HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

ins
INSTITUTE FOR
NETWORKED SOLUTIONS

Studienarbeit, Abteilung Informatik

Network Verification System

Hochschule für Technik Rapperswil

Frühlingssemester 2019
31. Mai 2019

Autor:	Luca Gubler, Alessandro Bonomo
Betreuer:	Laurent Metzger, Urs Baumann
Projektpartner:	INS Institute for Networked Solutions
Arbeitsperiode:	18.02.2019 - 31.05.2019
Arbeitsumfang:	240 Stunden, 8 ECTS pro Student

Inhaltsverzeichnis

Abstract

Während den Computernetz 1 & 2 sowie bei Kursen für externe Kunden müssen die Teilnehmer ein Netzwerk konfigurieren. Sobald bei der Konfiguration jedoch ein Fehler auftritt, wissen die Teilnehmer oftmals nicht, woran das Problem liegt. Ein Betreuer muss sich die gesamte Konfiguration anschauen um das Problem zu finden, was oftmals mehrere Minuten in Anspruch nimmt. Sobald mehrere Gruppen gleichzeitig die Hilfe eines Betreuers in Anspruch nehmen, gibt es teilweise sehr lange Wartezeiten.

Mit dem Network Verification System soll ein Programm erstellt werden, mit welchem dieses Problem umgangen werden kann. Die Betreuer können auf der Plattform Tests für das Netzwerk erstellen, ähnlich wie Unit Tests beim Programmieren. Um einen Test zu erstellen, muss lediglich angegeben werden, welcher Command ausgeführt werden soll, was das Expected Result ist und auf welchen Devices dieser Test ausgeführt werden soll.

Wenn ein Teilnehmer denkt, dass er sein Netzwerk konfiguriert hat, kann er auf der Plattform das Lab auswählen, welches er gelöst hat und die Tests starten. Sobald die Tests beendet sind, kann der Teilnehmer die Resultate der Tests ansehen. Er sieht, welche Aufgaben korrekt sind und welche Aufgaben er nochmals anschauen muss. Zudem kann er das Expected Result und das Actual Result vergleichen und bekommt so einen Hinweis, woran das Problem liegen könnte.

Mit dieser Lösung sind die Teilnehmer in der Lage, ihre Netzwerke selber zu Überprüfen und müssen die Betreuer nur in Anspruch nehmen, wenn sie Probleme haben, welche sie selber nicht lösen können. So reduziert sich der Stau bei den Betreuern und ein flüssiger Ablauf bei den Praktika wird gewährleistet.

Aufgabenstellung

Auf der nachfolgenden Seite befindet sich die vom Betreuer unterschriebene Aufgabenstellung.

Aufgabestellung SA NetVerif

The network verification tool is going to assess the correctness of the configuration and the functionality in a given network.
This tool is going to be used primarily in the CN Praktikum and in the Labs of the Cisco Network Academy. The tool should also be able to validate productive networks,

Based on assessment steps, the tool will validate that the task has been completed with success. The tool has to be capable to check the task in a lot of different ways (the states, the presence/absence of features in the configuration, connectivity and application checks)

A generic way to describe the tests for one lab or network environment should be used (for example one YAML files with all the definition of tests).
The tests should be executed in parallel wherever it makes sense. After the test have been run, a report has to be generated or another action has to be triggered (email or Webhook)

Use case 1	Select Test and Devices After he has completed a lab, a student can connect to the webpage and there will be a drop-down menu to select the lab that he wants to test	Mandatory
Use case 2	Run Test The user can click on a button "start test" and the tests will run. The user gets a web report.	Mandatory
Use case 3	CRUD Test The administrator is able to create new test or modify existing tests and assign them to a specific lab to specific devices (where necessary)	Mandatory
Use case 4	Manage User Account	Optional P2
Use case 5	Send Test Results to external Emails	Optional P2
Use case 6	Display of the command that fails on the web report and "Run the command" button if the user want to see the output.	Optional P1

PROF. L. METZLER

Management Summary

Ausgangslage

Die Teilnehmer im Lab konfigurieren Netzwerke unterschiedlicher Komplexität. Die Überprüfung der erstellten Netzwerke und das Troubleshooting kann unter Umständen sehr viel Zeit in Anspruch nehmen, vor allem wenn mehrere Personen gleichzeitig Hilfe in Anspruch nehmen möchten. Dementsprechend kann es zu längeren Wartezeiten führen, wodurch die Qualität des Labs leidet

Mit dem Network Verification System soll ein Tool erstellt werden, mit welchem die Teilnehmer die Netzwerke selbstständig testen können. So können sie ohne Hilfe eines Betreuers prüfen, ob die Aufgaben korrekt gelöst wurden oder welche Aufgaben nochmals angeschaut werden müssen.

Vorgehen / Technologien

Zu Beginn der Studienarbeit wurde eine Anforderungsanalyse gemacht. Da die Betreuer wünschten, dass Flask als Web Framework und Nornir als Automation Framework verwendet werden, wurden viele Entscheidungen bereits vorab abgenommen. Zusätzlich musste jedoch ein Configuration Management Tool evaluiert werden, welches in der Lage ist, Befehle auf Netzwerk Devices abzusetzen und den Output auszulesen. Für diesen Zweck entschied man sich für Netmiko, da es alle Bedürfnisse zufrieden stellt. Nach dem Abschluss der Anforderungsanalyse konnte schliesslich mit der Entwicklung des Network Verification Systems begonnen werden.

Ergebnisse

Nach Abschluss der Studienarbeit steht ein Tool zur Verfügung, mit welchem die Teilnehmer ihre Netzwerke im Lab selbstständig prüfen und testen können. Dazu können die Teilnehmer das gelöste Lab auswählen und die IP Adressen und Credentials angeben. Anschliessend erhalten sie einen Report, in welchem sie sehen können, welche Aufgaben korrekt gelöst wurden und welche Aufgaben nochmals angeschaut werden müssen. Somit müssen die Betreuer nur noch bei komplexeren Problemen Hilfestellung leisten und der gesamte Ablauf kann flüssiger von statten gehen.

1 Einleitung

1.1 Problemstellung

Für die durch das INS durchgeführten Module oder Kurse müssen die Teilnehmer für die Übungen oftmals selbstständig Netzwerke konfigurieren. Sobald jedoch Fragen oder Probleme auftreten, kommen die Betreuer schnell an ihre Grenzen. Oftmals möchten gleich mehrere Gruppen gleichzeitig die Hilfe der Betreuer in Anspruch nehmen. Da das Beheben dieser Probleme aber meistens mehrere Minuten benötigt, müssen die Teilnehmer unter Umständen längere Zeit warten, was zu viel Frust führen kann und die Qualität der Übungen leiden lässt.

Da alle Teilnehmer die gleichen Aufgaben erledigen, müssen die Betreuer bei jeder Gruppe die Konfiguration einzeln prüfen, was auch wieder unnötig viel Zeit beansprucht.

1.2 Aufgabenstellung

Diese repetitiven Arbeiten sollen durch ein Programm abgelöst werden. Es soll ein Network Verification System entwickelt werden, mit welchem die Teilnehmer ihre erarbeiteten Netzwerke ohne die Hilfe der Betreuer selbstständig überprüfen können. Somit müssen die Betreuer nur noch für das Troubleshooting beansprucht werden.

Die Betreuer sollen in der Lage sein Tests für das Netzwerk zu schreiben, ähnlich wie Unit Tests im Software Engineering Bereich. Die Tests sollen als YAML Dokument gespeichert werden, da dieses Format sehr einfach zu lesen und zu verstehen ist. Diese Tests können im Voraus geschrieben werden und anschliessend durch die Teilnehmer ausgeführt werden. Nach dem Ausführen der Tests erhalten die Teilnehmer einen Bericht, in welchem ersichtlich ist, welche Aufgaben richtig gelöst wurden und welchen Aufgaben mehr Zeit gewidmet werden muss.

Somit kann ein flüssigerer Ablauf der Übungen gewährleistet werden, wodurch die Teilnehmer wie auch die Betreuer entlastet werden können und die Qualität gewährleistet werden kann.

2 Anforderungen

2.1 Allgemeine Beschreibung

2.1.1 Produktperspektive

Mit dem Network Verification System wird ein Tool zur Verfügung gestellt, mit welchem die Teilnehmer eines Kurses des INS ihre erarbeiteten Netzwerke selbstständig testen können. Da alle Teilnehmer zur gleichen Zeit mit der Arbeit beginnen, sind sie auch häufig zur gleichen Zeit fertig. Wenn aber mehrere Teilnehmer zur gleichen Zeit die Hilfe der Betreuer in Anspruch nehmen, dann führt dies zu Engpässen bei den Betreuern und andere Teilnehmer verlieren viel Zeit, die sie mit warten verbringen.

Um diesem Problem entgegenwirken zu können, wird ein Tool bereitgestellt, mit welchem die Teilnehmer ihre Netzwerke selbstständig auf ihre Korrektheit prüfen können.

2.1.2 Produktfunktionen

Die Teilnehmer sollen in der Lage sein, ihre Netzwerke selbstständig testen zu können. Diese Tests sollen die Teilnehmer aufklären, welche Aufgaben korrekt gelöst wurden oder welche Aufgaben noch nicht richtig gelöst wurden. Zudem sollen diese Tests wiederholbar sein, so dass ein Teilnehmer stets den Status seines Fortschrittes prüfen kann.

Den Betreuern soll die Möglichkeit geboten werden, neue Test Cases zu erstellen und die bereits erstellten Test Cases fortlaufend anpassen zu können.

2.1.3 Benutzer Charakteristik

Die Zielpersonen dieses Tools sind zum einen die Betreuer, welche die Test Cases erstellen und bearbeiten und zum anderen die Teilnehmer, welche ihre Netzwerke mit diesen Test Cases überprüfen können.

2.1.4 Einschränkungen

Da das INS nur mit Cisco Devices arbeitet, müssen auch nur diese Devices durch das Network Verification System unterstützt werden. Das Tool wurde aber so aufgebaut, dass neue Devices ohne Probleme unterstützt werden können.

