Bachelor Thesis

Cisco DNA Center Tenant Manager v2

University of Applied Science Rapperswil

Department of Computer Science

24.02.2020 - 26.06.2020

Authors	Aaron Meier	**Advisor**	Professor Laurent Metzger
	Dennis Ligtenberg	**External Co-Examiner**	Laurent Billas
		Internal Co-Examiner	Professor Stefan F. Keller
Project Partner	Cisco Systems (Switzerland) GmbH		

Abstract

Introduction

In the scope of a term project thesis a Proof of Concept (PoC) was developed that allows for port configuration by multiple tenants on Cisco edge network switches. This was achieved by leveraging the API capabilities of Cisco's Software Defined Networking (SDN) platform DNA center. To further show external functionality the DNA platform enables through programmability, this thesis expands the student research project. Included in the scope is not only implementation but also finding and elaborating on new use cases together with the industry partner and the thesis advisor.

Approach

The base for the new features was a design thinking session with all involved project parties. It was decided to add a time based guest access feature with automatic rollback for the pre-existing port configuration functionality. Additionally on device configuration change management and notification should be added as a new feature. The resulted ideas were expanded on and their scope defined and limited by technical viability.
Before adding new features some updates where made to the existing base. This includes updates to the infrastructure, libraries and frameworks used plus improvements to the code architecture and quality.

Result

As a result three main features were added:

- Expanding port configuration and access management with time based access including automatic rollback of any configuration made during this access.
- Detecting configuration changes directly on edge switches.
- Notifying users about detected changes and approval workflow with automatic configuration rollback when denied.

Management Summary

Initial situation

The Cisco Digital Network Architecture (DNA) is a Software defined networking (SDN) solution that enables the full management of a campus network through a graphical user interface on the browser. Troubleshooting, monitoring and configuration are made more efficient and accessible through the frontend. Additionally third party integrations are made possible through a Application Programming Interface (API). In the scope of term project thesis a prove of concept was developed to add the option of multi tenancy on a third party website using the Cisco Intent API endpoints of DNA center. This thesis explores improving this base and to find and build new features on top of it.

Process

The project kickoff was marked by a design thinking session. Through this democratized approach for finding product ideas multiple improvements and new features were found. It was decided that three main features should be added.
The port configuration and access management was expanded with a feature that allows for creation of time limited access groups. Any configuration made by members of this group is automatically rolled back after the access expires. This could be a useful feature if one wants to give access to users in the scope of a lab course or workshop where DNA Center Interface configurations are involved. As the settings are automatically rolled back less cleanup has to be done after conducting such a course.
As a new feature detection on device configuration was implemented. If network wide changes can be detected compliance applications can be built with them so unapproved changes in a campus do not slip by administrators.
For detected changes a notification feature was implemented using a multitude of notification providers. As interface configurations are already structured in the Tenant Manager and a rollback mechanism was built for interface configuration changes an option for rolling back those changes was added.

Results

Most of the base project was updated and the architecture adjusted to better fit the growing size of the project. All technologies were upgraded to the most current version to ensure the maximum support length.
A fully functional PoC was implemented that covers all aforementioned features and a multitude of optionals. Multiple notification providers including ServiceNow, Cisco Webex Teams, E-Mail and in app where implemented.
Unfortunately no DNA Center interactions could be ported to official Intent API endpoints.

Outlook

The official DNA Center API still seems to be in more of a public testing state than fully productively usable. Rate limiting and missing features make the endpoints work for simple PoCs but not fit for full third party integration. At this point it is not recommendable to build productive implementations yet. However every DNA update improves the situation which certainly makes checking up on larger version updates worth it.
The change detection feature opens up the possibility for many interesting use cases. A feature that allows for rolling back any kind of configuration made on a device could be incredibly interesting. The change detection itself is not dependend on the DNA Center and could be used across networks outside of the campus.

Task description

Aufgabenstellung Bachelor Thesis (BA) " Multi-Tenant Manager v2 for Software Defined Access"

Software Defined Access is the next generation technology for networks in Campus.

The new requirements for mobility, security, automation and operations couldn't be met by the traditional networks.

- By building an overlay, Software Defined Access enables networks to be deployed with the flexibility of a software.
- By using a centralized Management called DNA Center, a high degree of automation can be reached.

The GUI of the DNA Center does not natively support multi-tenancy in the current version (1.3.3.x) and there is a need for a new layer of abstraction so that not only a few network administrators can modify the network parameters.

Empowering the users to modify their part of the network is a requirement that is going to be addressed in that Bachelor thesis (BA)

Such an application built on top of DNA Center, providing multi-tenancy has already been developed during the semester thesis (SA), and the students are now being given the objective to improve their existing app and add additional features.

The following use cases should be covered by the "improved" app:

Refactoring of the App	New architecture	MUST
Time based Access	Grant temporary access for a guest user to a limited number of ports	MUST
	Create an invite link	MUST
	The guest user accesses the platform using the invite link	MUST
	The guest user can list the ports that they are allowed to configure in the GUI.	MUST
	After the guess access expires, the configuration performed by the guest user is "cleaned" and set back to default.	OPTIONAL
Change Control	When a manual configuration is performed in an edge router, an event is raised.	MUST
	The event is sent to the GUI of the app.	MUST
	There is also the option to send the event to ServiceNow, a Webex	MUST

Contents

Introduction i
Approach i
Result i

I Technical Report **1**

1 Introduction and overview **2**
1.1 Problem definition and goals 2
1.2 Design thinking session 3
1.2.1 Ideate 3
1.2.2 Clustering and prioritization 3
1.2.3 Whiteboards 4
1.3 Scope and limitations 5
1.4 State of art 5

2 Requirements **6**
2.1 Use cases 6
2.1.1 Actors 6
2.1.2 Time Based Access Scenario 7
2.1.3 Change Control Scenario 9
2.1.4 Optional Use Cases 10
2.2 None functional requirements 11

3 Architecture Updates and improvements **12**
3.1 Original Architecture 12
3.1.1 Possible Improvements 13
3.1.2 Conclusion 13
3.2 Updates 14
3.2.1 DNAC Update 14
3.2.2 DNAC API 14

3.2.3 Worker Separation . . . 16
3.2.4 Introducing Pagination . . . 16
3.2.5 State management . . . 18
3.2.6 Context Provider . . . 19
3.2.7 Refactoring . . . 20

4 Architecture and design specification **21**
4.1 External interfaces . . . 21
4.1.1 DNA Center Intent API . . . 21
4.1.2 ServiceNow . . . 22
4.1.3 Webex Teams . . . 22
4.2 Domain . . . 23
4.2.1 Domain Model . . . 23
4.3 Technologies . . . 24
4.3.1 Programming languages . . . 24
4.3.2 Frameworks . . . 24
4.3.3 Libraries overview . . . 25
4.4 Design overview . . . 26
4.4.1 Frontend . . . 26
4.4.2 Backend . . . 27
4.4.3 Layers . . . 29
4.5 Containerization . . . 32

5 Time Based Access **33**
5.1 Passwordless Authentication . . . 33
5.2 Permission management . . . 35
5.3 Result . . . 36

6 Change Control **37**
6.1 Change Control Activity Diagram . . . 38
6.2 Configuration change detection . . . 39
6.2.1 Change detection methods . . . 39
6.2.2 Graylog . . . 41
6.2.3 Implementation . . . 42
6.3 Notifications . . . 43
6.3.1 Changes . . . 43
6.3.2 Result . . . 43
6.3.3 Notification Services . . . 44
6.3.4 In App Provider . . . 44
6.3.5 Webex Provider . . . 45
6.3.6 Mail Provider . . . 45
6.3.7 ServiceNow Provider . . . 45

7 Results and conclusion **47**

7.1 Achieved results . . . 47
7.1.1 Screenshots . . . 48
7.2 Possible improvements and additions . . . 52
7.3 Acknowledgements . . . 52

II Project documentation **53**

8 Deployment **54**
8.1 Updates . . . 54
8.2 Components . . . 55
8.2.1 Production Deployment . . . 56
8.3 Diagrams . . . 56
8.4 Infrastructure . . . 58
8.4.1 Updates . . . 58
8.4.2 Repositories . . . 58
8.4.3 Continuous Integration . . . 59

9 Data model **61**

10 API **63**
10.1 Endpoints . . . 63
10.2 Detailed documentation . . . 64

List of Figures **65**

List of Tables **67**

Glossary **68**

Bibliography **70**

III Appendix **72**

A Personal reports **1**
A.1 Aaron Meier . . . 1
A.2 Dennis Ligtenberg . . . 2

B Testing **3**
B.1 Backend . . . 3
B.1.1 Results . . . 3
B.2 Frontend . . . 3
B.3 NFR Validation . . . 4
B.4 Performance tests . . . 6
B.4.1 Scenario . . . 6

B.5 System tests . . . 8
B.5.1 Scenarios . . . 8
B.5.2 Test log . . . 9

C Metrics **11**
C.1 Sonarqube reports . . . 11
C.1.1 Frontend . . . 11
C.1.2 Backend . . . 12
C.2 Code . . . 13
C.2.1 Backend . . . 13
C.2.2 Frontend . . . 13
C.2.3 Total . . . 13

D Project planning **15**
D.1 Roadmap . . . 15
D.1.1 Updates . . . 16
D.2 Project management . . . 19
D.2.1 Phases and estimates . . . 19
D.2.2 Milestones and artifacts . . . 20
D.2.3 Time evaluation . . . 21
D.2.4 User Stories . . . 21
D.3 Meetings . . . 24
D.4 Responsibilities . . . 24
D.5 Infrastructure . . . 25
D.6 Development concepts . . . 26
D.6.1 Definition of Done . . . 26
D.6.2 Review . . . 26
D.7 Backups . . . 27
D.8 Risk management . . . 27
D.9 Quality assurance . . . 29
D.9.1 Exception Handling . . . 30

E Wireframes **31**
E.1 Time Based Access . . . 32
E.2 Change Control . . . 36

F Time tracking **41**
F.1 Time tracking by milestone . . . 41
F.2 Time tracking by category . . . 42
F.3 Time tracking by project members . . . 42
F.4 Time tracking to actual/planned comparison . . . 43

Part I

Technical Report

Chapter 1

Introduction and overview

1.1 Problem definition and goals

In the Student Research Project "Cisco DNA Center Multi Tenant Manager" a PoC was implemented where a website was built that allows for multiple tenants to access a simple set of port configurations. The platform offers an Administrator to grant port specific access to user groups. This was built using the DNA platform which offers a Centralized SDN solution to address the security, mobility and automation customers require in modern campus networks. This thesis explores additional features that can be built on top of the existing functionality. The goal is to find and develop use cases to solve real world problems in the networking environment that can benefit from automation. This can show Cisco's customers the value of the modern programmability features that the DNA center enables.

1.2 Design thinking session

To find ideas for additional features a design thinking session was conducted in the first week of the project. As the name implies design thinking is a methodology originally used by designers. It enables finding new features for products and is now widely used across many industries as a tool to drive product innovation. As this methodology is internally being promoted at Cisco it is a good fit. Following is a short description of the session and the rough outline of the fleshed out ideas that are later expanded on.
Because some participants took part remotely and all participants already knew each other well and had a good grasp on the project a less formal version of design thinking was used and some steps that would usually be part of a typical session were skipped. The main focus was on the "Ideate" phase.

1.2.1 Ideate

To get a big list of ideas, everyone had 10 minutes time to list new features that sprang into their minds. As the members where physically distributed this was done digitally on a Webex Teams whiteboard. A "no bad ideas" approach is used to find as many exiting new ideas as possible.

1.2.2 Clustering and prioritization

After the timer ran out the next step was to cluster all ideas into groups of similar functional areas. Those clusters then needed to be prioritized to find most interesting groups for all participants. This was done by giving everyone 3 votes to spread across all clusters. After everyone voted there where multiple clusters that attracted most attention. The clusters with a small amount of votes where eliminated and another iteration of voting was done to find 3 clusters to focus on.

1.2.3 Whiteboards

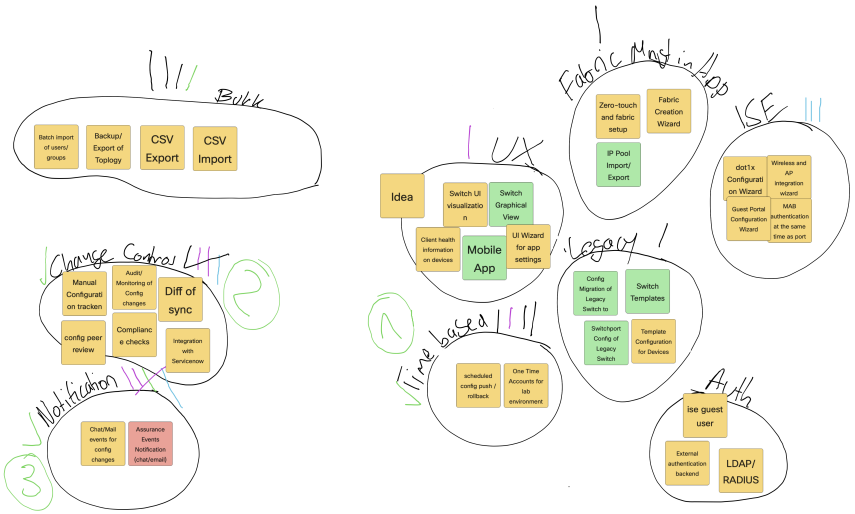


Figure 1.1: Clustered and prioritized ideas

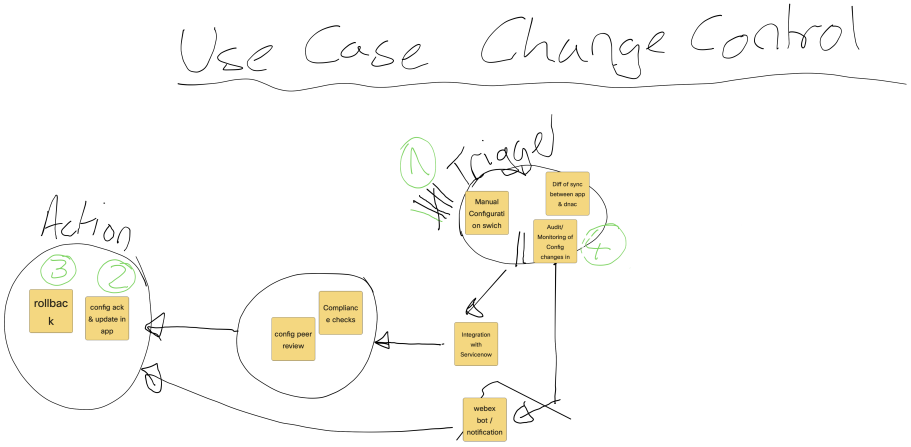


Figure 1.2: Refined change control flow

1.3 Scope and limitations

Network/Server infrastructure setup The setup and configuration of the network infrastructure including the DNA Center and fabric is in the responsibility of the Institute for Networked Solutions (INS) staff. The server infrastructure is provided as virtual machines that can be accessed, only the configuration of the applications running on the server is a responsibility of the thesis authors.

DNA API interraction As many API endpoints used are still unofficial and subject to change after any update the application is written for one specific release and not supported on other releases. Updates require manual testing of any DNA Center interactions and are not tested automatically.

Functionality The application is still being developer as a PoC, so functionality is to be preferred over interface polish and usability.

1.4 State of art

In the current DNA version (1.3.3.x) Role Based Access Control (RBAC) is not yet available. However starting with the upcomming release of 2.1.1.x some RBAC features are set to be added including some of the features covered in the term project thesis.

While some configuration change and compliance solutions in the networking domain exist there is no modern solution that fully integrates with the DNA center platform.

Time limited access features that offer sending users access links are implemented in many different apps nowadays. However no application specifically for interface configuration including a rollback feature exists for the DNA center.

Chapter 2

Requirements

2.1 Use cases

The Use cases are divided into two main groups: Time Based Access and Change Control. They share most of the same actors but broadly are functionally separated from each other.

2.1.1 Actors

System Administrator

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

Manager

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.

Guest

Guests are temporary users (e.g. from an event/lab session) with time-based access only.

Network Device Administrator

Any entity that has the ability to make changes on an edge device. This can include a user accessing the switch over the terminal or changes made by sources like Network Configuration Protocol (NETCONF) and the DNA center.

2.1.2 Time Based Access Scenario

The use cases listed below are either new use cases or existing use cases that have to be functionally altered based on new functionality.

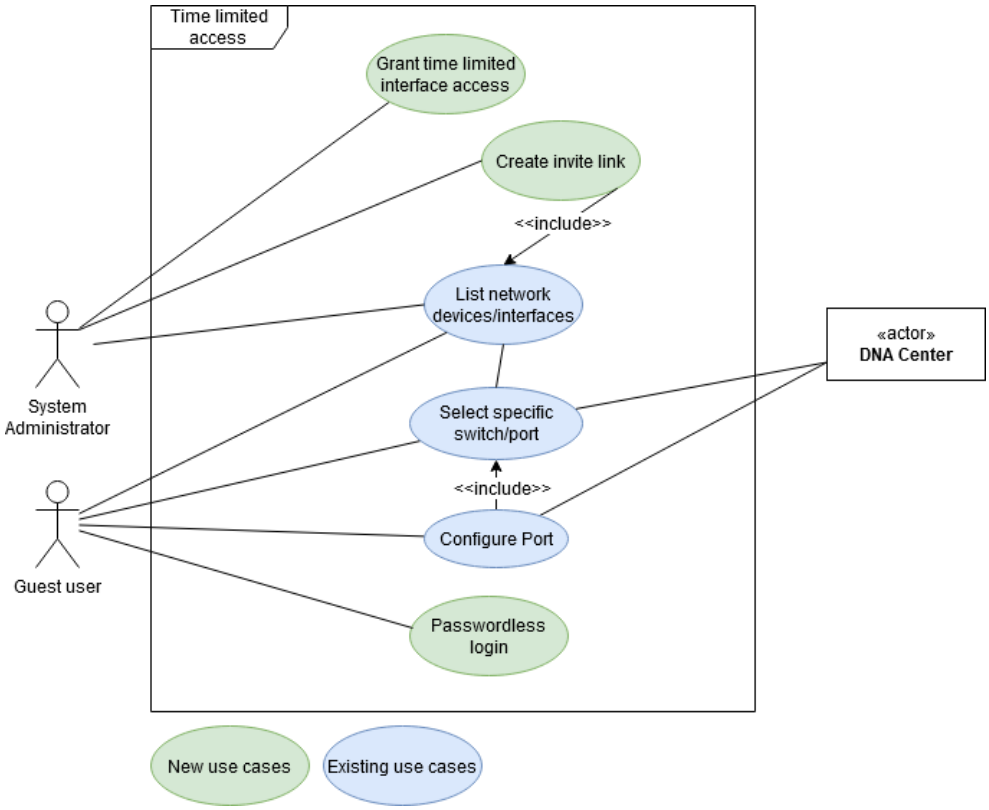


Figure 2.1: Use case diagram for time based access scenario

Use Cases

UC01 - Grant access

A System Administrator grants time limited access to a Guest user.

UC02 - Create invite link

A System Administrator creates an invite link.

UC03 - Guest access

A Guest user accesses the platform using an invite link.

UC04 - Guest interface list

A Guest user lists configurable interfaces.

UC05 - Guest interface configuration

A Guest user configures an interface.

UC06 - Schedule configuration Rollback

A System Administrator schedules a rollback for device configuration after Guest access expires.

2.1.3 Change Control Scenario

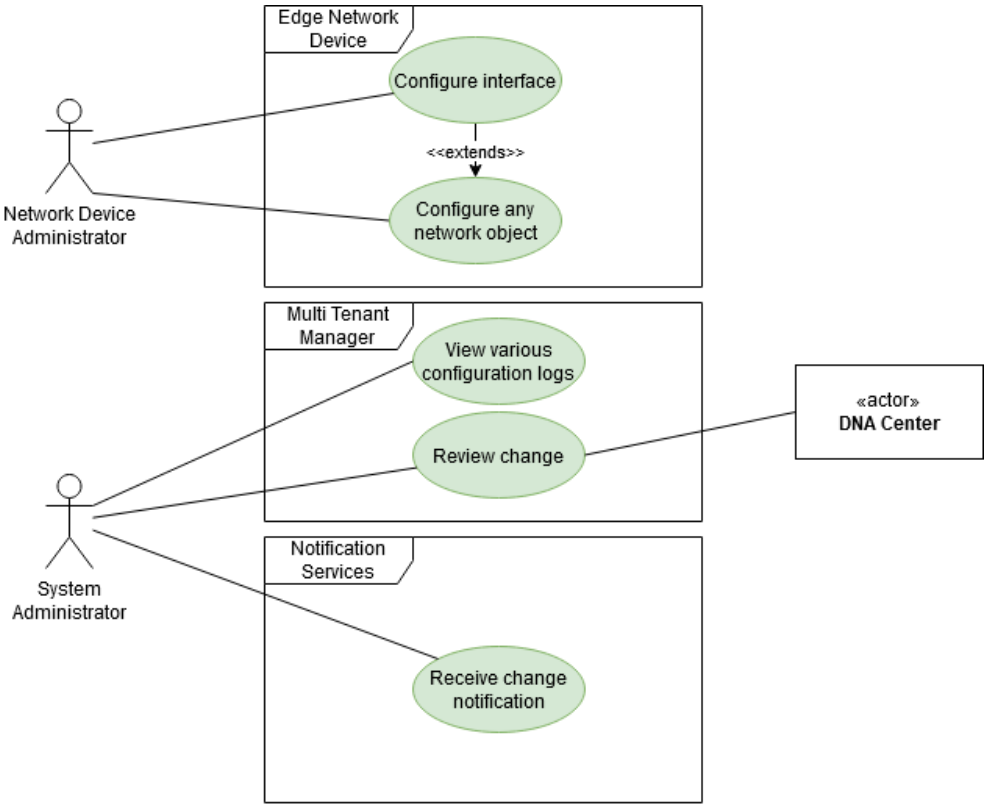


Figure 2.2: Use case diagram for change control scenario

Since all actions in the use case diagram require authentication it is left out in this diagram.

