the DNA center.

UC9 - Group devices/ports (optional)

A department IT manager creates groups of devices or ports for batch configuration using just the web interface.

UC10 - Configure grouped devices/ports (optional)

A department IT manager applies the same configuration to a group of devices/ports.

2.2 None functional requirements

#	Category	Description and acceptance criteria
NFR1	Functionality	Only authenticated users should have access to the system. Is authentication required?
NFR2	Functionality	Passwords should be protected trough modern measures. Are all passwords stored securely by default (Hash + Salt)?
NFR3	Functionality	Communication between the user and the system should be encrypted. Is the latest TLS-encryption supported?
NFR4	Reliability	On failed sync the app should use fallback data and indicate outdated data with an error If sync fails is there an error message suggesting that outdated data is used?
NFR5	Reliability	Failure of configuration changes should be traceable for administrators. Are configuration failures obvious in the systems log?
NFR6	Efficiency	The system should respond without a noticeable delay (=total of 10 seconds between request and response) under full workload (=100 simultaneous requests) for the login operation. Was the response time goal tested trough a load test?
NFR7	Maintainability	For future maintainability the system should depend on Ciscos intent APIs. Are all API calls made to intent based endpoints?
NFR8	Maintainability	The system should scale well on a typical campus networks growth with a maximum of 5,000 devices, a total of 480,000 ports, 256 VNs (based on Cisco DNAs limitation [6]). This means while the initial fetch of the above amount of information from DNA Center is running, response time of the application does not exceed 10s (time to display a loading screen/error message or the result). Was there a test made for the above scenario to validate the applications responsiveness?
NFR9	Portability	The systems installation should be straight forward and deployable trough Docker containers. Are containers defined for all software components?

Table 2.1: NFR according to ISO-9126 [10]

2.3 Excluded functionality

The following features are features that are not in the scope of this project but might want to be kept in mind when making architectural decisions.

External authentication back end A multitude of different authentication back ends/protocols (e.g. radius) could be implemented to make integration into existing environments easier.

Legacy switch migration By adding support for legacy switches none DNA fabric devices could be managed in the tool.

Synchronisation discrepancy handling When changes are made in the DNA center that are also managed in the Multi Tenant Manager discrepancies can occur. The applications could be expanded to handle those discrepancies in multiple ways.

Chapter 3

Architecture and design specification

3.1 External interfaces

3.1.1 DNA Center Intent API

The main external interface used in this project is the DNA Center Northbound Intent API. Northbound APIs provide the Interfaces to communicate between the SDN controller and an application used for network automation (as opposed to Southbound APIs that offer communication between the controller and network nodes in the fabric). The Intent API provides a policy-based abstraction for business intent. It is designed to focus on defining outcome rather than programming specific steps.[7]

Disclaimer

These findings where made using version 1.3.0.94 of DNA Center and 1.2.0.36 of the DNAC API Platform. DNA Center is a continuously developed and improved platform. There is no guarantee that the described endpoints have the same limitations in future releases.

A number of undocumented endpoints where used in this project. These endpoints are used without warranty and should not be used in commercial environments. Updating the DNA Center must be done with care. While intent endpoints guarantee stability there's no guarantee that undocumented endpoints are not subject to change.

Intent API limitations

As the Intent API is still growing in features there are multiple limitations that need to be taken into account when designing and implementing the system. This section describes the most important problems and solutions of those limitations. Specific endpoints used are described in the implementation section.

API rate limiting The standard rate limit for many API endpoints in DNA Center is 5 calls per minute (this is not for total API calls to the Intent API but for one specific REST endpoint). Any batch import will run for at least $\frac{|PORTS|}{5}$ minutes.

Get all edge devices Currently there is no way to retrieve a list of all edge devices in the fabric through the API. One solution would be settings manual tags on the DNA web interface to select a pre-defined list of edge devices. This isn't a very reliable and salable solution as manual work has to be done on the web interface to tag devices.

Another approach would be using the Fabric Wired Connectivity Intent endpoint to get information about an edge device for all devices in the fabric. If the device isn't in the fabric an error is responded. This endpoint is subject to the 5 calls/min rate limit and thus does not scale well either.

The final approach is using the same endpoint the DNA web interface used to get a device list. This endpoint is not documented but uses the same authentication token as official endpoints. All devices with relevant information can be requested in a single call. As this is the only properly scalable option this approach is used.

Port assignment (POST) The official endpoint for port assignment presents multiple issues. Firstly it has the same rate limitations as the GET endpoint, secondly only user devices can be configured on a port while there is an option for access points and servers in the host onboarding feature on the DNA Center web interface. As the rate limitations would make any batch configuration beyond 5 ports incredibly slow again the unofficial endpoint the web interface uses is used.
Beyond these two more unofficial endpoints are used in this project. As they are less vital they are described in the implementation section.

3.2 Domain

3.2.1 Domain Model

For a better understanding of the abstract behaviour of the application following domain model can be used. It is split into two groups, blue for user management and orange for configuration of network devices. The data classes and actual implementation may differ from the domain model. The basis for most of the domain model are the requirements. Only the Job/-Task process was added later, as its necessity was only found by the end of elaboration.

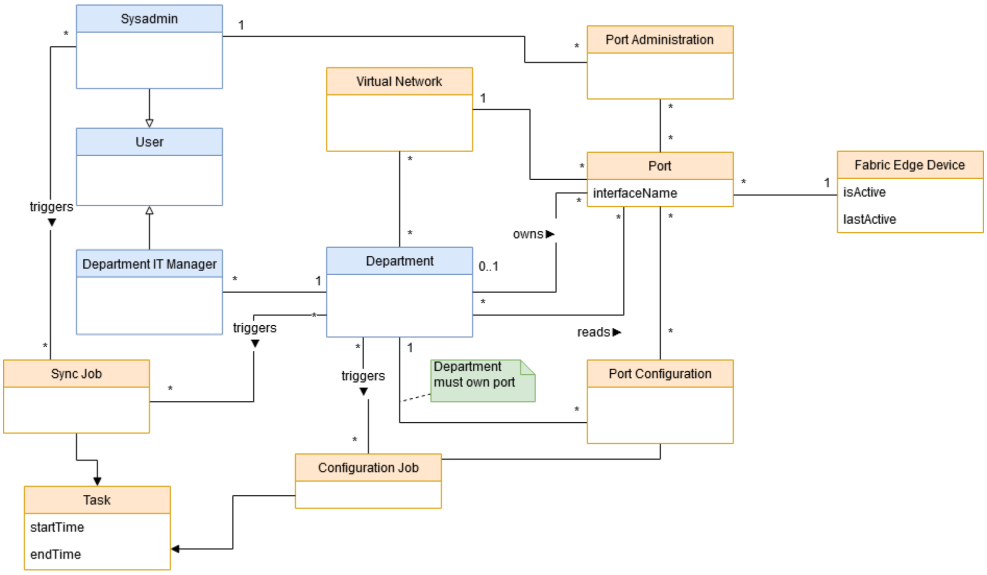


Figure 3.1: Domain model

3.3 Technologies

Additional to the external interfaces there are technologies that are given as a requirement, while some that have been assessed. This chapter gives an overview over the different technologies such as programming language and frameworks used.

3.3.1 Programming languages

The use of Python for the back end is required, or strongly recommended by the examiners since there's a good amount of preexisting knowledge in this language at the institute. As this is a language both project authors have some experience in, there is no need for further evaluation. The same applies to JavaScript/Typescript with the React framework for frontend development.

3.3.2 Frameworks

The frontend is based on the latest stable React release (16.12). The backend uses the latest Django LTS release (2.2) and the according Django REST (3.10) release.

3.3.3 Libraries overview

Following is a short overview of the most important libraries, which were used in front- and backend:

Frontend:

- **Axios** is used in place of polyfills fetch, as frontend boiler plate was not an issue and Axios improves browser support and adds auto JSON transformation.[3]
- **i18n** is used for translations. All translations are lazy loaded when loading the page. Translations on the frontend makes the app more responsive once loaded. While initial fetch time is an issue, but API calls impair responsiveness.[9]
- **Material UI** is used as main design guideline, as it comes with Typescript support, has less boiler plate than Bootstrap and is well received in the first MVP.[12]
- **Material Table** is used for every table, as it comes with builtin sort, data fetching capabilities and search.[11]
- **React Select** is used for drop down menus, based on experience in previous projects.[13]

Backend:

- **Django REST JWT** for JWT implementation
- **Gunicorn** for serving WSGI in production
- **Celery** as a worker solution
- **DNA Center SDK** for DNA center interaction
- **Dotenv** to enable Twelve factor compliant settings

3.4 Design overview

3.4.1 Frontend

The frontend is separated into multiple React components. By using small components their re usability is improved. Each component has its own life cycle and methods. To gain more stability Typescript is used.

Structure

The following overview shows how the components are organized:

Tenant Manager Web

```
 | src
 |
 |__ index.tsx
      |__ api
      |__ utils
      |__ resources
      |__ models
      |__ config
      |__ components
           |__ App.tsx
           |__ common
           |__ main
                |__ Main.tsx
                |__ MainStyles.tsx
```

Table 3.1: Frontend project structure

If a component has more than one file it is encapsulated in its own folder. Styles are separated from the main component to make the code more readable.

State management

The project relies only on native react state management. Additional libraries such as zustand, or react-redux are not used as these either showed no real benefits opposed to React Hooks with contexts or have not adapted to the new Hooks system yet.

3.4.2 Translation

Translations are configured on a string-matching basis. This makes development faster and messages in code can be used as a fallback (no additional code). Additionally i18N is wrapped in a suspense component, which shows a loading screen while updating translations.

Logging

To mitigate log messages in production a wrapper is used, which turns off messages on production. Logging of used components is set to only be enabled in the development environment.

3.4.3 Router

The setup of routing in frontend is designed so that configuration can be kept centrally. This has the advantage of being able to reuse routes between different views (user, admin). All routes except login are protected routes, this removes authentication state checks on components.

3.4.4 API

To interact with the backend API a separate layer is used. This improves re usability (components can reuse API features) and coupling. Backend messages are mapped and translated at the API layer level.

Models

Models are defined as interfaces (Interfaces over Types [16]) and also separated. Apart from the base model update models are defined to gain stronger typing. Mapping of backend data to a model is done directly in the API layer.

3.4.5 Backend

The following overview shows how backend components are seperated:

Tenant Manager Api

```
│  apps.api
│   ├── dnac
│   ├── migrations
│   ├── templates
│   ├── types
│   ├── models
│   ├── serializers
│   ├── utils
│   ├── views
│   └── urls.py
├── manage.py
├── config
├── tests
└── undocummented dnac client
```

Table 3.2: Backend project structure

Mostly a standard Django structure is used. Serializers and views are separated into multiple files and are organized in separate directories.

app.api The main Django app where most of the applications logic lives. The API is not separated into multiple apps to keep the structure simple.

types Contains custom enum types used by the database.

dnac Contains the core logic for any DNA center related tasks. Currently it consists of the sync and the host onboarding tasks.

undocumented dnac client A separate package for the custom written API client.

3.4.6 Layers

3.4.7 Layer diagram

Since Django REST framework is used the layering is largely given. The app can be split into 4 main layers:

Controller maps API calls to corresponding views.

Application handles API calls and does basic error handling.

Domain contains the main logic and data classes of the application. Business logic is mainly in the serializers while the data is represented in the models.

Utils contains utilities used by higher layers like communication with API endpoints.

Following diagram shows a simplified view of the layers and their most important components. To keep the diagram readable default Django packages used are excluded.

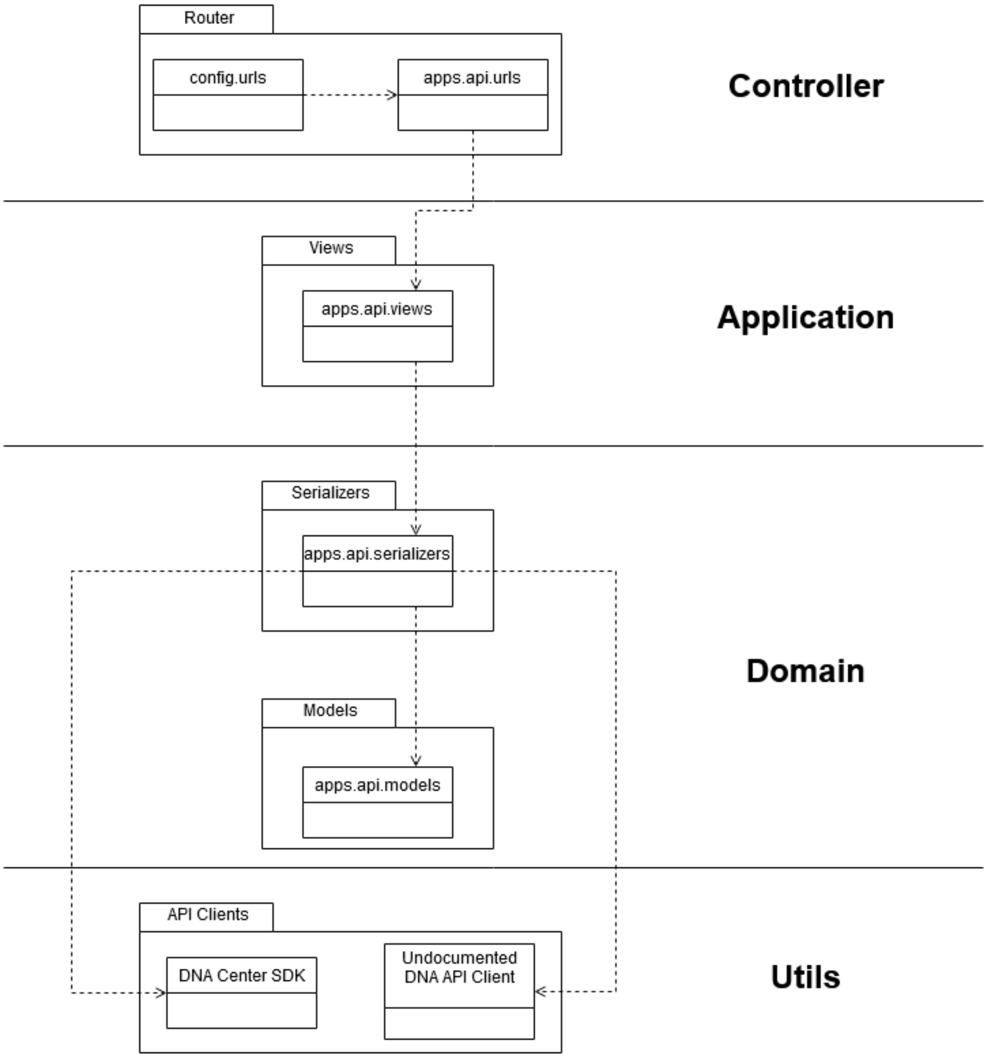


Figure 3.2: Backend layers

3.5 Containerization

The architecture and design specification is influenced on the configuration side by the NFR to use docker containers. The applications are designed to use environment variables for all configuration. The application is designed to be considered Twelve-Factor[15] compliant.

Chapter 4

Implementation

4.1 Authentication

In order to differentiate between different user roles (authorization), authentication is required. Django and Django Rest Framework already provide a solution for simple token- and session-based authentication. As the front- and backend are separated the HTTP-Token-based approach seemed useful, however this method is only appropriate for smaller apps and is not able to store additional user information.

Considering previous experience with JWT and the fact that front- and backend communication is based on JSON, JWT authentication was used based on the djangorestframework-jwt library. To get all user relevant information at once the JWT implementation was extended with a custom serializer.

4.2 Management

4.2.1 User and Groups

Djangos user and group management is one of the main reasons the framework has been considered very useful for this project. Unsolved were only permission related features that resulted in a customized user management: A role (network- or system administrator) and network objects (Ports, VNs, etc.) should be assignable to a user.

There are two methods to consider for assigning a role to a user:

- Use groups for roles and assign the user to the group
- Use an attribute for the role in the user model

The first approach is more dynamic, but adds more complexity. The second approach better aligns with the domain model, therefore attributes were used. As it is generally considered good practice, the Django User model was mapped to a custom Profile model referencing the User.

To assign a user to a network object, it has already been decided in the domain model to use groups (Departments in the domain model) to categorize permissions. As a benefit multiple users per Department are supported.s An assignment is defined trough simple group to network objects references.

4.2.2 Access Control

In order to fulfill the access control use case, an endpoint has been setup to assign access rights to groups. As a result, each group can be assigned to a device port either readonly or as an owner. Committing new values replaces existing assignments. This benefits the administrator in being able to mass-assign network objects.