2.2 Use Cases

2.2.1 Use Case Diagramm

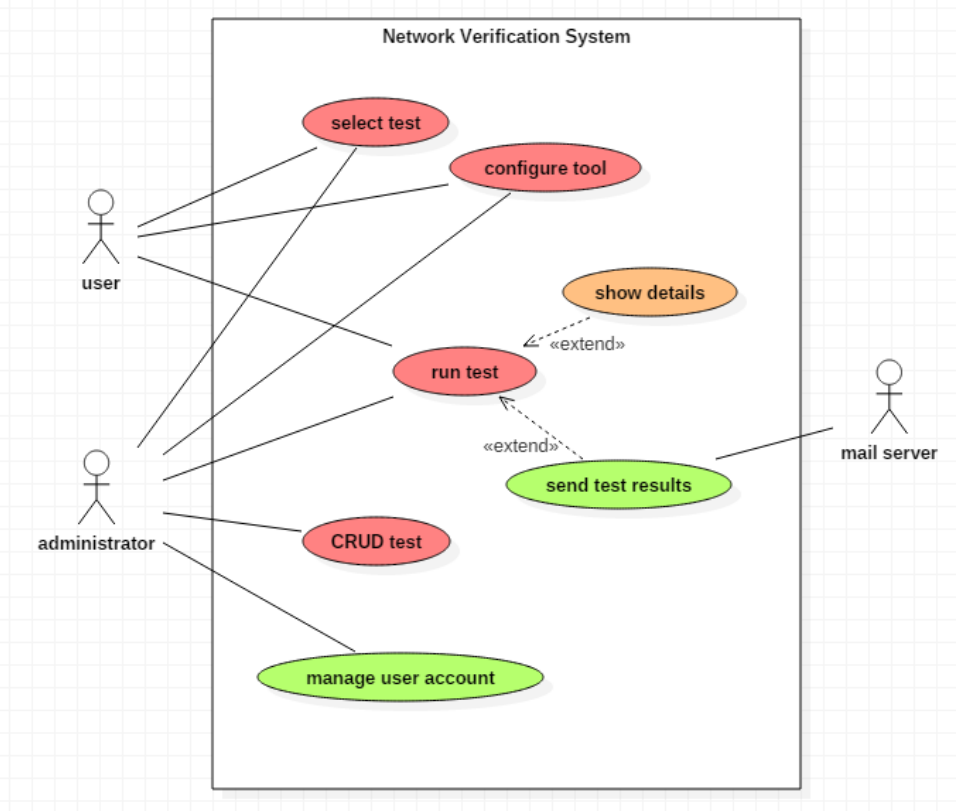


Abbildung 1: Use Case Diagramm

Die UseCases und die Aktoren werden im weiteren Verlauf des Kapitels genauer beschrieben. Der ”Mail-Server” wird als Aktor dargestellt, da das Tool ”starUML” keine besseren Alternativen anbietet.

Legende

Farbe	Priorität
rot	hoch
orange	mittel
grün	tief

2.2.2 Aktoren

Beschreibung der einzelnen Aktoren.

Aktor	**Beschreibung**
Benutzer	Student, der seine gelösten Aufgaben überprüfen will
Administrator	Dozent, der Lösungen für die Übungen bereitstellen, oder diese bearbeiten will
E-Mail-Server	Unterstützender Aktor, welcher die Ergebnisse der geprüften Aufgaben per Mail an die Benutzer versendet

2.2.3 Beschreibung der Use Cases

UC01: Select Test Der Benutzer ist in der Lage einen von mehreren vordefinierten Tests auszuwählen.

Primärer Aktor	Benutzer
Vorbedingungen	-
Nachbedingungen	Der vom Benutzer ausgewählte Test ist tatsächlich ausgewählt
Hauptszenario	
Aktor	**System Verantwortlichkeiten**
1. Der Benutzer wählt einen der vorgeschlagenen Tests aus	1. Das System lädt die bereits erstellten Tests
Auftrittswahrscheinlichkeit	hoch

UC02: Configure Tool Der Benutzer kann die nötigen Angaben zum Netzwerk machen (z.B. IP-Adresse der Router angeben, oder einen vordefinierten Pod wählen) und diese speichern.

Primärer Aktor	Benutzer
Vorbedingungen	-
Nachbedingungen	Die vom Benutzer eingegeben Daten werden temporär gespeichert. Nach dem neustarten der Applikation sind die Daten verloren.
Hauptszenario	
Aktor	**System Verantwortlichkeiten**
1. Die vorgegebenen Felder mit den nötigen Informationen füllen	1. Die Eingaben zwischenspeichern
Auftrittswahrscheinlichkeit	hoch

UC03: Run Test Nachdem ein Test ausgewählt und das Tool konfiguriert wurde, kann der Benutzer durch einen einfachen Klick den Test starten.

Primärer Aktor	Benutzer
Vorbedingungen	Einer der vordefinierten Tests wurde ausgewählt und das System wurde erfolgreich konfiguriert.
Nachbedingungen	Der Test wurde durchgeführt und der Benutzer wurde per Benutzerschnittstelle über das Ergebnis informiert.
Hauptszenario	
Aktor	**System Verantwortlichkeiten**
1. Der Benutzer startet den Test durch einen einfachen Mausklick	1. Das System liest die Konfiguration des Netzes und vergleicht sie mitt der erwarteten Lösung der Aufgabe.
Fehlerszenario	
Aktor	**System Verantwortlichkeiten**
	1. Das System erkennt Fehler und loggt diese.
Auftrittswahrscheinlichkeit	hoch

UC04: CRUD Test Für Administratoren ist es möglich neue Tests zu definieren. Die Tests werden in .yaml-Dateien gespeichert, so dass sie auch zu einem späteren Zeitpunkt eifach bearbeitet und wieder entfernt werden können.

Primärer Aktor	Administrator
Vorbedingungen	-
Nachbedingungen	Der Test wurde erfolgreich erstellt, ausgegeben, bearbeitet oder entfernt.
Hauptszenario	
Aktor	**System Verantwortlichkeiten**
Create: 1. Der Administator erstellt einen neuen Test. 2. Der Administator erfasst alle notwendigen Daten. 3. Der Administator speichert den Test im System. Read: 1. Alle bereits erstellten Tests werden in einer Liste dargestellt. Update: 1. Der Administrator wählt einen der bereits erstellten Tests. 2. Die Daten werden entsprechend geändert. 3. Der Test wird gespeichert. Delete: 1. Der Administator löscht einen Test aus dem System.	Create/Update: 1. Das System speichert den neuen Test. Read: 1. Das System stellt die Tests in Form einer Liste dar. Delete: 1. Das System löscht den gewählten Test.
Auftrittswahrscheinlichkeit	manchmal

UC05: Show Details Nachdem ein paar Tests durchgeführt wurden, soll es für den Benutzer möglich sein, die Details zu den einzelnen Tests anzeigen zu lassen. Dies ist hilfreich wenn ein Test unerwartet fehlgeschlagen ist.

Primärer Aktor	Benutzer
Vorbedingungen	Mindestens ein Test muss durchgeführt worden sein und die Resultate werden angezeigt.
Nachbedingungen	Alle gesammelten Daten zum ausgeführten Test werden dem Benutzer dargestellt.
Hauptszenario	
Aktor	**System Verantwortlichkeiten**
1. Der Benutzer klickt auf Details-Button.	1. Das System zeigt alle vorhandenen Informationen zum ausgewählten Test an.
Auftrittswahrscheinlichkeit	hoch

UC06: Send Test Results Sobald die Tests durchgelaufen sind, sollen die Ergebnisse per Benutzerschnittstelle an den entsprechenden Benutzer gesendet werden, damit dieser weiss, was er richtig und falsch gemacht hat.

Primärer Aktor	System
Vorbedingungen	Alle nötigen Angaben wurden gemacht, um einen Test laufen zu lassen. Ebenfalls wurde, falls gewollt, eine E-Mail-Adresse angegeben.
Nachbedingungen	Der Benutzer kann seine Ergebnisse einsehen (Falls Mail Adresse angegeben, auch per Mail).
Hauptszenario	
Aktor	**System Verantwortlichkeiten**
	1. Das System erstellt einen Bericht über den durchgeführten Test. 2. Das System versendet eine Mail mit den Testergebnissen.
Auftrittswahrscheinlichkeit	hoch

UC07: Manage User Account Es soll möglich sein, neue Administrator-Accounts anzulegen, welche dann selber Tests erfassen und bearbeiten können.

Primärer Aktor	Administrator
Vorbedingungen	
Nachbedingungen	Ein neuer Administrator wurde registriert.
Hauptszenario	
Aktor	**System Verantwortlichkeiten**
1. Der Benutzer klickt auf den Registrieren-Button. 2. Der Benutzer gibt die geforderten Angaben ein (unter anderem das Secret, welches nur Administratoren kennen) und bestätigen die Eingaben.	1. Das System prüft die Eingaben. 2. Das System erstellt den neuen Account.
Auftrittswahrscheinlichkeit	selten

2.3 Nicht Funktionale Anforderungen

Im nachfolgenden Kapitel werden die Nichtfunktionalen Anforderungen des Network Verification Systems angesprochen.

2.3.1 Qualität

Um die Qualität der Software sicherzustellen, wird das Modell ISO/IEC 9126 verwendet.

Maintainability

Das INS möchte die Software produktiv einsetzen. Dies bedeutet, dass sie über längere Zeit im Einsatz sein wird. Zu einem späteren Zeitpunkt können sich aber die Anforderungen an die Software ändern, weshalb die Software so gebaut werden soll, dass sie einfach modifizierbar ist.

Efficiency

Einzelne Tests können unter umständen mehr Zeit in Anspruch nehmen. Ein wichtiger Punkt ist, dass die einzlennen Teilnehmer nicht zu lange auf die Resultate ihrer Tests warten müssen. Darum müssen mehrere Tests parallel durchgeführt werden können.

Portability

Die Software soll auf unterschiedlichen Systemen laufen können. Die Scripts wurden in einer eigenen Virtual Environment programmiert. Somit wird sichergestellt, dass die Software auf unterschiedlichen Systemen laufen kann.

Reliability

Da es sich nicht um ein kritisches System handelt, müssen in dieser Hinsicht keine weiteren Vorkehrungen getroffen werden. Es muss jedoch darauf geachtet werden, dass die Verbindungen zu den Devices sauber auf und wieder abgebaut werden. Es soll nicht vorkommen, dass ein Student nicht mehr auf sein Device zugreiffen kann, weil eine Telnet Verbindung nicht sauber terminiert wurde.

Functionality

Vorerst soll die Software nur Cisco Devices und die Befehle unterstützen, welche im Lab ”router-case-study-solution” verwendet werden. Es muss jedoch darauf geachtet werden, dass die Befehle dynamisch eingegeben werden können. Falls man einen Test mit einem neuen Befehl durchführen möchte, soll man nicht zuerst das Backend um weitere Funktionalität erweitern müssen.

Usability

Die Software soll einfach zu handhaben sein. Die Teilnehmer sollen mit wenigen Klicks die Tests auf ihren Netzwerken laufen lassen. Den Betreuern soll es möglich sein, neue Tests zu erstellen und bereits erstellte Tests einfach anzupassen.

Security

Um die Tests laufen lassen zu können, muss man sich nicht authentifizieren, man muss lediglich im selben Netzwerk sein, in welchem die Software läuft. Da die Tests nur im Labor verwendet werden, müssen auch keine weiteren Sicherheitsmassnahmen getroffen werden.

Um jedoch neue Tests zu erfassen oder Tests zu bearbeiten, muss sich der Betreuer einloggen. Für den Beginn gibt es ein Login für alle Betreuer. Es ist jedoch möglich neue Logins zu erstellen.

2.3.2 Schnittstellen

Benutzerschnittstellen

Die Teilnehmer und Betreuer können auf die Software mittels eines Webbrowsers zugreifen.