Use Cases

UC07 - Detect change in interface configuration

A user configures an interface on an edge device. The change is detected by the Multi Tenant Manager.

UC08 - Review change

A System Administrator reviews a change.

UC09 - Approve/Deny change

A System Administrator approves or denies a new interface configuration. If it is denied it is automatically rolled back

UC10 - Configuration change logs

A user views various device configuration logs.

UC11 - Get Notification

A System Administrator is notified about a device configuration change.

2.1.4 Optional Use Cases

Some optional use cases where defined that either add to the usability of the frontend or add interesting functionality. The use cases are ordered by priority based on interest and estimated implementation time.

1. **App/System Configuration**
   Configure various access parameters in the frontend instead of using environment variables.
2. Review of Changes in App
   The ability to review changes on the frontend with a more structured view.
3. **Regex Filter**
   Filtering of change notifications by setting a regex filter.
4. **Manual Configuration Snapshot/Rollback/Differentiating**
   Snapshots of interface configurations including a rollback mechanism to a specific state.
5. **In-App Configuration of ServiceNow, Webex Bot**
   The ability for users to configure their preferred notification providers for changes.
6. **Compliance Rules**
   Set custom compliance rules with automatic actions attached for configuration changes.

2.2 None functional requirements

The NFRs are defined and tested to ensure that multiple quality categories of the application are ensured.

#	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	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 [1]). 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?
NFR8	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 [11]

Chapter 3

Architecture Updates and improvements

This project started as the continuation of a Student research project (SA). Before implementing new features the architecture is refactored to account for the growing number of features that are to be added. Additionally open improvements defined in the previous project are considered for implementation. To ensure the maximum possible support time, programming languages, libraries and other technologies are updated to the most current stable release.

3.1 Original Architecture

The previous architecture was split between three main components:

- **Frontend**: React/Typescript application
- **Backend**: Django-Rest/Python application
- **DNAC Synchronization**: DNAC API client module, responsible for all DNAC interactions

As the new use cases are primarily focused on the existing architecture, it is not necessary to fully switch or rewrite one of the main components. However there some parts can benefit from general improvements.

This includes the programming languages themselves, frameworks, CI/CD and especially the DNA Center Platform. Furthermore there are possible improvements that can be implemented to further strengthen the quality of the functional base. In order to keep the amount of work efficient it has been decided that such changes are implemented before adding new functionality.

3.1.1 Possible Improvements

The SA concluded the following possible improvements in the results chapter:

- Better separation of layers: On the backend application components with similar domains of responsibility can be isolated into several Django-Apps. Separating functionality into multiple apps reduces the complexity and makes it easier to navigate through the source project.
- More testing: Most parts of the backend application were covered by automated tests. However for there currently is no testing for external DNAC API stability. Introducing such tests would save time on future API changes. Additionally frontend testing can be automated also to save time.
- Increase performance for higher datasets: It is currently possible to request all device + port information in a single request. This results in timeouts and that can be prevented with a pagination mechanism.
- Reduce code redundancy: On the frontend there are some components, which share similarities and can be made more generic to reduce code redundancy.
- UI/UX Improvements: The main focus of the previous project was on functionality. Therefore UI/UX of the application can be greatly improved.
- Use official DNAC API calls: Unfortunately, the official DNAC APIs have not allowed the respective features for the use cases. If they support the use cases without limitations, API calls can be migrated to official to greatly improve future maintainability.

3.1.2 Conclusion

The above improvements are all valid for future development. However some cuts had to be made to focus on the most important features. In the initial design thinking session, no UX use cases were identified as important. UI improvements were considered welcome after feature completeness is reached, making it a low priority improvement. Manual system testing is still sufficient enough for frontend tests.

3.2 Updates

As fundamental changes would have more impact on later development work, architecture updates were planned to be done at the beginning of the project.

Relevant technologies are updated to maximise support time after construction is completed and respectively granting access to the newest features.

3.2.1 DNAC Update

To get access to the newest features the DNA Center was updated as early as possible.

Updating the DNA Center

Before updating the DNA Center a backup was made to keep the opportunity of rolling back to an older version if changes in APIs broke features in a way that where cannot easily be ported to the new features. When backing up, the storage often ran out of space, so older backups had to be manually deleted. A useful feature to build in the future would be a simple script that periodically cleans up older backups when space runs out. To ensure that backups persist even when the DNA Center fails, the Tenant Manager webserver was configured as remote backup host.
After the first update a problem came up where switches would not apply interface configurations on edge devices. This was not immediately notices as the DNA web interface did not directly notify users about failed configuration but kept it in a pending state indefinitely. To fix this the DNA Center was updated a second time about half way into the project and all switches updated to the newest available software images witch were added manually.

3.2.2 DNAC API

Due to the updates the official Intent API endpoints where re-evaluated to check if any interaction before done using unofficial endpoints could be migrated. Additionally some unofficial API endpoints changed their behaviour and required an update in the python sync library written.

Evaluating updated DNA API endpoints

In the new version many new endpoints were added to make the official Intent API more viable. It is now possible to create and manage Virtual Networks, Scalable Groups and even Fabrics through the API. However the problem of not being able to list many of the required network objects required for interface configuration still persists. While one can create and get information about a specific VN or Scalable group it is not possible to list all usable records. This results in those endpoints only being usable if all objects are created and stored within the third party integration. If a VN is created in the DNA Center web interface there is no way to retrieve information about it without knowing its UUID.
Furthermore most interface related endpoints are still rate limited to 5 calls per minute. As only one interface can be configured at the time the endpoints are not viable for this project. In contrast to this the undocumented API endpoints allow for configuration of all interfaces of a network device in one call which to this point has not had rate limiting issues.

Updating to changed unofficial endpoints

The updates to the DNA Center caused unofficial endpoints to change. Most changes where small and required little adjustment on the Tenant Manager Sync side. The most impactful change is authentication profiles not being globally configured and usable anymore but site specific. This resulted in authentication profiles having a different UUID depending on what site a device is assigned to. As sites could be ignored in the implementation before, devices and authentication profiles need an added siteprofile UUID field in their data models. This is however not reflected on the frontend. Devices are still displayed independent of their assigned site.

3.2.3 Worker Separation

Because they are often time intensive all interactions with the DNA are done using asynchronous workers. This was achieved using Celery for the task processing and Redis for the task queue.

In the previous project it has been noted it would be a best practice to separate the Worker from the other micro services. At that point introducing worker and threads was merely a PoC, but it has been proven to work well for immediate configuration management or synchronization with DNA.

The Time Based Access use case made the advantage of having an independent worker process reasonable. In fact, rollback of configuration at a scheduled point is more reliable. E.g. On a roll out of an application update the run of scheduled tasks is less affected as downtime of a worker can be reduced greatly by either switching between different workers or if the update only affects backend code, the roll out does not affect the worker process anymore.

3.2.4 Introducing Pagination

Adding support for pagination is the best way to improve performance with a minimal amount updates. Timeouts when calling the API with requsts that would otherwise result in large responses are fully eliminated when using pagination (See load balacing tests B.4).

The implementation on the backend is straight forward. Django-REST offers pagination as configurable feature on class views.

The frontend required more complicated adjustments. The API classes and its models had to be updated from the ground up and the change impacted several parts of the UI application.

These new features make pagination with 100 objects at a time possible (20 objects are used in development).

It is worth mentioning, that a minor request from the previous project was the support of `Sticky headers`. Meaning, the headers in an UI table are also displayed (with a scrollbar) when the amount of records outsizes the screen size. This was added tohether with the pagination feature.

Another impact is the loss of frontend-based filtering and searching functionality on all objects. This functionality is only available for the currently loaded objects and depends on the pagination size.

To solve this two possible solutions were identified:

- Adding server-based filtering and searching
- Fetch all records for frontend filtering

As the server-based filtering would add a considerable amount of work and this use-case was introduced as a nice to have in the former project, the decision was to cache all records for filtering (like before). This means per default filtering is turned off when the amount of available records are greater than the currently displayed records. If an admin wants to filter more than 100 devices, it is possible increase the number of items loaded on the UI. Filtering features are enabled dynamically.

Unfortunately, because of the above impacts, it was not possible to use the built-in *Remote data fetching* capability with pagination support of the Material Table library and the implementation had to rely on customizing the API.

3.2.5 State management

Trough re-evaluation of used dependencies and reduction of redundancy, the choice of state management library was re evaluated. In the previous project there were no real benefits of using additional libraries for state management in the Frontend application. However with the planned functionality sharing states trough different components became necessary.

Possible solutions

Possible solutions came down to three basic choices:

- Using a state management library: State container for predictable state management. Most used in combination with React.
- Using custom hooks: Use React hooks to implement a solution to manipulate state.

Conclusion

Redux was implemented in parts of the application as a Prototype. The amount of newly introduced complexity and necessary migration work that came with Redux was similar to rewriting most of the UI. Alternative libraries of Redux, that promise a more forward approach (like zustand.js) were also considered. However those approaches generally use the same mechanism to manipulate states (stores with reducers and actions) and would introduce the same amount complexity as Redux considering the different states.

Finally, it was decided to use self written custom hooks and not introduce any additional libraries for the following reasons:

- The application does not require a huge amount of states.
- States are mostly loaded when a page is opened initially while they are mostly updated with API interactions.
- Learning a new technology on the side would require a larger amount of initial learning time. In favor of feature development this should be reduced to a minimum.

API Hook

To share state across components the following React Hook was implemented.

```
export const useApi = <T extends any>(usePagination: boolean = true) => {
  const [isLoading, setIsLoading] = useState(false);
  const [total, setTotal] = useState();
  const [data, setData] = useState<T>();
  const [dataList, setDataList] = useState<T[]>([]);
  const fetch = <T extends any>(promise: Promise<T>, onSuccess: (data: T) =>
      void) => {...};
  const getDevices = (page: number = 0, pageSize: number =
      DEFAULT_PAGINATION_SIZE) => {...};

  return {
    data,
    dataList,
    isLoading,
    total,

    getDevices,
  }
};
```

Listing 3.1: Shortened version of the API Hook implementation

Thanks to the Typescripts Generic this code can be reused in different components and models while still ensuring type safety. The hook holds api actions (e.g. *getDevices* above), which can then used by other components to load accessible data.

3.2.6 Context Provider

React Context Providers are useful for small states, that are shared globally. As the messaging component on the frontend is re-used on various sub-components it is a key component that was refactored to minimize redundancy.

Additionally, a Theme settings provider was introduced to support a selectable light and dark theme.

3.2.7 Refactoring

- **API Classes**: The API classes were rebuilt with a better abstraction on the base API class to remove redundancy. Additionally, error handling was fixed and more mappings were added for higher quality feedback towards the user.
- **Generalize components**: Many shared components (e.g. Material Table Dialogs), were generalized to support re-usability. Large components were split into smaller sub-components for better maintainability.

Chapter 4

Architecture and design specification

As the architecture builds up on a pre-existing project only changes and additions for new use cases are covered in this section.

4.1 External interfaces

A number of external interfaces are required in this project. While in the SA only the DNA Center was required, more interfaces where added for the notification use cases.

4.1.1 DNA Center Intent API

The limitations and uses of the DNA Center Intent API and undocumented API endpoints have been extensively covered in the SA under "3.1.1 DNA Center Intent API" and thus will not be further covered in this thesis beyond the changes mentioned in the updates and improvements chapter.

4.1.2 ServiceNow

ServiceNow is used as one of the notification services for change notifications and approvals. It is an industry leading product for enterprise workflow and operations. As it is the go to solution for ticketing and change management for many large companies it is a good fit to show how Cisco's programmability platforms can integrate into third parties. It is possible to acquire a developer instance of ServiceNow for free, this was used ins the development and demonstration of this project. The instance automatically goes into hibernation when not used for a couple of hour, and gets completely reclaimed if not used for 10 days. When the instance is reclaimed all settings are wiped and not recoverable. Because of those limitations it would be advisable to run a dedicated developer instance on an internal infrastructure.[14]
ServiceNow offers a REST API that offers full control over opening and editing of all types of change request.

4.1.3 Webex Teams

Webex Teams is a Cisco collaboration solution that fully integrates meetings, chats and conferences on one simple platform. As it is a Cisco product and enables a "chatops" integration it is a natural choice as service to use for notifications.
Webex Teams offers a REST API that enables easy implementation of chatbots. There are no relevant restrictions and costs attached for writing an implementation using the APIs.[17]

4.2 Domain

4.2.1 Domain Model

A multitude of changes are required on the old domain model in order to account for new features.

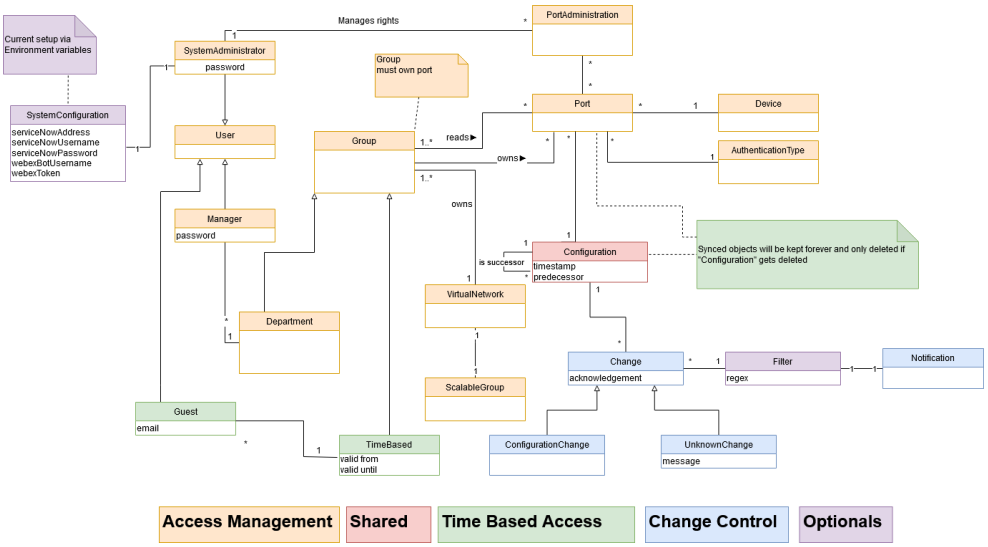


Figure 4.1: Domain model

Access management

The domain for access and user management (orange) is extended with fields for guest (green) access. Guests are part of a special kind of access group that only has access to the platform for a limited time.

Configuration history

In the old version of the Tenant Manger only the current port configuration was relevant. As the change management and guest access require rollbacks to specific states a history of configurations is required. This is represented as a chain of configurations that each point to their predecessor (red). Each interface has one currently active configuration associated to it.

Notification

Changes can trigger a notification. Notifications can have a regex filter associated to them (optional).

4.3 Technologies

The following chapter gives a roundup and overview of updated/currently used programming language and frameworks used.

4.3.1 Programming languages

The previously used programming languages remain to be used and are updated to their respective latest version. Moving from Python 3.7 to 3.8 has not had any direct impact on backend code as the newly introduced features were minimal [12]. However, the upgrade theoretically promises a performance boost.

On the opposite, the migration from Typescript 3.6 to 3.8 introduced new syntactic features. Especially the introduction of *Option Chaining* and *Null Coalescing* has made the code more readable.

4.3.2 Frameworks

On the frontend, React has been updated to the v16.13.x to have better long term support.

A major change was the move to the latest Django version (3.0) and the according Django REST (3.10) release. The idea was to support a version as close as possible to the upcoming LTS release (v3.2) to minimize future migration work.

4.3.3 Libraries overview

Frontend: The libraries used in the frontend were all updated to its latest versions. To keep the overall complexity low, a minimal amount of additional dependencies have been added.

Backend:

- **Whitenoise**: Used as a Django/Nginx replacement to serve static files. The change was caused by the new deployment architecture 8, but has the advantage to have a consistent static file serving configuration for production and development.
- **DRFPasswordless**: Used for passwordless authentication as part of the Time Based Access use case.
- **Channels**: Used for Websocket support as part of the Change Control/Notification use case.
- **Webex Teams SDK** Used for webex notification support

4.4 Design overview

4.4.1 Frontend

The frontend is separated into multiple React components. Most of the components were further improved for reuse-ability and reduction of complexity.

Structure

The following shows a simplified updated overview of the component structure:

Tenant Manager Web

```
src
└── index.tsx
    ├── api
    ├── resources
    ├── models
    ├── router
    ├── services
    ├── utils
    ├── components
    │   ├── App.tsx
    │   ├── common
    │   └── main
    │       ├── Main.tsx
    │       └── MainStyles.tsx
    └── config
```

Table 4.1: Updated frontend project structure

Components are separated into their own folders and styles are separated from main components to improve readability. A new folder *services* has been added, including custom hooks and contexts. Also the initial *serviceWorker* settings, which came with the project have been re-added to allow Progressive Web Apps (PWA).

4.4.2 Backend

The backend has been refactored to a new API specification with better naming and improved separation of applications.

Structure

The following overview shows how backend components are seperated:

Tenant Manager API
tenantmanager ├── apps │ ├── account │ ├── application │ ├── authentication │ ├── core │ ├── network │ └── notification └── dnac

Table 4.2: Updated backend project structure

The previous (mostly standard Django structure) was refactored to accommodate separated applications. The application names reflect the public api endpoints as the endpoints already separate the level of responsibility, reducing external dependencies.

Each application has their own set of serializers and views, organized in different files and directories.

Following a list of apps and their responsibilities:

account Managing users, groups and guest access.

application Log history or change system preferences.

authentication User authentication with JWT and passwordless authentication.

core Shared functionality used across apps.

network Anything that applies to DNAC; synced objects, configuration management, filter management.

notification Notification providers and functionality to acknowledge changes.

dnac Logic for any DNA center interaction (previously located in a separate module, relocated for easier maintainability).

4.4.3 Layers

The fundamental layer separation remains structurally unchanged. However additional features and refactoring have added multiple new components and dependencies between backend layers. The descriptions are copied and modified from the previous project.

Controller maps API calls to corresponding views.

Application handles API calls and error handling towards external callees.

Domain contains the main logic and data classes for the applications. Business logic is mainly contained in the serializers while the data is represented in the models, not every Djano app has serializers or Models associated to them.

Services communication services used by higher layers like communication with API endpoints and notification services. Renamed from "Utils" to better reflect its functionality.

For comparison both the old and new layer separation with their respective packages are depicted in following layer diagrams.