Finally, to reduce frontend logic when mass-assigning device ports to groups, another endpoint has been created to reset port assignments for devices.

4.3 DNA Center API endpoints

Where possible official endpoints where used. The official documentation for those can be found on Cisco Devnet[5] or on the DNA Center web interface.

4.3.1 Official endpoints

The descriptions are taken from the official API documentation.

GET /dna/intent/api/v1/task/taskId Returns a task by specified id.

This endpoint is used to monitor tasks startet on the Platform. It returns information about the task including completion and failure info.

GET /dna/intent/api/v1/interface/network-device/deviceId/startIndex/recordsToReturn Returns the list of interfaces for the device for the specified range.

This endpoint returns basic information about all interfaces of a device. It does not include any information about host onboarding, for which an undocumented endpoint is used.

4.3.2 Undocumented DNA Center endpoints

As a result of rate limits and other issues a number of unofficial endpoints were used in the project. Following a list of those endpoints with short descriptions of what they do and why they where used:

GET /api/v2/data/customer-facing-service/DeviceInfo If used without parameters this endpoint returns all devices on the fabric including a "roles" field that marks edge devices.

If a URL parameter with the name id is given the configuration of a specific device including configured interfaces through the host onboarding feature can be queried by using the devices UUID.

PUT /api/v2/data/customer-facing-service/DeviceInfo Update all relevant host onboarding settings of a device in the fabric. The main drawback of this endpoint is that it requires all settings to be present. So all ports with all settings must be included in every call to this endpoint. It is recommended to use this specific endpoint with caution as wrong parameters have broken edge devices to a point where a reset was necessary while developing. It is best to use all data from the GET request and only replace the port configurations.

GET /api/v2/data/customer-facing-service/VirtualNetwork Lists all configured virtual networks with basic information. Currently there is no way to get a list of VNs through the official intent API so this endpoint has to be used.

GET /api/v2/data/customer-facing-service/virtualnetworkcontext Lists all configured virtual networks with less information about the VN itself but includes a list of all scalable groups associated with it.

GET /api/v2/data/customer-facing-service/scalablegroup/access Lists all scalable groups configured with all necessary information. Like before there is no official endpoint to achieve this.

GET /api/v2/data/customer-facing-service/Segment Lists all segments (IP Pools) configured. In the intent API only new segments can be created but not listed.

GET /api/v1/siteprofile?namespace=authentication Lists all authentication profiles configurable on the fabric. In the intent API they can again only be created and not listed.

4.4 DNA Center API clients

For interacting with the DNA Center APIs there are two clients required. For official endpoints an existing client can be used while for undocumented endpoints a new custom client is required.

4.4.1 DNA Center SDK

No official Python SDK for the API is released alonside DNA Center. However there are multiple community made clients available for the official API. Many of them are outdated and seem to have been inactive for a long time. The DNA Center SDK (dnacentersdk)[8] is the most frequently updated option. It is owned and maintained by the Enterprise Networking Business team at Cisco and gets updated as soon as there is an update on the platform.

4.4.2 Undocumented API client

For undocumented endpoints there is no existing client, so a simple client was developed to be used in this project. To make it reusable for other projects it is separated in a standalone package. It was decided to make the client as simple as possible, since the undocumented API can change at any time. For that reason no extensive test where written for it and relies mainly on system tests.

4.5 Workers

Interactions with the DNA Center API often take a long time. Not only because many requests are made and rate limiting might apply but also because POST/PUT requests to the API start tasks in the DNA Center itself. The same task API is used for documented and undocumented APIs. If this is done synchronously responses to the frontend can take a long time or even time out. This would effectively block user interaction. To solve this long tasks are processed using workers. For this Celery[4] is used as worker and Redis[14] for the message queue. Both are popular open source projects and default choices when implementing workers in a Python project. It is also the main recommendation in Two Scoops of Django for larger projects [1].

Due to the worker system a task endpoint and database entries are added to the API so the frontend can display and monitor ongoing tasks similar to the DNA Center web interface.

4.6 DNA Center Synchronization

One of the main architectural challenges is defining how the application synchronizes and updates topology changes on the fabric. As defined in the requirements changes made in DNA Center or on the devices directly overwrite any defined configuration in the app itself. This section explores possible solutions/approaches to this problem.

4.6.1 Getting all edge nodes

The first problem encountered was trying to get all edge nodes. There is no official endpoint that contains information about edge nodes. Possible solutions:

- Use two API calls (One to get the device and one to check, whether if its an edge device)
  This results in a slow sync for more than 5 devices because of rate limiting.
- Use tags in DNA Center (predefined by the DNA center administrator)
  This is as reliable as the DNA center admin is.
- Use defined roles in DNA Center (predefined by the DNA center administrator)
  This is as reliable as the DNA center admin is.
- Use the same undocumented API endpoint as the web interface.
  The main drawback of this approach is that no stability is guaranteed. The endpoint might change on a platform update.

Decision: It was decided with the examiner that the best method is using unofficial APIs as it is the most automated approach. In fact this approach is used for most data gathering as the Intent API does not offer options to list VNs, Scalable Groups, IP Pools and Authentication profiles.

4.6.2 Sync job

The synchronisation can be triggered by any user and is limited to the networking objects the user has permissions to. The site administrator can trigger a full sync of all objects. Any group can only run one sync job at a time, this way the risk of concurrency problems is minimized. The sync job takes a while and makes many calls to the DNA Center API, if a user could trigger multiple syncs at the same time this could lead to significant traffic and load of the appliance.

4.6.3 Sync flow

When a sync is triggered on the web portal the sync request is added to the message queue, and a running job object is returned. The worker will, when ready, start the synchronization job and write all updates to the database directly. When finished it also updates the job object in the database as finished or failed. There currently is no polling done by the fronted to check for job completion but a list of jobs a specific group has access to can be displayed. If a user tries to run a job while one is still active in their group an error message is shown.

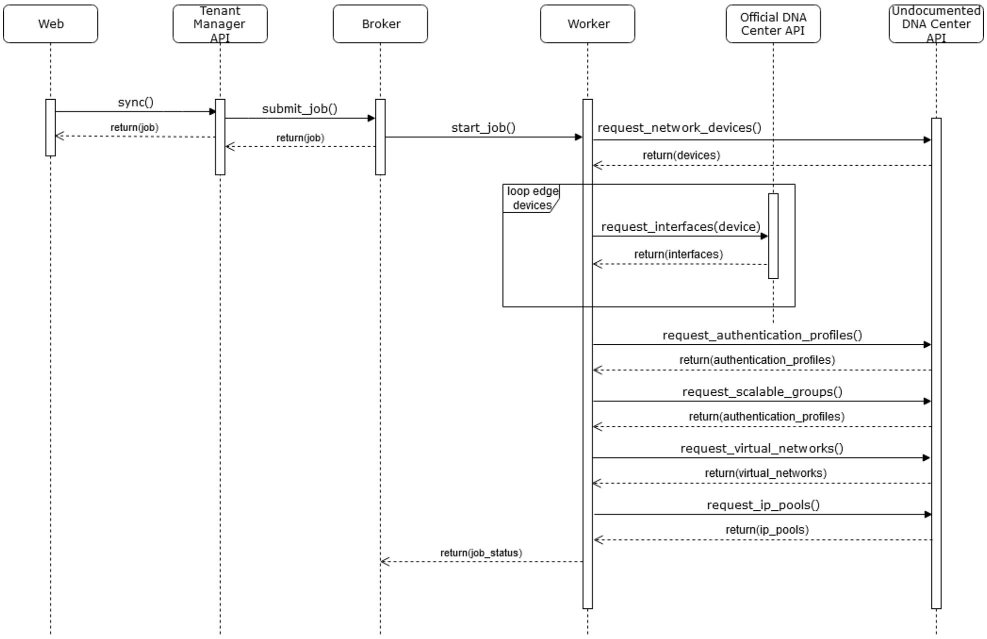


Figure 4.1: Sequence diagram of sync

4.7 Host onboarding

The task management for the host onboarding feature works in the same way as for the sync. Before a task is started multiple validations are done to ensure that the user who triggered the job has access permissions to all interfaces, VNs and Scalable groups used in the request. If the validation fails an error is returned and no job is started.

If the request is valid the job is started and handed to the message broker. The worker first sets each interfaces status to pending in the database, and a completion status when it is done. This ensures that the same interface can't be configured by two processes at the same time.

Before any configuration can be pushed old device configuration must be gotten. This must be done for every PUT to the device as multiple IDs change after every request.
There are multiple issues that can arise when directly updating interface configuration. For example an interface type (e.g. from user device to server) cant be changed, it must first be cleared and then changed. The web interface of DNA Center just displays an error message when trying to overwrite the type. Additionally when a port is already configured it has 2 additional ids that have to be added to the request. To make the configuration easier and more user friendly the port assignments are first cleared and only written to the device afterwards, because of this the worker first has to wait for completion of the interface clearing task before posting any new configuration.

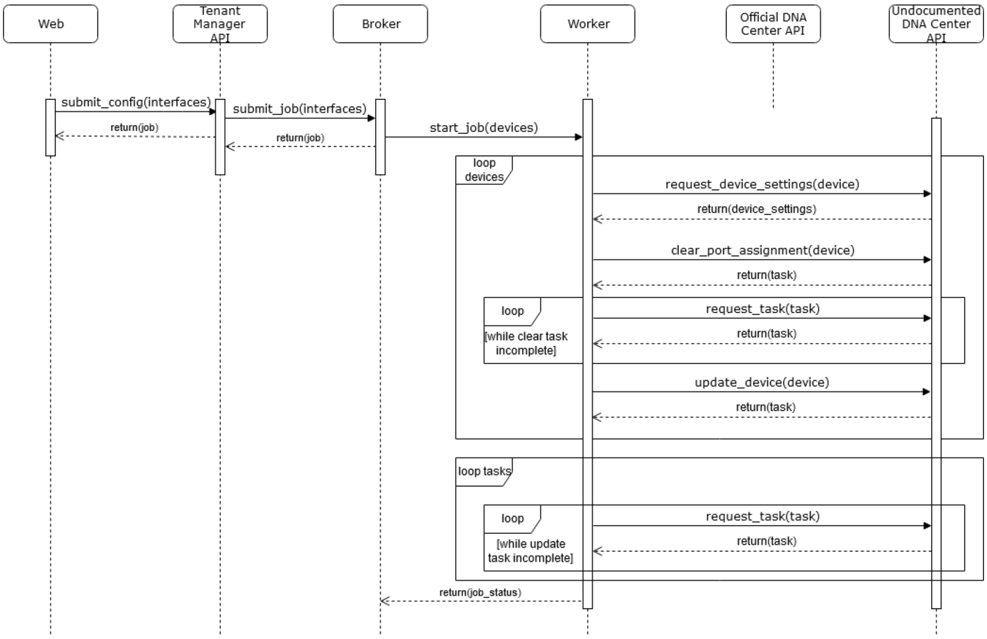


Figure 4.2: Sequence diagram of host onboarding

Chapter 5

Results

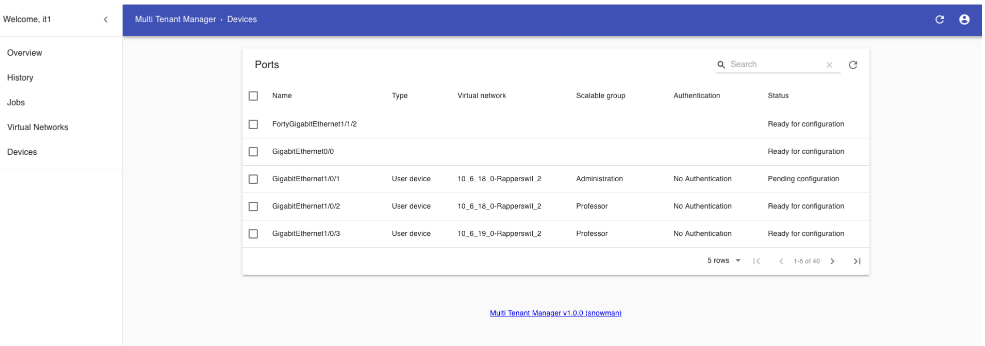
5.1 Achieved results

A web platform was created to allow for multi tenancy in the host onboarding feature. System administrators can create users, groups and grant groups permission not only for configuring interfaces but also assign specific IP pools to groups. All required features where implemented in sufficient quality for a prove of concept.

A client for interacting with undocumented endpoints on the DNA Center was built. Because most interactions with the DNA Center are time consuming a job system was build that uses workers to interact with DNA Center asynchronously. This system allows any user to monitor the status of any jobs they created.

The optional features interface grouping and CSV batch import of interface configurations could not be implemented because of time constraints.

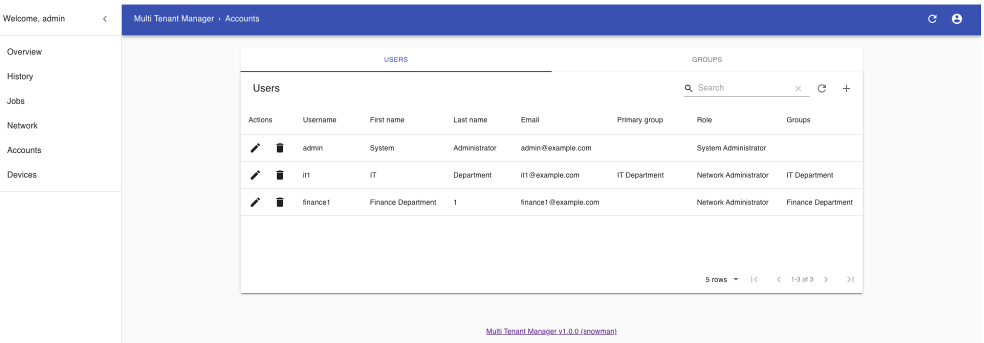
5.1.1 Screenshots



Ports

□	Name	Type	Virtual network	Scalable group	Authentication	Status
□	FortyGigabitEthernet1/1/2					Ready for configuration
□	GigabitEthernet0/0					Ready for configuration
□	GigabitEthernet1/0/1	User device	10_6_18_0-Rapperswil_2	Administration	No Authentication	Pending configuration
□	GigabitEthernet1/0/2	User device	10_6_18_0-Rapperswil_2	Professor	No Authentication	Ready for configuration
□	GigabitEthernet1/0/3	User device	10_6_19_0-Rapperswil_2	Professor	No Authentication	Ready for configuration

Figure 5.1: Screenshot of the port configuration view



Users

Actions	Username	First name	Last name	Email	Primary group	Role	Groups
	admin	System	Administrator	admin@example.com		System Administrator	
	it1	IT	Department	it1@example.com	IT Department	Network Administrator	IT Department
	finance1	Finance Department	1	finance1@example.com		Network Administrator	Finance Department

Figure 5.2: Screenshot of the user management view

5.2 Possible improvements

There are several improvements that can be made to get the application from a prove of concept state to a productive tool that can be used in a live environment.

Backend On the backend testing could certainly be improved by separating different layers with dependency injection and using more mocks in unit tests and generally writing more tests. Additionally the unofficial API client currently relies on system tests to ensure that endpoints have not changed after updating DNA Center. By creating automated tests to check for API stability the overall maintainability of the backend can be improved.

Frontend On the frontend no usability tests have been made because the focus was on delivering a working prove of concept and not an application for productive use. By conducting usability tests and getting feedback by users the overall quality of the frontend can be improved. Small possible UI changes like making lists scrollable have already been noted. Also there are multiple code quality changes that can be done to reduce code redundancy with type-mapping.

DNA Center API Many undocumented endpoints where used to achieved all required functionality. The DNA Center API should be monitored in case endpoints are changed or added so that undocumented endpoints can be replaced with official ones.

High loads During load testing it was discovered that the backend system does not scale well. When updating 500 entries the backend stops responding within an acceptable time. This could be solved by implementing a pagination mechanism.

5.3 Possible additional features

RADIUS support Currently an internal system is used for user and group management. For use in a productive system integration into third party authentication systems would be crucial. One possible specific system to support would be RADIUS.

Legacy switches An interesting feature to add would be support of host onboarding on legacy switches. With this feature switches that are not in the fabric could be configured which would be a significant advantage to the native DNA Center host onboarding.

Creation of virtual networks, IP pools and scalable groups In this version of the tool all relevant objects for host onboarding must be configured on the DNA Center frontend as a pre-requisite. If the creation of those objects could be done on the Multi Tenant Manager itself the system administrator would not need to switch between multiple tools.