Netzwerkschnittstellen

Damit die Software die Tests auf den Netzwerk Devices ausführen kann, müssen diese über Transmission Control Protocol / Internet Protocol (TCP/IP) erreichbar sein. Zudem müssen die Devices über Secure Shell (SSH) oder Telnet erreichbar sein, da sonst die Tests nicht ausgeführt werden können.

3 Evaluation

3.1 Frontend

3.1.1 Bootstrap

Da das Webinterface nur ein Formular zum Ausfüllen der Verbindungs-Informationen ist und anschliessend die Resultate darstellen soll, wurde das Frontend einfach gehalten. Bootstrap wurde als CSS Framework gewählt, da dieses Framework einfach zu benutzen ist und bereits viel Funktionalität mit sich bringt.

Mit dem Bootstrap Template hat man zum Beispiel schon ein optisch ansprechendes Design, ohne das viele Anpassungen notwendig sind.

3.2 Backend

3.2.1 Flask

Flask ist ein Microwebframework für Python, basierend auf Werkzeug und Jinja2 [6] und wird von Firmen wie Red Hat, Airbnb oder Netflix verwendet. [12]

Flask ist ein Microframework, weil es an sich keine weiteren Tools oder Libraries verwendet. So existiert in Flask zum Beispiel keine Datenbankschicht oder eine Formularvalidierung. Diese Komponenten müssen separat dazu installiert werden. Es existiert jedoch eine vielzahl von Add-ons, welche verwendet werden können. [18]

Flask eignet sich sehr gut für kleine Webseiten oder um einen Prototypen zu erstellen. Zudem kann man selber entscheiden, wie man die Applikation aufbauen möchte. Dies hat aber auch den Nachteil, dass es mehrere Ansätze gibt, wie man etwas umsetzen könnte. Dies kann manchmal zu etwas Verwirrung führen.

Von den Betreuern wurde gewünscht, dass Flask als Webserver verwendet wird, da es in den vom Institute for Networked Solutions (INS) genutzten Technologie-Stack passt.

3.2.2 Django

Als Alternative zu Flask könnte man Django einsetzen.

Django ist ein Full-Stack Webframwork und basiert auf dem MVC Pattern [2] und wird von Firmen wie Instagram, Spotify oder YouTube eingesetzt [14].

Django wird eher für komplexere Projekte oder datenbanklastigen Webseiten eingesetzt. Im Gegensatz zu Flask bietet Django bereits sehr viele vorinstallierte Komponenten an. So sind die Komponenten für ORMapper oder die Formularvalidierung im Gegensatz zu Flask bereits integriert [10].

Ein grosser Vorteil von Django ist, das bereits alle Komponenten vorhanden sind. Gleichzeitig hat dies aber auch den Nachteil, dass alles zusammen deployed wird. Zudem kann es zu stark gekoppelten Applikationen führen. [11]

3.3 Nornir

Nornir ist ein Automation Framework für Python. Es gibt viele verschiedene Automation Tools wie Ansible, Salt, Chef oder Puppet. Nornir unterscheidet sich jedoch von diesen Tools, indem man eigenen Python Code schreibt um das Tool zu verwenden. Ansible ist zum Beispiel auch in Python geschrieben, verwendet jedoch eine eigene Domain-Specific Language (DSL), welche dazu verwendet wird, das Tool zu bedienen. [13]

Im Gegensatz zu anderen Tools braucht Nornir keine DSL sondern der Code wird in Python geschrieben. Das bringt mehrere Vorteile mit sich, denn eine DSL kann mühsam und kompliziert werden, sobald nicht nur die Standard Funktionen verwendet werden. Zudem ist der Code einfacher zu debuggen, da bereits existierende Python Logging und Debugging Tools verwendet werden können. Zudem hilft die Integrated Development Environment (IDE) bei der Autovervollständigung und der Code kann direkt in existierende Python Scripts integriert werden. [13]

Da sich Nornir bereits im verwendeten Technologie Stack des INS befindet, wurde von den Betreuern gewünscht, dass Nornir verwendet wird.

3.3.1 Netmiko

Netmiko ist ein von Kirk Byers entwickeltes Tool, welches das SSH Management zu Netzwerk Devices vereinfacht. Netmiko basiert auf der Paramiko SSH Library. Netmiko erlaubt es, dass man eine SSH Verbindung zu einem Netzwerk Device aufbauen kann. Anschliessend können einfache Show oder Konfigurations Befehle abgesetzt werden. Für den Zweck dieser Studienarbeit ist es wichtig, dass Behfele auf Cisco Devices abgesetzt werden können. Netmiko selber ist aber herstellerunabhängig. [7]

3.3.2 Napalm

Network Automation and Programmability Abstraction Layer with Multivendor support (NAPALM) ist eine Python Library, welche Funktionen implementiert, mit welchen über eine Application Programming Interface (API) zugegriffen werden kann. Auf den ersten Blick wirken NAPALM und Netmiko ziemlich ähnlich. Der grosse Vorteil von NAPALM ist aber, dass die Herstellerunabhängigkeit gegeben ist. Die Funktionen get_facts() oder get_interfaces() funktioniern, egal ob es sich um ein Cisco IOS XE, Cisco IOS XR oder um ein Juniper Device handelt. [8]

Eine wichtige Anforderung an das Network Verification System ist, dass das Tool generisch ist. Möchte man eine neue Information auf einem Netzwerk Device abfragen,

soll man nicht erst eine neue Funktion implementieren müssen, welche dies ermöglicht. Mit NAPALM wäre die Umsetzung nicht ganz einfach gewesen. Man hätte für jede der knapp 30 Funktionen, welche in NAPALM implementiert sind, eine eigene Funktion im Backend implementiere müssen.

Mit Netmiko ist dies einfacher, da man den selben Befehl angeben kann, welchen man auch über die Command Line Interface (CLI) eingeben würde. NAPALM ist herstellerunabhängiger als Netmiko. Da im Lab jedoch nur Cisco Devices zum Einsatz kommen, ist dieser Punkt nicht so wichtig. Viel wichtiger ist, dass das Tool generisch ist. Dieser Punkt kann mit Netmiko besser abgedeckt werden, weshalb man sich für die Verwendung von Netmiko entschied.

3.4 Datenbank

Bei der Auswahl der Datenbank stand man vor der Entscheidung, ob man MySQL oder PostgreSQL verwenden wollte. Da bei dieser Applikation jedoch nur eine sehr kleine Datenbank mit sehr wenigen Features zum Einsatz kommt, entschied man sich für MySQL. Dies aus dem einfachen Grund, dass sich die Studenten mehr Erfahrung mit MySQL aneignen können.

Es wurde allerdings darauf geachtet, dass man die Datenbank ohne Probleme wechseln kann. Zu einem späteren Zeitpunkt kann man zum Beispiel eine PostgreSQL Datenbank verwenden, falls man dies bevorzugt.

4 UI Design

Bevor man mit der Entwicklung des Frontends beginnen konnte, musste geplant werden, wie das User Interface genau aussehen soll. Mit Hilfe der Use Cases konnte ein erster Entwurf erstellt werden.

4.1 Mockup

Exercise Page

Man wollte die Exercise Page für die Studenten sehr simpel halten. Die Studenten sollen nur sehr wenige Informationen eingeben müssen und die Seite soll selbsterklärend sein.

Abbildung 2: Exercise Page

4.2 Dashboard

Das Dashboard steht nur den Betreuern zur Verfügung. Man muss registriert und eingeloggt sein, damit auf diese Seite zugegriffen werden kann. Hier sind alle Aufgaben sichtbar und neue Aufgaben können hinzugefügt oder entfernt werden.

Abbildung 3: Dashboard

Add Exercise

Auf dieser Seite können die Betreuer neue Aufgaben erfassen. Um auf diese Seite zugreifen zu können, muss man ebenfalls eingeloggt sein.

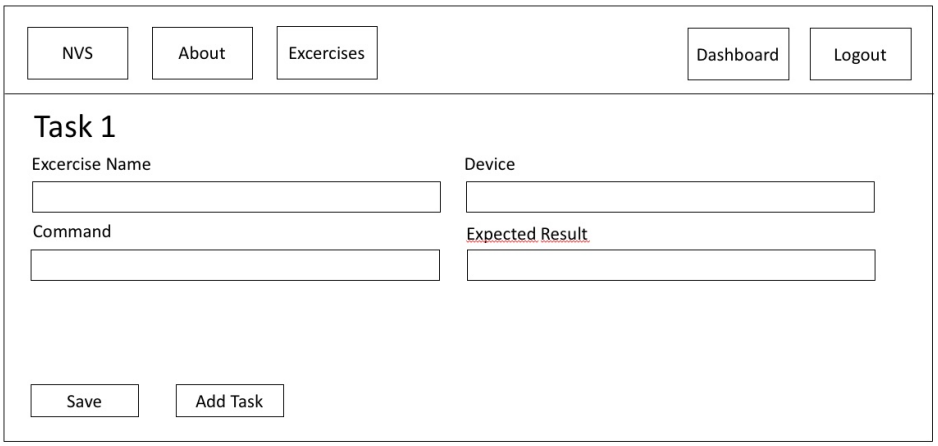


Abbildung 4: Exercise Page

4.3 Effektive Webseite

Mit diesen Entwürfen konnte schliesslich die Webseite umgesetzt werden. Im Verlauf der Arbeit änderten sich jedoch die Anforderungen, weshalb das Endresultat nicht genau den Entwürfen entpricht. Das Endresultat kann jedoch alle Ansprüche zufrieden stellen.

Startseite

Die Startseite ist die erste Seite, welche ein Benutzer zu sehen bekommt. Aus diesem Grund wurde darauf geachtet, dass diese Seite einfach und übersichtlich gestaltet ist.

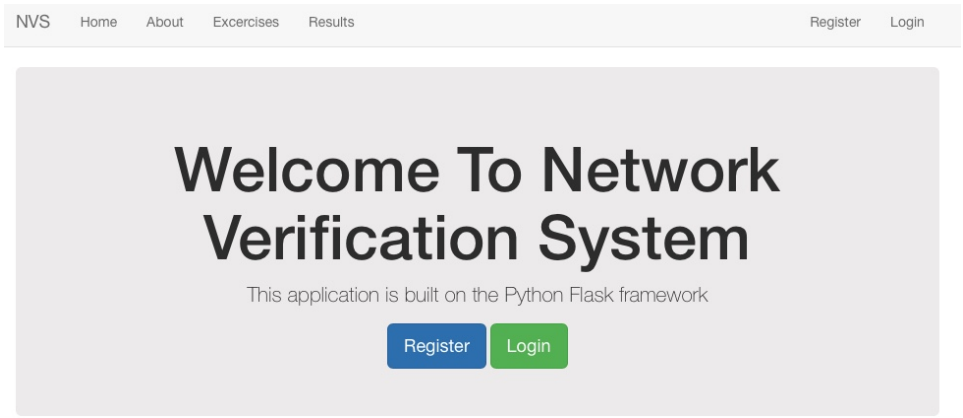


Abbildung 5: Startseite

Test Mode

Es gibt zwei verschiedene Möglichkeiten, wie ein Benutzer sein Netzwerk testen kann. Falls seine Devices als Pod hinterlegt sind, dann kann der Benutzer einfach ”Select a pod” auswählen. Dies hat den Vorteil, dass die Informationen aus einem vordefinierten Pod File geholt werden können und der Benutzer die Informationen nicht manuell eingeben muss.