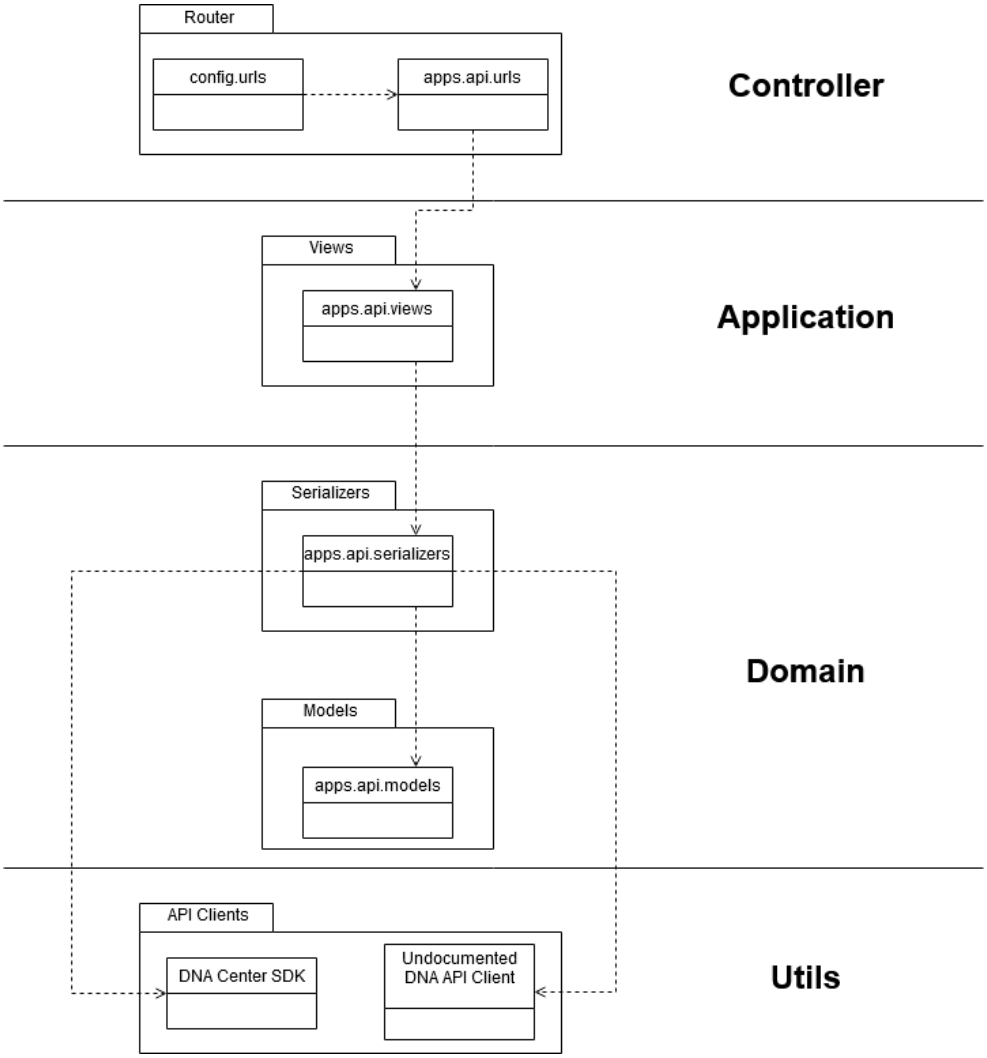


Figure 4.2: Old backend layers

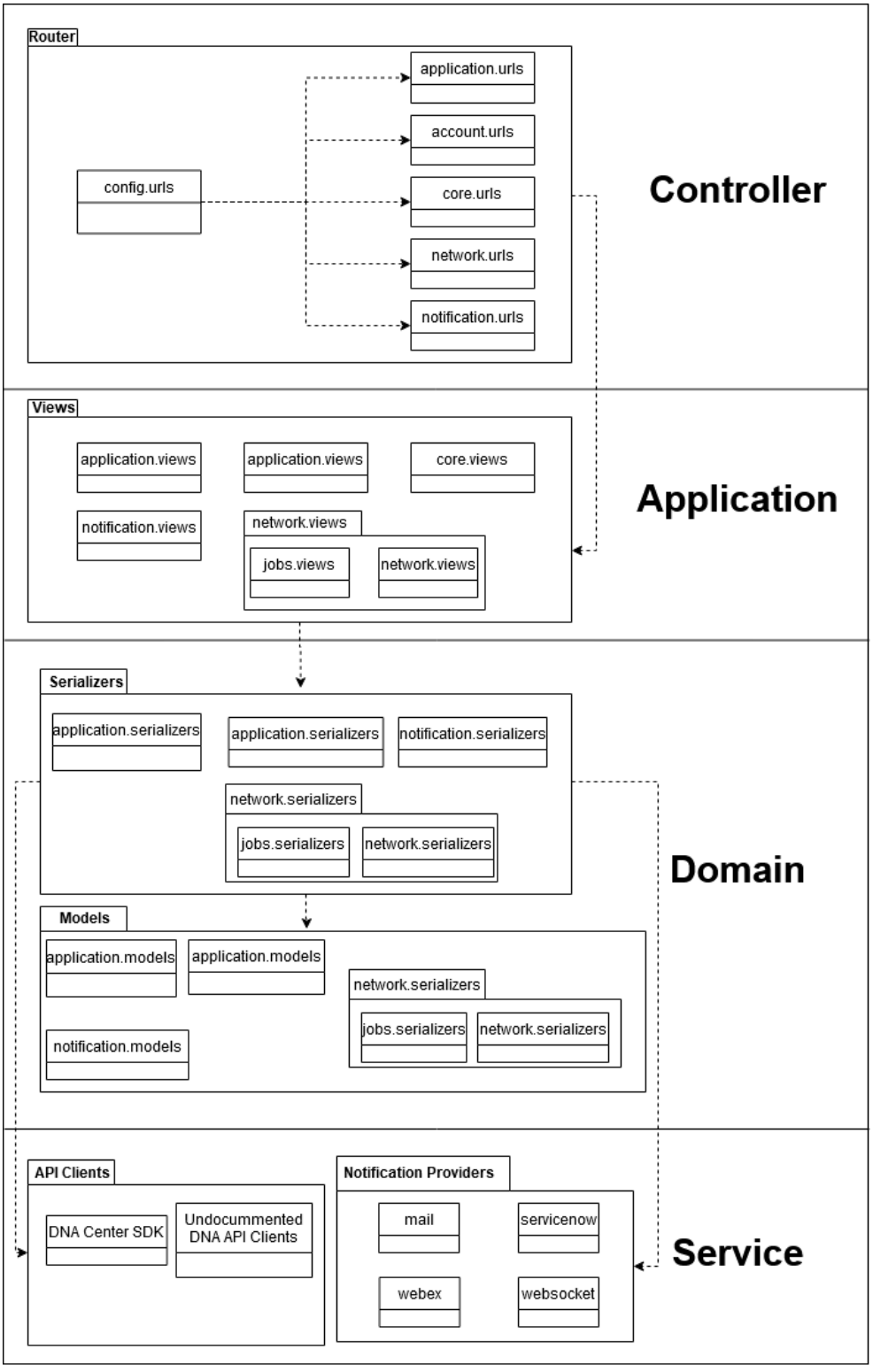


Figure 4.3: Updated backend layers

4.5 Containerization

As the NFR of supporting docker images (See 2.1) was transitioned from the previous project and is still valid, the architecture for the deployment was improved and updated to accommodate for additional requirements.

Newly introduced deployment changes (Separate Worker container, Traefik as Reverse Proxy, Graylog for log management) are all container based. New settings configuration was defined trough environment variables, to keep the application Twelve-Factor[16] compliant.

Chapter 5

Time Based Access

The following chapter describes the implementation of Time Based Access relevant Use Cases (UC01-UC06).

5.1 Passwordless Authentication

Any user facing functionality in the time based access category depends on passwordless authentication. Therefore this core feature was implemented first.

Possible solutions

In order to support passwordless authentication in Django REST, two main options are available: Either use a third-party library or create a custom Django authentication backend. The latter would be much more time consuming not only for implementation, but also for future maintenance.

So an authentication backend with the following features is necessary:

- Python 3.8, Django 3.0 and optional Django REST 3.11 support
- **Support for token based authentication**: A user should be able to connect to the system with a pre-generated secure token.
- **Mail support**: Receiving an invite link with the token for authentication would be the most forward way, as the Mail backend in Django already has been setup.
- **Maintainment**: The library should be actively maintained and upgraded to new Django versions in a timely manner.

With the above requirements the following solutions were considered to be feature complete:

- **drfpasswordless**[8]: Has the most users and is maintained well. Additional support for mobile messages and Django REST endpoints are available. Supports customization of templates.
- **django-tokenauth**[5]: Very simple approach with customization support and rate-limiting. Less actively maintained.
- **django-mail-auth**[6]: Supports custom user models and mobile messages. Well maintained.

Conclusion

All libraries are good choices, but only drfpasswordless included Django REST endpoints out of the box. The other solutions require additional work such as creating custom serializers/views etc.

`drfpasswordless` provides the best customization support and is more actively maintained than the other libraries.

Result

As a result, passwordless authentication was implemented with **drfpasswordless**. Templates (HTML, plain) were customized to have consistent notifications. Additional support of mobile messages was disabled and token expiration window adjusted.

JWT Support

As **drfpasswordless** comes with its own authentication backend, a new authentication system was added.

The system uses application tokens on successful authentication. However, as the frontend was designed to use JWT, the new authentication system does not work with JWT. Downgrading from JWT to a simple token based authentication system is not a viable option. Therefore JWT was integrated in drfpasswordless. Fortunately, the library's serializer and token generator are customizable.

A serializer for JWT tokens and a custom token generator were implemented. The generator uses the JWT functions provided by *djangorestframework-jwt*.

5.2 Permission management

In order to create guests and assign permissions on different network objects, the pre-existing permission management is used. This is possible as the domain regarding assignments has not changed from the last project.

The existing solution uses groups for permission references and a user profile for group selection. Therefore straight forward approach is to extend group functionality with time-based attributes.

Rollback and maintenance tasks

In order to allow for Rollbacks of configurations created by guest users, a scheduler is necessary. The configuration system is built using a Celery worker for configuration jobs, scheduling can be implemented by using the the built-in periodic tasks functionality from Celery.

The following tasks were added:

- **Account maintenance**: Runs periodically at 1:00 AM. Sends account notifications for guest users if the account access expires in the next 7 days. Also activates/disables guest accounts based on their assigned time based access attributes.
- **Network maintenance**: Runs periodically at 3:00 AM. Runs scheduled rollbacks based on the initially created jobs. The configuration to rollback is chosen by the last known configuration state referenced by the Time Based Groups assigned device ports before the access started. If no such configuration state can be found, the configuration of the device ports is cleared to no configuration (Fallback).

5.3 Result

The following workflow explains the guest access and creation process implementation in detail:

1. An API Call to create a new Time based group is made. The Call includes the group name, a from and to validity date and a list of guest emails.
2. If the request was valid, the backend generates guest users based on the emails with random usernames. Guest users are assigned to the GuestAdministrator role and cannot being promoted to other roles. As a constraint, duplicate usernames and emails are not allowed.
3. A notification with a login URL is sent to the newly created guest users. If the start validity date is in the future, the guest account remains disabled until the access begins.
4. On each time based group creation a RollbackJob will be automatically created. Making sure that when the guest access expires the rollback of configuration is conducted.
5. After success full creation the System Administrator can update the time based groups permissions to network objects. It is possible to update time based access attributes or delete the group which triggers an immediate Rollback.
6. To make the invitation of guests more secure, the initial token has an expiry timer. If a guest token has expired a guest can request a new token via email by himself.

Chapter 6

Change Control

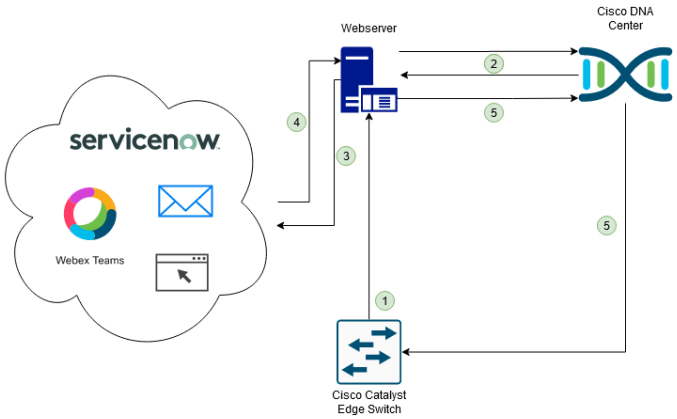


Figure 6.1: Final change control flow

The change control epic can be split into three smaller groups of user stories:

- Change detection
- Notifications
- Change approval and rollback of configuration

This section goes into detail on the different problems and solutions for the implementing each of those categories.

6.1 Change Control Activity Diagram

The implemented change control system can be represented by following activity diagram:

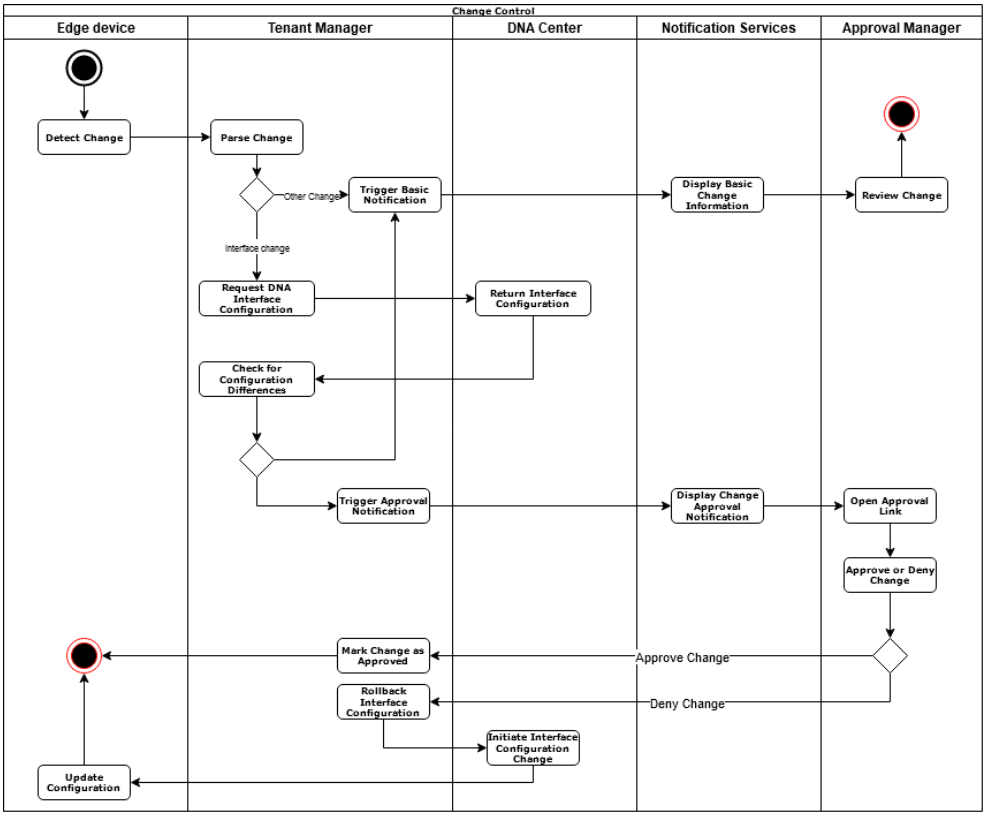


Figure 6.2: Change control activity diagram

6.2 Configuration change detection

6.2.1 Change detection methods

Detecting configuration changes made on edge switches is the core challenge of the change control system as it provides the source information for any following steps. In order for the resulted change data and notification to be use full it needs to meet a number of requirements:

- Changes must be detected in a timely manner so that any action taken by change approvers can be executed as quickly as possible. Ideally they do not have to be polled but are triggered as soon as the change happens.
- A before and after state of the device configuration needs to be obtainable so that a difference of the changes can be displayed.
- Changes should be detected on a per session base so that no additional work has to be done to merge multiple change events to a full session.
- (Optional) Detect the source (CLI, RESTCONF, DNA Center etc.) and user that initiated the change. This is not required but might be useful for future extensions of the app.

To assess the different options to achieve the required functionality a PoC was built for viable options.

Polling changes

One of the most forward options for change detection is periodically polling running configuration on switches directly from the Tenant Manager web server. Configuration differences can be computed by saving the last polled configuration and comparing it to the new one. As this approach does not support session detection and information about the configuration source and is not triggered it is not a viable solution and is not further explored.

DNAC event subscription

The DNA Center platform offers a wide collection of subscribable events. Event subscriptions can be added on the web interface, a custom REST endpoint is definable that is called whenever a event occurs. This would effectively add webhook that is automatically triggered. Unfortunately the events currently covered in the catalogue mainly focus on standard IT service management (ITSM) use cases. There is no event that is triggered by direct configuration changes directly on a switch.[7]

Collection of logs using an aggregator

Cisco switches can send logs to a remote syslog server. In case of a standard DNAC fabric setup sending messages to the DNAC log is enabled by default. While already having a third party logging tool (KIBANA) installed, additional remote logging servers can be configured. On this external logging tool any changes can be detected and an event forwarded to the Tenant Manager webserver. To test the feasability of this approach a graylog instance was setup and advanced logging enabled on an edge switch.
With this approach changes can be detected reliably. However it adds the additional complexity of a logging server. Additionally there is no straight forward way to get a difference between old and new configurations using this approach and not only configuration changes are raised but also other events/commands conducted on the devices. Furthermore logs are not created on a per session basis but as each input happens.[10][3]

IOS Change Notifcation and Logging

As the name implies the Change Notification and Logging feature specifically covers logging of new configuration changes. Changes are detected on a per session basis and contain user information and Notifications so a remote syslog can be enabled.
This approach fulfills most necessary requirements, and works reliably in the PoC. However there is no easy way to get a difference between old and new config as only new config is written to the syslog. Furthermore as well as the standard logging solution the complexity of an external logging server has to be added.[4]

Embedded Event Manager (EEM)

The Cisco IOS Embedded Event Manager (EEM), which offers real time event detection directly on supported Cisco devices, has been a part IOS for the past decade. There are multiple way to add actions based on EEM events. The recent addition of the Guestshell adds the possibility of handling EEM events in Python scripts running on a virtualized Linux environment. This opens supported devices up to a wide range of programmability options.[2]

A Cisco DevNet sample python script is already publicly available that publishes configurations changes to specified Cisco Webex (Cisco Spark at the time) chat.[13] This implementation can be used as base for a implementation of the required use case. Rewriting the python script to write changes to a file proved the functionality of this script.
This approach checks all required boxes for the use case. As it is event driven changes can be detected and acted on immediately. The ability to use a python script enables direct communication to the Tenant Manager web server which keeps the implementation and setup more simple. The biggest drawback is that enabling and configuring the Guestshell feature is required on all edge devices that are to be monitored. However this could be automated through Cisco Network Plug and Play.

6.2.2 Graylog

While not used in the final solution, Graylog instance was a useful tool for debugging. When configured to log to the remote syslog server errors and debug logs from scripts written on switches can easily be accessed. Whenever the DNA Center pushes new configuration to a switch a number of logging messages can be observed helping confirm a successful deployment. The resulting logs provided a deeper inside on how the DNA Center interacts with switches within the fabric.

6.2.3 Implementation

As it best meets the requirements the EEM based approach was selected.

Setup and limitations

EEM is available on most newer Cisco network devices.[9] However the guestshell feature only runs on fairly recent Catalyst or ISR models, see the official configuration guide for more information.[15] In order for this solution to work edge switches must have access to the network of the Tenant Manager server.

Python script and Tenant Manager endpoint

The script that is executed whenever a change is detected is a heavily modified version of the open source *Config Diff to Cisco Spark* script from the Cisco Devnet.[2]
Whenever a change event is triggered the script reads the running configuration of the device using the Python eem library that allows for running of any IOS commands inside the script. The contents of the running configuration are compared to the configuration state when the event was last triggered. The old configurations are saved in a backup file on the devices Flash, at the end of its execution the current configuration is written into this backup file. The difference between both versions are sent to a REST endpoint on the Tenant Manager web server with two fields, added configuration and removed configuration. Using the ciscoconfparse package on the web server the differences are parsed. If the change is on an interface a check is made if the current interface configuration on the DNA Center differs from the Tenant Manager. Is this the case a notification is sent with an change approval option. Is the change not on a specific interface only a change notification is triggered.

6.3 Notifications

An important decision of the architecture was to separate functionality different Django applications. This decision is especially useful for the notifications as it can be decoupled from other components.

6.3.1 Changes

Changes are stored in two ways:

- **Configuration Changes**: These changes can be mapped to an interface and are versioned as a chain of configurations referencing port configurations.
- **Unknown changes**: Fulltext changes, which can't be mapped to an interface directly.

This means that notifications can be either triggered directly when saving the changes or trough an event subscription. As calculating differences and generating actual notifications can take longer, the most stable and robust way is to use the latter. To achieve that, **Django signals** are used to subscribe to events when a notification should be triggered.

6.3.2 Result

Whenever a new change is created the respective handler is triggered. This can either be the configuration handler or the a handler for unstructured messages. For the configuration handler the following steps are performed:

- First a notification for System Administrators will be generated. All differences of port configurations between the configuration referenced in the change and between the previous configuration will be calculated.
- Then the notifications are be generated for Network Administrators or Guest Administrators (Owner or Read-Only). The calculation will be done on the previously calculations to speedup performance.
- Finally, the notifications are saved as full text to mitigate losses and recalculations.

The unknown change handler is a little different. First, each group can create custom Regex filters.

- On new changes, custom defined regex filters are be checked applied.
- If a match is found a notification will be created.

6.3.3 Notification Services

Notification services are designed as Plug-Ins and are dynamically loaded. A notification provider can be enabled/disabled in the settings and provide the implementation for the relevant notification service. This enables easy adaption of new notification services in the future.

```
class BaseNotificationProvider(ABC):
    notification_type = NotificationType.APP
    enabled = False
    template = 'notification'

    def notify(self, notification) -> None:
        raise NotImplementedError

    def acknowledge(self, notification) -> None:
        raise NotImplementedError
```

Listing 6.1: Notification Provider Interface

The above code shows the original design draft for a notification provider. Besides to the *notify()* function it is designed to support acknowledgment for all message providers except mail that can easily be implemented in the future.

Acknowledging changes this way has the disadvantage of requiring custom implementations and endpoints for each chosen technology. However, as an optional planned feature was to also allowing for reviewing changes in the Web UI itself, it was decided to only use notification providers to send messages providing a URL to review changes. This implements the optional features and allows for central approval.

Sending notifications

Notifications services can be enabled/disabled by the user. Notification providers are invoked using signals. The notifications are sent to all enabled services.

6.3.4 In App Provider

In-App notifications are sent to websockets. The implementation of Websockets is implemented using Django Channels. The frontend app subscribes to messages of the assigned primary group. System Administrators use the unassigned group id to receive all notifications. Once a new notification has been created, the backend will send a message with the type of notification (Change or Filter) to the specific channel. The frontend then updates the notification count and allows Administrators to view a details of the notification.

6.3.5 Webex Provider

The Webex provider adds the possibility of notifying the administrator about changes on Webex Teams. Using Webex Teams SDK a custom message is sent containing basic information about changes. While Webex Teams offers some message formatting using markdown the user is provided with a link to the change for a better and more detailed overview.

The provider dynamically loads a different markdown template for the message body depending on the type of message (Filtered, approval, general change)

6.3.6 Mail Provider

The mail provider functions the same as the Webex provider but renders a HTML template for the message body.

6.3.7 ServiceNow Provider

The ServiceNow provider implements the creation of a ServiceNow change request when a change occurs. The change request is assigned to a predefined administrator and displays basic information in the description field. The user is again referenced to the Tenant Manager for further information and if applicable Approval/Denial.

As Service Now change requests can only display pure text, a text template is provided to its provider.

Possible extension with approval automation

Normal change requests on ServiceNow are subject to approval by default, custom policies can be defined for specific requests. It is possible to create a trigger on the change request approval that sends an approval or denial REST message to a defined endpoint. A small PoC was developed for this feature. However this feature was not implemented in favour of adding a change approval view to the Tenant Manager directly.