Part II

Project documentation

Chapter 6

Deployment

6.1 Components

There are multiple physical and virtual components working together. Following is a small overview of all components:

- **Client/Browser**: The client loads frontend (React app) from server and communicates encrypted with the server
- **Server**: The server is a virtual machine, which currently runs on the infrastructure of the INS. Ubuntu 18.04 is used as an operating system and deployed trough Ansible roles (fully automated deployment).
- **Reverse Proxy**: Responsible for communication against the outside world. Acts as MITM/proxy for the docker containers. Two virtualhosts are configured, which forward requests to the right containers. The reverse proxy is configured for TLS encryption and also forces redirection to encrypted communication.
- **Docker**: Docker runs on the server for Container services
- **Containers**
  - **Web**: Runs Reacts development web server
  - **Api**: Runs Django development/built-in web server and Celeries worker process
  - **Db**: Runs latest Postgres database server
  - **Redis**: Runs latest Redis broker server

Production Deployment has additional and some changed components:

- **Containers**
  - **Proxy**: Runs a production ready (sane config values for production-use) Reverse proxy for other containers.
  - **Web**: Runs an Nginx webserver for serving the production build of React App.
  - **Api**: Runs Django with Gunicorn and production config. Uses Whitenoise for serving static files.

- **Optional: Web Application Firewall (WAF)**: In production environments its best practice to use a WAF for filtering web traffic between the internet and application servers.

6.2 Diagrams

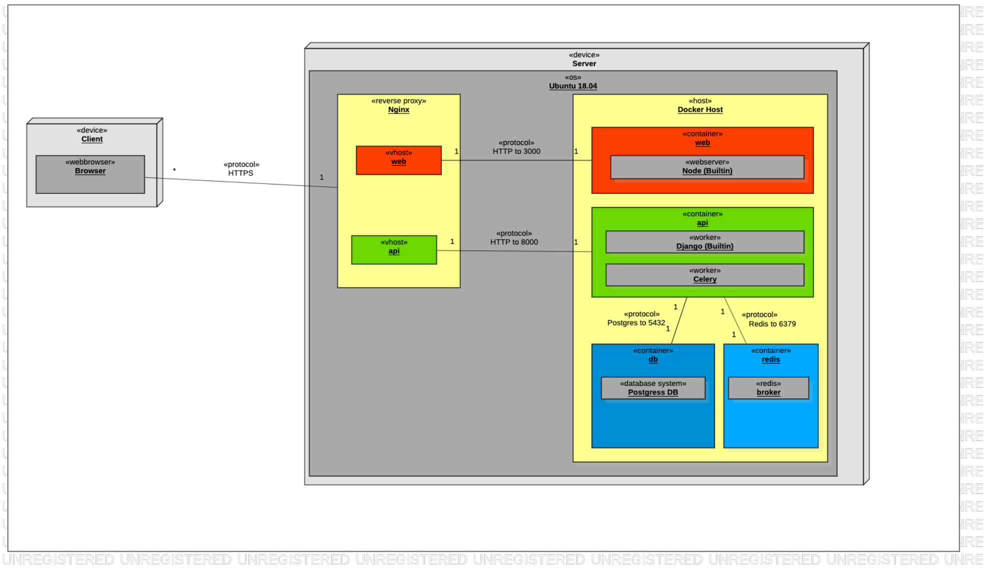


Figure 6.1: Deployment Diagram in Development

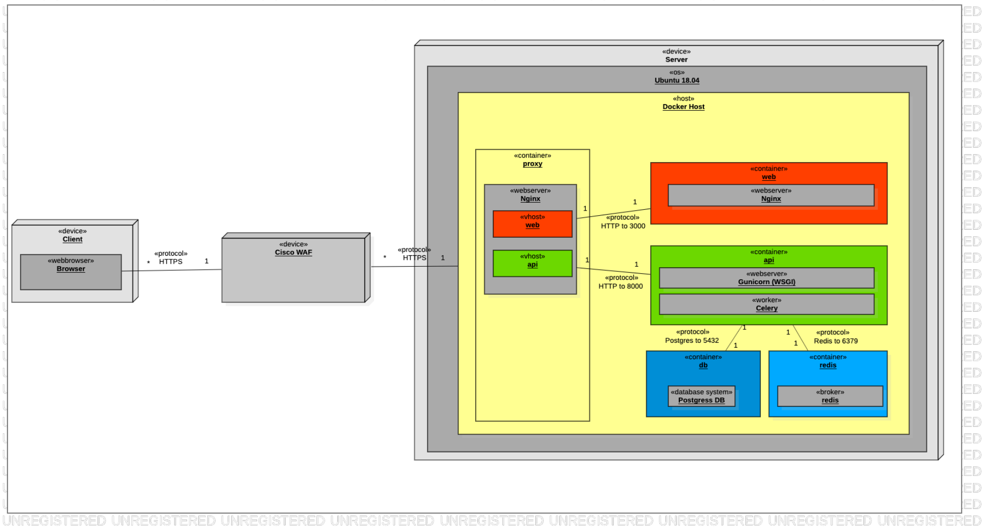


Figure 6.2: Deployment Diagram in Production

6.3 Infrastructure

6.3.1 Repositories

The project is hosted at Gitlab[1] and split into the following repositories:

- **web**: Frontend code
- **api**: Backend code
- **thesis**: Mirror of current documentation hosted at Overleaf
- **docs**: Repository for additional documentation such as drawing, projectplan, domain model etc.
- **setup**: Installation manual, Docker Compose config for development and production
- **infrastructure**: Ansible roles for VM setup and Dockerfiles for ci builder images
- **load-test**: Python scripts for load testing

[1] https://gitlab.com/cisco-dna-center-multi-tenant-manager

6.3.2 Continuous Integration

As previously mentioned, projects deployment is based on multiple repositories. By using Ansible, the server development server, can be deployed automatically. Built images are stored in Gitlab's Container registry.

The following list briefly describes a usual code deployment cycle:

1. Run Tests (Linter, Unit, Integration, Sonar Gate) for API/Web
2. Build docker images for API/Web and push them to Gitlab's container registry and Sync images to the setup project
3. Deploy images to the VM trough docker-compose
   Two compose files are provided: One for development and one for production-use.
   Development uses a special debugging configuration and no reverse proxy.
   The production version is provided for future installation trough customers and to fullfill NFRs.

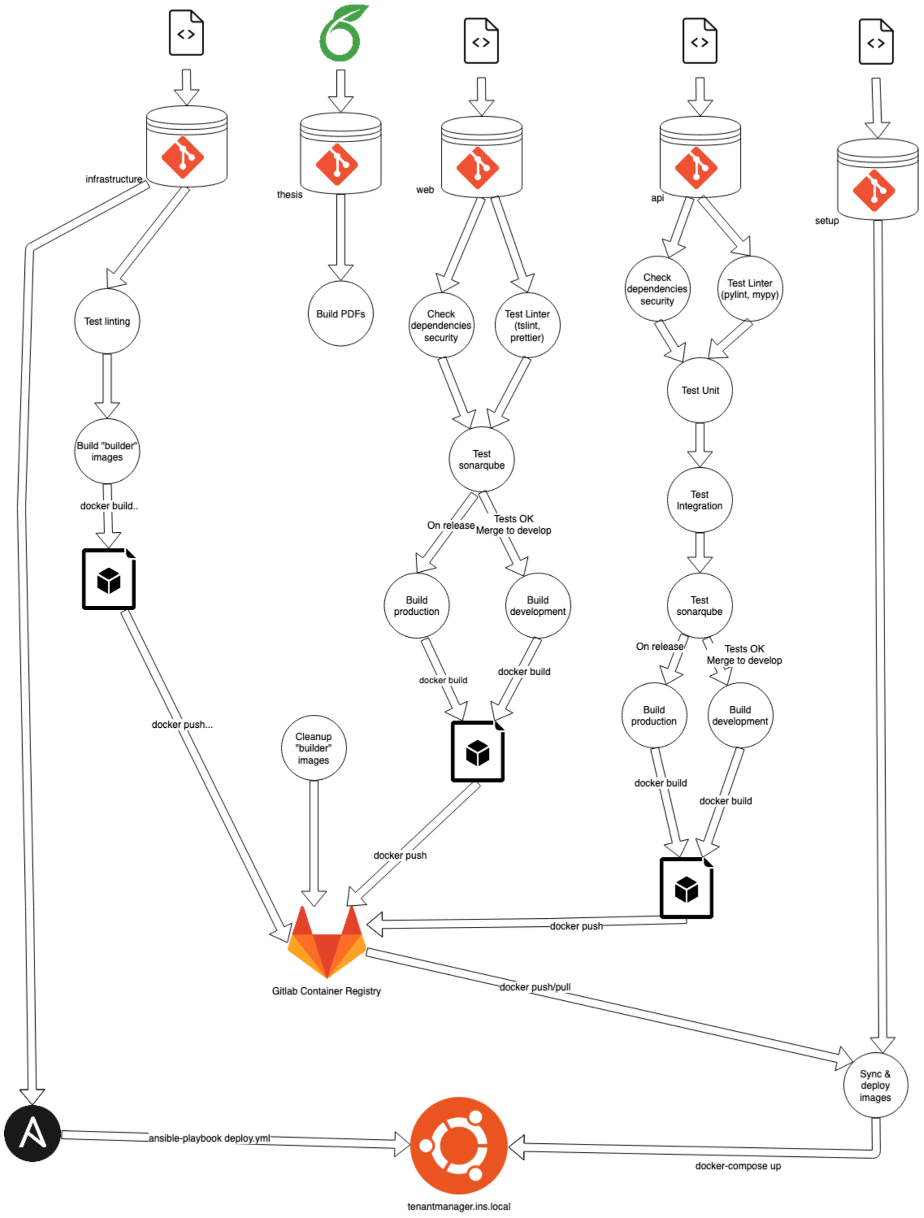


Figure 6.3: CI/CD

Chapter 7

Use cases fully dressed

7.1 Introduction

The fully dressed use cases are used for system tests and as a basis for the tickets in the project planning. The previously defined use cases in brief descriptions and actors are the basis for the fully dressed use cases. As many features changed during the initial planning stage the fully dressed use cases where rewritten after the elaboration stage for implemented features only.

7.2 Use cases

Name	UC01: List network devices
Description	A department IT manager views a list with an overview of of all network appliances they have read access to.
Actors	Department IT manager
Precondition	• User must be logged in • Devices must be synced • User has ownership of at least one port
Postcondition	• All devices the user has ownership of is displayed
Success Story	1. Navigate to the devices tab
Extensions	-

Table 7.1: UC01: List network devices

Name	UC02: Grant and revoke access
Description	A sysadmin grants or revokes a specific group read/write access to a specific switch/port
Actors	System admin
Precondition	• User must be logged in • User must be admin
Postcondition	• Users in group can view/edit specific interfaces • After refreshing assignment is displayed on interface overview
Success Story	1. Select multiple ports 2. Open assignment popup 3. Select group to assign 4. Save assignment
Extensions	• Remove assignment • Edit assignment

Table 7.2: UC02: Grant and revoke access

Name	UC03: CRUD users and groups
Description	A sysadmin creates/updates/deletes users and groups and adds users to groups.
Actors	System admin
Precondition	• User must be logged in • User must be admin
Postcondition	• Users are assigned a group • User account is created/removed
Success Story	1. Create an account 2. Create a group 3. Assign a group to account
Extensions	• Edit user group • Edit group • Delete user • Delete group

Table 7.3: UC03: CRUD users and groups

Name	UC04: View changelog
Description	Any user views a history of changes made on a specific switch they have access too.
Actors	System admin, Department IT manager
Precondition	• User must be logged in
Postcondition	• An interface configuration has been made in the users group
Success Story	1. History of own group is displayed, if logged in as admin all history is shown
Extensions	• Filter by group as admin

Table 7.4: UC04: View changelog

Name	UC05: Configure port
Description	A department IT manager sets different preconfigured port settings like port type and authentication method.
Actors	Department IT manager
Precondition	• User must be logged in • User must have ownership of multiple interfaces
Postcondition	• The interface configuration is saved • A job to deploy the settings on DNA Center is created
Success Story	1. Select multiple interfaces in the device overview 2. Set port type, IP Pool, scalable group and authentication method
Extensions	

Table 7.5: UC05: Configure port

Name	UC06: Trigger DNA center sync
Description	Any authenticated user manually triggers a sync to update all data of the application with the current state from the DNA center.
Actors	System admin, Department IT manager
Precondition	• User must be logged in
Postcondition	• A sync job is created • The user is forwarded to the job overview page • All objects from the DNA Center are updated to the DNA Centers current state after the job is finished
Success Story	1. Trigger a sync using the sync button
Extensions	

Table 7.6: UC06: Trigger DNA center sync

Chapter 8

Data model

The database was defined using an application first strategy, meaning that the database is created using the defined Django models. Upgrades/migrations to the database are done using Djangos migration mechanism. Following class was automatically generated using the Django extensions library.

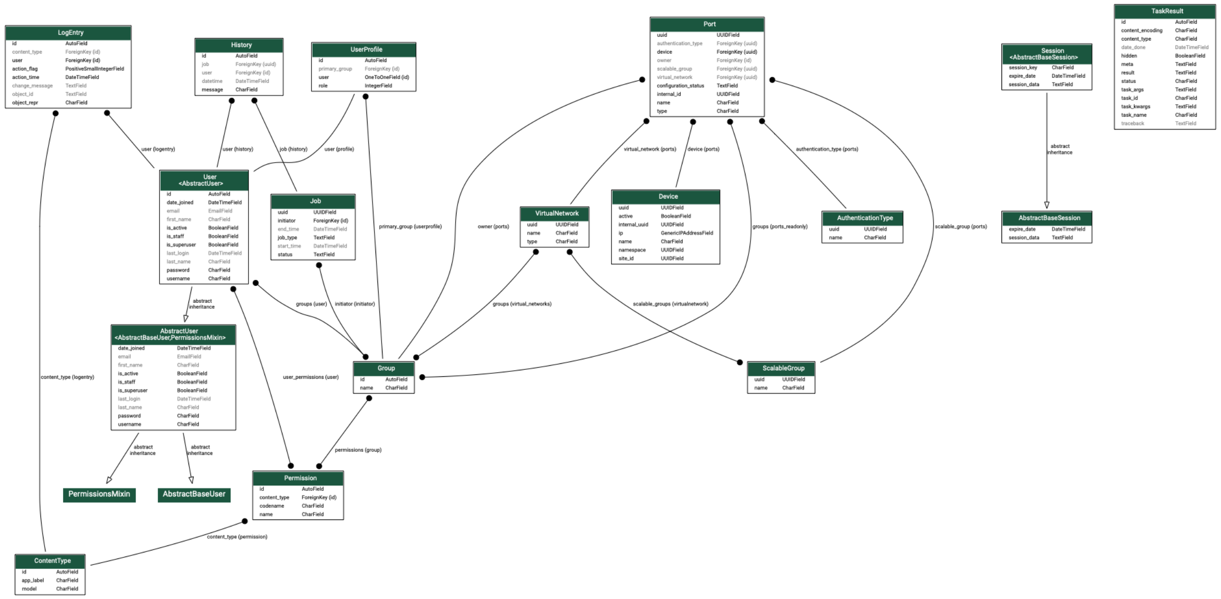


Figure 8.1: Class diagram

Chapter 9

API

9.1 Endpoints

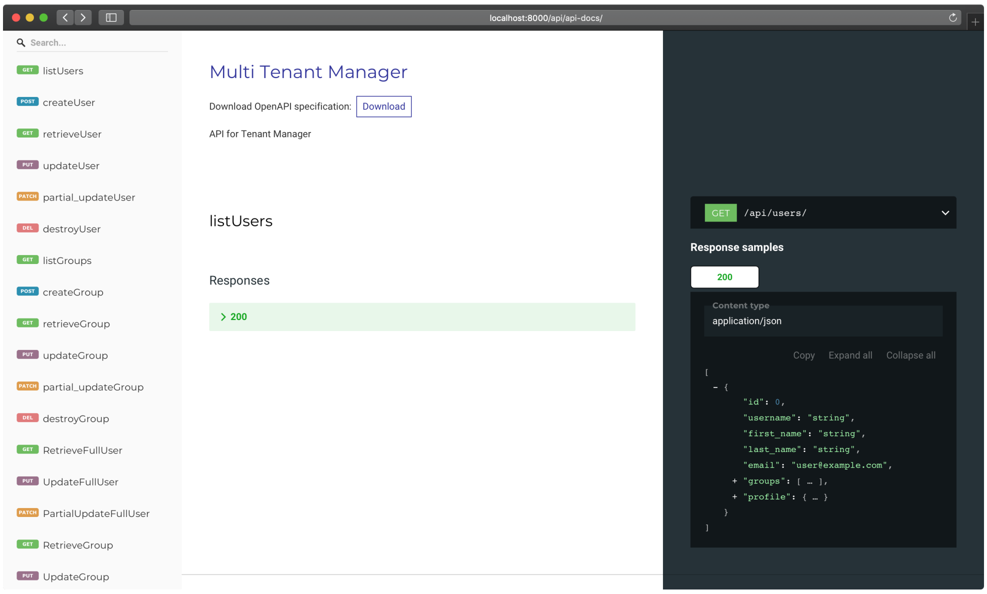
Following is a short overview of the most important endpoints at root level:

- **/users** User management (CRUD) for admins
- **/user** Change of user preferences for each user himself
- **/groups** Group management (CRUD) and access right management for admins
- **/authentication** Authentication, refresh and verify of JWT
- **/scalable-groups** Retrieval of available scalable groups
- **/virtual-networks** Retrieval of available virtual networks,
- **/authentication-types** Retrieval of available authentication-types
- **/devices** Retrieval of available devices and reset of access rights
- **/jobs** Retrieval of jobs
- **/sync** Trigger of sync jobs
- **/port-configuration** Configure assigned ports
- **/history** Retrieval of history
- **/admin** Administration of the Database (useful for development purposes)

All API functions are filtered by the users permission.