Befinden sich die Devices aber nicht in einem Pod, dann kann der Benutzer ”Enter information manually” auswählen und die Informationen manuell eingeben.

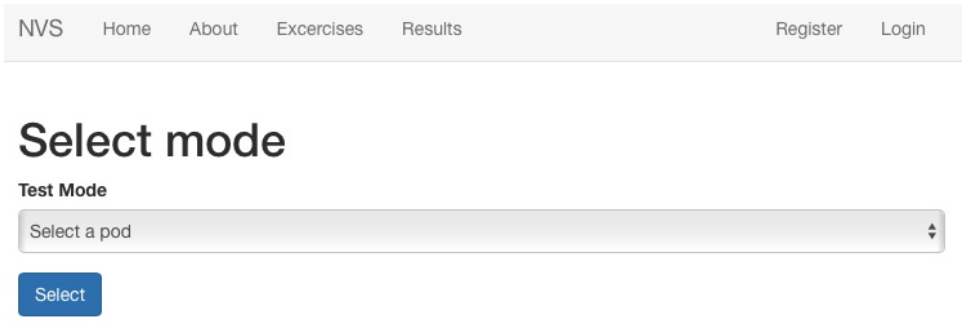


Abbildung 6: Test mode

Select Pod

Falls der Benutzer ”Select a pod” ausgewählt hat, wird er auf diese Seite weitergeleitet. Hier muss er lediglich den Pod auswählen, auf welchem er seine Konfiguration vorgenommen hat. Des weiteren muss er noch angeben, welche Aufgaben getestet werden sollen, ob per SSH oder Telnet auf die Devices zugegriffen werden soll und den Benutzernamen respektive das Passwort. Zudem kann der Benutzer noch angeben, ob er eine E-Mail mit den Ergebnissen erhalten möchte. Dies ist jedoch optional.

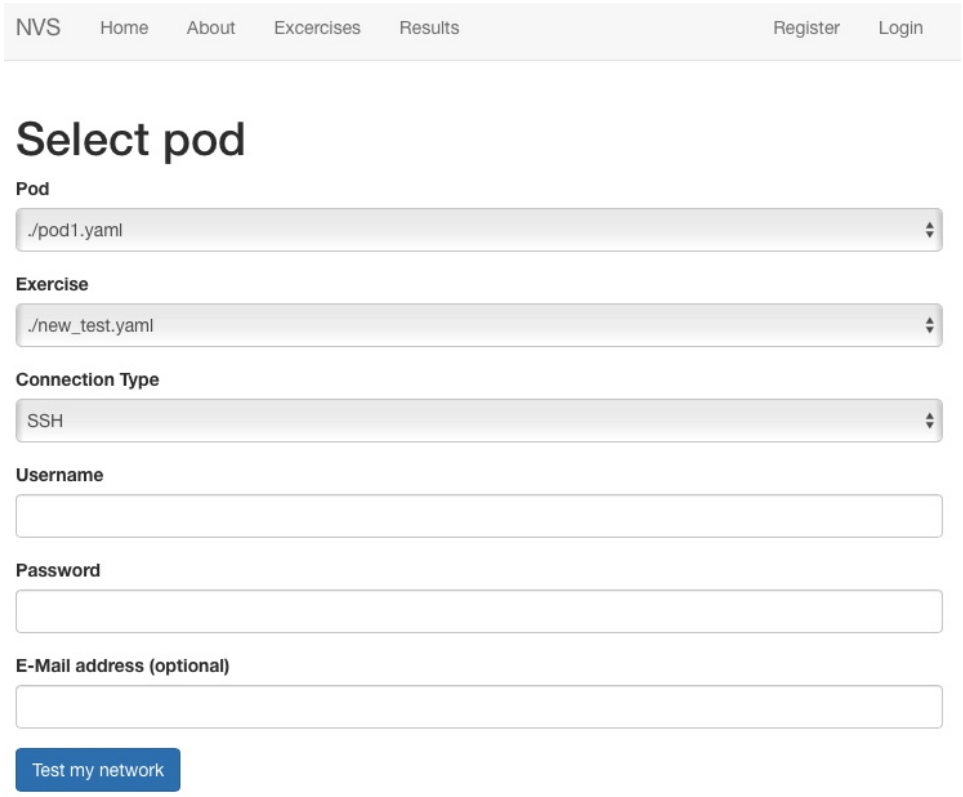


Abbildung 7: Pod Modus

Result

Sind die Tests durchgelaufen, kann der Benutzer die Resultate anschauen. Auf dieser Seite sind die Test Cases aufgelistet, welche getestet wurden, und das Resultat, ob die Tests bestanden wurden oder etwas schief gelaufen ist, wird angezeigt.

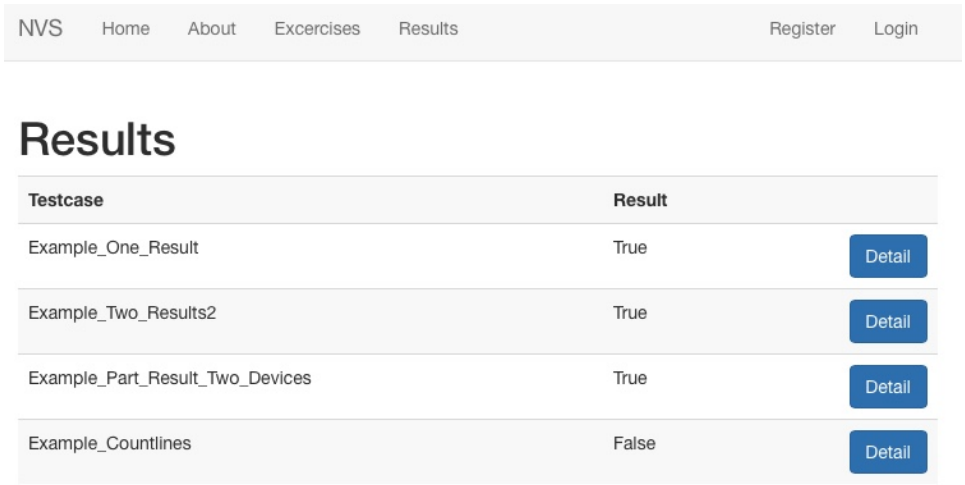


Abbildung 8: Resultat Zusammenfassung

Result Detail

Möchte der Benutzer die Details eines einzelnen Test Cases anschauen, wird er folgende Seite sehen. Hier ist ersichtlich, welcher Befehl auf welchem Device ausgeführt wurde. Zudem sieht der Benutzer, welches Resultat erwartet wurde und welches Resultat tatsächlich erhalten wurde.

Abbildung 9: Resultat Detail

Dashboard

Die Betreuer haben Zugriff auf das Dashboard. Hier sind alle Aufgaben aufgelistet. Zudem können neue Aufgaben erstellt werden und bereits existierende Aufgaben können angepasst, oder gelöscht werden.

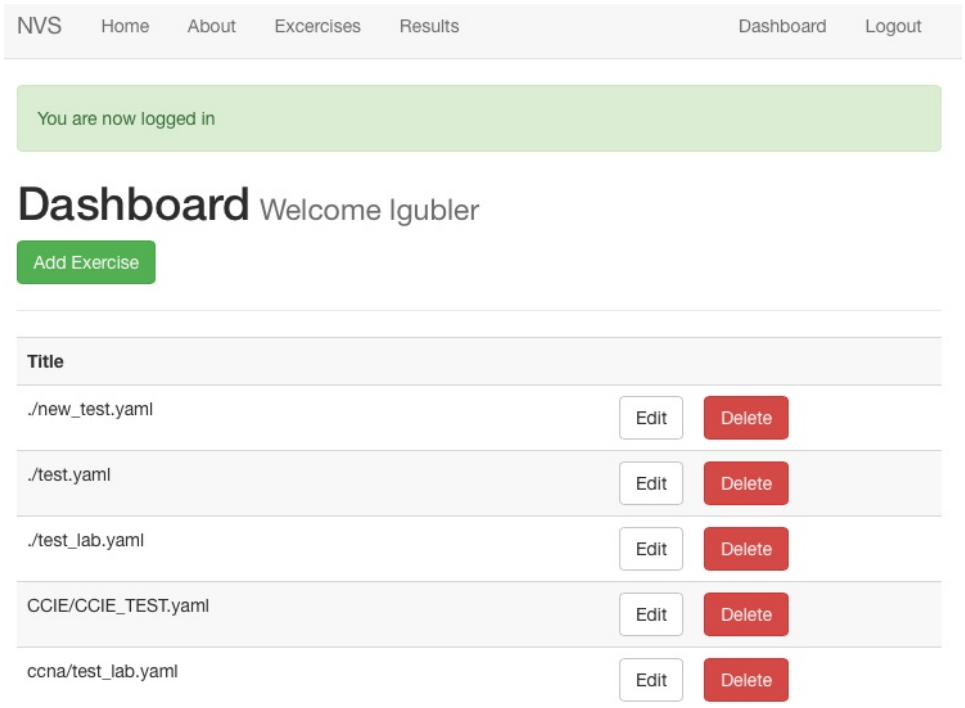


Abbildung 10: Dashboard

Add Exercise

Im Vergleich zum Mockup hat sich diese Seite wohl am stärksten geändert. Zu Beginn war geplant, dass für jede benötigte Information ein einzelnes Eingabefenster erstellt wird. Um dies umzusetzen hätte man jedoch einen grossen Mehraufwand gehabt. Aus diesem Grund stehen nun nur zwei Eingabefelder zur Verfügung. Im Feld ”Title” kann der Name für die neu erstellte Exercise angegeben werden. Im Feld ”Body” können dann die einzelnen Test Cases erfasst werden. Die Test Cases werden anschliessend als YAML Datei abgespeichert.