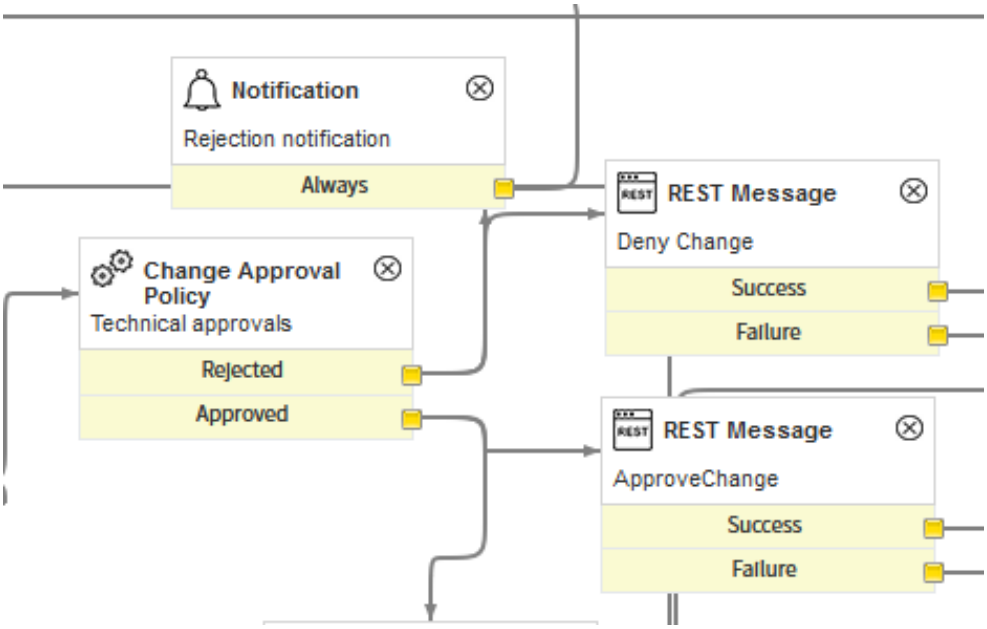


Figure 6.3: External REST calls added to the ServiceNow change request workflow using the workflow editor

Chapter 7

Results and conclusion

7.1 Achieved results

The project architecture was updated to better reflect its size and growing potential. All used technologies were updated to the latest versions.

The option to add time limited guest access groups was added to the interface configuration management. Settings made during this access are rolled back automatically. Guest users can access the site passwordless using the guest URL provided via E-Mail.

A change control system was implemented including change detection on network devices, notifications on multiple platform and the option to roll back changes on interfaces.

7.1.1 Screenshots

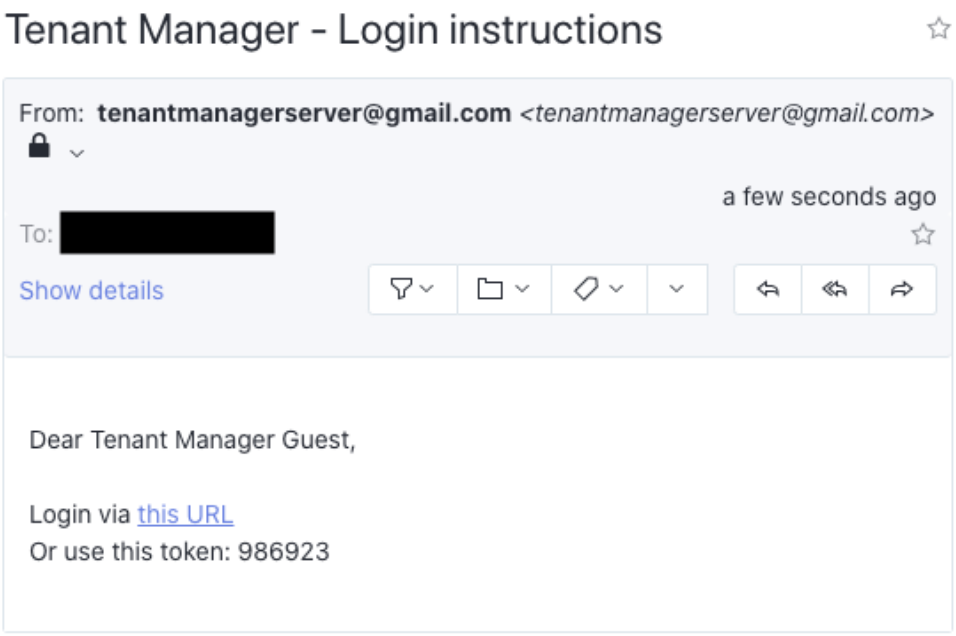


Figure 7.1: A simple invite link to access the DNA Center passwordless

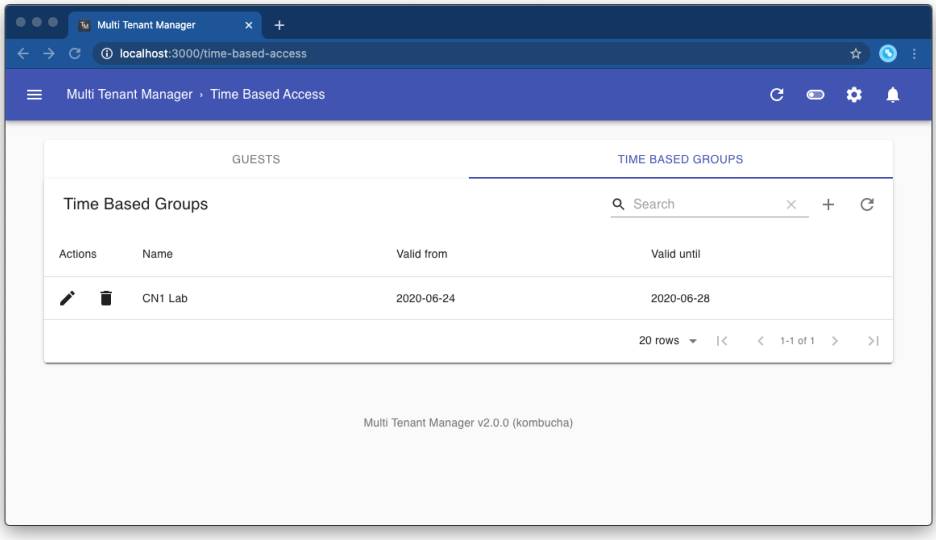


Figure 7.2: View to fully manage time based access group

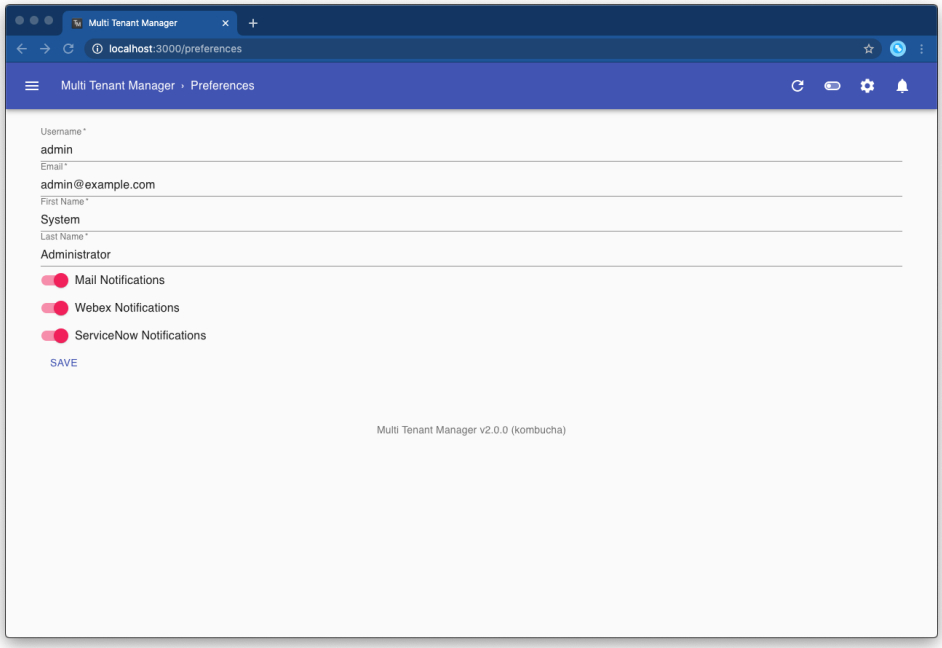


Figure 7.3: In App settings of user preferences

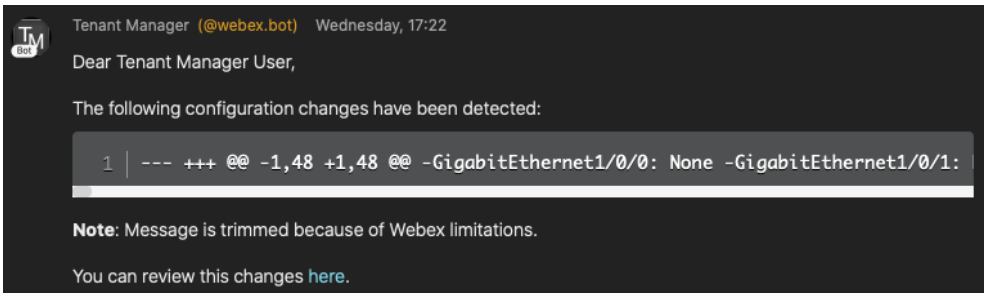


Figure 7.4: Webex Teams configuration change notification

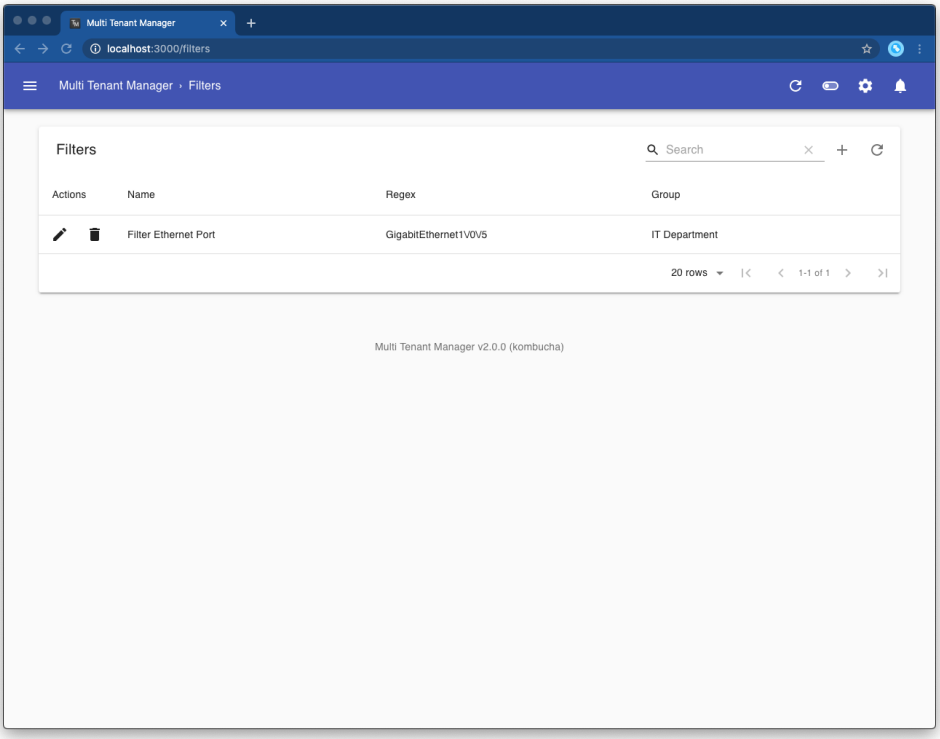


Figure 7.5: Option to filter notifications by defining a regex

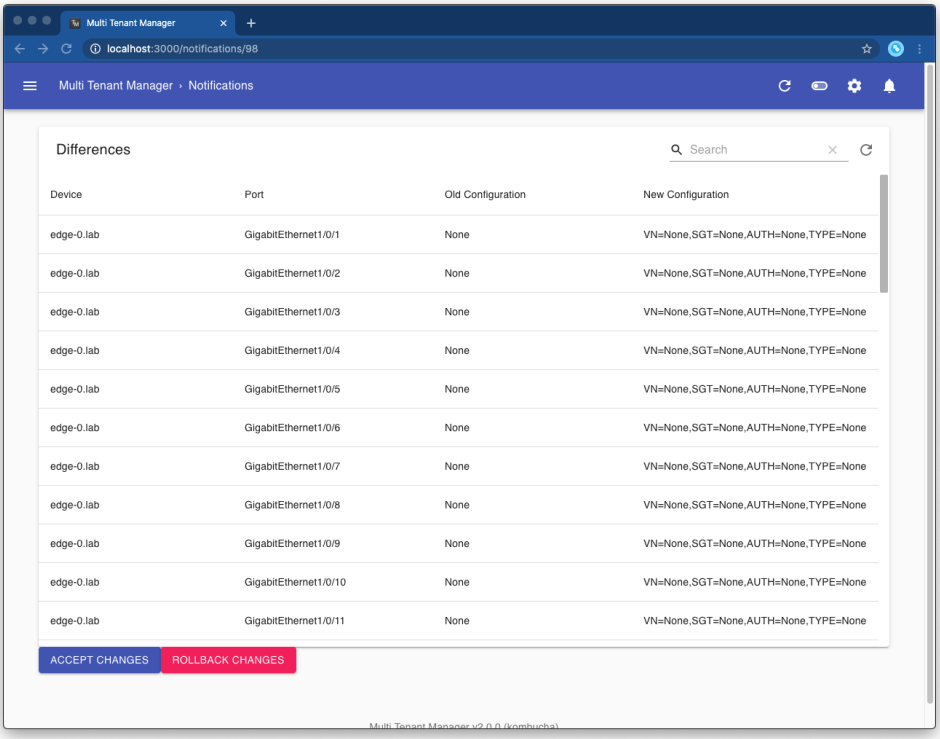


Figure 7.6: Interface change review view

7.2 Possible improvements and additions

Some improvements are possible to make the app more stable or usable.

Full ServiceNow and Webex Teams bot integration As already covered a full ServiceNow integration with automatic rollbacks of device configuration when the change request state is denied would be possible. Additionally a Webex Teams bot could be extended to roll back configuration by simply replying "approve" or "deny" to a change notification.

Improve DNA interaction While interactions with the DNA Center (specifically pushing interface configuration) where improved they are still prone for unexpected errors. Currently jobs fail and offer little information as to why. While the web interface of the DNA Center is prone for the same errors a more usable feedback is given as to why a configuration job has failed. However as these interactions are still built on unofficial endpoint it might not be worth the effort.

Previously identified changes Not all improvements and additions from the SA could be implemented. This includes legacy switch support, third party authentication backends through RADIUS and in-app creation and management of VNs, IP pools and scalable groups.

7.3 Acknowledgements

We would like to thank all involved parties that made this thesis the positive experience it was, continuing the trend after a successful SA.
It was a blessing to have Patrick Mosimann as an industry partner. It is rare that an industry partner attends almost every weekly meeting. The general input and help with the DNAC and Switch trouble shooting sessions where invaluable.

Thanks to Jessica Hilti for reacting quickly to any requests and problems we had on the infrastructure side. It was also helpful to have someone that just recently went through the same process of writing a Thesis at the University of Applied Science Rapperswil (HSR) that helped keep formalities in order and meetings organized.

Finally thanks to Prof. Laurent Metzger for making this project possible. It was great having an examiner that allowed for our own input to be included in the scope of the project. This kind of flexibility made the project feel like it's truly ours.

Part II

Project documentation

Chapter 8

Deployment

8.1 Updates

The following changes were introduced to the deployment since the previous project:

- Nginx was replaced with Traefik to support the NFR 2.1 better and unify production/development deployment. Additionally Let's encrypt support for production use was setup. The development version uses a self-signed certificate.
- The celery worker was separated into a specific container
- Graylog and additional dependencies (Elasticsearch, Mongo DB) were added for log tracking.

8.2 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
- **DNAC**: The DNAC host sends log data to the Graylog instance.
- **Server**: The server is a virtual machine, which runs on the infrastructure of the INS. Ubuntu 18.04 is used as an operating system and deployed trough Ansible roles (fully automated deployment).
- **Docker**: Docker runs on the server for Container services. HTTP/S ports are forwarded to the reverse proxy container.
- **Containers**
  - **Proxy**: Runs a traefik instance with access to Read access to Docker information. Acts as proxy for public docker containers (API and Web). The reverse proxy is configured for TLS encryption and also forces redirection to encrypted communication.
  - **Web**: Runs Reacts development web server
  - **Api**: Runs Django development/built-in web server
  - **Worker**: Runs Celery worker process
  - **Db**: Runs latest Postgres database server
  - **Redis**: Runs latest Redis broker server
  - **Graylog**: Runs a Graylog instance
  - **Mongo**: Runs a Mongo dabase for Graylog
  - **Elasticsearch**: Runs an Elasticsearch instance for Graylog

8.2.1 Production Deployment

Production Deployment shares most configuration from development, but is different in the following ways:

- **Containers**
  - **Proxy**: Traefik uses Let's Encrypt
  - **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.
  - **Graylog, Mongo, Elasticsearch**: These containers are disabled in production.

8.3 Diagrams

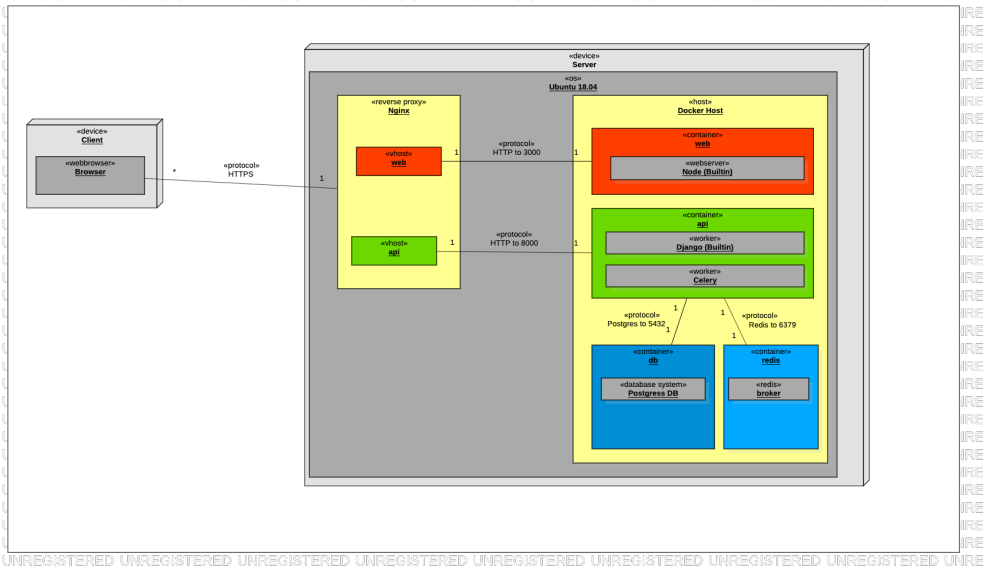


Figure 8.1: Original Deployment Diagram in Development

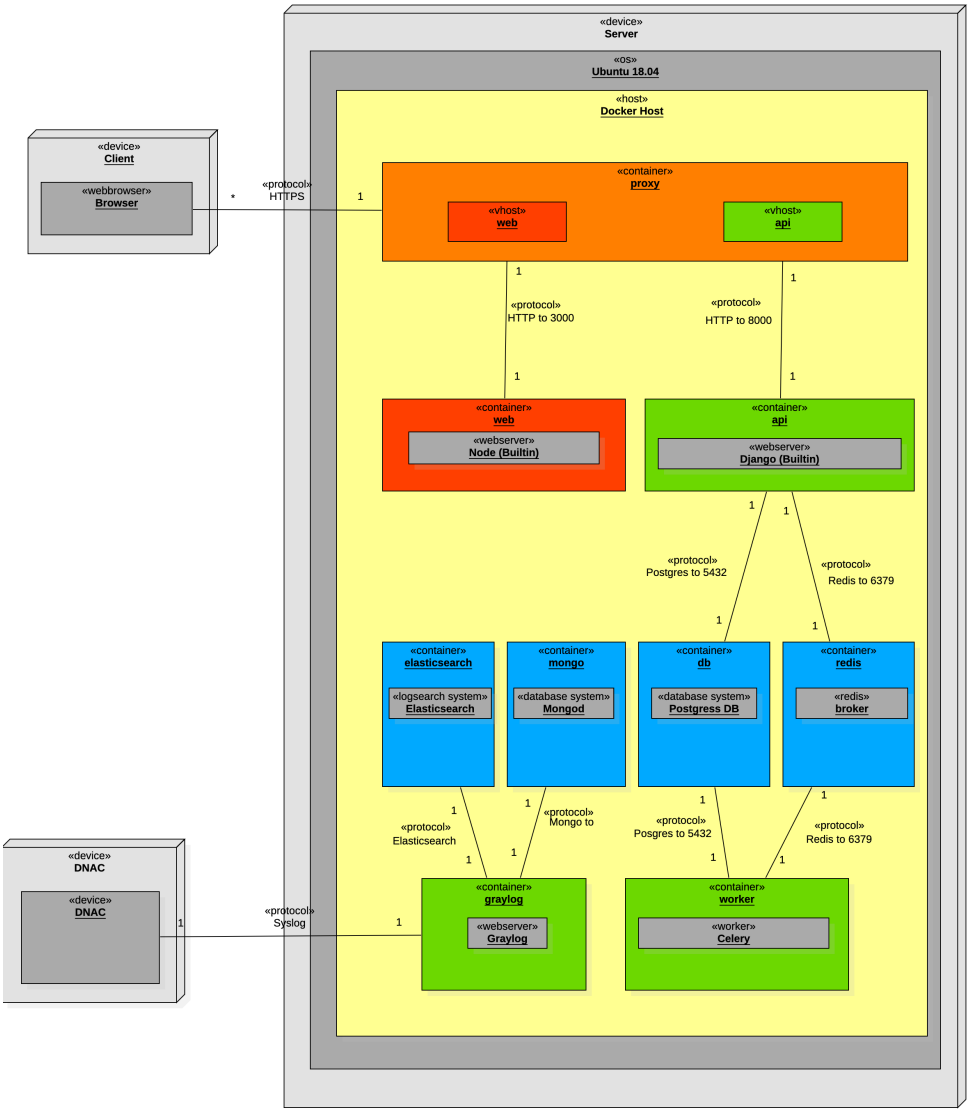


Figure 8.2: Updated Deployment Diagram in Development

8.4 Infrastructure

8.4.1 Updates

The following changes have been introduced to the infrastructure of the previous project:

- TSLint was migrated to ESLint, because of deprecation
- Time tracking automation has been added
- Continous Integration configuration has been improved to allow parallel pipelines and skipping of jobs with tags. Automatic builds for production relases have been setup on the master branch.