9.2 Detailed documentation

A detailed API documentation is created using ReDoc and OpenAPI. This documentation includes schemas for requests and replies for all available endpoints. The latest API documentation can be retrieved from the backend application under http://BASE-URL/api-docs.



Multi Tenant Manager

Download OpenAPI specification: Download

API for Tenant Manager

listUsers

Responses

> 200

GET /api/users/

Response samples

200

Content type
application/json

```
[
 - {
      "id": 0,
      "username": "string",
      "first_name": "string",
      "last_name": "string",
      "email": "user@example.com",
    + "groups": [ … ],
    + "profile": { … }
   }
]
```

Figure 9.1: API documentation

List of Figures

List of Tables

Glossary

API Application Programming Interface. 2, 4, 5, 8, 10, 16, 53, *see* Application Programming Interface

Application Programming Interface Programming interface in software. Usable and extendable by other software components. 2

CI Continuous Integration. 22, 53, *see* Continuous Integration

Cisco Digital Network Architecture An overlay based SDN solution by Cisco Systems. 2

Continuous Integration Automated build environment. 22

DNA Cisco Digital Network Architecture. 2–4, 53, *see* Cisco Digital Network Architecture

HSR University of Applied Science Rapperswil. 21

Mixed Scrum Project management process based on Scrum mixed with the Unified Process model. 18

NFR None-functional Requirements. 5, 6, 8, 52, 53, *see* None-functional Requirements

PoC Proof of Concept. 3, 4, 18, 24, 53, *see* Proof of Concept

Proof of Concept Minimal product to proof a theory/concept. 4

SDN Software defined networking. 2, 10, 53, *see* Software defined networking

Software defined networking Technology that enables network automation through programming. 2

VCS Version Control System. 22, 54, *see* Version Control System

Version Control System System for versioning source files. 22

Bibliography

[1] Audrey Roy Greenfeld Daniel Roy Greenfeld. *Two Scoops of Django 1.11. Best Practices for the Django Web Framework*. Two Scoops Press, 2017.

[2] Prof. Laurent Metzger. "CN2 Lecture 9 - Software Defined Access v1.0". 2019.

[3] *Axios*. URL: `https://github.com/axios/axios`.

[4] *Celery Worker*. URL: `https://docs.celeryproject.org/en/latest/userguide/workers.html` (visited on 12/12/2019).

[5] *Cisco Devnet Intent API Documentation*. URL: `https://developer.cisco.com/docs/dna-center/api/1-3-1-x/` (visited on 12/12/2019).

[6] *Cisco DNA Center Datasheet*. URL: `https://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/dna-center/nb-06-dna-center-data-sheet-cte-en.html#CiscoDNACenter1310appliancescaleandhardware` (visited on 03/10/2019).

[7] *Cisco DNA Center overview*. URL: `https://developer.cisco.com/docs/dna-center/` (visited on 03/10/2019).

[8] *DNA Center SDK*. URL: `https://github.com/cisco-en-programmability/dnacentersdk` (visited on 12/12/2019).

[9] *i18n*. URL: `https://www.npmjs.com/package/i18n`.

[10] *ISO/IEC 9126*. URL: `https://en.wikipedia.org/wiki/ISO/IEC_9126` (visited on 03/10/2019).

[11] *Material Table*. URL: `https://material-table.com/#/`.

[12] *Material UI*. URL: `https://material-ui.com/`.

[13] *React Select*. URL: `https://react-select.com/home`.

[14] *Redis*. URL: `https://redis.io/` (visited on 12/12/2019).

[15] *Twelve-Factor*. URL: `https://12factor.net/` (visited on 12/12/2019).

[16] *Typescript Interfaces over Types.* URL: https://www.typescriptlang.org/docs/handbook/advanced-types.html#interfaces-vs-type-aliases.

Part III

Appendix

Appendix A

Personal reports

A.1 Aaron Meier

At the beginning of the project I was not sure if the product is to network heavy as software development is the discipline I want to focus on in my studies at the HSR. It was a great surprise that we where able to mainly focus on software engineering while working with networks as an abstract concept by using APIs.
Learning about DNA Center was more interesting than expected, I have already worked with enterprise networks but never worked with an infrastructure based on a Software Defined Networking approach.
It was a pleasure working with the INS and Cisco, issues where addressed quickly and we had the opportunity to adapt our project goals dynamically when there where unexpected technical limitations. I greatly look forward to writing the bachelor thesis.

A.2 Dennis Ligtenberg

As someone with a software engineering background this project was interesting for me from the very beginning as it combines the world of software engineering with modern networking. Instead of just writing some custom software we had the ability to write software that interacts with a special technical subject matter. Because of my work at Cisco I have obviously already heard and looked at the DNA Center before starting this project but never got to do a deep dive into it and especially its APIs. At first understanding many of the new concepts like Virtual Networks and scalable groups was challenging as I have only been taught "old school" networking at the network related courses I have taken at the HSR. This project was a great chance to learn a lot of new concepts that I would otherwise not have encountered.

The project naturally wasn't without its pitfalls. Reverse engineering undocumented API endpoints that at first glance have seemingly random behaviour could get frustrating. But one thing that the problems that arised taught me is that an agile approach to project management is great for projects where the subject matter is largely unknown. Thanks to this we where able to quickly adapt and change our project and planning to handle problems.

The work with all of the involved parties was one of the highlights of this project. Most meeting felt positive and productive while problems where always addressed quickly. No meeting ever left a sour taste in one's mouth but always ended on a positive note. Because of this and the interesting product we are working on I am really looking forward to continuing working together during the bachelor thesis.

Appendix B

Testing

Trough projects development cycle, it has been defined that testing is focused on the app and no external systems, such as DNA Center.

B.1 Backend

Unit tests are used for general code coverage. For integration Sunny Case tests are used (test only normal use cases). As the projects scope is considered more PoC oriented, this scenario allows more development time on features. Testing heavily relies on PyTest[1] and Djangos respective Django REST TestCases classes. As mentioned in Two Scoops of Django[1, P. 305] tests should cover everything except parts of the software that already are covered trough Django core and third-party packages.

B.1.1 Results

There were 38 integration tests and Unit tests defined which results in a 70% code coverage.

```
---------- coverage: platform darwin, python 3.7.5-final-0 -----------
Coverage HTML written to dir htmlcov
Coverage XML written to file coverage.xml

38 passed in 59.86s
```

Listing B.1: PyTest results

[1] https://pytest.org/

B.2 Frontend

As initially defined by the project partner, the projects scope relies on functionality over design. Therefore it was agreed to not unit/e2e test the frontend application and solely rely on system tests.

B.3 NFR Validation

In order to check NFR, following is a checklist for completed NFR:

#	Description and acceptance criteria	State	Notes
NFR1	Only authenticated users should have access to the system. Is authentication required?	Accepted	Satisfied trough Djangos authentication backend, JWT implementation, permissions and testing
NFR2	Passwords should be protected trough modern measures. Are all passwords stored securely by default (Hash + Salt)?	Yes	Satisfied trough Djangos authentication backend
NFR3	Communication between the user and the system should be encrypted. Is the latest TLS-encryption supported?	Yes	Satisfied trough the reverse proxy ssl configuration
NFR4	On failed sync the app should use fallback data and indicate outdated data with an error If sync fails is there an error message suggesting that outdated data is used?	Yes	Inplementen on back and frontend
NFR5	Failure of configuration changes should be traceable for administrators. Are configuration failures obvious in the systems log?	Yes	Visible in system logs
NFR6	The system should respond without a noticeable delay (=total of 10 seconds between request and response) under full workload (=100 simultaneous requests) for the login operation. Was the response time goal tested trough a load test?	Yes	Proven in load testing, returns an error on large loads
NFR7	For future maintainability the system should depend on Ciscos intent APIs. Are all API calls made to intent based endpoints?	No	Because of technical limitations undocumented endpoints had to be used. Workaround was accepted by the examiner.

NFR8	The system should scale well on a typical campus networks growth with a maximum of 5,000 devices, a total of 480,000 ports, 256 VNs (based on Cisco DNAs limitation [6]). This means while the initial fetch of the above amount of information from DNA Center is running, response time of the application does not exceed 10s (time to display a loading screen/error message or the result). Was there a test made for the above scenario to validate the applications responsiveness?	Yes	Is covered by replying with an error before timeout
NFR9	The systems installation should be straight forward and deployable trough Docker containers. Are containers defined for all software components?	Yes	All components are in a container and deployable with a few easy steps

Table B.1: NFR Checklist

B.4 Performance tests

In order to full fill the NFR8 (System should scale well on typical campus networks), load tests have been created to check whether the application needs additional changes.

B.4.1 Scenario

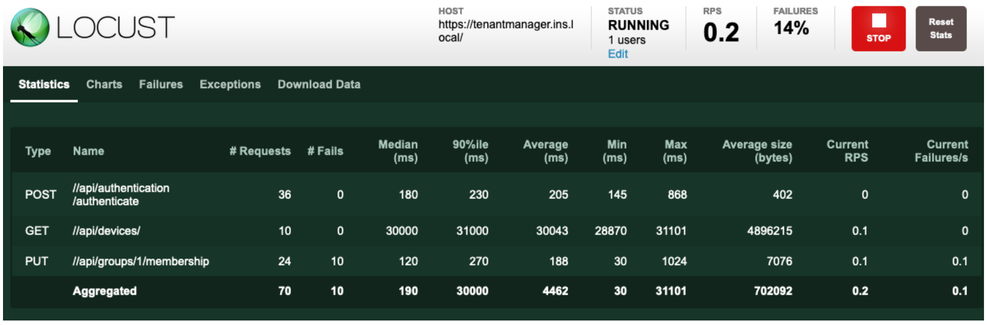
As a testing scenario the assignment of device port owner rights to a group has been chosen, as this is one of the more resource intensive task, that does not rely on DNA center (external systems) and can be tested with sample data.

- **POST authentication**: Authenticate as system administrator
- **GET devices**: Load all available device ports
- **POST group membership**: Set group as owner on all device ports
- **POST group membership**: Remove group as owner on all device ports

The definition of maximum devices/ports, based on DNA centers limitations is 5′000 devices with 480′000 ports. As reasonable load testing values first a batch of 120′000 ports has been tested (5′000 devices with each 26 ports minus 2 uplinks).

B.4.2 Results

At first there was a bottleneck because SQLite is used in development, this leads to a I/O speed limitation. With SQLite it was not even possible to generate and save 5′000 devices in nominal time.
By switching to a Postgres database on the development environment, sample data could be generated. Unfortunately, loading the amount of devices information timed out (response time > 60s).
Finally, the amount of sample data was reduced to 500 devices or 12000 device ports and fetching devices was successfully.



HOST https://tenantmanager.ins.local/

STATUS RUNNING 1 users

RPS 0.2

FAILURES 14%

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
POST	//api/authentication /authenticate	36	0	180	230	205	145	868	402	0	0
GET	//api/devices/	10	0	30000	31000	30043	28870	31101	4896215	0.1	0
PUT	//api/groups/1/membership	24	10	120	270	188	30	1024	7076	0.1	0.1
	Aggregated	**70**	**10**	**190**	**30000**	**4462**	**30**	**31101**	**702092**	**0.2**	**0.1**

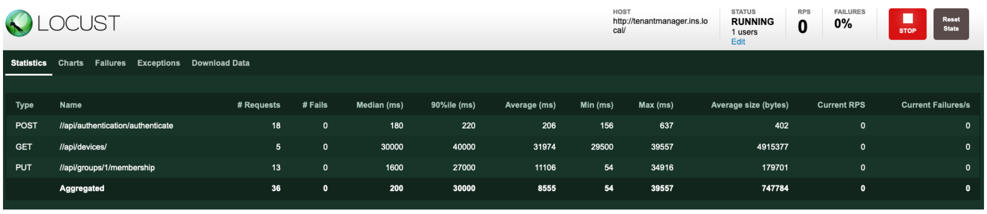
Figure B.1: Load test results

Another issue is that assigning 12000 ports is not allowed by Django (Limit is 1000), therefore assigning a new port membership fails with the following message:

```
exception The number of GET/POST parameters exceeded
    settings.DATA_UPLOAD_MAX_NUMBER_FIELDS.
```

Listing B.2: Load test exception

This was solved by increasing `DATA_UPLOAD_MAX_NUMBER_FIELDS` to 100'000.



HOST http://tenantmanager.ins.local/

STATUS RUNNING 1 users

RPS 0

FAILURES 0%

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
POST	//api/authentication/authenticate	18	0	180	220	206	156	637	402	0	0
GET	//api/devices/	5	0	30000	40000	31974	29500	39557	4915377	0	0
PUT	//api/groups/1/membership	13	0	1600	27000	11106	54	34916	179701	0	0
	Aggregated	**36**	**0**	**200**	**30000**	**8555**	**54**	**39557**	**747784**	**0**	**0**

Figure B.2: Load test results without Django limits

This result shows that, membership assignment of 12'000 device ports is indeed possible, but fetching the same amount of data exceeds the predefined maximum response time of 10 seconds. A possible solution would be to introduce pagination[2] in the relevant endpoints. However as predefined, the NFR is still in its acceptance area, as an error message is displayed in the frontend suggesting an invalid response.

[2] https://www.django-rest-framework.org/api-guide/pagination/

B.5 System tests

B.5.1 Scenarios

The following scenarios

#	Use case	Description
Test 1	List network devices	A department IT manager views a list with an overview of of all network appliances they have read access to.
Test 2	Grant and revoke access	A sysadmin grants or revokes a specific group read/write access to a specific switch/port
Test 3	CRUD users and groups	A sysadmin creates/updates/deletes users and groups and adds users to groups.
Test 4	View changelog	Any user views a history of changes made on a specific switch they have access too.
Test 5	Configure port	A department IT manager sets different preconfigured port settings like port type and authentication method.
Test 6	Trigger DNA center sync	Any authenticated user manually triggers a sync to update all data of the application with the current state from the DNA center.

Table B.2: System test scenarios

B.5.2 Test logs

This is the test log of the most recent system tests. The system tests where run whenever a feature was finished or improved.

#	Use case	Accepted	Notes
Test 1	List network devices	Yes	None
Test 2	Grant and revoke access	Yes	None
Test 3	CRUD users and groups	Yes	None
Test 4	View changelog	Yes	A bit simple, could contain more information.