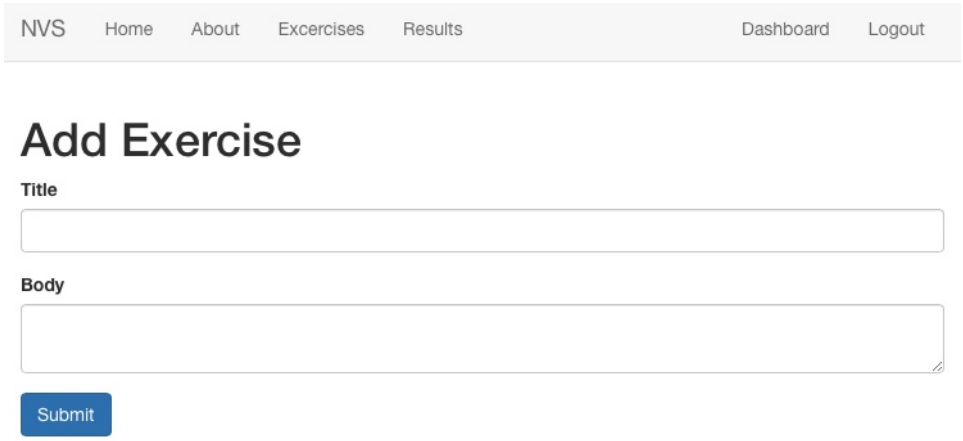


Abbildung 11: Add Exercise

Edit Exercise

Wenn ein Betreuer auf dem Dashboard auf ”Edit” drückt, wird er auf diese Seite weitergeleitet. Hier wird die existierende Datei geladen und kann nun durch den Betreuer angepasst werden.

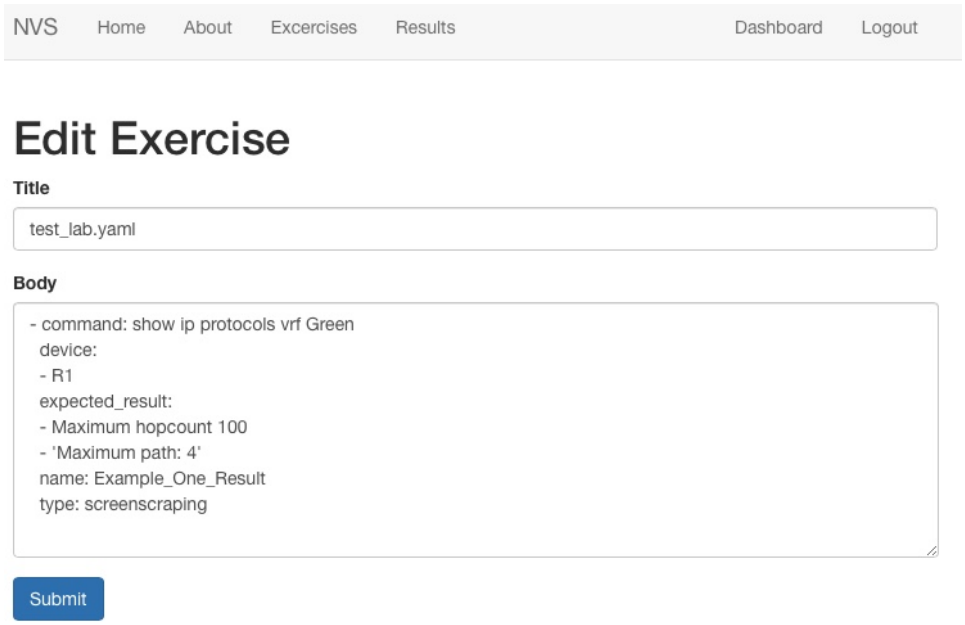


Abbildung 12: Edit Exercise

5 Technologien

5.1 Flask

Flask ist ein Micro Webframework geschrieben in Python. Es ist als Microframework klassifiziert, da keine speziellen Tools oder Libraries erforderlich sind. [6]

Werkzeug

Werkzeug ist eine Web Server Gateway Interface (WSGI) Application Library. [3] WSGI ist eine Spezifikation, in welcher beschrieben wird, wie der Web-Server mit der Web-Applikation kommuniziert und wie die Web-Applikation die Anfragen bearbeitet. [4]

Jinja 2

Jinja 2 ist eine moderne Templating Language für Python und wurde nach Django's Templates modelliert. [5] Mit einer Templating Engine kann ein Template verarbeitet werden. So kann die Webseite dynamisch gestaltet werden und es muss weniger Hypertext Markup Language (HTML)-Code geschrieben werden. [17]

5.2 Nornir

In diesem Abschnitt wird eine kurze Einführung in Nornir gegeben. Damit Nornir funktioniert, braucht es zusätzlich noch ein "Hosts.yaml" File, in welchem die einzelnen Devices gelistet werden und ein "Groups.yaml" File, bei welchem die Credentials hinterlegt sind.

Hosts.yaml

Das Hosts File wird verwendet, um die Informationen zu den einzelnen Devices zu verwalten. Nachfolgend ist ein Beispiel eines Hosts Files, wie es im Network Verification System verwendet wird:

Listing 1: Example Hosts File

```
R1:
  hostname: 152.96.11.31
  port: 2041
  username: admin
  password: cisco123
  platform: cisco_ios
R2:
  hostname: 152.96.11.31
  port: 2042
  username: admin
  password: cisco123
  platform: cisco_ios
```

Groups.yaml

Das Network Verification System braucht eigentlich kein Groups File, da die Credentials direkt ins Hosts File geschrieben werden. Nichts desto trotz muss ein leeres Groups File vorhanden sein, da Nornir sonst eine Exception wirft. Möchte man zu einem späteren Zeitpunkt trotzdem nocht ein Groups File erstellen, könnte dieses folgendermassen aussehen:

Listing 2: Example Groups File

```
core:
  username: admin
  password: S3cr3tP455w0rd
  platform: cisco_ios
lab:
  username: admin
  password: password123
  platform: cisco_ios
```

Beim Hosts File muss man lediglich noch angeben, zu welcher Gruppe dieses gehören soll.

Listing 3: Hosts File with groups

```
R1:
  hostname: 152.96.11.31
  port: 2041
  groups:
  - core
R2:
  hostname: 152.96.11.31
  port: 2042
  groups:
  - core
```

Der grosse Vorteil des Groups File liegt darin, dass die Informationen, welche bei allen Hosts gleich sind, in ein separates File ausgelagert werden können und man sich nicht wiederholen muss. Wenn es mehrere Devices gibt, welche alle denselben username haben, dann muss man diese Information nicht zu jedem Host angeben.

Ausführung

Nornir erlaubt nun das Ausführen von Befehlen auf den einzelnen Devices oder Gruppen. Folgender Code ist für die Ausführung der Befehle zuständig und wird im Modul ”runner.py” verwendet:

Listing 4: Example Runner

```
def run_test(self, test_case):
    result = self.nr.run(
        task=netmiko_send_command,
        enable=True,
        command_string=test_case.get_command(),
        on_failed=True
    )
    return result
```

Nornir ist nur für die Orchestrierung zuständig. Für die Ausführung der Befehle auf den Devices kann Netmiko oder Napalm verwendet werden. task=netmiko_send_command und command_string=test_case.get_command() sind dazu da, um die einzelnen Befehle auszuführen. Mit netmiko_send_command hat man den Vorteil, dass der exakt gleiche Befehl angeben werden kann, welchen man auch auf dem Device ausführen würde. So ist man extrem flexibel, da man jeden Befehl eingeben kann, welchen man möchte. enable=True führt dazu, dass die Befehle im EXEC mode ausgeführt werden. So hat man keine Probleme, weil man die Befehle mit falschen Berechtigungen ausführen möchte.

6 Software Architektur

6.1 Systemübersicht

Die Systemübersicht gibt einen Überblick über die verschiedenen Komponenten des Systems. Nachfolgend sind die einzelnen Komponenten genauer beschrieben.

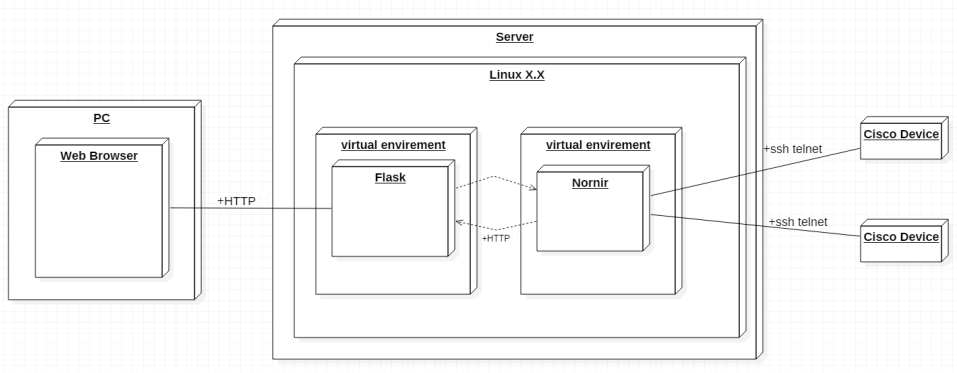


Abbildung 13: Systemübersicht

Client

Der Client greift via Browser seines Desktop-PCs auf die Webseite zu. Da der Webserver nicht von extern erreicht werden muss, wird HyperText Transfer Protocol (HTTP) als Protokoll verwendet

Webserver

Auf dem Webserver läuft ein Flask-Microframework. Hier kann der User die nötigen Angaben machen, zum Beispiel welches Lab und welche Devices er gerne überprüfen möchte. Der Webserver erstellt die benötigten Dateien und übergibt diese dem NetworkRunner.

NetworkRunner

Der NetworkRunner validiert die vom Webserver erstellen Dateien. Wenn die Dateien valide sind, wird Nornir gestartet und sorgt dafür, dass eine Verbindung zu den Devices aufgebaut wird und die Befehle ausgeführt werden. Nornir gibt die Resultate an den NetworkRunner zurück. Dieser überprüft, ob die Resultate mit den erwarteten Werten übereinstimmen. Die Resultate werden an den Webserver zurückgesendet, welcher diese in einer Datenbank abspeichert.

Cisco Device

Nornir stellt eine Verbindung zu den Cisco Devices her und setzt Befehle auf diesen Devices ab. Vorerst werden nur Cisco Devices unterstützt.

6.2 Schnittstellen

Die Schnittstelle zwischen dem Webserver (Flask) und dem NetworkRunner (Nornir) ist nicht ganz trivial und besteht aus verschiedenen Komponenten. Sobald ein Netzwerk geprüft werden soll, startet der Webserver eine Instanz des NetworkRunners und gibt folgende Argumente mit:

1. Pfad zur Host.yaml Datei
2. Pfad zur Exercise.yaml Datei
3. Cookie
4. Mail Adresse

Auf die Eigenheiten der einzelnen Argumente wird in den nachfolgenden Kapiteln eingegangen. Sobald die Tests durchgeführt wurden, wird das Ergebnis mittels Webhook zurück an den Webserver übermittelt.

host.yaml

Durch den Inhalt dieser Datei wird übermittelt, welche Devices Mitglieder des zu testenden Netzwerkes sind. Die Datei muss folgendermassen strukturiert sein:

Listing 5: Beispiel Host.yaml

```
R1:
  hostname: 152.96.11.31
  password: password
  platform: cisco_ios_telnet
  port: 2041
  username: username
R2:
  hostname: 152.96.11.31
  password: password
  platform: cisco_ios_telnet
  port: 2042
  username: username
```

Pro Devicename muss die IP-Adresse, der Port, die Art des Zugriffes (telnet oder ssh) und die Logindaten angegeben werden. Die Liste der Devices kann beliebig lang sein.

exercise.yaml

Durch den Inhalt dieser Datei wird übermittelt, welche Tests auf welchen Devices ausgeführt werden sollen.