8.4.2 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
- **time-tracking**: Python scripts for automated time tracking exports

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

8.4.3 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.

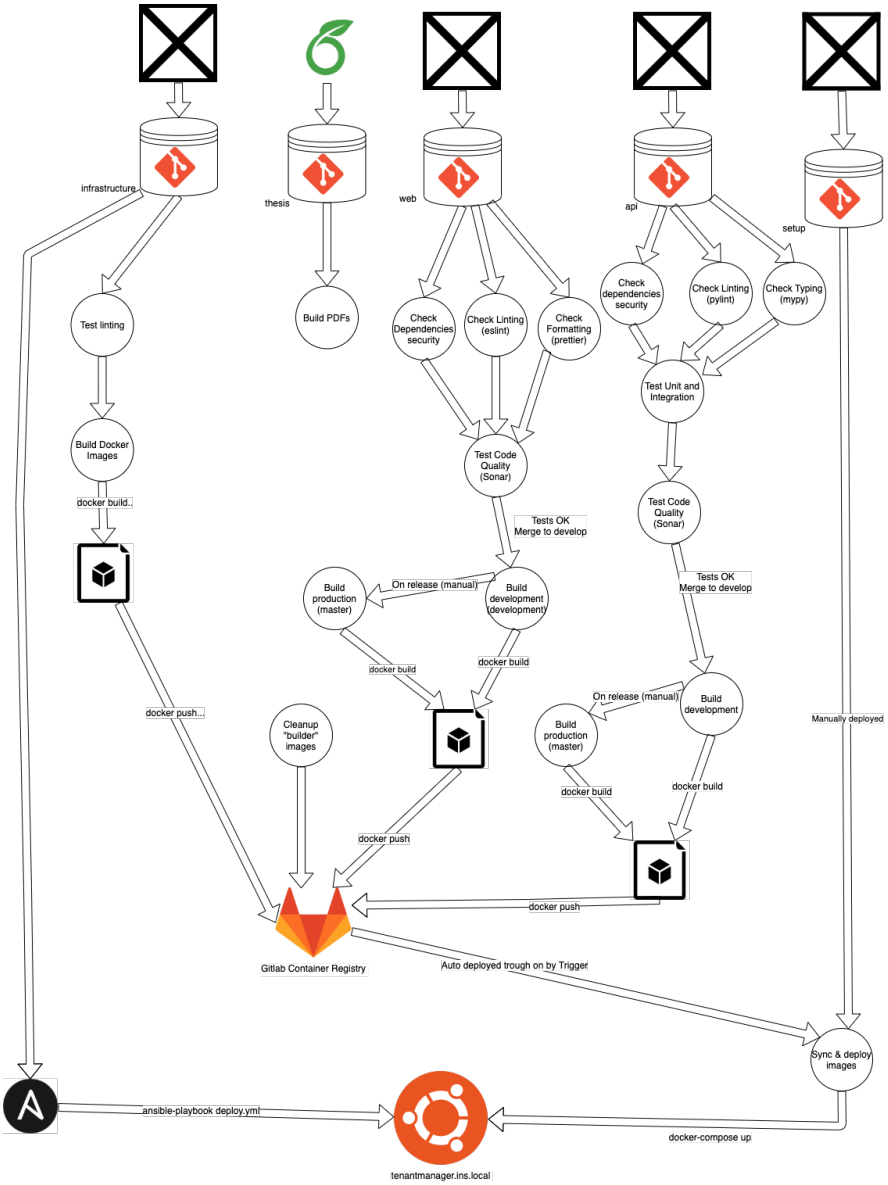


Figure 8.3: Updated CI/CD

Chapter 9

Data model

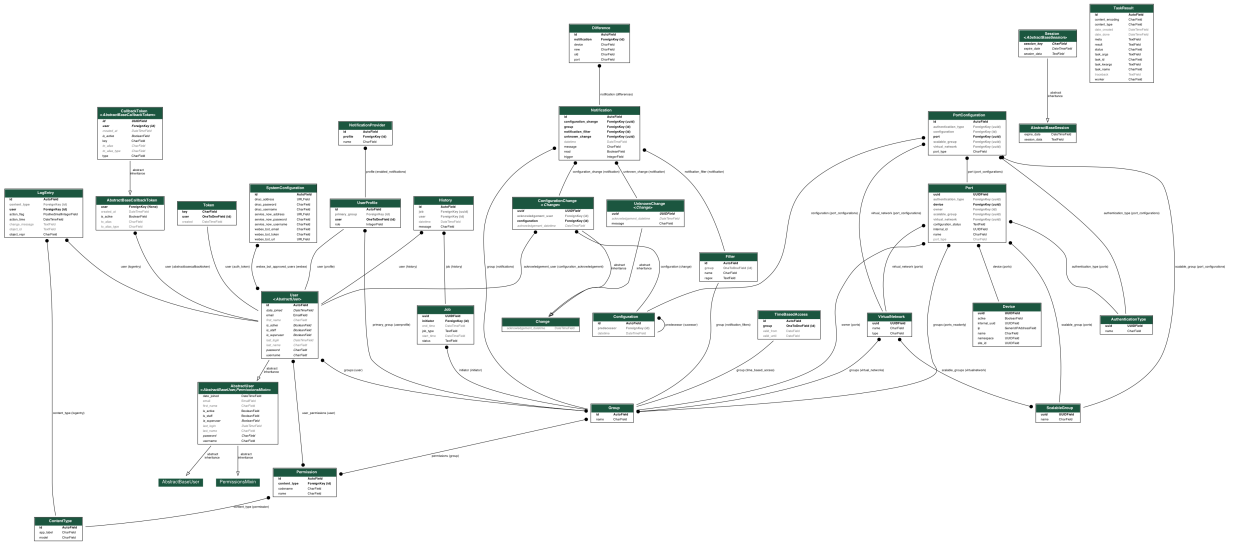


Figure 9.1: Class diagram

Chapter 10

API

10.1 Endpoints

Listed below are the most important API endpoints/endpoint groups in the application, a detailed description of each endpoint can be found on the web server:

- **/application/history/** History of user interactions on the site
- **/network/** Various endpoints for retrieving information about edge devices, VNs, scalable groups and authentication types.
- **/network/jobs/** DNA center related jobs. Includes open/closed/failed jobs.
- **/network/configurations/** Interface configurations, contains current and historic configs.
- **/network/filters/** Filters that can be applied to notifications.
- **/notification/** Endpoints for notification management and aknowledgement
- **/account/** User management endpoints. Includes normal groups and accounts and guest groups and membership information.
- **/authentication/** JWT authentication endpoint.
- **/network/sync/** Endpoint to manually trigger a network sync.

10.2 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.

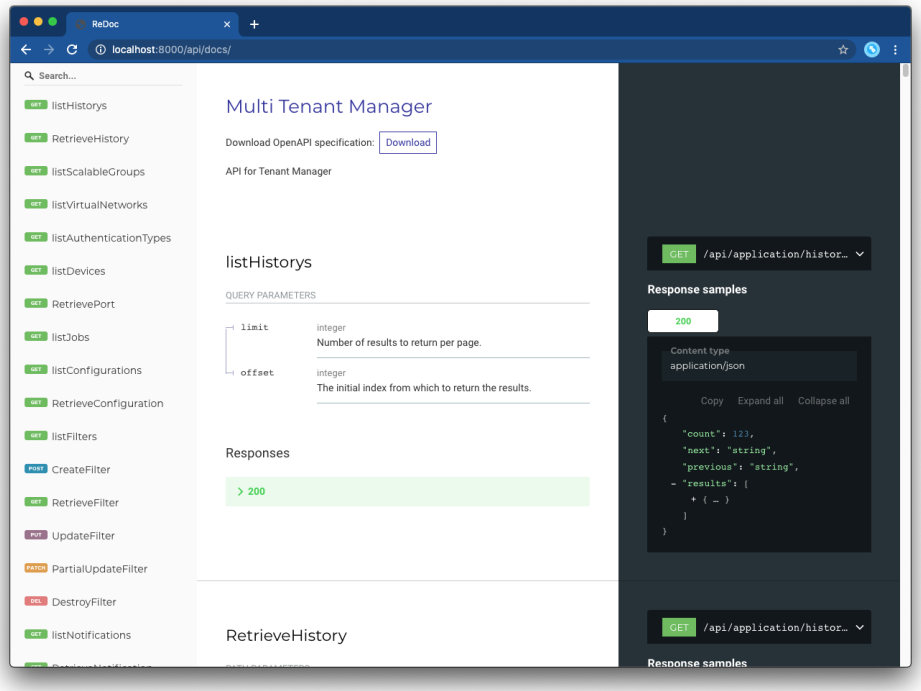


Figure 10.1: API documentation

List of Figures

1.1 Clustered and prioritized ideas 4
1.2 Refined change control flow 4

2.1 Use case diagram for time based access scenario 7
2.2 Use case diagram for change control scenario 9

4.1 Domain model 23
4.2 Old backend layers 30
4.3 Updated backend layers 31

6.1 Final change control flow 37
6.2 Change control activity diagram 38
6.3 External REST calls added to the ServiceNow change request workflow using the workflow editor 46

7.1 A simple invite link to access the DNA Center passwordless . . . 48
7.2 View to fully manage time based access group 48
7.3 In App settings of user preferences 49
7.4 Webex Teams configuration change notification 49
7.5 Option to filter notifications by defining a regex 50
7.6 Interface change review view 51

8.1 Original Deployment Diagram in Development 56
8.2 Updated Deployment Diagram in Development 57
8.3 Updated CI/CD 60

9.1 Class diagram 62

10.1 API documentation 64

B.1 Load test results without Django limits 7

C.1 Sonarqube web 11

C.2 Sonarqube api 12

D.1 Original Project Roadmap 17
D.2 Updated Project Roadmap 18

F.1 Time by milestone 41
F.2 Time by category 42
F.3 Time by project members 42
F.4 Time comparison 43

List of Tables

2.1 NFR according to ISO-9126 [11] 11

4.1 Updated frontend project structure 26
4.2 Updated backend project structure 27

B.1 NFR Checklist 5
B.2 System test scenarios 9
B.3 System test scenarios 9

D.1 Changelog Project planning 15
D.2 Responsibilities 24
D.3 Risk list 28
D.4 Quality Assurance 29

Glossary

API Application Programming Interface. ii, 68, *see* Application Programming Interface

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

CI Continuous Integration. 25, 68, *see* Continuous Integration

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

Cisco IOS Embedded Event Manager An IOS subsystem for real time event detection. 40

Continuous Integration Automated build environment. 25

DNA Cisco Digital Network Architecture. ii, 2, 5, 6, 16, 68, *see* Cisco Digital Network Architecture

EEM Cisco IOS Embedded Event Manager. 40, 42, 68, *see* Cisco IOS Embedded Event Manager

HSR University of Applied Science Rapperswil. 24, 52, 68

INS Institute for Networked Solutions. 5, 68, *see* Institute for Networked Solutions

Institute for Networked Solutions The department at the HSR responsible for teaching network related topics. 5

IT service management Managing it services inside an organization. 39

ITSM IT service management. 39, 68, *see* IT service management

Mixed Scrum Project management process based on Scrum mixed with the Unified Process model. 19

NETCONF Network Configuration Protocol. 6, 69, *see* Network Configuration Protocol

Network Configuration Protocol An network management protocol that allows network device configuration through a standardized XML format. 6

NFR None-functional Requirements. 5, 11, 67, 69, *see* None-functional Requirements

PoC Proof of Concept. i, iii, 2, 3, 5, 16, 19, 27, 39, 40, 46, 69, *see* Proof of Concept

Progressive Web Apps Web application designed to work offline. 26

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

PWA Progressive Web Apps. 26, 69, *see* Progressive Web Apps

RBAC Role Based Access Control. 5, 69, *see* Role Based Access Control

Role Based Access Control A method of restricting access to certain networking/security objects based on roles. 5

SA Student research project. 8, 12, 13, 21, 69, *see* Student research project

SDN Software defined networking. ii, 2, 69, *see* Software defined networking

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

Student research project Student research project on which this project follows. 12

VCS Version Control System. 25, 69, *see* Version Control System

Version Control System System for versioning source files. 25

Bibliography

[1] *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 06/20/2020).

[2] *Cisco EEM configuration guide and overview*. URL: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/166/b_166_programmability_cg/guest_shell.html (visited on 06/20/2020).

[3] *Cisco IOS logging configuration*. URL: https://community.cisco.com/t5/networking-documents/how-to-configure-logging-in-cisco-ios/ta-p/3132434 (visited on 06/20/2020).

[4] *Configuration Change Notification and Logging*. URL: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/config-mgmt/configuration/15-sy/config-mgmt-15-sy-book/cm-config-logger.pdf (visited on 06/20/2020).

[5] *django-mail-auth Library*. URL: https://pypi.org/project/django-mail-auth/ (visited on 06/20/2020).

[6] *django-tokenauth Libraty*. URL: https://pypi.org/project/django-tokenauth/ (visited on 06/20/2020).

[7] *DNA Center Platform Events*. URL: https://www.cisco.com/c/en/us/td/docs/cloud-systems-management/network-automation-and-management/dna-center-platform/1-3-1-0/user_guide/b_dnac_platform_ug_1_3_1_0/b_dnac_platform_ug_1_3_1_0_chapter_0110.html#id_98583 (visited on 06/20/2020).

[8] *drfpasswordless Libraty*. URL: https://pypi.org/project/drfpasswordless/ (visited on 06/20/2020).

[9] *EEM data sheet*. URL: https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-embedded-event-manager-eem/datasheet_c78-492444.html (visited on 06/20/2020).

[10] *Graylog*. URL: https://www.graylog.org/ (visited on 06/20/2020).

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

[12] *New Python 3.8 Features*. URL: https://docs.python.org/3/whatsnew/3.8.html (visited on 06/20/2020).

[13] *Sample script to publish configuration changes to Webex/Spark*. URL: https://github.com/CiscoDevNet/python_code_samples_network/tree/master/eem_configdiff_to_spark (visited on 06/20/2020).

[14] *ServiceNow developer platform*. URL: https://developer.servicenow.com/dev.do#!/home (visited on 06/20/2020).

[15] *Setup guide for Cisco guestshell*. URL: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/166/b_166_programmability_cg/guest_shell.html (visited on 06/20/2020).

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

[17] *Webex Teams for developers*. URL: https://developer.webex.com/docs/api/getting-started (visited on 06/20/2020).

Part III

Appendix

Appendix A

Personal reports

A.1 Aaron Meier

I was very happy to be back working on the Tenant Manager v2 (to keep the name short). There were some interesting optional features at end of last semester that I'd really wished to finish, but haven't had enough time. The project itself was very demanding and exposed a lot of possibilities in different areas (Network- and Software Engineering, Design etc.). Starting the project with a Design Thinking session was a huge advantage for us to directly collaborate on new ideas. I was really surprised with so many great ideas that came up in such a short time. With the refinement and wireframing of the ideas it soon became clear, how huge the scope of the project was. More custom features were demanded than ever, the official DNAC APIs still weren't enough and new updates introduced breaking changes (not that we haven't already had enough challenges with the impact of COVID-19 and other delays). I had to push back my side-projects in order to fully concentrate on the project. Looking back, I think it was more difficult to work on an existing solution rather than starting a project from scratch. However, it was very cool to learn and work with new technologies. And mixing software with network engineering was already fun to work in the last project.

Altough most meetings took place remotely, it was never an problem to reach someone in case of an issue. Meetings were generally very productive thanks to the excellent support on technical issues and the open feedback to questions/decisions.

A.2 Dennis Ligtenberg

This thesis was undoubtedly written during a special time. Just when the project started to get rolling the entire status quo was completely changed by the COVID-19 lockdown. For a week the project stood still awaiting for the HSR and ourselves to adapt to the "new normal". While the change heavily impacted my daily life, the transition to work from home at the thesis was surprisingly smooth. Webex was already used for the meetings and all required infrastructure already had to be accessed using a VPN anyways. The HSR displayed unexpected flexibility by extending the due date for the thesis.
As for the project, working with the DNA Center and network automation in general was immensely interesting and educational again. It is obvious that this approach is the future of networking and I am exited to see what comes of it even if I don not end up in this industry after graduating. Updating and reverse engineering the unofficial API did not get any less frustrating compared to the SA, but routine at least made it less time consuming.
Once again it was a pleasure to work with all involved parties. The meetings always felt productive and positive. Having a external industry partner that not only shows up at the beginning and the presentation of the project is a great experience. Having most people attending the weekly meeting helped with the gathering of productive feedback and support when problems arose. To conclude this once again was a very positive experience this time in a very special setting.

Appendix B

Testing

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 a PoC, this scenario allows more development time on features.

B.1.1 Results

There were 72 integration and unit tests defined, which resulted in 65% code coverage.

```
---------- coverage: platform darwin, python 3.8.3-final-0 -----------
Coverage HTML written to dir htmlcov
Coverage XML written to file coverage.xml

72 passed in 155.28s
```

Listing B.1: PyTest results

B.2 Frontend

As already defined in the previous project, the 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

#	Description and acceptance criteria	State	Notes
NFR1	Only authenticated users should have access to the system. Is authentication required?	Fulfilled	Any views that should not be public are protected, protected views can only be accessed with a JWT token
NFR2	Passwords should be protected trough modern measures. Are all passwords stored securely by default (Hash + Salt)?	Fulfilled	The Django authentication backend encrypts password with a Hash and Salt
NFR3	Communication between the user and the system should be encrypted. Is the latest TLS-encryption supported?	Fulfilled	The Traefik is configured with a Let's encrypt certificate. Any traffic between the user and the gateway is encrypted by default
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?	Fulfilled	A basic error notification is given when a sync fails. On sync error the configuration falls back to the state of the DNA Center,
NFR5	Failure of configuration changes should be traceable for administrators. Are configuration failures obvious in the systems log?	Fulfilled	Errors are logged in a system log and on the frontend.
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?	Fulfilled	Pagination ensures a timely response to any request to the server, load tests result in responses under 10 seconds

NFR7	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 [1]). 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?	Fulfilled	Through pagination the system can handle a large amount of network entries
NFR8	The systems installation should be straight forward and deployable trough Docker containers. Are containers defined for all software components?	Fulfilled	All infrastructure components are fully containerized and can be easily deployed.

Table B.1: NFR Checklist

B.4 Performance tests

In order to test the full fillment of the NFR8 (System should scale well on typical campus networks) with the pagination functionality. The load test from the previous project have been updated and reused.

B.4.1 Scenario

As a testing scenario the following has been choosen:

- **POST authentication**: Authenticate as system administrator
- **GET devices**: Load all available device ports

The same data limits of the load tests from the previous project are used with generated sample data (500 devices with 24 ports each) on Postgres. SQLite tests were skipped as previous runs have shown limitations. Assignment of configurations were not changed trough pagination support and therefore out of scope of for the load tests.

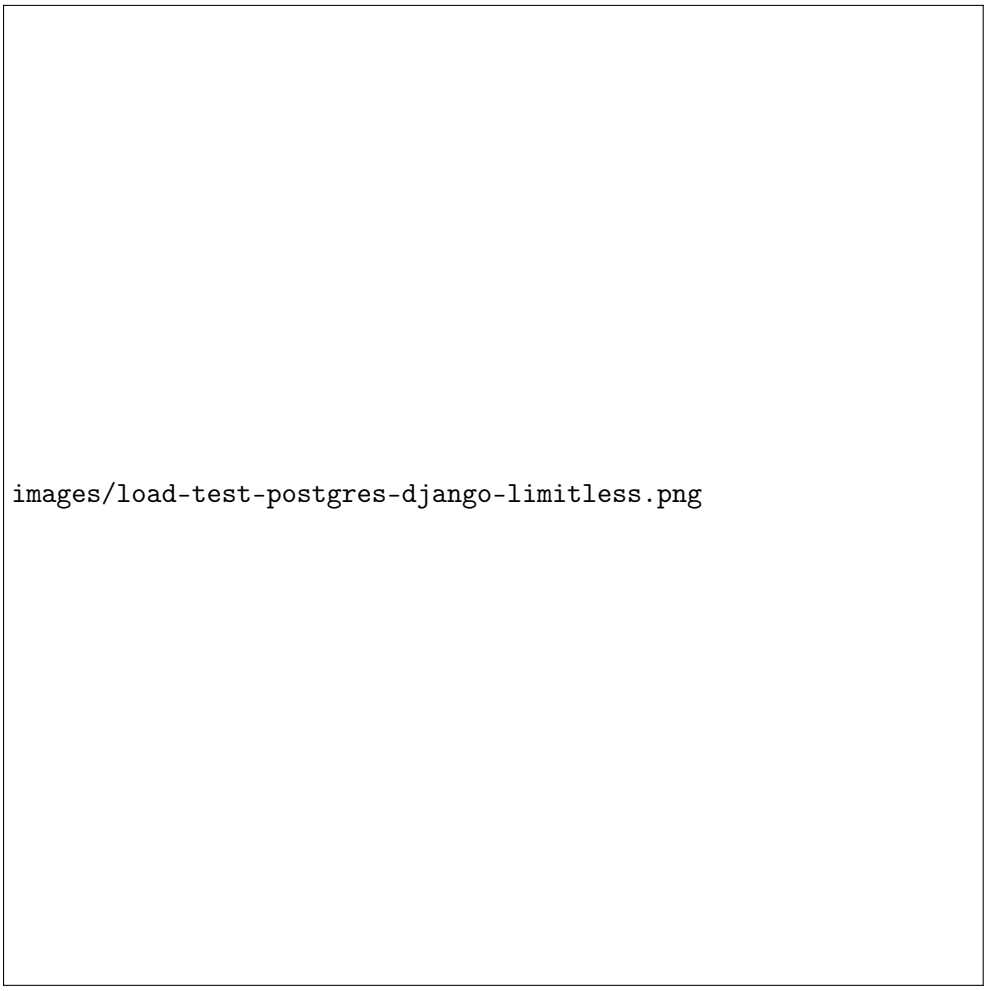


Figure B.1: Load test results without Django limits

This result shows that with active pagination no requests have timeouts. The defined NFR is still is accepted as before, but the requirement is better full filled.