Test 5	Configure port	Yes	None, works with the same limitations as in DNA Center.
Test 6	Trigger DNA center sync	Yes	None

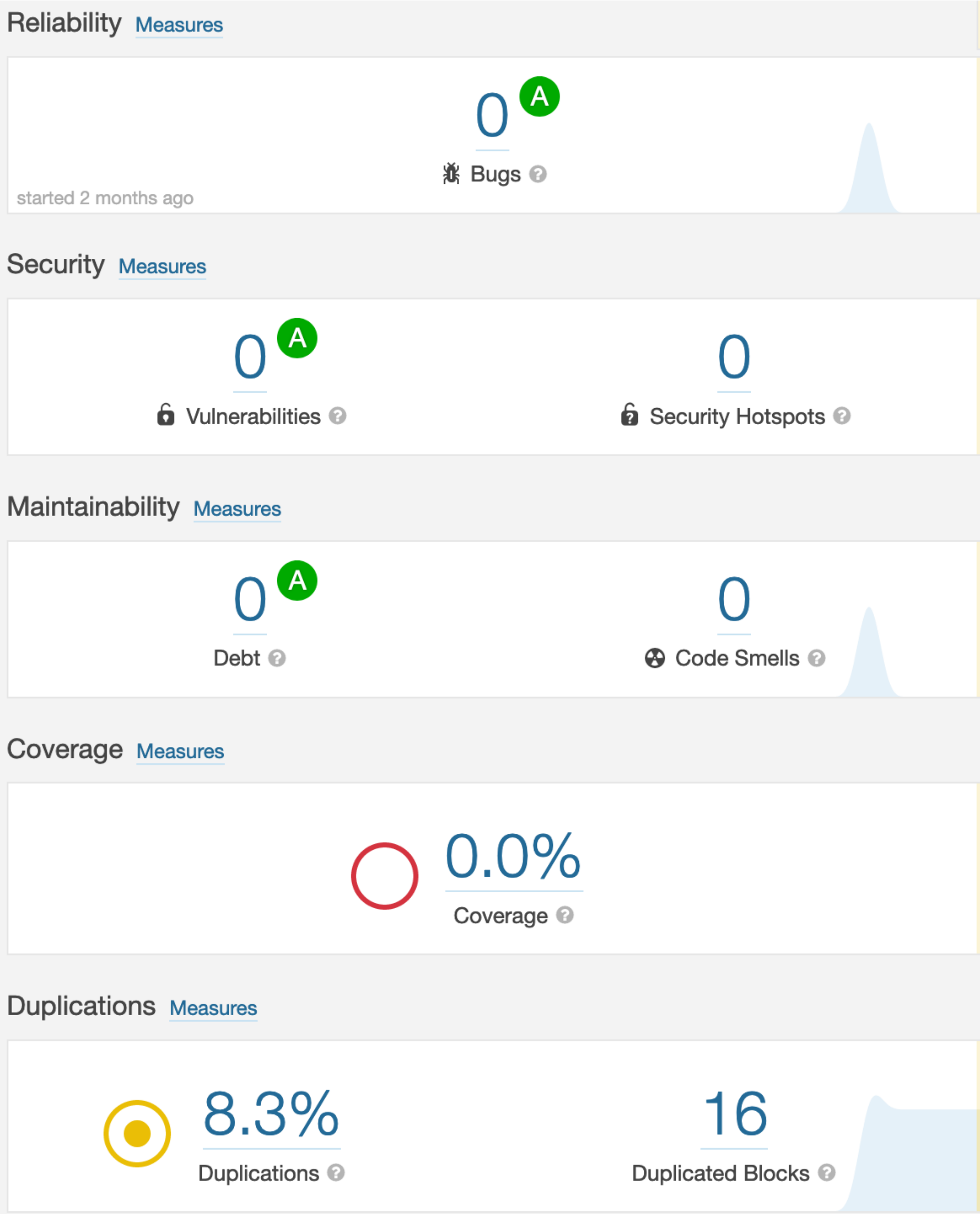
Table B.3: System test scenarios

Appendix C

Metrics

C.1 Sonarqube reports

C.1.1 Frontend



Reliability Measures

0 A

Bugs

started 2 months ago

Security Measures

0 A

Vulnerabilities

0

Security Hotspots

Maintainability Measures

0 A

Debt

0

Code Smells

Coverage Measures

0.0%

Coverage

Duplications Measures

8.3%

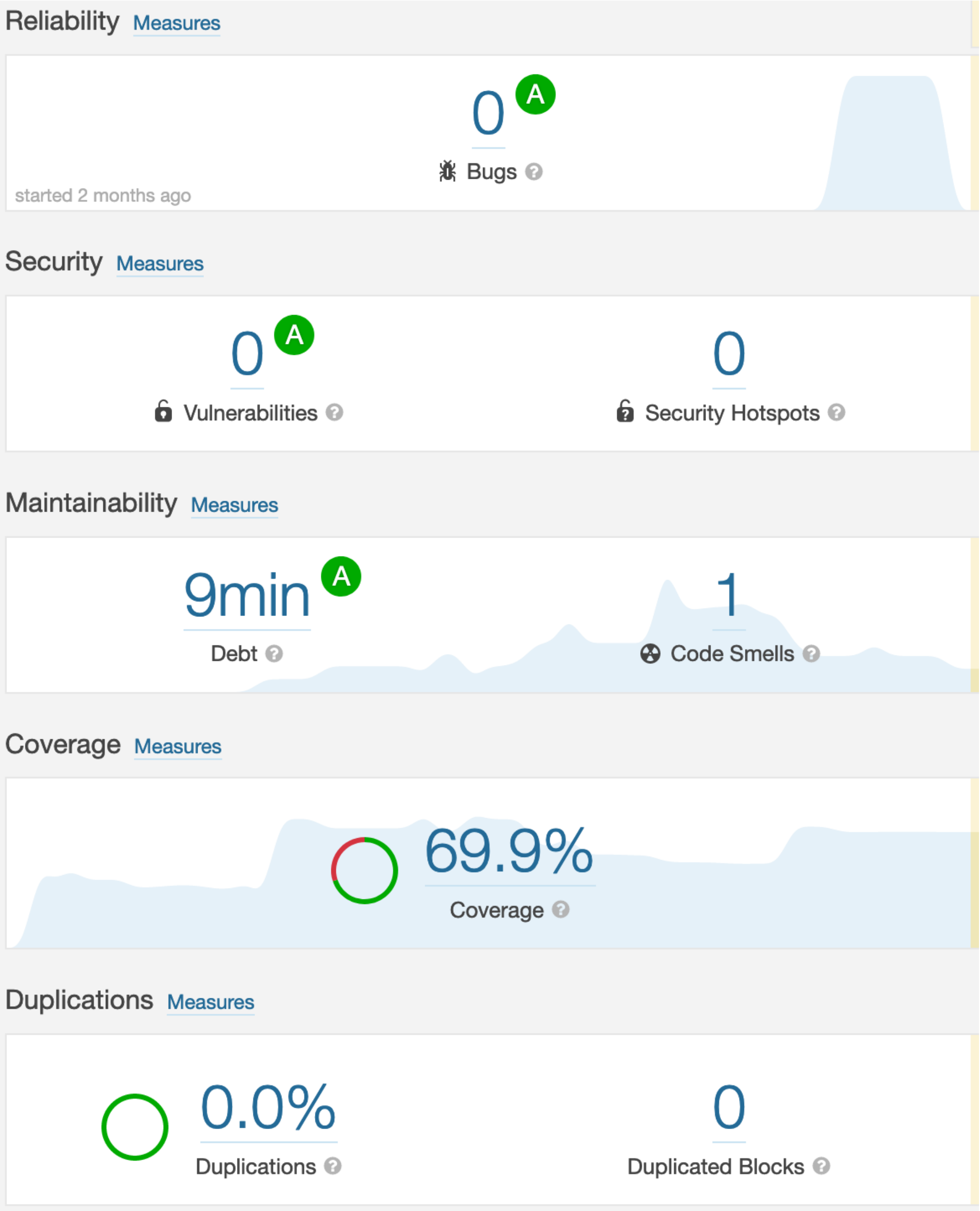
Duplications

16

Duplicated Blocks

Figure C.1: Sonarqube web

C.1.2 Backend



Reliability Measures

0 A

Bugs

started 2 months ago

Security Measures

0 A

Vulnerabilities

0

Security Hotspots

Maintainability Measures

9min A

Debt

1

Code Smells

Coverage Measures

69.9%

Coverage

Duplications Measures

0.0%

Duplications

0

Duplicated Blocks

Figure C.2: Sonarqube api

The achieved metrics are within the defined acceptable limits. The single code smell on the backend is due to cyclic complexity caused by processing the data passed from the DNA Center API and can not easily be resolved.

C.2 Code

Following is an overview of Lines of code (calculated with cloc) of the main code bases.

C.2.1 Backend

```
------------------------------------------------------------------------------
Language                  files        blank       comment         code
------------------------------------------------------------------------------
Python                       31          341            60         1240
HTML                          1            0             2           19
------------------------------------------------------------------------------
SUM:                         32          341            62         1259
------------------------------------------------------------------------------
```

Listing C.1: App lines of code

```
------------------------------------------------------------------------------
Language                  files        blank       comment         code
------------------------------------------------------------------------------
Python                       11           52            24          200
------------------------------------------------------------------------------
SUM:                         11           52            24          200
------------------------------------------------------------------------------
```

Listing C.2: DNAC client lines of code

```
------------------------------------------------------------------------------
Language                  files        blank       comment         code
------------------------------------------------------------------------------
Python                        6          119             3          484
------------------------------------------------------------------------------
SUM:                          6          119             3          484
------------------------------------------------------------------------------
```

Listing C.3: Tests lines of code

C.2.2 Frontend

```
------------------------------------------------------------------------------
Language                  files        blank       comment         code
------------------------------------------------------------------------------
TypeScript                   77          380            42         3835
------------------------------------------------------------------------------
SUM:                         77          380            42         3835
------------------------------------------------------------------------------
```

Listing C.4: Frontend lines of code

C.2.3 Total

This is the total of all written code (including infrastructure and documentation code):

```
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
JSON                             7              2              0          16204
TypeScript                      77            380             42           3835
TeX                             49            549            163           2544
Python                          56            581            152           2204
YAML                            20            116             17           1142
Dockerfile                       8             38             12            180
Markdown                         7             56              0            165
reStructuredText                10            114            191            126
Bourne Shell                     5             13              3             53
XML                              5              0              0             39
HTML                             2              0              2             37
INI                              2              4              0             28
-------------------------------------------------------------------------------
SUM:                           248           1853            582          26557
-------------------------------------------------------------------------------
```

Listing C.5: Total lines of code

C.3 Gitlab

- **Merge requests**: 43 MRs (42 successfully merged)
- **Issues**: 24 Issues
- **Commits (Code related)**: 255 Commits
- **Commits (Infrastructure, testing and documentation**: 279 Commits

Appendix D

Project planning

This chapter explains how the project will be done.

26.09.2019	0.1	Initial definition	Aaron Meier
03.10.2019	0.2	Update milestones	Aaron Meier
22.10.2019	0.3	Update with infrastructure setup	Aaron Meier

Table D.1: Changelog Project planning

D.1 Roadmap

This projects planning is based on typical estimates for milestones/phases by previous Software Engineering modules which were taught at the HSR.

The following project preconditions were known after the projects Kick-Off:

- **Start**: 19.09.2019
- **Resources**: 480h ($2 Members * 8 ECTS * 30h$)
- **Presentation**: 20.12.2019
- **End**: 20.12.2019 17:00
- **Working days**: Weekly on Tuesday and Friday

This results in a weekly work amount of 17 hours with a total of 28 working days.

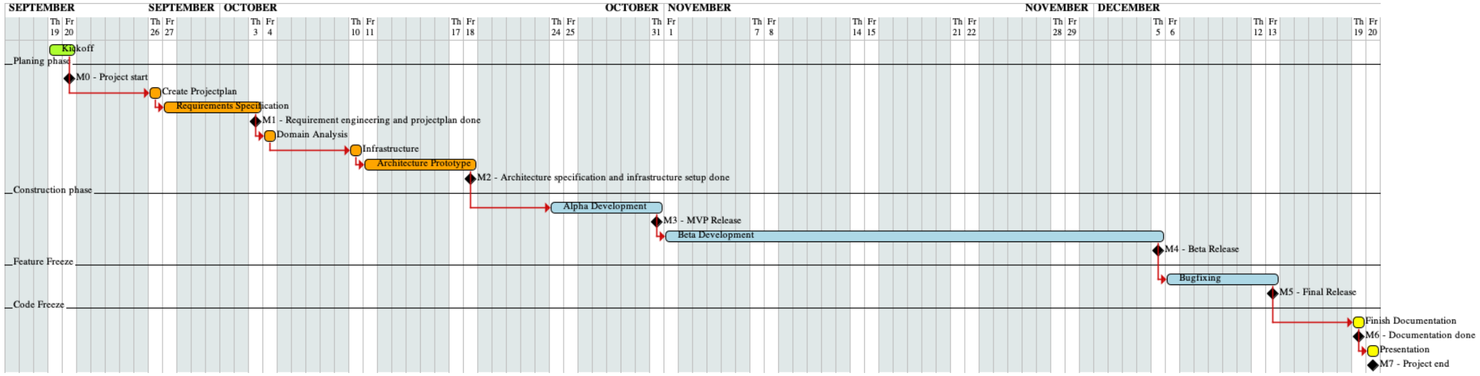


Figure D.1: Project Roadmap

D.2 Project management

Our project management is based on the method Mixed Scrum. This means work is split into Inception, Elaboration, Construction, followed by a short Transition phase. The advantage of this is a thorough analysis of requirements without loosing the flexibility in the Construction phase. As the projects goals are generally more PoC based and very new to us, it is important to have this. As can be seen on the projects roadmap

D.2.1 Phases and estimates

This is an general overview of project phases and estimates, which can also be seen in the projects roadmap.

- **Inception (colored green)**: 1 week (~7%)
- **Elaboration (colored orange)**: 4 weeks (~29%)
- **Construction (colored blue)**: 8 weeks (~57%)
- **Transition (colored yellow)**: 1 week (~7%)

D.2.2 Milestones and artifacts

The project is split into the following milestones (in bold) and their artifacts to measure its progress.

0. **Project start**
   - Task description
1. **Requirement engineering and project plan done**
   - Project plan
   - Project management
   - Textual project plan
   - Requirements (Specification and NFRs)
   - Project Scope
   - Use Cases
   - Use Case Diagram
   - Domain model
   - Wireframes (UI Mockups)
2. **Architecture specification and infrastructure setup done**
   - Infrastructure setup:
     - IDE
     - VCS
     - CI
     - Testing
     - Static Code analysis
     - Deployment
     - Bug tracking
   - Plan to minimize risks (Risk management)
   - Working units defined as User stories
   - Architecture Protoype with general Subsystem, Interfaces
3. **MVP Release**: Release of MVP (Version 0.0.1, Codename "Spooky")
4. **Beta Release**: Release of Beta (Version 0.1, Codename "Snowflake")

5. **Final Release**: Release of RTM (Version 1.0, Codename "Snowman")
   - Define and run system tests
   - Final Release with resolved bugs
6. **Documentation done**: Full Documentation
7. **Project end**: Presentation

D.2.3 Time evaluation

Time will be tracked trough toggle based on the defined Milestones and type of work. Type of work is defined in the following categories:

- **Development**
- **Research**
- **Documentation**

D.3 Meetings

Advisory meetings occur weekly, usually at the University of Applied Science Rapperswil (HSR), and take one hour. Contents of the meetings reflect the current projects state.
Advisory meeting protocols are written in English and sent to all members as soon as possible.

As both project members work in the same room, additional meetings are not necessary. If something important comes up, we track that in our bug tracker by creating an issue.

To assure bug reports quality for code we also use a predefined template for code related issues.

Every friday afternoon, further project steps are discussed. While in construction phase this also works as a weekly review of the ongoing sprint.

D.4 Responsibilities

Both project members are full stack developers. Trough previous projects we found out, that assigning responsibilities to detailed results in a more separated work environment, which is not what we like. Therefore responsibilities for front- and backend development have not been separated.

Member	Task
Aaron Meier	DevOps, Developer
Dennis Ligtenberg	Product Owner, Developer

Table D.2: Responsibilities

D.5 Infrastructure

Test environment (Lab) hardware:

- DNA Center (IP: 10.6.10.10)
- ISE (IP: 10.6.10.20)
- WLC (IP: 10.6.3.30)
- DHCP (IP: 10.6.3.50)
- Access Point (DHCP Range)
- Raspberry PIs (DHCP Range)

Development environment:

- **Version Control System (VCS), Continuous Integration (CI), Container Registry, Issue and bug tracking**: Gitlab
- **Documentation**: Overleaf
- **Code Analysis**: Sonarqube
- **Live Demo and user testing**: VM with Ubuntu by INS
- **Communication**: Cisco Webex, Mail
- **IDE**: IntelliJ Pycharm (Backend), IntelliJ Webstorm (Frontend)
- **Tool chain**: Docker/Docker-Compose (Deployment), VScode/Ansible (Inftrastructure setup), Plantuml (Planning), Draw.io (Drawing), StarUML (UML), Postman, Google Chrome Developer Tools, Firefox Developer Edition (Reverse engineering)

D.6 Development concepts

If no other specification the basis concept is Git flow. Initial setup/work of the project is done together in discussion to make sure every member knows all the projects dependencies. After that, every team member assigns the issue/working unit he is planning to working on and starts developing in their own branch. Branches are devided into feature-/issue- prefixes. The member then should commit as early and often as possible (track changes in small parts) into the branch. After successfully implementing a feature and reaching D.6.1 the branch gets merged with current development-branch. In this state a new development release gets deployed and can be tested via the live system. Releasing a new production version is done by tagging and merging to master branch.