Listing 6: Beispiel Exercise.yaml

```
- command: ping vrf Green 10.1.12.2
  device:
  - R1
  expected_result:
  name: Example_One_Result
  type: ping
- command: show ip protocols vrf Green
  device:
  - R1
  expected_result:
  - Maximum hopcount 100
  - 'Maximum path: 4'
  name: Example_One_Result
  type: screenscraping
```

Wie die Tests genau aufgebaut werden, ist im Kapitel ”Testaufbau” beschrieben.

Cookie

Für jeden Benutzer der Applikation wird ein Cookie erstellt, mit welchem der Benutzer eindeutig identifizierbar ist. Somit kann der Benutzer Resultate ansehen, ohne dass ein Login eingerichtet werden muss.
Dieses Cookie wird beim Ausführen der Tests an den NetworkRunner übergeben. Sobald die Tests durchgeführt wurden und die Ergebnisse zurück an den Webserver gesendet werden, wird das Cookie ebenfalls mitgesendet, damit die Resultate einem Benutzer zugeordnet werden können.

Mail-Adresse

Die Mail-Adresse wird gleich wie das Cookie an den NetworkRunner übergeben und zusammen mit den Ergebnissen wieder an den Webserver gesendet. Dies ist notwendig, da der Webserver den Benutzer über den Ausgang der Tests informiert und daher die Adresse einer bestimmten Testdurchführung zugeordnet werden können muss.

Webhook

Sobald die Tests durchgeführt wurden, wird das Ergebnis per Webhook zurück zum Webserver gesendet. Die Daten werden als yaml Datei übertragen.

Listing 7: Beispiel Webhookdaten

```
- name: Example_Part_Result:
  command: 'show ip protocols vrf Green'
  device:
  - R1
  expected_result:
  - 'Active Timer: 3 min'
  - 'Maximum hopcount 100'
  result: 'Effektives Resultat des abgesetzten Befehls'
  status: true
  cookie: 'iasnfdzofk23r23sdfdg'
  mail: 'hansmuster@hsr.ch'
```

6.3 Logische Architektur

Da das Network Verification System aus zwei Applikationen besteht, gibt es auch zwei logische Architekturen. Zum einen eine 3-Tier Architektur, mit ”Presentation Layer”, ”Business Layer” und ”Database Layer” für den Webserver und zum anderen eine 1-Tier Architektur für den NetworkRunner. Das Projekt besteht aus zwei verschiedenen Programmen, sodass diese so gut wie nur möglich unabhängig voneinander sind. Durch diese Entscheidung ist es relativ einfach, einzelne Instanzen des Runners hochzufahren, welche jeweils die Tests für einen Benutzer durchführen.

Webserver

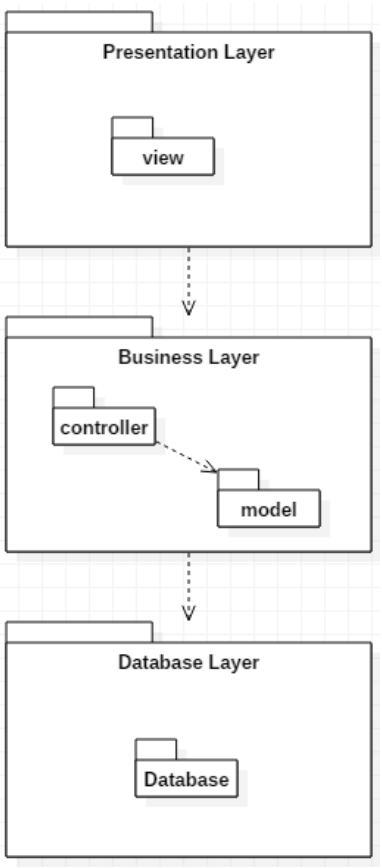


Abbildung 14: Architektur Webserver

In den nachfolgenden Kapiteln werden die einzelnen Layer genauer beschrieben.

Presentation Layer

Im Presentation Layer befindet sich die Implementation des GUIs. Das GUI wird mit Hilfe des Python Microframework Flask und der Templating Engine Jinja2 implementiert.

View

Die View rendert die Daten und stellt sie für den Benutzer dar.

Business Layer

Der Business Layer hat verschiedene Zuständigkeiten. Zum einen ist er dafür verantwortlich, dass das Routing einwandfrei funktioniert. Zum anderen müssen auch die einkommenden Daten für den jeweiligen Gebrauch aufbereitet werden. Zusammenhängend mit dem Aufbereiten der Daten, müssen auch die verschiedenen Schnittstellen besetzt werden.

Controller

Im Controller befindet sich das Routing, mit welchem anschliessend auf die einzelnen Blueprints zugegriffen wird. Zudem ist die Logik in den Controllern implementiert.

Module

Damit die Modularität beibehalten wird, wird die Logik in eigenen Modulen implementiert. Diese können somit einfach hinzugefügt, erweitert oder entfernt werden.

Database Layer

Der Database Layer ist für die Erstellung der Datenbank und den dazugehörigen Tabellen zuständig. Der Zugriff auf die Datenbank wird jedoch nicht zentral verwaltet.

6.4 Projektstruktur

Jedes Modul, abgesehen vom Modul ”extensions”, ist gleich aufgebaut und besteht aus einem Controller und einer beliebigen Anzahl Forms.

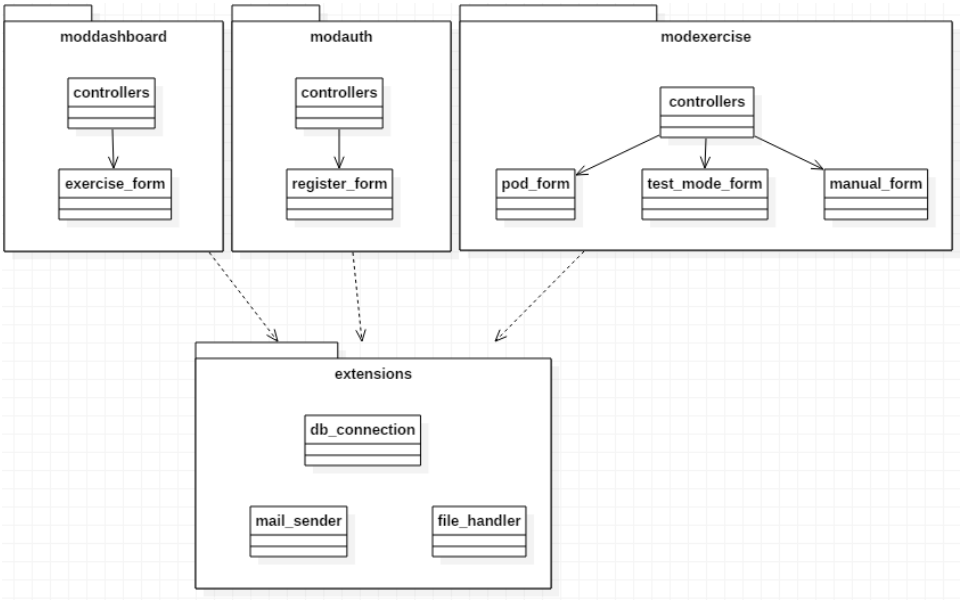


Abbildung 15: Packagediagramm

Package mod*

Klasse	Beschreibung
controllers	In den Controller Klassen befindet sich die Logik, für die Aufbereitung der anzuzeigenden Daten und die Weiterleitung auf andere Seiten.
forms	Forms sind Klassen, welche für das übermitteln von Formulardaten auf andere Seiten benötigt werden.

Package ”Extensions”

Klasse	Beschreibung
db_connection	Die ”db_connection” ist ein Abbild der Datenbank.
mail_sender	Ist zuständig für das Versenden der Mails.
file_handler	Enthält die Logik, um Dateien auszulesen und diese zu bearbeiten.

6.5 NetworkRunner

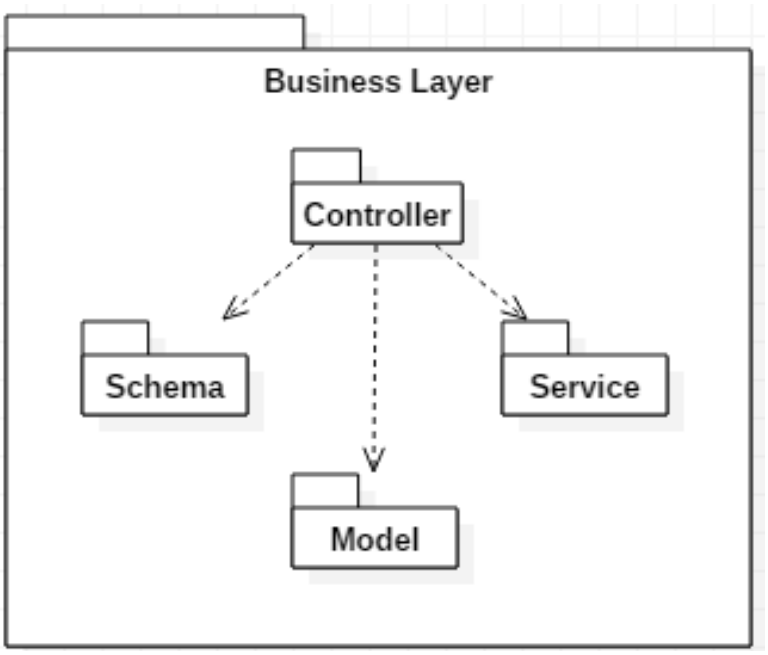


Abbildung 16: Architektur NetworkRunner

Business Layer

Der Business Layer des NetworkRunners ist zuständig für das Ausführen der Tests. Dazu gehört das Entgegennehmen der Benutzerangaben aus dem Webserver und das daraus resultierende einlesen der angegeben Dateien und dessen Validierung. Die wohl wichtigste Aufgabe ist jedoch das Absetzten der Befehle auf den einzelnen Devices und die Evaluierung des Resultats.

Controller
Die Controller kontrollieren den kompletten Testablauf.

Schema
Hier befinden sich die definierten Schemas, mit welchen der Inhalt der Testdateien validiert wird.

Service
Die Service-Klassen bieten alle nötige Funktionalität an um Tests auszuführen und die ausgeführten Tests zu evaluieren.

Model
Im Model werden die internen Daten gespeichert, damit diese einfacher verwendet werden können.