B.5 System tests

The basis for the system tests are the implemented user stories, where possible they are bundled to a larger test. Only user stories with direct user interaction are tested in this section. System tests are conducted whenever a feature is implemented, at the end of the project all system tests were conducted again. Tests for old features that are not in the scope of the projects are not listed but still conducted using the defined tests and protocols from the SA.

At the point of final tests edge switches have not yet been configured to reach the network of the production server. Any test including change detection is mocked using a simple Postman call with real world data. The script running on the switches was tested by changing it to write any notification to a local file.

B.5.1 Scenarios

#	Use case	Description
Test 1	Time based group - Future	The system administrator creates a time based guest groups that has access to a device in the future and invites a user. The user should not have access to the management page when clicking the invite link.
Test 2	Time based group - Present	The sysadmin creates a time based guest groups that has access to a device in the present and invites a user. The user should have access to the management site and be able to configure devices when clicking the invite link. The user can also login using the acquired token.
Test 3	Guest config rollback	Configuration made by guests is rolled back to its previous state when access expires.
Test 4	Remove guest access	A system administrator creates and removes a guest access group. The rollback job is unregistered.
Test 5	Detect any change	A system administrator gets notified about configuration changes made on edge devices.

Test 6	Approve interface change	A system administrator gets notified about configuration changes made on interfaces outside the Tenant Manager. The change can be approved in the browser.
Test 7	Deny interface change	A system administrator gets notified about configuration changes made on interfaces outside the Tenant Manager. The change can be denied in the browser. When denied the configuration is rolled back automatically.

Table B.2: System test scenarios

B.5.2 Test log

Following are the test logs of the most recent full system test:

#	Use case	Accepted	Notes
Test 1	Time based group - Future	Yes	None
Test 2	Time based group - Present	Yes	None
Test 3	Guest config rollback	Yes	As the rollbacks are a scheduled job that only occur once a day the rollback time has to be changed so this test does not require waiting for the normal rollback.
Test 4	Remove guest access	Yes	None
Test 5	Detect any change	Yes	Has to be mocked at the time, however test using real data from the switch succeed.
Test 6	Deny interface change	Yes	Same as previous test
Test 7	Deny interface change	Yes	Same as previous test, rollback however works fully as it is done using the DNA Center which has access to the switches.

Table B.3: System test scenarios

Appendix C

Metrics

C.1 Sonarqube reports

C.1.1 Frontend

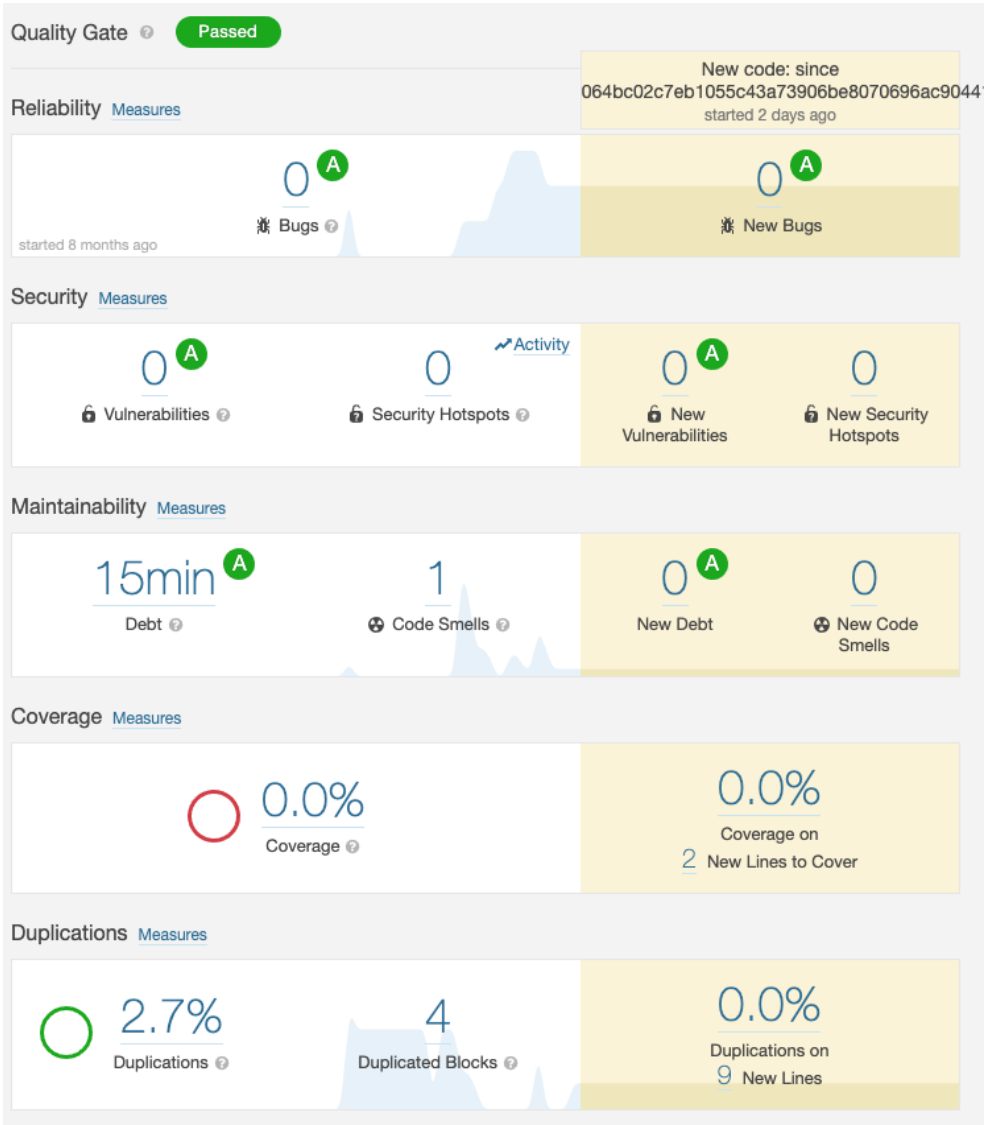


Figure C.1: Sonarqube web

C.1.2 Backend

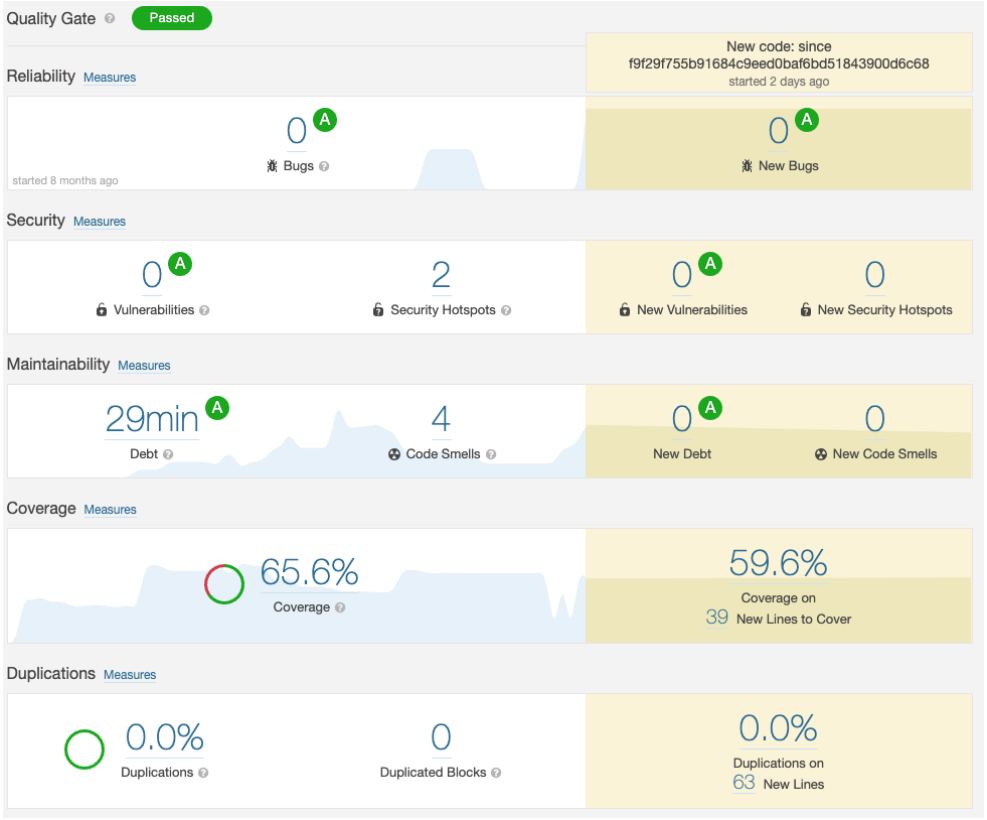


Figure C.2: Sonarqube api

C.2 Code

C.2.1 Backend

```
-------------------------------------------------------------------------------
Language                     files         blank       comment          code
-------------------------------------------------------------------------------
Python                          91           868           192          3149
HTML                             7             4             0           119
Markdown                         2            12             0            11
-------------------------------------------------------------------------------
SUM:                           100           884           192          3279
```

Listing C.1: Backend lines of code

```
-------------------------------------------------------------------------------
Language                     files         blank       comment          code
-------------------------------------------------------------------------------
Python                          14           132             8           599
-------------------------------------------------------------------------------
SUM:                            14           132             8           599
-------------------------------------------------------------------------------
```

Listing C.2: Tests lines of code

C.2.2 Frontend

```
-------------------------------------------------------------------------------
Language                     files         blank       comment          code
-------------------------------------------------------------------------------
TypeScript                      89           685            36          5500
-------------------------------------------------------------------------------
SUM:                            89           685            36          5500
-------------------------------------------------------------------------------
```

Listing C.3: 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                            10             1             0         16900
TypeScript                      89           685            36          5500
Python                         109          1075           246          4034
TeX                             47           426           210          1757
YAML                            20           123            16          1348
Markdown                        15           156             0           410
Dockerfile                       6            25             9           188
```