D.6.1 Definition of Done

Feature/Issue is mergeable to development branch if the following conditions are met:

- Tests written/updated (common sense typical cases) and run without errors
- Code style guide goals are met
- Static code analysis has no errors
- Code has been reviewed by another person
- Quality gate of sonarqube has been reached: 50% coverage, No code smells, no OWASP errors

D.6.2 Review

We use the code review feature of Gitlab with additional merge request templates. The following process will be done on each review from another person:

- Checkout feature/issue branch and test if feature works or issue has been resolved
- Unit/Integration tests have been defined
- The implementation is architecture/design compliant
- Variable and class names are meaningful and have been well selected
- Other code smells, which cannot easyly be found by static code analysis (e.g. solution sprawl) do not exist in added code

D.7 Backups

The projects main storage solution is based on Git repositories. These are decentrally stored and can be easily restored from one or more destinations. Also the important branches are all protected and minimize risk of accidental overwrite. Additionaly we mirror the main documentation from Overleaf to Gitlab and rely mainly on well known public platforms.

D.8 Risk management

As mentioned earlier this project is based more on PoC and therefore the usual potential risks are rather small. The same applies to the impact of such a risk. If a risk occurs we have enough time planned to handle it, but it might decreases the chance to implement additional features.

#	Description	E[1]	S[2]	Mitigation	Action at occurence
1	Lab Infrastructure will not be available on defined date	2	1	Plan other work units to be flexible in working capacity	Do other project relevant tasks first
2	Synchronisation/API can not be done trough missing API features	4	7	As a backup solution parsing DNA centers web UI can always being considered. Define project scope exact and document differences.	Search for different possible solutions and present to project partner. Inform project advisor/-partner and explain differences in possible solutions
3	Cisco DNA Center API features do not work	6	9	Plan enough time for API research	Inform Cisco (make a wish), inform project advisor/-partner and explain possible workarounds.

Table D.3: Risk list

[1] Possibility for Occurance
[2] Weighted Damage

D.9 Quality assurance

Assurance	Time
Code review for merges into development branch with another person	After pull request
Proofreading of documentation	Transition phase
Unit and Integration tests	Continuously
Use branches (issue/feature)	Continously
Dependency checks	On every push
Use of linters	On every push
Use of quality gates	On every merge request

Table D.4: Quality Assurance

D.9.1 Exception Handling

Backend

1. Log error and stack trace
2. Show error message in debug mode in output
3. Disable spreading of error and further processing of higher components if possible
4. In API request respond with an error code and message based on the HTTP standard

As availability is not very important (based on NFRs), we will not add any error handling code for monitoring (e.g. watchdog, mailing). On the other hand consistency in sync is important, therefore we use rollbacks if errors occur.

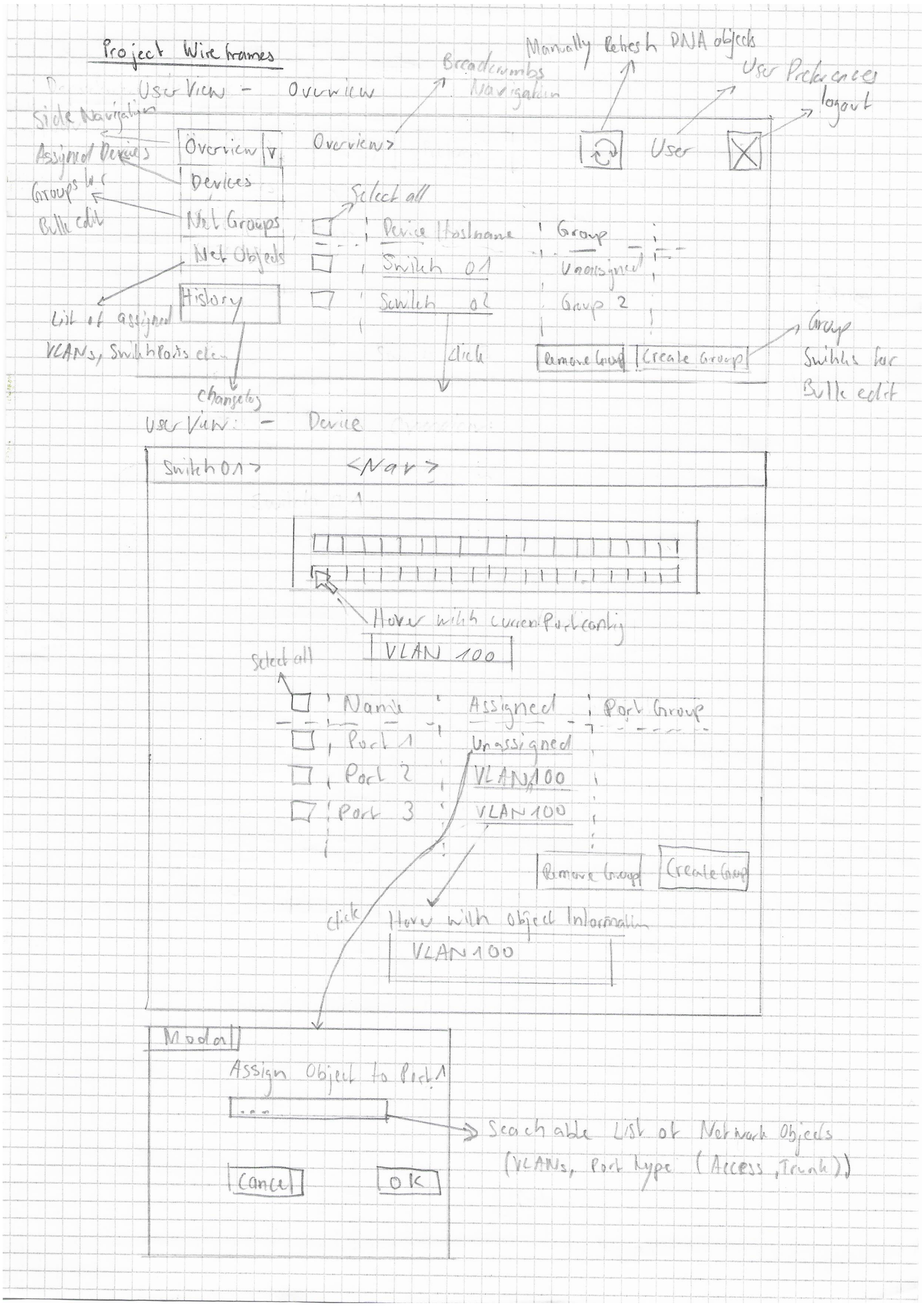
Frontend The frontends implementation relies on the Backend codes and doesn't evaluate Backend messages.

1. Errors from API requests will be handled directly by its error code. Then a message gets assigned and the error will be shown trough a notification alert to the user.
2. In debug mode error messages are logged to the web browsers console.

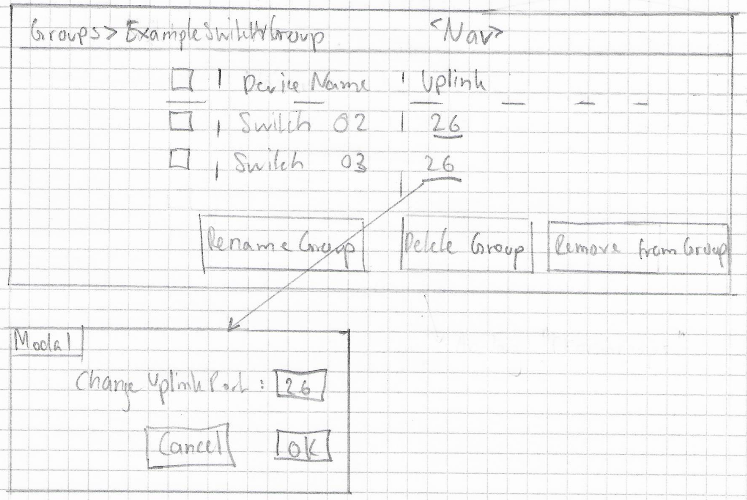
Appendix E

Wireframes

E.1 User Views

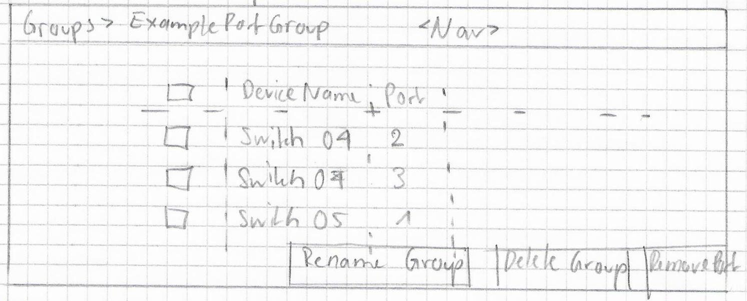


UsrView - NetGroup /Switch

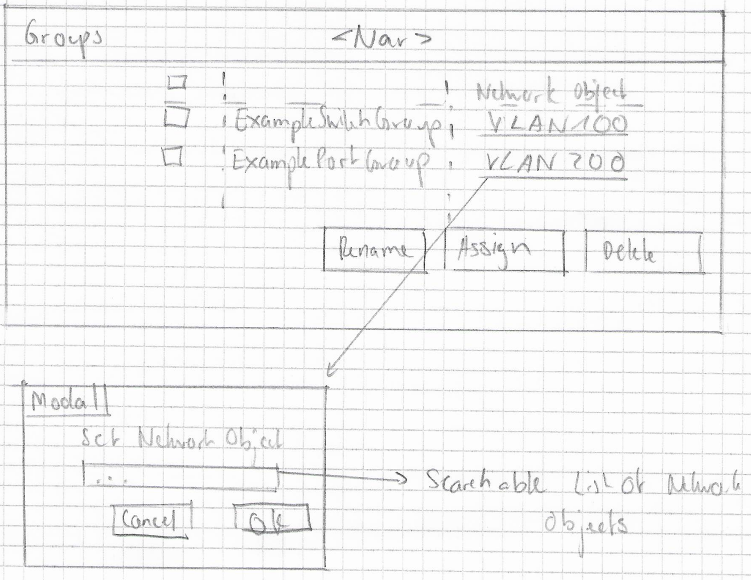


□	Device Name	Uplink
□	Switch 02	26
□	Switch 03	26

UsrView - Net Group / Port

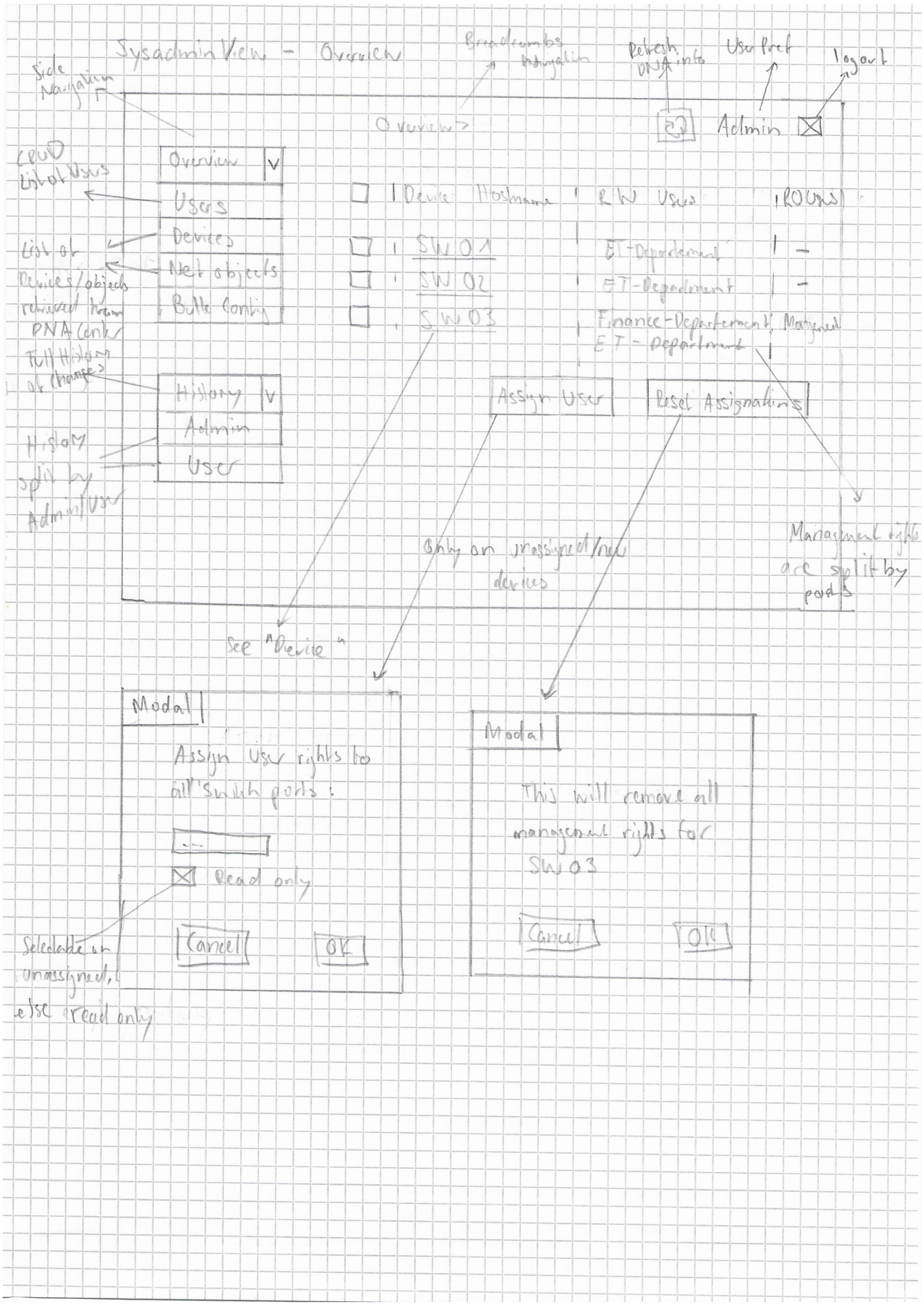


□	Device Name	Port
□	Switch 04	2
□	Switch 07	3
□	Swith 05	1

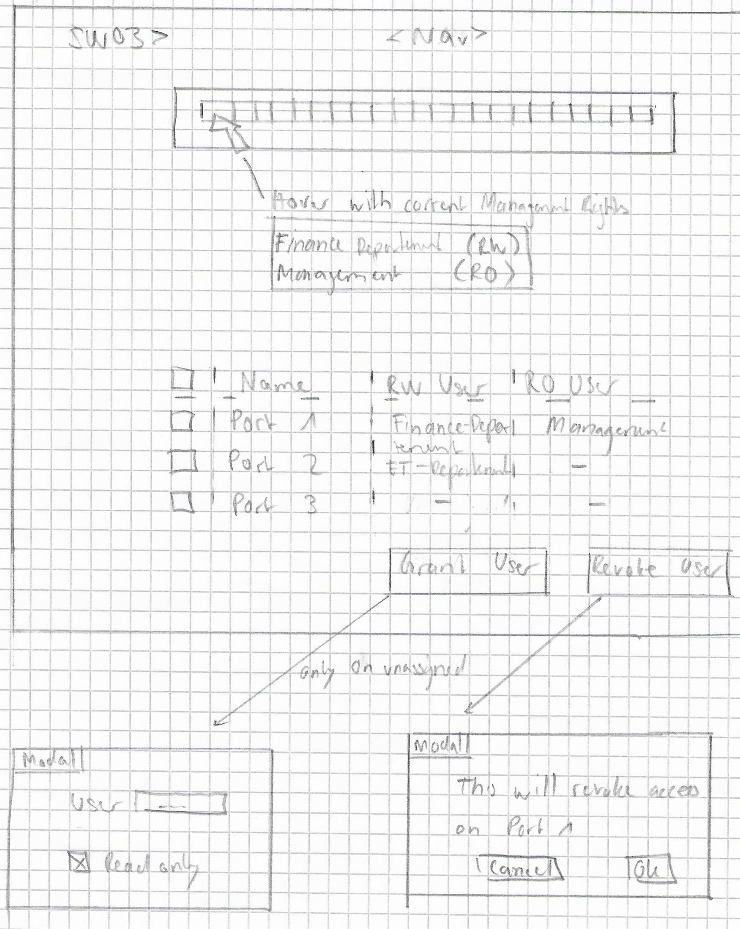


□		Network Object
□	ExampleSwitchGroup	VLAN100
□	ExamplePortGroup	VLAN 200

E.2 Admin Views


Sysadmin View - Overview
Breadcrumbs Navigation
Refresh DNA info
User Pref
logout
Side Navigation
Overview>
Admin ☒
CRUD List of Users
Overview v
Users
Devices
Net objects
Bulk Config
List of Devices/objects retrieved from DNA Center
Full History of changes
History v
Admin
User
History split by Admin/User
□ | Device Hostname | RW Users | RO Users
□ | SW 01 | ET-Department | -
□ | SW 02 | ET-Department | -
□ | SW 03 | Finance-Departement, Management ET - Department |
Assign User
Reset Assignations
Only on unassigned/new devices
Management rights are split by ports
See "Device"
Modal
Assign User rights to all Switch ports:
...
☒ Read only
Cancel
OK
Selectable on unassigned, else read only
Modal
This will remove all management rights for SW 03
Cancel
OK