6.6 Klassenstruktur

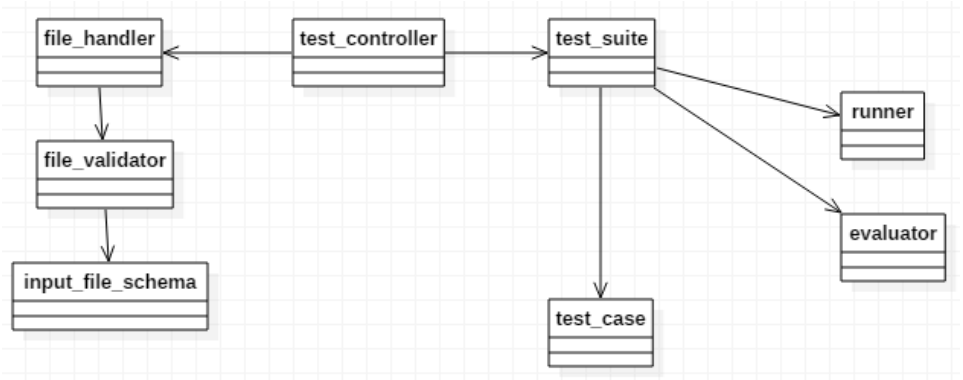


Abbildung 17: Klassendiagramm

Package ”Controller”

Die Klassen im Package ”Controller” steuern den Datenfluss und den Ablauf des Tools.

Klasse	Beschreibung
test_controller	Steuert den kompletten Ablauf und dient zusätzlich als Schnittstelle zum Frontend.
test_suite	Die TestSuite stellt die auszuführenden Tests dar und ist zuständig, dass die Befehle auf den richtigen Devices ausgeführt werden.

Package ”Service”

Die Klassen im Package ”Service” bieten jeweils Logik an, welche von den Controllern gesteuert wird.

Klasse	Beschreibung
file_handler	Enthält die Logik, Dateien auszulesen und diese zu bearbeiten.
file_validator	Der FileValidator stellt sicher, dass die vom FileHandler eingelesenen Daten korrekt sind.
runner	Der Runner führt die Befehle auf den entsprechenden Devices aus.
evaluator	Der Evaluator vergleicht die Ergebnisse des Runners mit den angegeben erwarteten Ergebnissen.

Package ”Model”

Die Klassen im Package ”Model” enthalten Daten, welche notwendig sind um die Tests korrekt auszuführen.

Klasse	Beschreibung
test_case	Stellt einen auszuführenden Test dar mit dem dazugehörenden erwarteten Ergebnis.

Package ”Schema”

Die Klassen im Package ”Schema” definieren wie die einzulesenden Daten strukturiert sein müssen.

Klasse	Beschreibung
input_file_schema	Das InputFileSchema definiert, wie die Testdatei strukturiert sein muss.

6.7 Sequenzdiagramm

Die Grafik unten stellt den Ablauf abstrakt dar, da nur auf die wichtigsten Dinge eingegangen wird.
Der gesamte unten dargestellte Ablauf läuft asynchron. Das heisst, für jeden Benutzer, der sein Netzwerk testen will, wird eine eigene Runner-Instanz erstellt, welche unabhängig der anderen Instanzen das gegebene Netzwerk prüft. Dies ist notwendig, damit Benutzer ihre Netzwerke zeitgleich testen können. Wichtig hierbei ist jedoch, dass das gleiche Netzwerk nicht von mehreren Benutzern zur selben Zeit getestet wird.

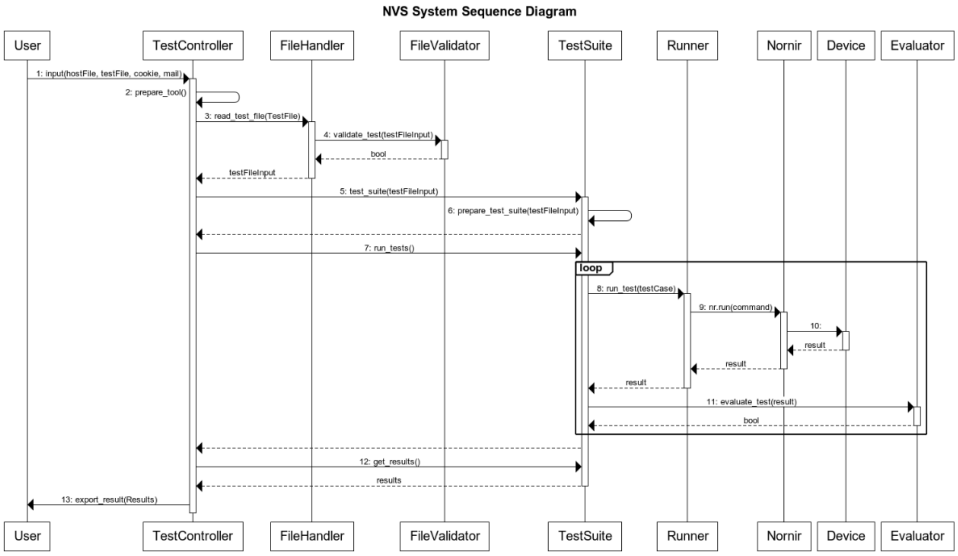


Abbildung 18: Sequenzdiagramm

1. Wenn der User einen Test startet, wird dem TestController der Pfad zur Hosts.yaml Datei und der Pfad zur ausgewählten Übung geschickt. Zusätzlich wird noch das persönliche Cookie des Users und dessen Mail-Adresse mitgesendet.

2. Der TestController bereitet sich auf das Ausführen der Tests vor. Er generiert die notwendigen temporären Konfigurationsdateien, damit die Tests unabhängig voneinander auf den Devices ausgeführt werden können.

3. Der TestController dirigiert dem FileHandler, das TestFile einzulesen.

4. Der FileHandler übergibt dem FileValidator den Inhalt der Datei. Dieser überprüft, ob der Inhalt der Datei korrekt ist. Anschliessend gibt er einen Bool an den FileHandler zurück und teilt diesem so mit, ob der Inhalt valide ist oder nicht.

5. Der TestController übergibt nun die ausgelesenen Tests der TestSuite.

6. Für jeden Test wird nun ein Test-Objekt erstellt, welches den auszuführenden Test darstellt. Zusätzlich wird für jedes Device eine eigene Runner-Instanz inizialisiert.
7. Die Tests sollen nun ausgeführt werden.
8. In einem Loop wird dem Runner nun Test für Test übergeben und soll ausgeführt werden.
9. Der Runner konfiguriert das Nornir und startet dieses.
10. Der Befehl wird auf dem Device abgesetzt. Das Ergebnis wird bis zur TestSuite zurückgeleitet.
11. Der Evaluator vergleicht nun das Resultat mit dem erwarteten Resultat. Anschliessend wird ein bool an die TestSuite zurück gegeben.
12. Wenn alle Tests durchgelaufen sind, holt der TestController alle Resultate bei der TestSuite.
13. Der TestController nimmt alle Resultate und erstellt daraus ein Reporting. Dies wird per Webhook dem Frontend gesendet.

6.8 Datensicherung und Backup

Die Applikation erstellt keine Backups der Exercise und Podfiles oder der Datenbank. Da in der Datenbank aber keine relevanten Daten abgelegt sind, ist dies nicht weiter notwendig.

Es wird jedoch empfohlen, dass Backups von den Exercise und Pod Files erstellt werden. Dies könnte mit einem einfachen Cronjob umgesetzt werden, welcher folgendermassen aussehen könnte:

Listing 8: Example Cronjob

```
0 1 * * * /bin/mv /path/to/files/* /path/to/backup/directory
```

Des Weiteren können die einzelnen Files mit einem Git-Repository verwaltet werden.

6.9 Datenbank

Da man die Applikation simpel halten wollte, wurde nur eine Datenbank mit zwei Tabellen erstellt. Zwischen den beiden Tabellen existiert keine Assoziation.

User

In der Tabelle ”user” werden die einzelnen User verwaltet. Nur die Betreuer sind in der Lage, sich als Benutzer anzumelden. Die Teilnehmer werden anhand eines Cookies identifiziert. Möchte man zu einem späteren Zeitpunkt, dass sich die Teilnehmer auch registrieren und anmelden können, muss die Tabelle ”user” so erweitert werden können, dass zwischen Benutzer und Administrator unterschieden werden kann und eine Assoziation zur Tabelle ”result” muss hinzugefügt werden.

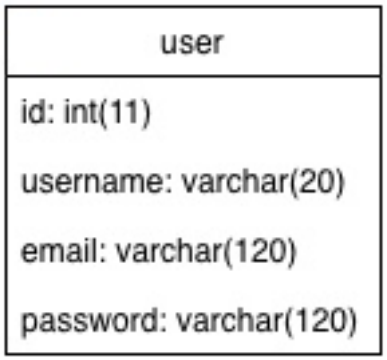


Abbildung 19: User Tabelle

Result

In der Tabelle ”result” befinden sich die Resultate der durchgeführten Tests. Zusätzlich zum eigentlichen Resultat wird noch das Cookie des Benutzers und eine Session gespeichert. Zum jetztigen Zeitpunkt werden die Benutzer anhand eines Cookies identifiziert. Möchte der Benutzer auf ältere Resultate zugreifen, kann er dies mit Hilfe der Session bewerkstelligen.

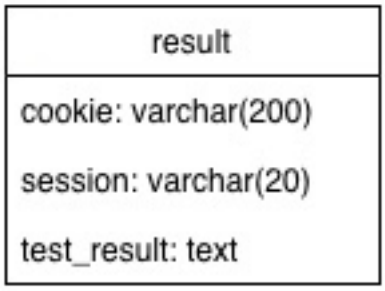


Abbildung 20: Result Tabelle

Relationale Datenbank

Der Webserver wurde mit einer MySQL Datenbank entwickelt und getestet. Schlussendlich ist es jedoch dem Enduser überlassen, welche Datenbank eingesetzt werden soll. Um die Datenbank zu wechseln, muss lediglich die Variable ”SQLALCHEMY_DATABASE_URI” im File config.py angepasst werden.

Nachfolgend sind Beispiele, wie der Connection String für eine MySQL oder Postgres Datenbank aussehen könnte.

Listing 9: Database Connection String

```
# MySQL Connection String
SQLALCHEMY_DATABASE_URI = 'mysql://root:start123@localhost/nvs'

# Postgres Connection String
SQLALCHEMY_DATABASE_URI = 'postgresql://root:start123@localhost:5432/nvs'
```

6.10 API Endpoints

Nachfolgend wird mit einer Grafik die API Endpoints beschrieben.