```
HTML                            8           4           0         137
reStructuredText                9         115         193         126
Bourne Shell                   11          24           3          90
XML                             9           0           0          81
SVG                             1           0           0          43
INI                             2           4           0          31
Modula3                         1           2           0           5
-------------------------------------------------------------------------------
SUM:                          337        2640         713       30650
```

Listing C.4: Total lines of code

Appendix D

Project planning

This section goes into detail of the project planning methods, milestones and organization.

25.02.2020	0.1	Initial definition	Aaron Meier
28.03.2020	0.2	Update Projectplan, Roadmap	Aaron Meier

Table D.1: Changelog Project planning

D.1 Roadmap

Following are the final project conditions and dates:

- **Start**: 17.02.2020 (Original), 24.05.2020 (Actual)
- **Resources**: 720h ($2 Members * 12 ECTS * 30h$)
- **Interim presentation submission**: 16.05.2020
- **Presentation for Cisco**: 06.07.2020
- **Exam presentation**: 08.07.2020
- **Thesis due date**: 26.06.2020 17:00
- **Standard working days**: Weekly on Mondays (8h), Tuesdays (8h) and Wednesdays (4h) and two full weeks (5 days, each 8h) from 1.06.2020

This results in a weekly work amount of 20 hours with a total of 35 working days during the semester and two weeks (10 days) with 8 hours each after the semester. The holiday week from 13.04.2020 to 19.04.2020 is closed.

D.1.1 Updates

Trough the outbreak of COVID-19 and the project Kick-Off not starting before 24.02, it has been decided to re-evaluate and update the project planning. As a result, the holiday week is after all used for project work. Because of the COVID-19 related break in the middle of the project the due date for the thesis was moved up by 2 weeks from 12.06.2020 to 26.06.2020.

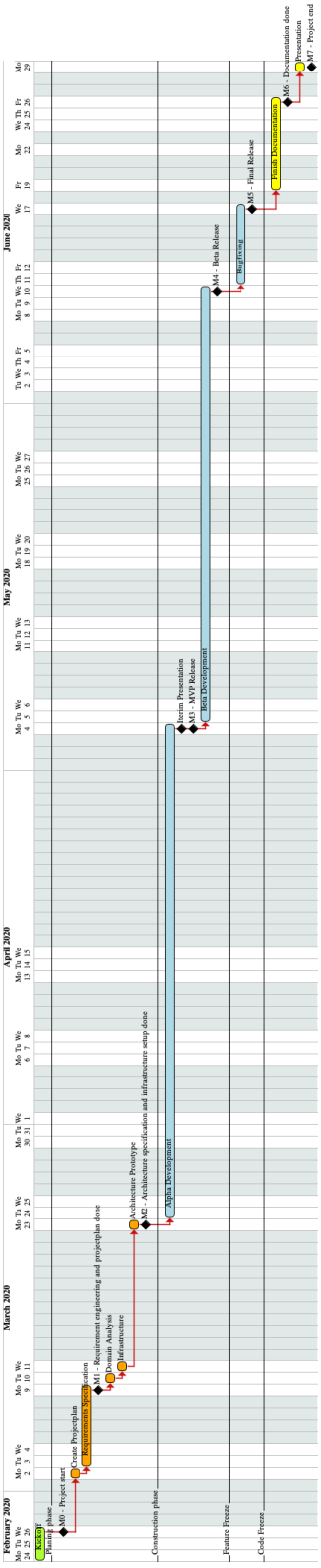


Figure D.1: Original Project Roadmap

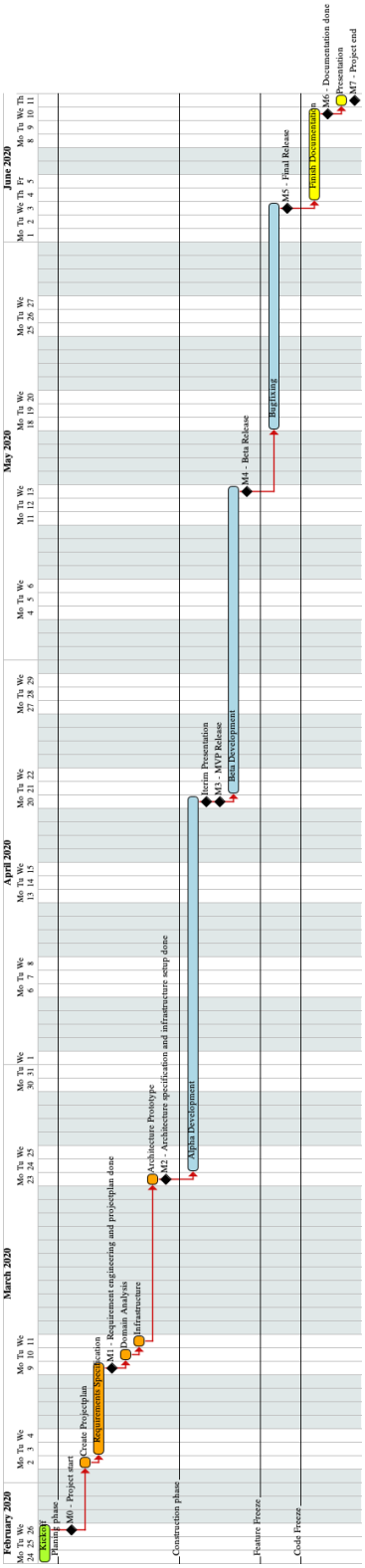


Figure D.2: Updated Project Roadmap

D.2 Project management

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

Evaluation of the previous research project showed that the project will benefit from larger construction (more agile) phase. As the projects goals are again PoC based, it is important to maximize time spent on construction.

An already planned Design Thinking session will increase the initial productivity of Requirements engineering (Elaboration phase). Additionally, the Infrastructure from the previous project will be used. Only updates and improvements are necessary, which results in time saving in the elaboration phase.

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)**: ~6%
- **Elaboration (colored orange)**: ~16%
- **Construction (colored blue)**: ~67%
- **Transition (colored yellow)**: ~11%

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**
    - Design Thinking Session
    - Rough 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**
    - Update Infrastructure setup:
        - VCS (archive/recreate repositories)
        - Setup Time tracking
        - Improve CI (more automation and speed)
        - Update images, VM
    - Risk management
    - User stories
    - Update Architecture
3. **MVP Release**: Release of MVP (Version 1.0.x, Codename "Ice Tea")
4. **Beta Release**: Release of Beta (Version 1.x.0, Codename "Matcha")

5. **Final Release**: Release of RTM (Version 2.0.0, Codename "Kombucha")
6. Final task description
    - Define and run system tests
    - Final Release with resolved bugs
7. **Documentation done**: Full Documentation
8. **Project end**: Presentation

D.2.3 Time evaluation

Time will be tracked using [1]https://toggl.com/. Every time entry is tagged with a milestone and category. Following categories are defined:

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

D.2.4 User Stories

User stories, based on the defined use cases, form the base for the task management. Each story is represented by a GitLab issue. Following stories include optionals. The completion state of each story is tracked on GitLab. The use cases are referenced in parentheses: (USE CASE NUMBER)

Grant permissions to Time-Based-Access Groups (UC01) As a System Administrator I want to assign permissions of available VNs and Ports to a TBA group, this permissions then get assigned immediately to the group (reserving it from other groups).

Update Time-Based-Access Groups lifetime (UC01) As an System Administrator I want to update the access time (from, until) from a TBA Group in order to increase the accessible lifetime of guest access.

Add guests to Time-Based-Access Groups (UC02) As an System Administrator I want to add guests to a TBA Group in order to make them able to access the system for a limited time.

Allow guest access by token in valid time-frames (UC03) As a Guest I want to access the system in the valid time-frames trough emails (passwordless) in order to manage my assigned objects.

[1]Toggl

Allow guest to list port configuration (UC04) As a Guest I want to read my assigned objects (Networks, VNs) in the valid timeframes.

Allow guest to manage port configuration (UC05) As a Guest I want to manage my assigned objects (Networks, VNs) in the valid timeframes.

Rollback of guest configurations (UC06) As a System Administrator I want to automatically schedule rollbacks for expired TBA groups in order to restore network configuration.

Rollback of guest permissions (UC06) As a System Administrator I want to automatically schedule rollbacks for expired TBA groups in order to restore permissions on assigned objects.

Detect and store change in port configuration from DNAC (UC07) As a System Administrator I want to track port configuration changes from DNAC for future actions.

Detect and store change in port configuration from Device (UC07) As a System Administrator I want to track port configuration changes directly from Devices for future actions.

Detect and store other configuration changes from DNAC (UC06, UC09) As a System Administrator I want to track other configuration changes from DNAC for future actions.

Detect and store other configuration changes from Device (UC06, UC09) As a System Administrator I want to track other configuration changes directly from Devices for future actions.

Rollback of port configurations (UC06) As a System Administrator I want to rollback previous states of port configurations configuration.

Display configuration changes (Admin) (UC08) As a System Administrator I want to display all tracked configuration changes to later apply filters or manually review changes.

Display configuration changes (Users) (UC10) As a User I want to display all tracked configuration changes to for my assigned devices to later apply filters or manually review changes.

Dispatch notifications based on configuration change type (UC07) As a System Adminsitrator I want to automatically calculate differences between port configuration changes for future notifications.

Send notifications in Tenant Manager (Push) (UC11) As a System Adminsitrator I want to get In-App notifications via Tenant Manager when a port configuration change is detected.

Send notifications via Mail (UC11) As a System Adminsitrator I want to get Mail notifications when a port configuration change is detected.

Send notifications via Webex (UC11) As a System Adminsitrator I want to get Webex notifications when a port configuration change is detected.

Send notifications as a ServiceNow Ticket (UC11) As a System Adminsitrator I want to automatically create a ServiceNow Ticket as a notification when a port configuration change is detected.

Create filters for configuration changes (OPTIONAL) As a System Adminsitrator I want to create filters for notifications for configuration changes.

Update of user settings (OPTIONAL) As a User (Guest, System Administrator, Network Administrator) I want to update my user settings (first-, lastname, email) to make sure the information is correct.