Sysadmin View - Device



Finance Department (RW)
Management (RO)

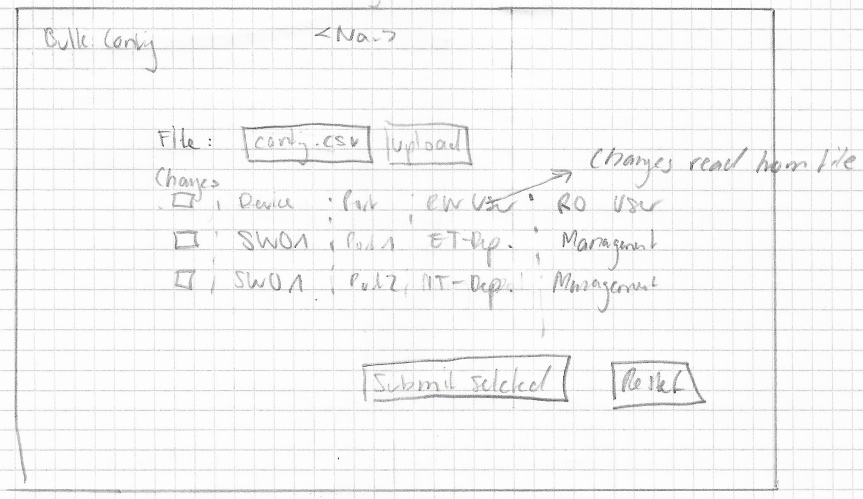
□	Name	RW User	RO User
□	Port 1	Finance-Department	Management
□	Port 2	IT-Department	–
□	Port 3	–	–

User

☒ Read only

This will revoke access on Port 1

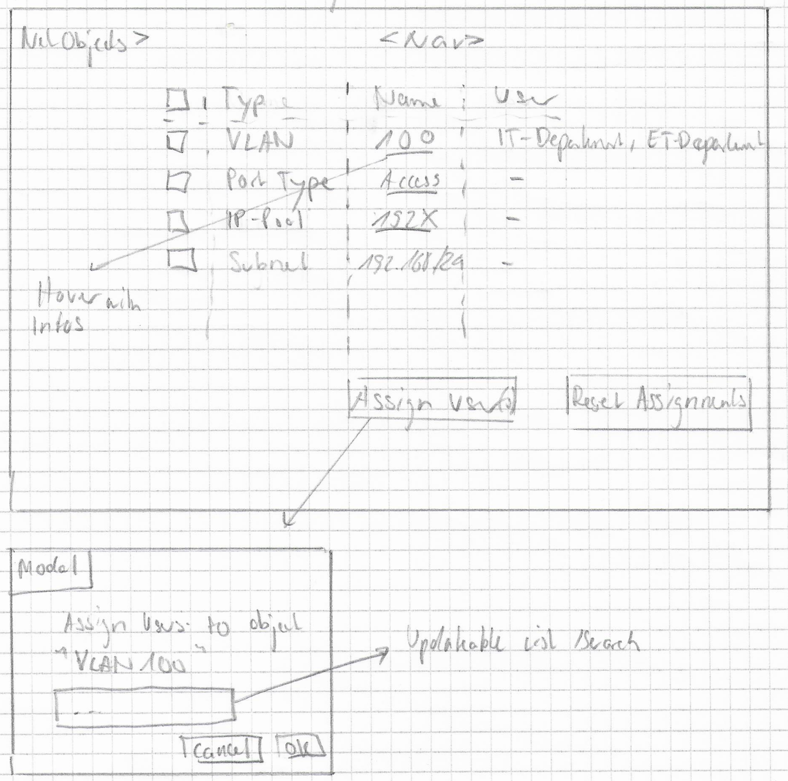
SysadminView - Bulk Config



Changes

	Device	Port	RW User	RO User
□	SW01	Port1	ET-Dep.	Management
□	SW01	Port2	IT-Dep.	Management

SysadminView - Net objects



□	Type	Name	User
□	VLAN	100	IT-Department, ET-Department
□	Port Type	Access	-
□	IP-Pool	192X	-
□	Subnet	192.168/24	-

Assign Users to object "VLAN 100"

...

Appendix F

User interface screenshots

F.1 Login view

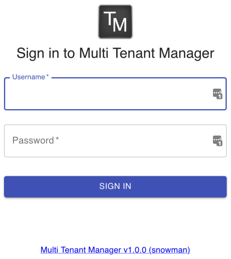
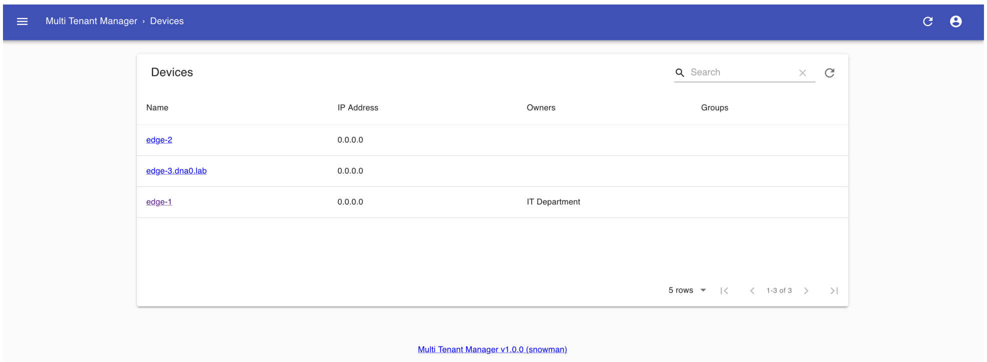

TM
Sign in to Multi Tenant Manager
Username *
Password *
SIGN IN
Multi Tenant Manager v1.0.0 (snowman)

Figure F.1: Login view

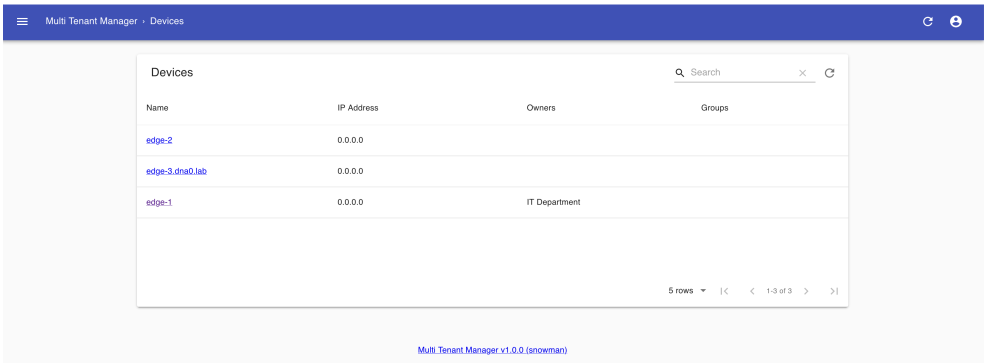
F.2 Devices view


Multi Tenant Manager › Devices
Devices
Search
5 rows
1-3 of 3
Multi Tenant Manager v1.0.0 (snowman)

Name	IP Address	Owners	Groups
edge-2	0.0.0.0		
edge-3.dna0.lab	0.0.0.0		
edge-1	0.0.0.0	IT Department	

Figure F.2: Devices view

F.3 History view



Multi Tenant Manager › Devices

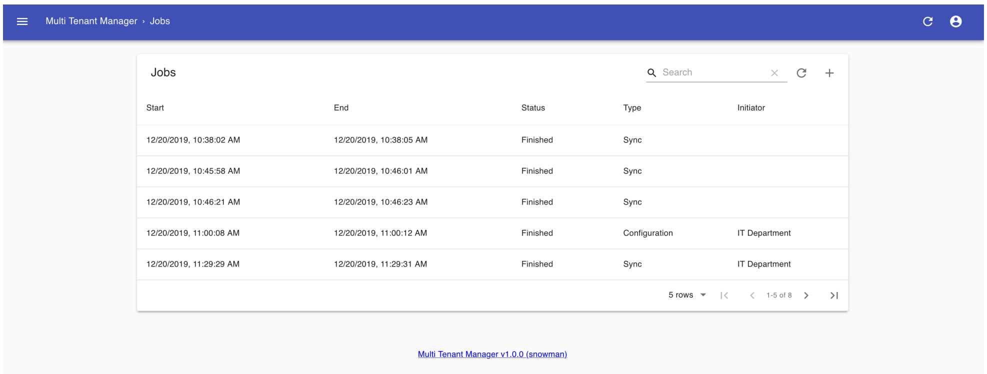
Devices

Name	IP Address	Owners	Groups
edge-2	0.0.0.0		
edge-3.dna0.lab	0.0.0.0		
edge-1	0.0.0.0	IT Department	

5 rows
1-3 of 3
Multi Tenant Manager v1.0.0 (snowman)

Figure F.3: History view

F.4 Jobs view



Multi Tenant Manager › Jobs

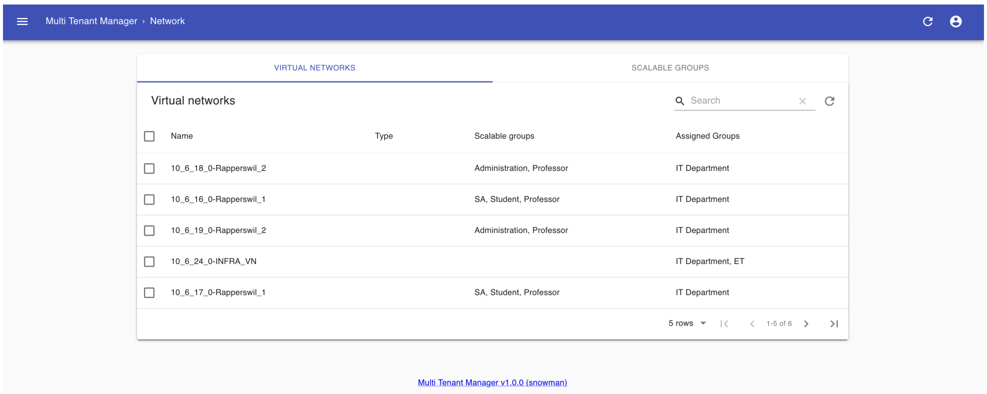
Jobs

Start	End	Status	Type	Initiator
12/20/2019, 10:38:02 AM	12/20/2019, 10:38:05 AM	Finished	Sync	
12/20/2019, 10:45:58 AM	12/20/2019, 10:46:01 AM	Finished	Sync	
12/20/2019, 10:46:21 AM	12/20/2019, 10:46:23 AM	Finished	Sync	
12/20/2019, 11:00:08 AM	12/20/2019, 11:00:12 AM	Finished	Configuration	IT Department
12/20/2019, 11:29:29 AM	12/20/2019, 11:29:31 AM	Finished	Sync	IT Department

5 rows
1-5 of 8
Multi Tenant Manager v1.0.0 (snowman)

Figure F.4: History view

F.5 Network view


Multi Tenant Manager › Network
VIRTUAL NETWORKS
SCALABLE GROUPS
Search

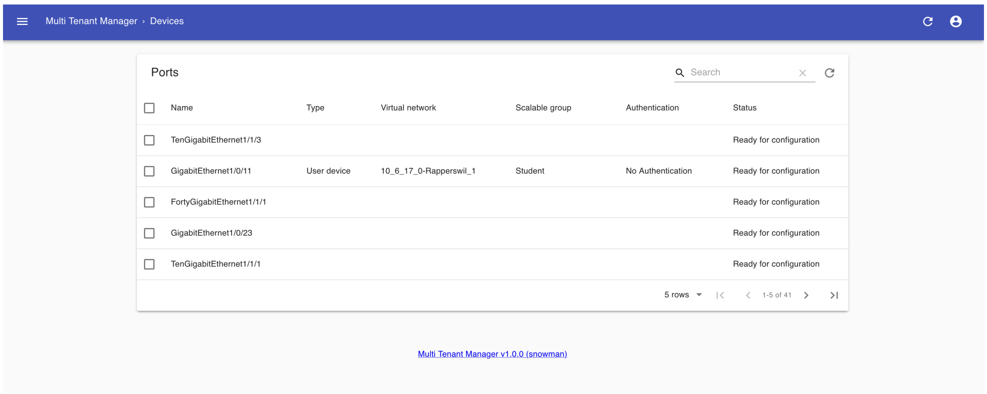
Virtual networks

☐	Name	Type	Scalable groups	Assigned Groups
☐	10_6_18_0-Rapperswil_2		Administration, Professor	IT Department
☐	10_6_16_0-Rapperswil_1		SA, Student, Professor	IT Department
☐	10_6_19_0-Rapperswil_2		Administration, Professor	IT Department
☐	10_6_24_0-INFRA_VN			IT Department, ET
☐	10_6_17_0-Rapperswil_1		SA, Student, Professor	IT Department

5 rows
1-5 of 6
Multi Tenant Manager v1.0.0 (snowman)

Figure F.5: Network view

F.6 Interface view



Ports

☐	Name	Type	Virtual network	Scalable group	Authentication	Status
☐	TenGigabitEthernet1/1/3					Ready for configuration
☐	GigabitEthernet1/0/11	User device	10_6_17_0-Rapperswil_1	Student	No Authentication	Ready for configuration
☐	FortyGigabitEthernet1/1/1					Ready for configuration
☐	GigabitEthernet1/0/23					Ready for configuration
☐	TenGigabitEthernet1/1/1					Ready for configuration

Figure F.6: Interfaces view

Appendix G

Time tracking

G.1 Time tracking by milestone

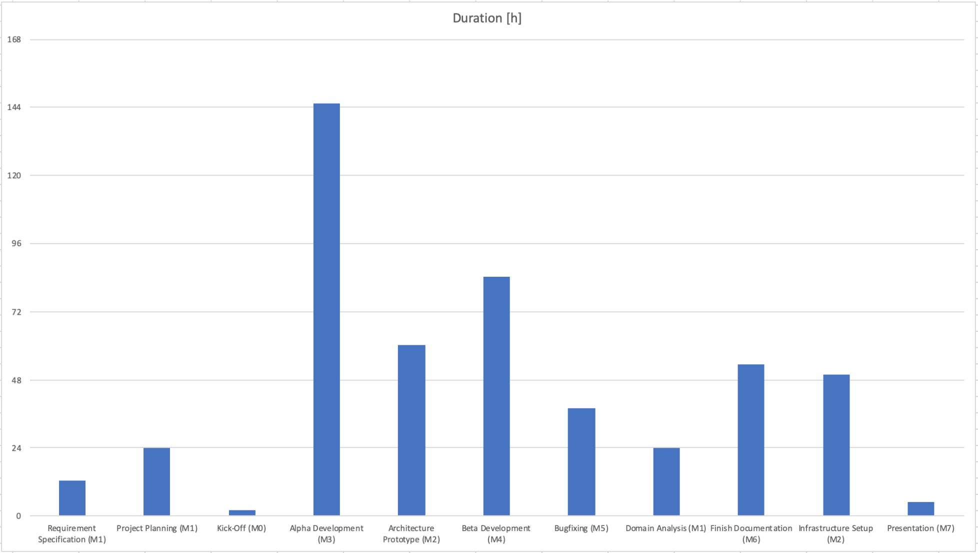


Figure G.1: Time by milestone

G.2 Time tracking by category

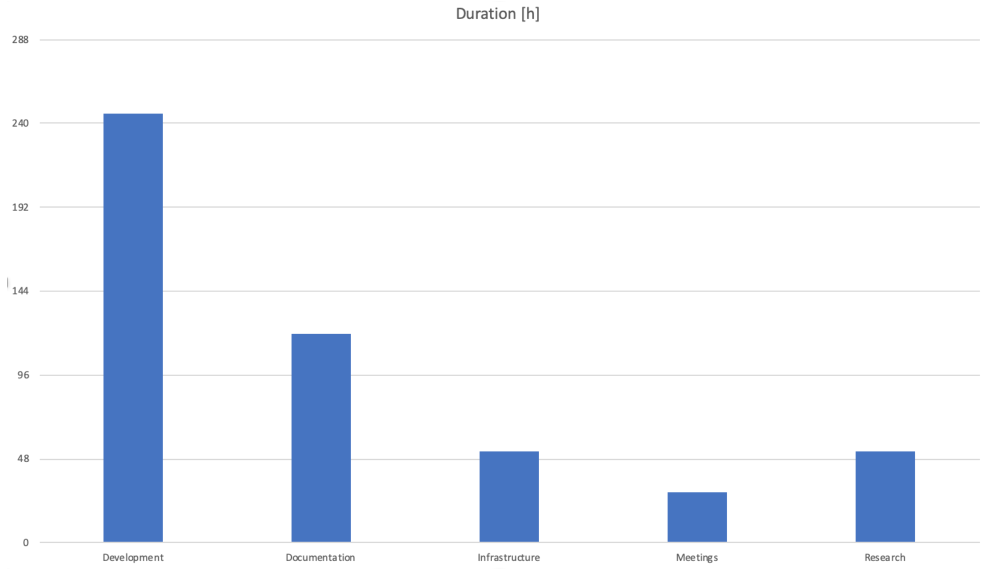


Figure G.2: Time by category

G.3 Time tracking by project members

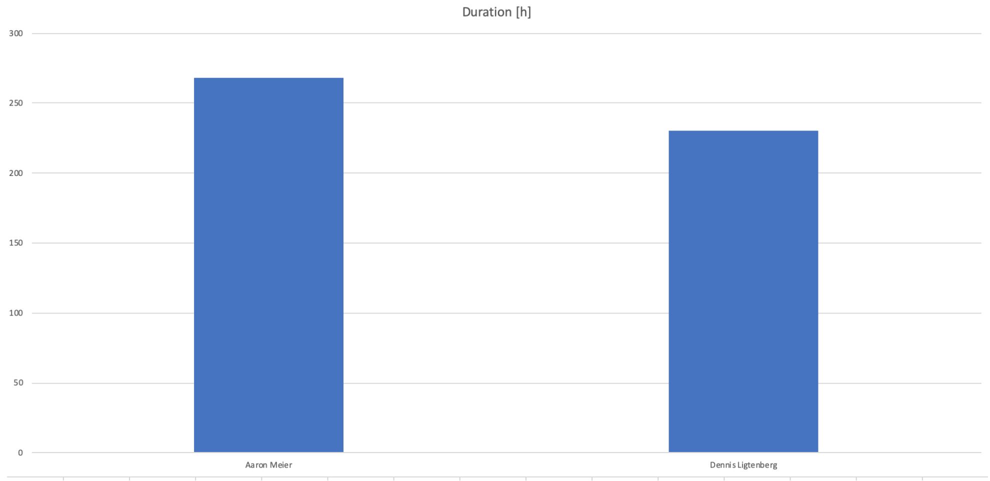


Figure G.3: Time by project members

G.4 Time tracking to actual/planned comparison

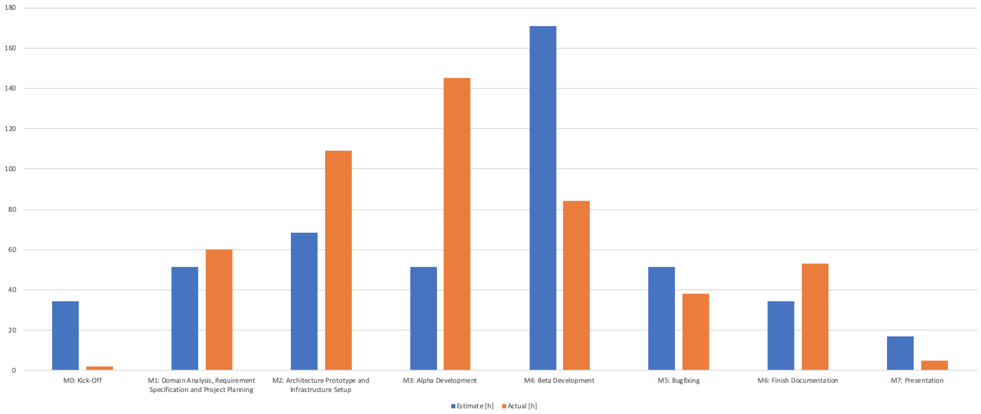


Figure G.4: Time comparison

Appendix H

Code and installation documentation

The following code and installation documentation is auto generated with Sphinx and is also available in HTML form within the backend.

Multi Tenant Manager - Code and installation documentation

Release 1.0.0

Aaron Meier, Dennis Ligtenberg

Dec 19, 2019

CONTENTS

CHAPTER

ONE

INSTALLATION

The following documents the installation for the `Multi Tenant Manager Project`.

1.1 Dependencies

- Docker
- Docker-Compse
- Access to the docker images (hosted on Gitlab)

1.2 Download images

```
docker login registry.gitlab.com

# Image locations
WEB_IMAGE=registry.gitlab.com/cisco-dna-center-multi-tenant-manager/setup/web
API_IMAGE=registry.gitlab.com/cisco-dna-center-multi-tenant-manager/setup/api

# Production
docker pull $WEB_IMAGE:latest-production
docker pull $API_IMAGE:latest-production

# Development
docker pull $WEB_IMAGE:latest-development
docker pull $API_IMAGE:latest-development
```

1.3 Manual build

Prebuilt Images are stored in the registry. If you have access to the source code you may also build the images yourself trough their Dockerfiles. After that you should update `.env` with the new image locations.

1.4 Run Development

```
docker-compose up
```

1.5 Run production

```
docker-compose -f docker-compose.production.yml up
```

CHAPTER

TWO

CONFIGURATION

Configuration is supported only trough environment variables. By using the predefined setup repository, access to the `.env` file is already given. This file holds all configuration data and can be adjusted to preference.

2.1 Reference

If you want to customize the deployment even more, the following tables lists all available settings.

2.1.1 Backend settings:

Environment variable	Default Value	Description
SECRET_KEY	SECRET_KEY	Secret key the backend uses as a seed
PRODUCTION	False	Enable production mode
CONTEXT_PATH	api	Set the path were the api is available
LANGUAGE_CODE	en-US	Set the language
TIME_ZONE	UTC	Set the time zone
DATABASE_ENGINE	sqlite	Choose between sqlite or postgres
DATABASE_PATH	./	Path for sqlite
DATABASE_NAME	tenantmanager	Database name
DATABASE_USER	postgres	Database user
DATABASE_PASSWORD	postgres	Database password
DATABASE_HOST	127.0.0.1	Database host
DATABASE_PORT	5432	Database port
EMAIL_HOST	localhost	Email host
EMAIL_HOST_USER	username	Email user
EMAIL_HOST_PASSWORD	password	Email password
EMAIL_PORT	587	SMTP port
EMAIL_USE_TLS	True	Enable encryption for SMTP
LOG_LEVEL	INFO	Choose between DEBUG, INFO and WARN
DNA_DEMO_MODE	False	Enable demo mode, loads sample data
DNA_USERNAME	admin	DNA center administrator for connections
DNA_PASSWORD	password	DNA center password for connections
DNA_ADDRESS	10.1.1.1	DNA center address
DNA_TIMEOUT	5	Timeout for DNA center connections
DNA_ALLOW_INSECURE_SSL	False	Disable SSL verification, useful if self-signed certificates are used
REDIS_URL	redis://	URL for redis broker
RESULT_BACKEND	django-db	Results backend for broker, should not be changed

Continued on next page

Table 1 – continued from previous page

Environment variable	Default Value	Description
SUPERUSER_USERNAME	admin	Root username
SUPERUSER_EMAIL	admin@example.com	Root email
SUPERUSER_PASSWORD	password	Root password
LOAD_TEST_ENABLED	False	Enable load testing

2.1.2 Frontend settings:

Environment variable	Default Value	Description
NODE_ENV	production	Set either production or development modus
REACT_APP_API_URL	/api	Backend location

CHAPTER

THREE

CODE DOCUMENTATION

3.1 apps package

3.1.1 Subpackages

3.1.1.1 apps.api package

Subpackages

apps.api.dnac package

Submodules

apps.api.dnac.exceptions module

exception `apps.api.dnac.exceptions.`**`DNABaseException`**

Bases: `Exception`

Base exception

exception `apps.api.dnac.exceptions.`**`DNACConfigException`**

Bases: *`apps.api.dnac.exceptions.DNABaseException`*

Config exception

exception `apps.api.dnac.exceptions.`**`DNACSyncException`**

Bases: *`apps.api.dnac.exceptions.DNABaseException`*

Sync exception

apps.api.dnac.helpers module

`apps.api.dnac.helpers.`**`wait_for_task_completion`**`(`*task_id*`)`

apps.api.dnac.port_assignment module

apps.api.dnac.sync module

Module contents

apps.api.serializers package

Submodules

apps.api.serializers.accounts module

apps.api.serializers.jobs module

apps.api.serializers.networks module

Module contents

apps.api.types package

Submodules

apps.api.types.choices_enum module

class apps.api.types.choices_enum.**ChoicesEnum**

Bases: `enum.Enum`

An enumeration.

choices = <bound method ChoicesEnum.choices of <enum 'ChoicesEnum'>>

apps.api.types.configuration_status module

class apps.api.types.configuration_status.**ConfigurationStatus**

Bases: *apps.api.types.choices_enum.ChoicesEnum*

An enumeration.

FAILED = 'FAILED'

PENDING = 'PENDING'

READY = 'READY'

apps.api.types.interface_type module

class apps.api.types.interface_type.**InterfaceType**

Bases: *apps.api.types.choices_enum.ChoicesEnum*

An enumeration.

ACCESS_POINT = 'ACCESS_POINT'

SERVER = 'SERVER'

USER_DEVICE = 'USER_DEVICE'

apps.api.types.job_status module

class apps.api.types.job_status.**JobStatus**

Bases: *apps.api.types.choices_enum.ChoicesEnum*

An enumeration.

FAILED = 'FAILED'

FINISHED = 'FINISHED'

PENDING = 'PENDING'

apps.api.types.job_type module

class apps.api.types.job_type.**JobType**

Bases: *apps.api.types.choices_enum.ChoicesEnum*

An enumeration.

CONFIGURATION = 'CONFIGURATION'

SYNC = 'SYNC'

apps.api.types.user_roles module

class apps.api.types.user_roles.**UserRoles**

Bases: enum.IntEnum

An enumeration.

NetworkAdministrator = 2

SystemAdministrator = 1

choices = <bound method UserRoles.choices of <enum 'UserRoles'>>

apps.api.types.virtual_network_type module

class apps.api.types.virtual_network_type.**VirtualNetworkType**

Bases: *apps.api.types.choices_enum.ChoicesEnum*

An enumeration.

DEFAULT = 'DEFAULT'

GUEST = 'GUEST'

INFRA = 'INFRA'

ISOLATED = 'ISOLATED'

Module contents

apps.api.views package

Submodules

apps.api.views.accounts module

apps.api.views.jobs module

apps.api.views.misc module

apps.api.views.networks module

Module contents

Submodules

apps.api.admin module

apps.api.apps module

class apps.api.apps.**ApiConfig**(*app_name*, *app_module*)

Bases: `django.apps.config.AppConfig`

label = 'api'

verbose_name = 'Multi Tenant Manager API'

apps.api.mixins module

class apps.api.mixins.**FilterGroupMixin**(***kwargs*)

Bases: `rest_framework.generics.GenericAPIView`

filter_queryset_group(*queryset*)

get_queryset()

Get the list of items for this view. This must be an iterable, and may be a queryset. Defaults to using *self.queryset*.

This method should always be used rather than accessing *self.queryset* directly, as *self.queryset* gets evaluated only once, and those results are cached for all subsequent requests.

You may want to override this if you need to provide different querysets depending on the incoming request.

(Eg. return a list of items that is specific to the user)

class apps.api.mixins.**UserContextMixin**(***kwargs*)

Bases: `rest_framework.generics.GenericAPIView`

get_serializer_context()

Extra context provided to the serializer class.

apps.api.models module

apps.api.permissions module

class apps.api.permissions.**IsNetworkAdministrator**

Bases: `rest_framework.permissions.BasePermission`

This permission is used as a reference for network administration over Tenant manager It allows the user to create assign their networks to devices

has_permission(*request: rest_framework.request.Request*, *view: django.views.generic.base.View*) → bool

Return *True* if permission is granted, *False* otherwise.

class apps.api.permissions.**IsSystemAdministrator**

Bases: `rest_framework.permissions.BasePermission`

This permission is used as a reference for full control over Tenant manager It allows the user to create user groups and assign access rights to them

has_permission(*request: rest_framework.request.Request*, *view: django.views.generic.base.View*) → bool

Return *True* if permission is granted, *False* otherwise.

apps.api.tasks module

apps.api.urls module

apps.api.utils module

Module contents

3.1.2 Module contents

3.2 undocumented_dnac_client package

3.2.1 Subpackages

3.2.1.1 undocumented_dnac_client.api package

Submodules

undocumented_dnac_client.api.authentication module

class undocumented_dnac_client.api.authentication.**AuthenticationApi**(*base_url: str*)

Bases: `object`

list(*username: str*, *password: str*) → Dict[str, str]

undocumented_dnac_client.api.authentication_profiles module

class undocumented_dnac_client.api.authentication_profiles.**AuthenticationProfilesAPI**(*base_ur str*, *access_tok str*)

Bases: object

list()

undocumented_dnac_client.api.devices module

class undocumented_dnac_client.api.devices.**DevicesAPI**(*base_url: str*, *access_token: str*)

Bases: object

get_device(*device_id: str*)

list()

update_ports(*device_id: str*, *ports: List*) → str

undocumented_dnac_client.api.scalable_groups module

class undocumented_dnac_client.api.scalable_groups.**ScalableGroupsAPI**(*base_url: str*, *access_token: str*)

Bases: object

list()

undocumented_dnac_client.api.segments module

class undocumented_dnac_client.api.segments.**SegmentsAPI**(*base_url: str*, *access_token: str*)

Bases: object

list()

undocumented_dnac_client.api.virtual_networks module

class undocumented_dnac_client.api.virtual_networks.**VirtualNetworksAPI**(*base_url: str*, *access_token: str*)

Bases: object

list()

list_context()

Module contents

class `undocumented_dnac_client.api.`**DNACenterAPI**(*base_url: str*, *username: str*, *password: str*)

Bases: `object`

DNA Center Undocumented API endpoints client

Creates a client to use undocumented/unofficial DNA center endpoints.

3.2.2 Submodules

3.2.3 undocumented_dnac_client.exceptions module

exception `undocumented_dnac_client.exceptions.`**RateLimitException**

Bases: *undocumented_dnac_client.exceptions.UndocumentedDNACenterAPIException*

Exception for rate limiting.

exception `undocumented_dnac_client.exceptions.`**UndocumentedDNACenterAPIException**

Bases: `Exception`

Base for exceptions.

3.2.4 undocumented_dnac_client.response_codes module

Undocumented DNA Center response codes Copyright (c) 2019 Cisco and/or its affiliates. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

These codes are taken from the community built dna center SDK https://github.com/cisco-en-programmability/dnacentersdk/blob/master/dnacentersdk/response_codes.py

3.2.5 undocumented_dnac_client.utils module

`undocumented_dnac_client.utils.`**check_response**(*response*, *expected*)

3.2.6 Module contents

PYTHON MODULE INDEX

INDEX