Update of session settings (OPTIONAL) As a User (Guest, System Administrator, Network Administrator) I want to update session settings (language, theme) to save this for future use.

Update of system settings (OPTIONAL) As a System Administrator I want to update Tenant Manager settings, which are otherwhise set via ENV-Variables. This affects credentials for ServiceNow, Webex and DNAC. This would be helpful for initial setup of the system.

Review of configuration change in Tenant Manager (OPTIONAL) As a System Adminsitrator I want to Review port configuration changes in Tenant Manager itself.

Create Compliance Rules (OPTIONAL) As a System Adminsitrator I want to create a rule about how configuration should be, this results in easy checks for compliance.

D.3 Meetings

Advisory meetings occur weekly, usually at the 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, it will be tracked in Gitlab issues.

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

Every monday afternoon, further project steps are discussed. While in construction phase this will be used as a weekly review of the ongoing sprint.

D.4 Responsibilities

Both project members are full stack developers. This means each developer is responsible for both, front- and backend.

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), Postman, Google Chrome Developer Tools, Firefox Developer Edition

D.6 Development concepts

The basis for code management and configuration 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 divided into feature-/issue- prefixes. Each member 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

For code review the builtin-feature of Gitlab with additional merge request templates is used. 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 easily 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 distributed and therefore easily restore-able from one or more destinations. Also the important branches are all protected and minimize risk of accidental overwrite.

Additionally the documentation is mirrored from Overleaf to Gitlab. For stability, well known public platforms are preferred.

D.8 Risk management

This project results in a PoC and therefore the usual potential risks are rather small. The same applies to the impact of such a risk. If a risk occurs, enough time is planned during the construction phase to handle it. On the other hand, this might decrease the chance to implement additional features. This is mitigated by priority/weighting of Use Cases.

#	Description	E[2]	S[3]	Mitigation	Action at occurence
1	Synchronisation with DNAC API is broken due an accidental update	1	7	A backup from DNAC working state will be performed.	Rollback to working state
1	On initial update to the latest DNAC version, synchronisation with the DNAC API is no longer possible and updating the back-end sync module takes longer than planned	8	5	Run the update procedure in an early project stage and plan enough time to resolve any issues	Update backend sync module to a working state (preferably with official API calls).
1	Rollback of tracked changes, is technically not possible (e.g. due DNAC rate limits, API limitations).	3	7	Plan enough time for alternative solutions. Mitigate by thoroughly researching early in Elaboration. Include other use-cases as fallback	Define different solutions and present to project partner/advisor. Change projects scope to other use-cases.

Table D.3: Risk list

[2]Possibility for Occurance
[3]Weighted Damage

D.9 Quality assurance

Assurance	Time	Goal
Code review for merges into development branch with another person	After pull request	Higher code quality
Proofreading of documentation with final reviews of other persons	Transition phase	Less typos and content mistakes
Unit and Integration tests	Continuously	Less bugs
Use branches (issue/feature)	Continuously	Less merge conflicts and better tracking of changes
Dependency checks	On every push	Better security
Use of Linters	On every push	Less code smells and compliant code style guide
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), no error handling code for monitoring (e.g. watchdog, mailing) will be added on occurrence. On the other hand consistency in sync is important, therefore rollbacks are used on occurrence.

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

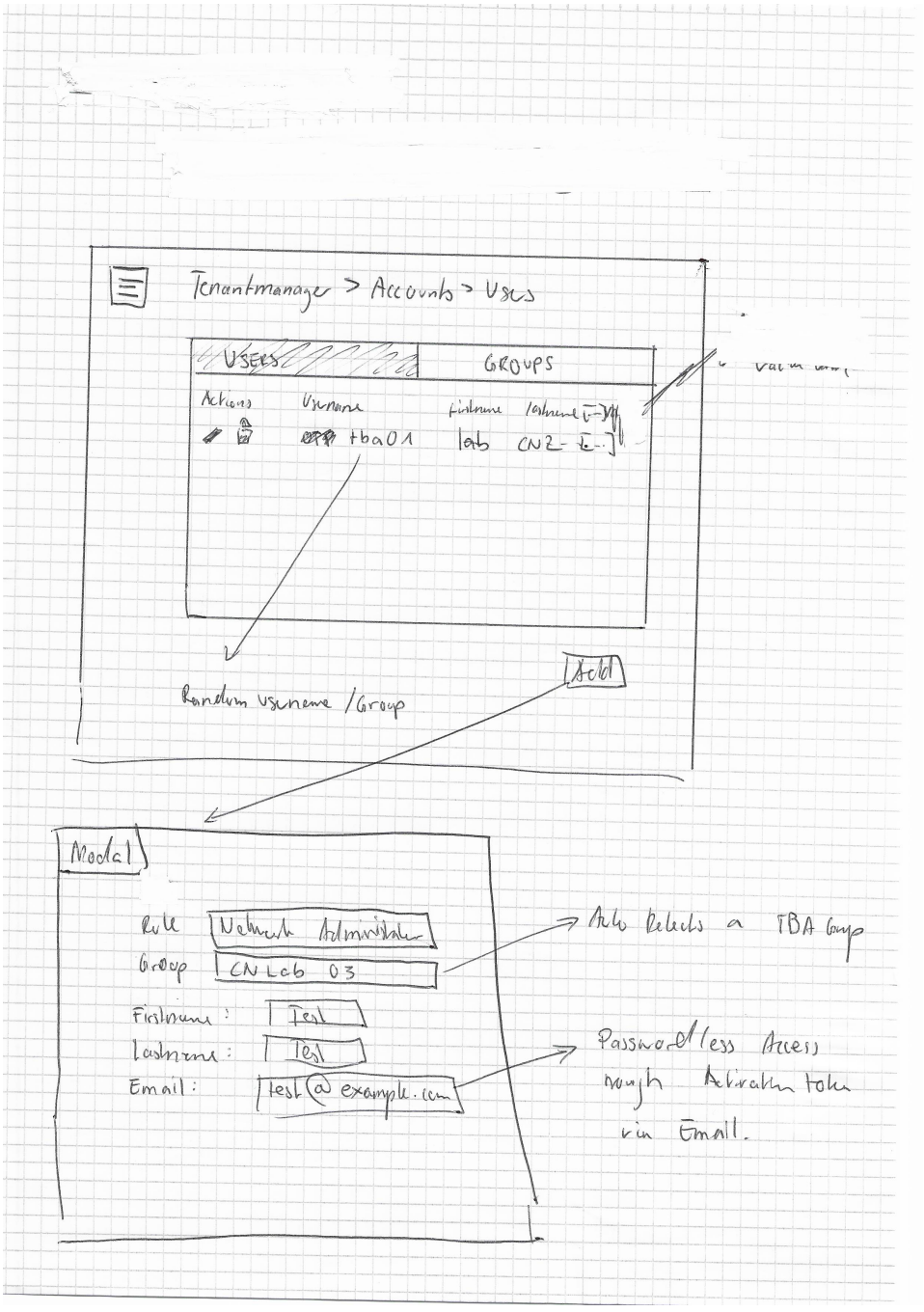
1. Errors from API requests will be handled directly by its error code. Then a message gets assigned and the error will be shown trough a notification alert to the user.
2. In debug mode error messages are logged to the web browsers console.

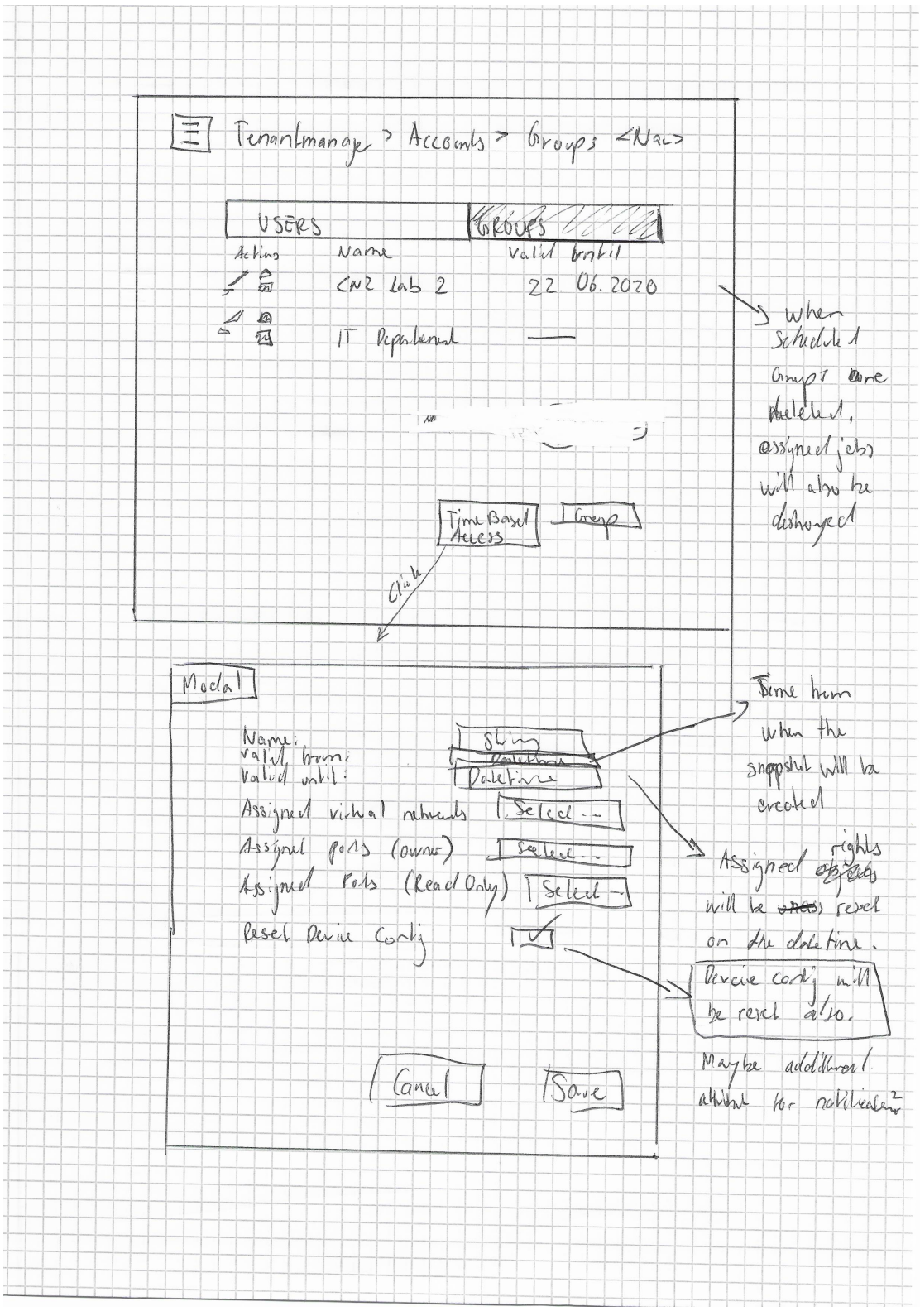
Appendix E

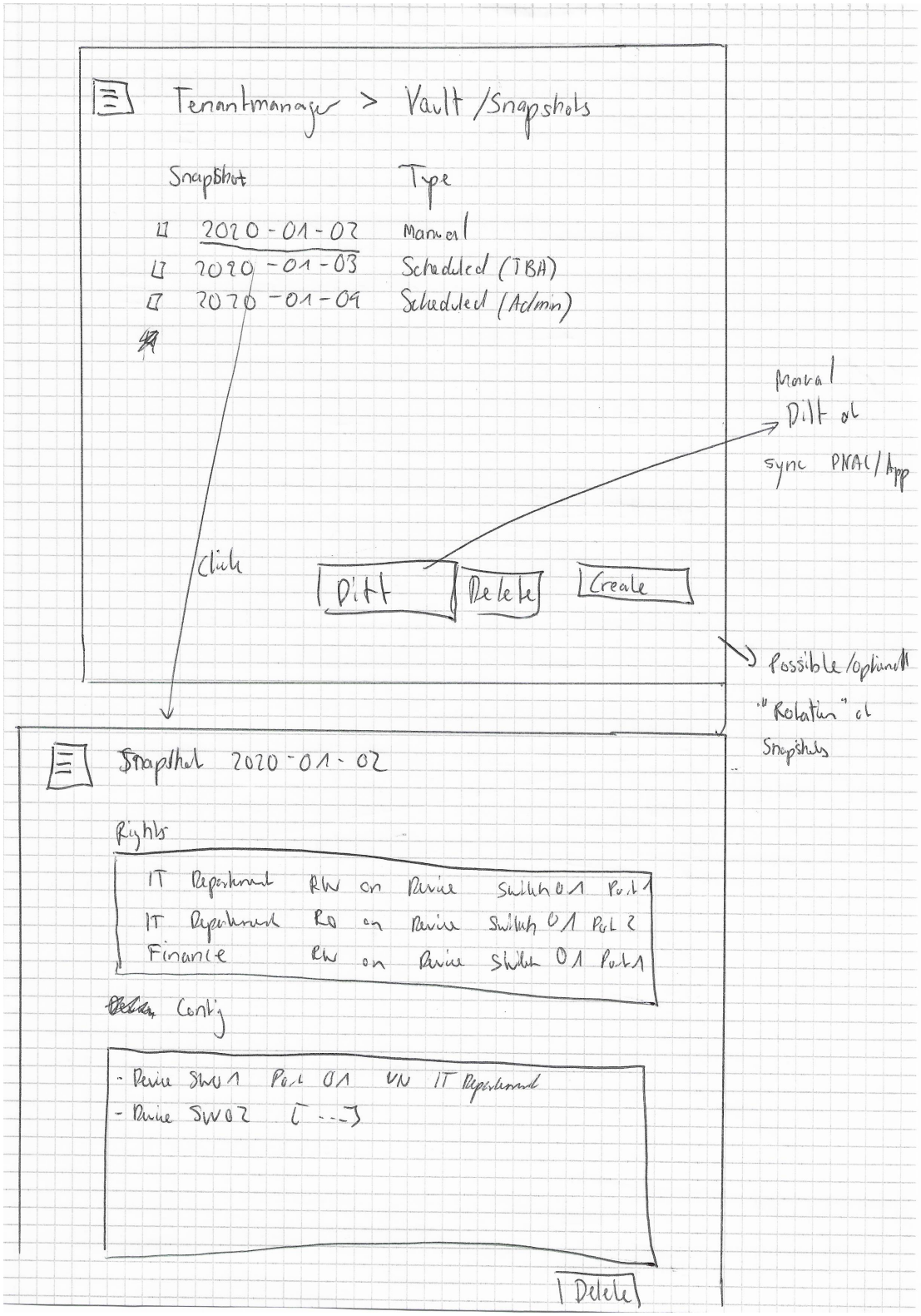
Wireframes

Following wireframes were used to communicate the covered functionality and are not used as basis of the UX design.

E.1 Time Based Access




Tenantmanage > Accounts > Groups <Nal>
USERS
GROUPS
Actions
Name
Valid until
CN2 Lab 2
22. 06. 2020
IT Department
—
when Scheduled Groups are deleted, assigned jobs will also be dismissed
Time Based Access
Group
Modal
Name:
Valid from:
Valid until:
Slim
Datetime
Time from when the snapshot will be created
Assigned virtual networks
Select --
Assigned pools (owner)
Select --
Assigned Pools (Read Only)
Select --
Reset Device Config
Assigned rights will be revert on the datetime.
Device config will be reset also.
Maybe additional attribut for notification?
Cancel
Save


Tenantmanager > Vault / Snapshots
Snapshot
Type
2020-01-02
Manual
2020-01-03
Scheduled (TBA)
2020-01-09
Scheduled (Admin)
Manual
Diff of
sync PNAC / App
Click
Diff
Delete
Create
Possible / optional
"Rotation" of
Snapshots
Snapshot 2020-01-02
Rights
IT Department RW on Device Switch 01 Port 1
IT Department RO on Device Switch 01 Port 2
Finance RW on Device Switch 01 Port 1
Config
- Device SW01 Port 01 VN IT Department
- Device SW02 [---
Delete

Tenantmanager > Jobs

	Start	End	Status	Type	Initiator
[redacted]	01/01/2020	–	Pending	Config	IT Department
[redacted]	01/04/2020	–	Scheduled (Snapshot)	Sync	Admin
☒	01/06/2020	–	Scheduled (Rollback)	Config	Admin

Cancel

Click

Modal — Cancel Scheduled Jobs

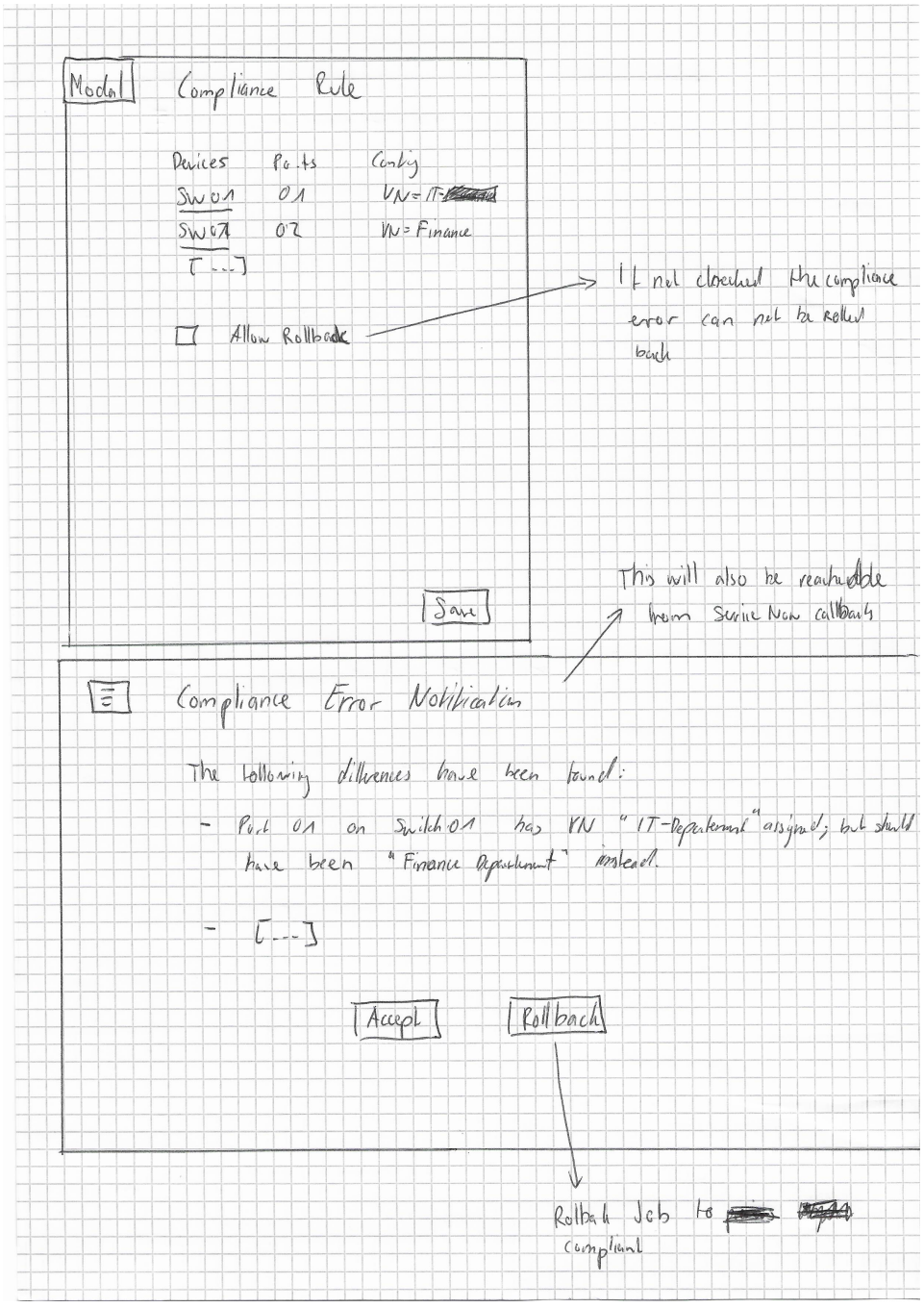
Do you really want to cancel the scheduled job for 01/06/2020?
This will create a new job to Rollback to Snapshot 01/04/2020.

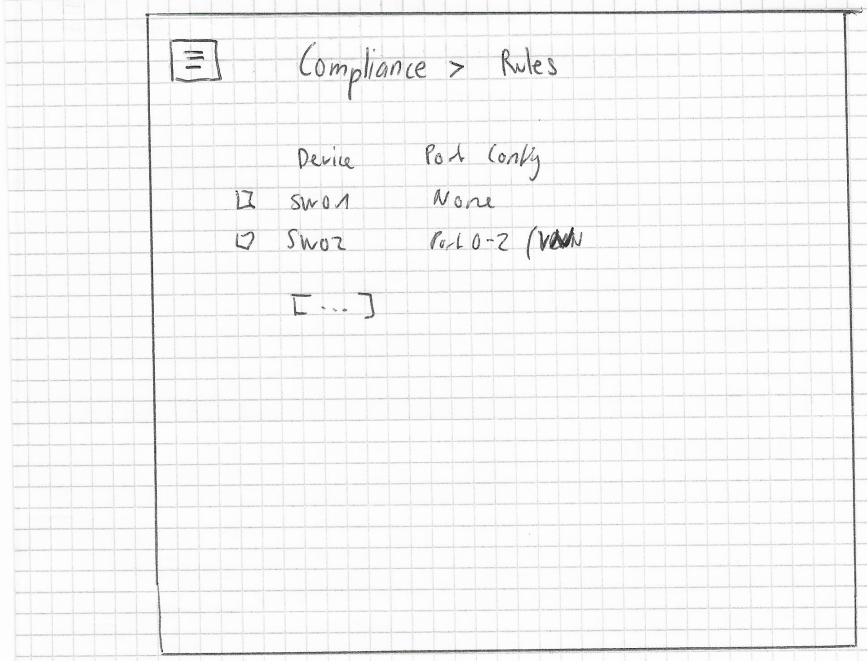
Cancellation of Snapshots: only possible if no Rollback depends on it

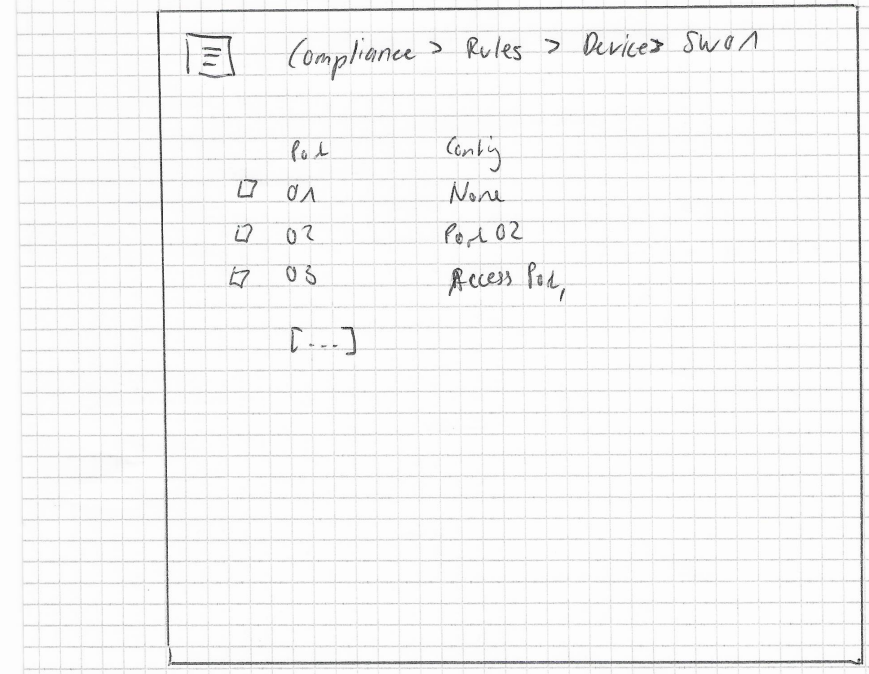
Cancellation of Rollbacks:

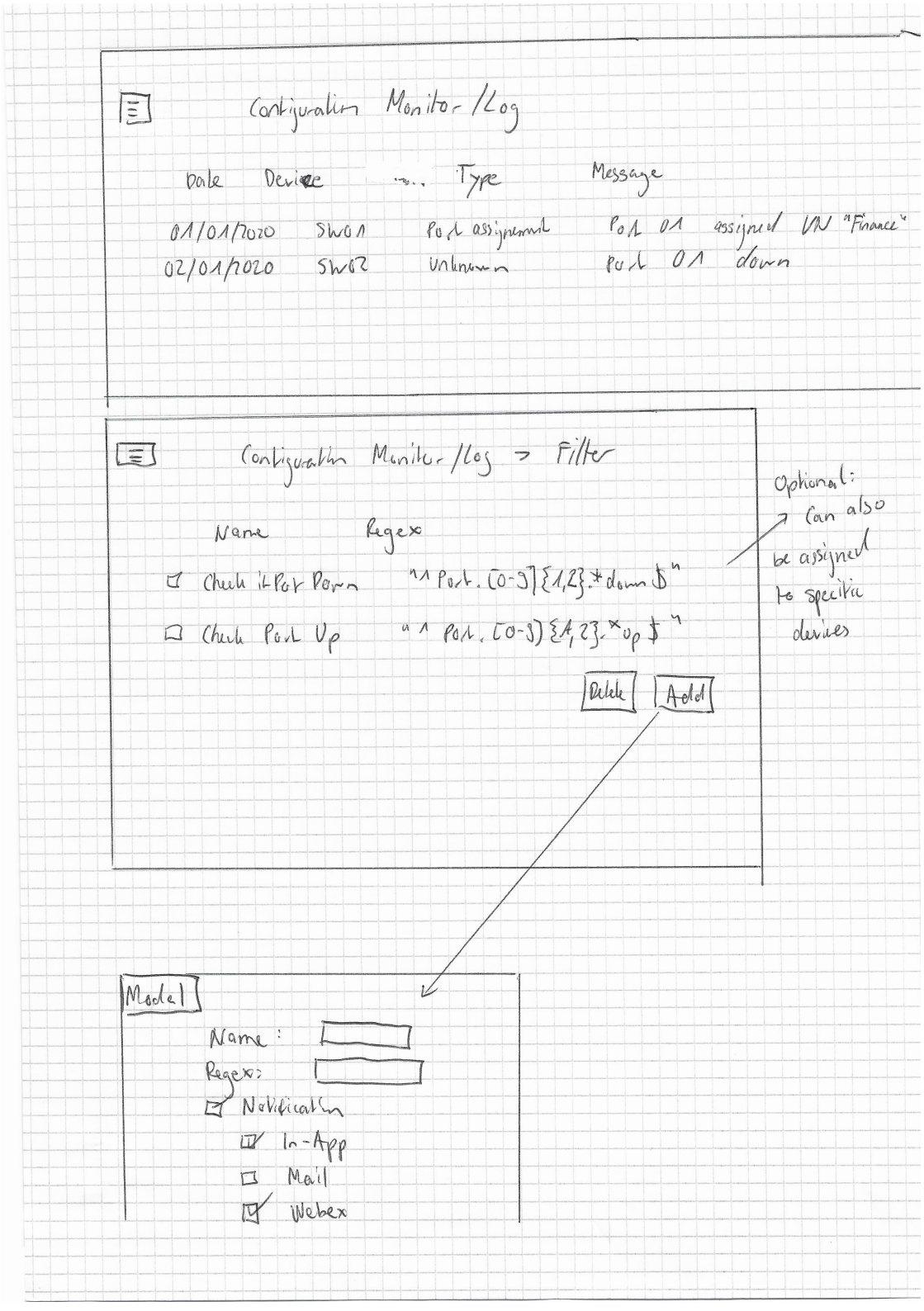
- Depending snapshot is not created
  → Both will be deleted
- Snapshot already created
  → Rollback will be deleted
  + new Rollback job with previous Snapshot

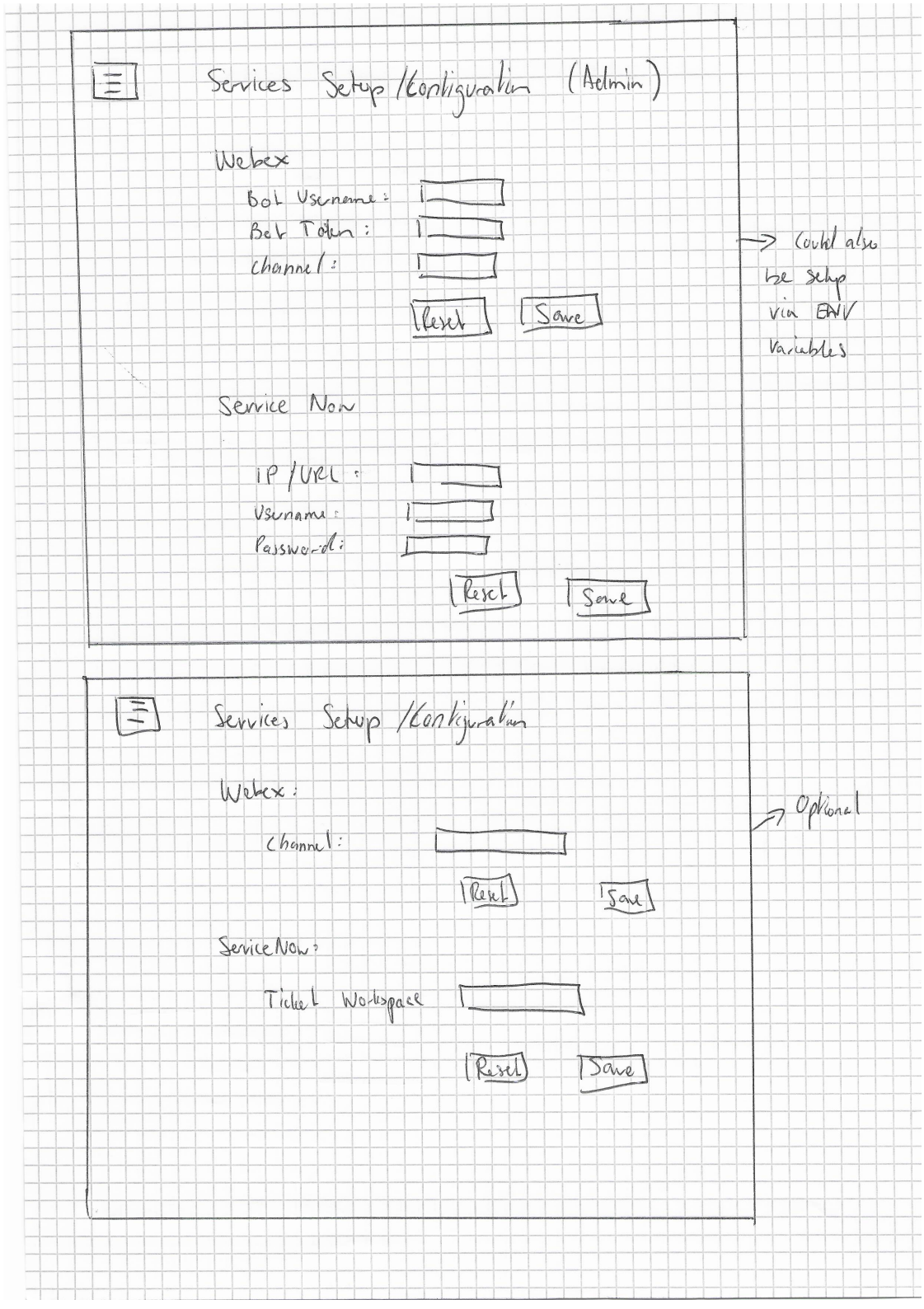
E.2 Change Control

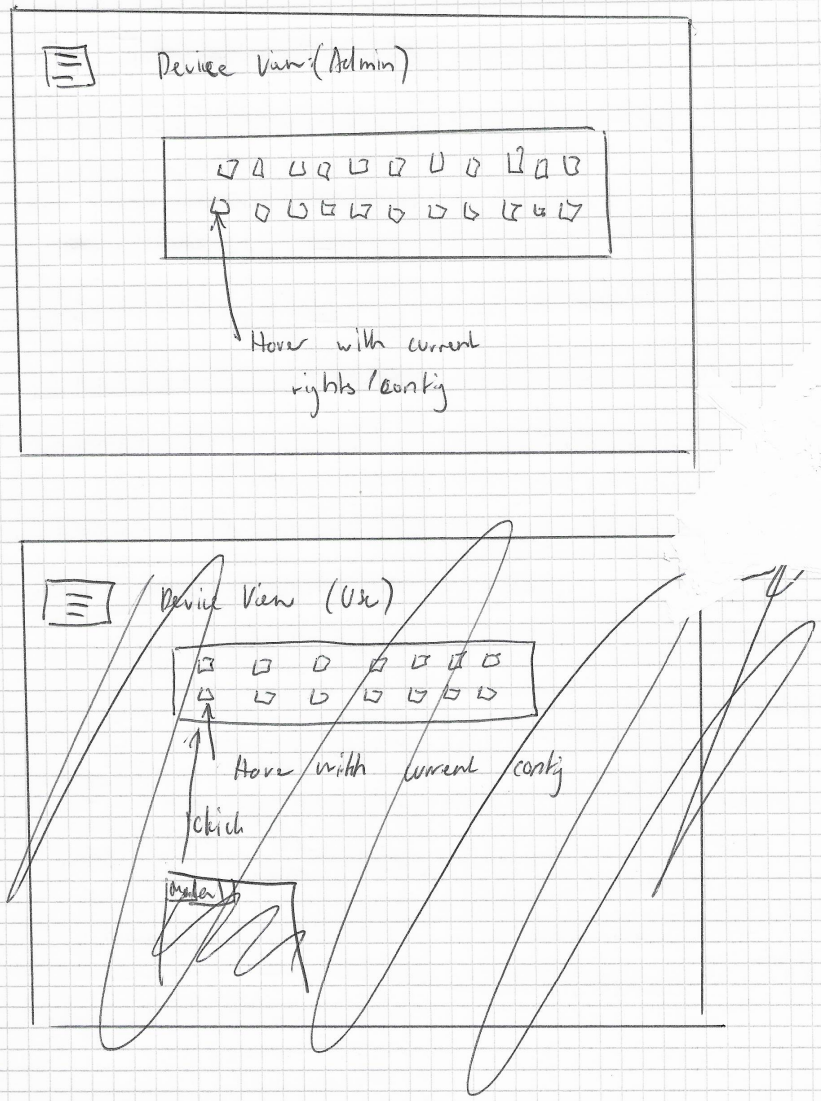



Compliance > Rules
Device
Port Config
SW01
None
SW02
Port 0-2 (VLAN
[...]


Compliance > Rules > Device > SW01
Port
Config
01
None
02
Port 02
03
Access Port
[...]


Configuration Monitor / Log
Date Device ... Type Message
01/01/2020 SW01 Port assignment Port 01 assigned VN "Finance"
02/01/2020 SW02 Unknown Port 01 down
Configuration Monitor / Log > Filter
Name Regex
Check if Port Down "^Port.[0-9]{1,2}.*down$"
Check Port Up "^Port.[0-9]{1,2}.*up$"
Delete
Add
Optional:
Can also be assigned to specific devices
Model
Name:
Regex:
Notification
In-App
Mail
Webex


Services Setup / Konfiguration (Admin)
Webex
Bot Username:
Bot Token:
channel:
Reset
Save
→ could also be setup via ENV variables
Service Now
IP / URL:
Username:
Password:
Reset
Save
Services Setup / Konfiguration
Webex:
channel:
Reset
Save
→ Optional
ServiceNow:
Ticket Workspace
Reset
Save


Device View (Admin)
Hover with current
rights / config
Device View (User)
Hover with current config
click

Appendix F

Time tracking

F.1 Time tracking by milestone

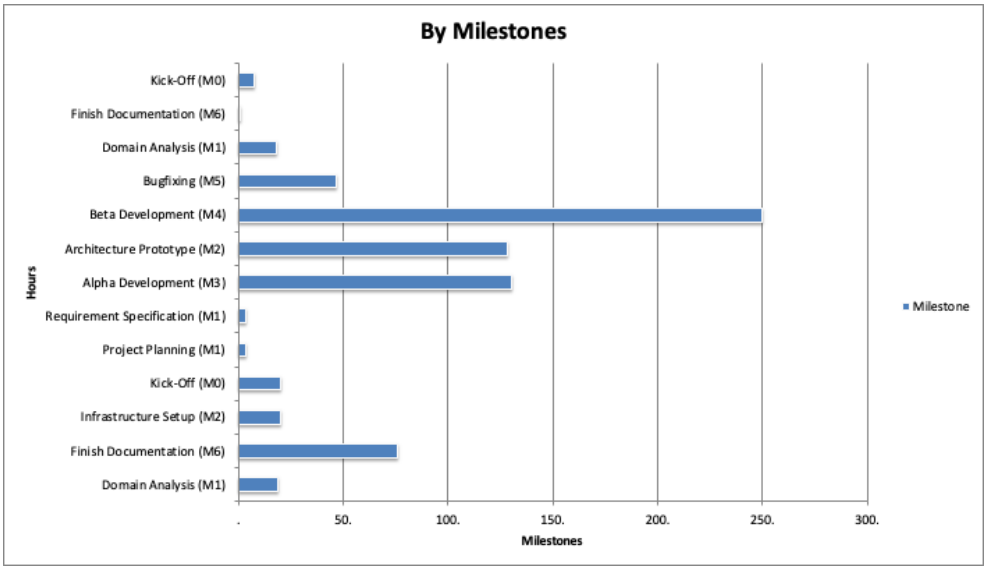


Figure F.1: Time by milestone

F.2 Time tracking by category

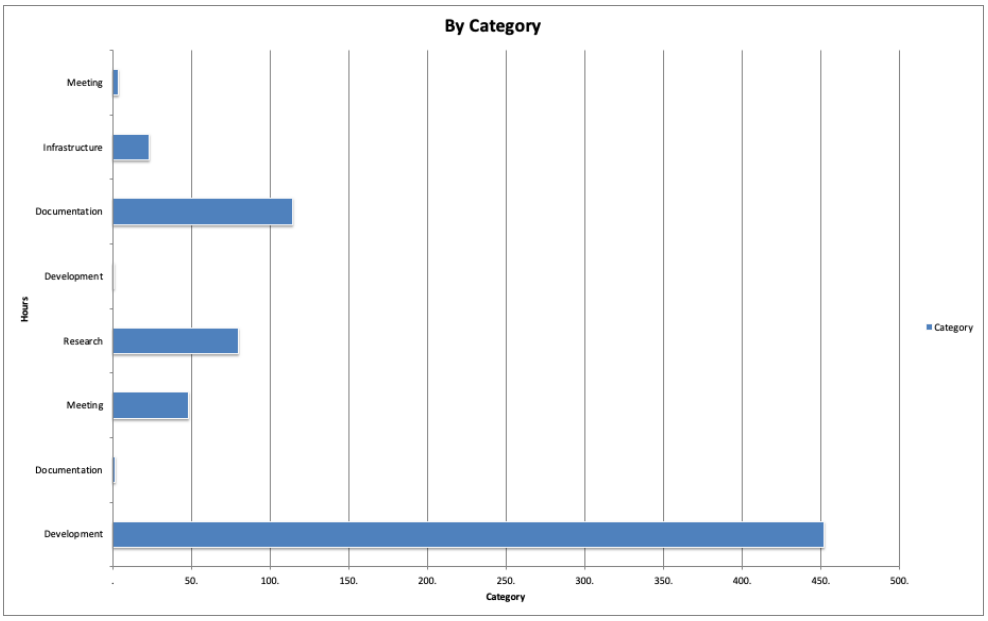


Figure F.2: Time by category

F.3 Time tracking by project members

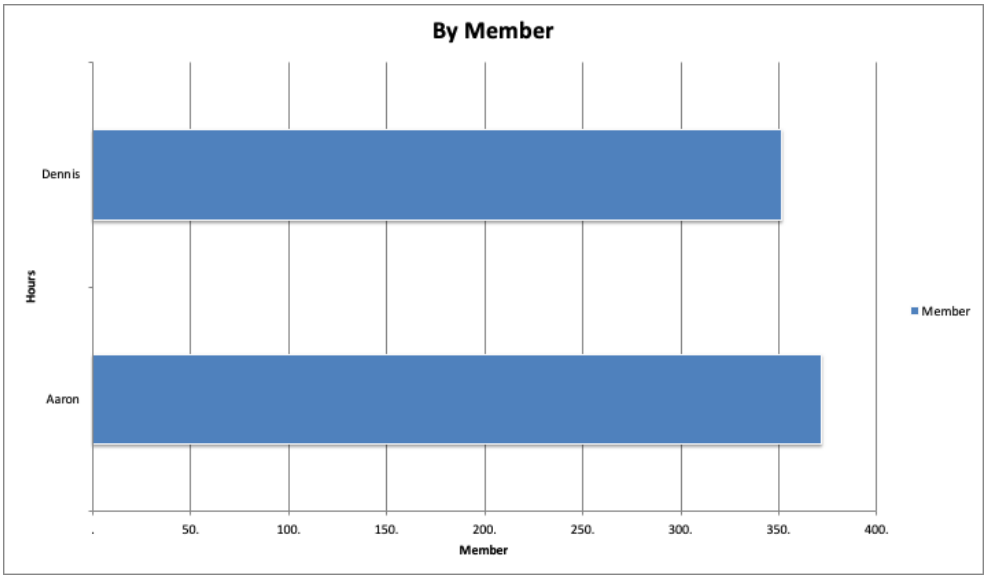


Figure F.3: Time by project members

F.4 Time tracking to actual/planned comparison

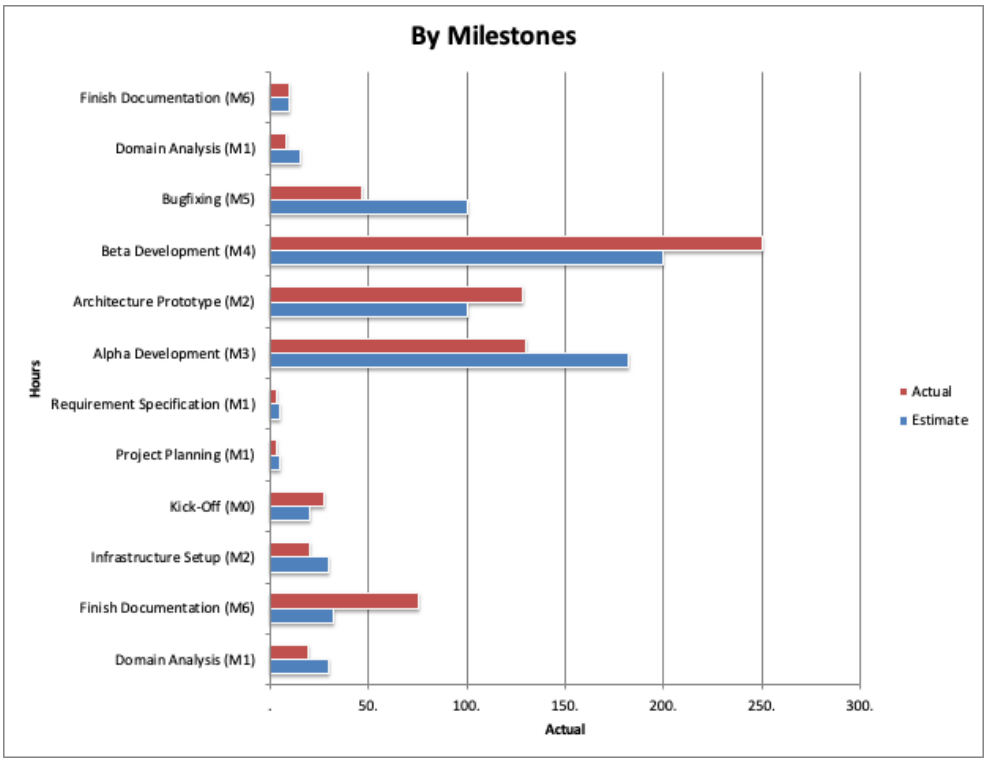


Figure F.4: Time comparison