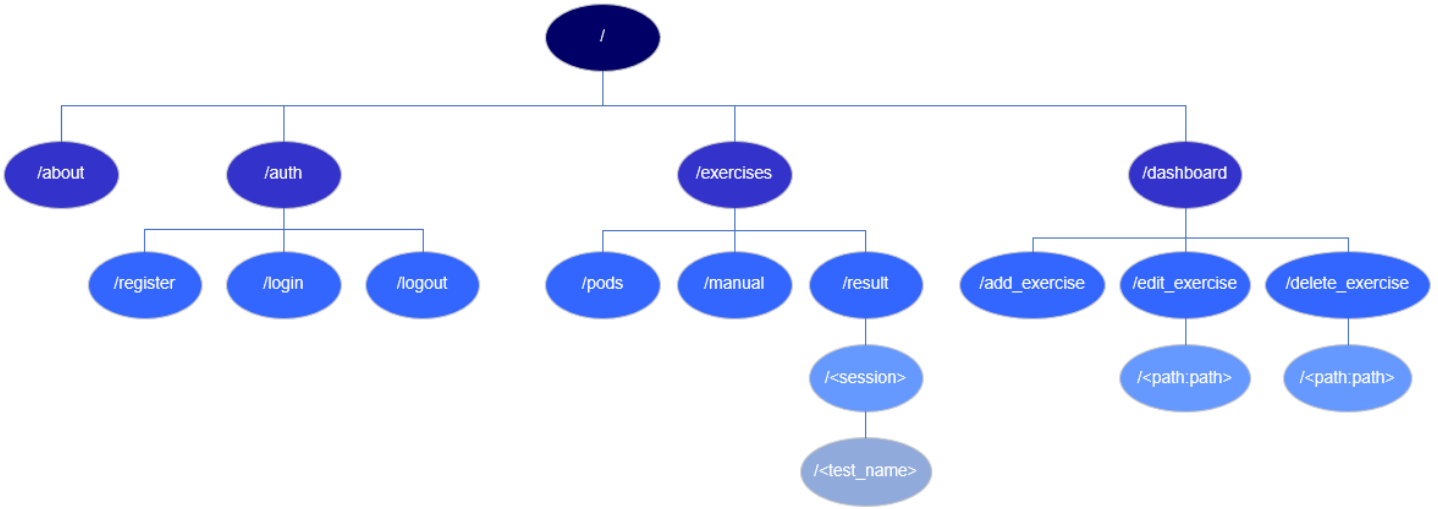


Abbildung 21: Beschreibung der API Endpoints

Homepage

Title	**Darstellung Homepage**
URL	/
Method	GET
URL Params	-

About

Title	**Allgemeine Informationen**
URL	/about
Method	GET
URL Params	-
Beschreibung	Informationen zu den Studenten hinter diesem Projekt

Auth

Title	**Darstellung Registrierungs Seite**
URL	/auth/register
Method	GET
URL Params	-

Title	**Registrierung eines Benutzers**
URL	/auth/register
Method	POST
URL Params	-

Title	**Darstellung Login Seite**
URL	/auth/login
Method	GET
URL Params	-

Title	**Login**
URL	/auth/login
Method	POST
URL Params	-
Beschreibung	Falls der angegebene Benutzer existiert und das Passwort übereinstimmt, dann wird der Benutzer auf das Dashboard weitergeleitet. Ansonsten wird die Login Seite neu geladen und eine Fehlermeldung eingeblendet.

Title	Logout
URL	/auth/logout
Method	GET
URL Params	-
Beschreibung	Die Session wird gelöscht und der Benutzer wird auf die Login Seite weitergeleitet.

Exercises

Title	Darstellung der Test Modi
URL	/exercises
Method	GET
URL Params	-
Beschreibung	Falls beim Benutzer kein Cookie gesetzt wird, wird hier eines gesetzt.

Title	Auswahl Pod oder Manuell
URL	/exercises
Method	POST
URL Params	-
Beschreibung	Weiterleitung des Benutzers auf die entsprechende Seite.

Title	Darstellung Pod Seite
URL	/exercises/pods
Method	GET
URL Params	-
Beschreibung	Die vordefinierten Pods und Exercises können durch den Benutzer im Dropdown Menü ausgewählt werden. Die Dateien müssen im Ordner abgelegt werden, welcher im Config File definiert wurde.

Title	Ausführung der Tests
URL	/exercises/pods
Method	POST
URL Params	-
Beschreibung	NetworkRunner wird als Subprocess aufgerufen. Sobald alle Tests durchgelaufen sind, meldet sich der NetworkRunner mit einem POST Request bei /exercises/result.

Title	**Darstellung manuelle Eingabe**
URL	/exercises/manual
Method	GET
URL Params	-
Beschreibung	Die Exercises können durch den Benutzer im Dropdown Menü ausgewählt werden. Die Dateien müssen im Ordner abgelegt werden, welcher im Config File definiert wurde.

Title	**Ausführung der Tests**
URL	/exercises/manual
Method	POST
URL Params	-
Beschreibung	NetworkRunner wird als Subprocess aufgerufen. Sobald alle Tests durchgelaufen sind, meldet sich der NetworkRunner mit einem POST Request bei /exercises/result.

Title	**Darstellung der Resultate**
URL	/exercises/result
Method	GET
URL Params	-
Beschreibung	Zuerst wird getestet, ob beim Benutzer ein Cookie gesetzt ist und ob Resultate zu diesem Cookie in der Datenbank gefunden wurden. Existiert ein Eintrag mit dem entprechenden Cookie in der Datenbank, werden die Resultate dargestellt. Ansonsten wird die Seite ohne Einträge geladen. Da mehrere Einträge mit dem selben Cookie in der Datenbank existieren können, wird in absteigender Reihenfolge nach dem Cookie gesucht. So wird stets der neuste Eintrag gefunden.

Title	**Schreiben der Resultate in Datenbank**
URL	/exercises/result
Method	POST
URL Params	-
Beschreibung	Der NetworkRunner meldet sich bei dieser Route mit einem POST Request und die Daten werden in die Datenbank geschrieben. Hat der Benutzer eine E-Mail Adresse angegeben, wird eine E-Mail mit dem Link zu seinen Resultaten geschickt.

Title	Darstellung der Resultate
URL	/exercises/result/<session>
Method	GET
URL Params	**session** Wird zum Laden von Einträgen aus der Datenbank verwendet. Beispiel: /exercises/result/6ipA3Qw72UY
Beschreibung	Existiert ein Eintrag mit der entprechenden Session in der Datenbank, werden die Resultate dargestellt. Ansonsten wird eine Exception geworfen.

Title	Detailansicht eines einzelnen Test Cases
URL	/exercises/result/<session>/<test_name>
Method	GET
URL Params	**session** Wird zum Laden von Einträgen aus der Datenbank verwendet. Beispiel: /exercises/result/6ipA3Qw72UY **test_name** Wird zur Darstellung eines einzelnen Test Cases verwendet. Beispiel: /exercises/result/detail/6ipA3Qw72UY/Test1
Beschreibung	Wenn ein Eintrag zum entsprechenden test_name existiert, werden die Details zu diesem Test angezeigt.

Dashboard

Title	Liste der Exercises
URL	/dashboard
Method	GET
URL Params	-
Beschreibung	Hier wird eine Liste der Exercises dargestellt. Es ist darauf zu achten, dass sich die Exercises im Ordner befinden, welcher im Config File angegeben ist. Sind keine Exercises im Ordner vorhanden, wird eine Warnung gezeigt.

Title	Darstellung zum hinzufügen einer neuen Exercise
URL	/dashboard/add_exercise
Method	GET
URL Params	-
Beschreibung	Ein leeres Formular wird angezeigt, mit welchem eine neue Exercise erfasst werden kann.

Title	**Erstellung einer neuen Exercise**
URL	/dashboard/add_exercise
Method	POST
URL Params	-
Beschreibung	Sind alle Angaben gemacht, wird eine neue Exercise im Ordner erstellt, welcher im Config File angegeben wurde. Es ist darauf zu achten, dass ".yaml" als Dateiendung verwendet wird. Die Dateien können auch in einem Subfolder abgelegt werden. Beispiel: /CCIE/Lab_01/AwesomeTest.yaml

Title	**Darstellung zum hinzufügen einer neuen Exercise**
URL	/dashboard/edit_exercise/<path:path>
Method	GET
URL Params	**<path:path>** Stellt den Pfad zum Exercise File dar. Beispiel: /dashboard/edit_exercise/test_lab.yaml /dashboard/edit_exercise/CCIE/lab101.yaml
Beschreibung	Falls das File im angegebenen Pfad existiert, wird der Inhalt des Files im Formular dargestellt und kann durch den Betreuer angepasst werden. Existiert kein File beim angegebenen Pfad, wird eine Error Page dargestellt.

Title	**Anpassung einer existierenden Exercise**
URL	/dashboard/edit_exercise/<path:path>
Method	POST
URL Params	**<path:path>** Stellt den Pfad zum Exercise File dar. Die Dateien können auch in einem Subfolder abgelegt werden. Beispiel: /dashboard/edit_exercise/test_lab.yaml /dashboard/edit_exercise/CCIE/lab101.yaml
Beschreibung	Der Inhalt des Files kann angepasst und abgespeichert werden.

Title	**Löschen einer Exercise**
URL	/dashboard/delete_exercise/<path:path>
Method	POST
URL Params	**<path:path>** Stellt den Pfad zum Exercise File dar. Die Dateien können sich auch in einem Subfolder befinden. Beispiel: /dashboard/delete_exercise/test_lab.yaml /dashboard/delete_exercise/CCIE/lab101.yaml
Beschreibung	Das File wird gelöscht.

7 Implementation

7.1 Pod Files

Bei der Benutzung des Network Verification Systems wird die Konfiguration der angegebenen Devices überprüft. Dazu braucht das Tool die IP Adressen und den Port. Im Lab stehen den Teilnehmern oftmals Pods zur Verfügung. Ein Pod ist nichts anderes als ein Zusammenschluss mehrerer Devices.
Anstatt die Informationen jedes Mal manuell einzufügen, können die Teilnehmer einen Pod auswählen.

Pods definieren

Damit das Tool die Pods erkennt, muss zuerst ein YAML Dokument mit den benötigten Informationen erstellt werden. Sobald das Dokument im entsprechenden Verzeichnis abgelegt wird, wird es vom Tool automatisch erkannt.
Um einen neuen Pod zu definieren, müssen folgende Schritte durchgeführt werden:

1. Erstellen einer neuen YAML Datei
2. Informationen zu den Devices in die Datei schreiben
3. Die Datei im entsprechenden Verzeichnis abspeichern. Das Verzeichnis ist im Config File des Webservers angegeben.

Sobald die Datei mit dem neu erstellten Pod im entsprechenden Verzeichnis abgelegt wurde, kann dieses verwendet werden. Dazu kann man auf das Dropdown-Menü 'Select Pod' klicken und den neu erstellten Pod auswählen.

Pods Struktur

Im nachfolgenden Kapitel wird genauer darauf eingegangen, welche Informationen benötigt werden, um einen Pod zu erstellen und wie diese eingetragen werden müssen. Pro Pod können beliebig viele Devices erfasst werden.
Für jedes Device müssen folgende Angaben gemacht werden:

- **Name**
  Der Name muss innerhalb desselben Pods eindeutig sein und sollte aussagekräftig gewählt werden. Es empfiehlt sich, dass hier derselbe Name gewählt wird, welcher auch in der Aufgabenstellung verwendet wurde. Der Name wird später beim Erstellen der Tests benötigt.

  - **Hostname**
    Der Hostname entspricht der IP-Adresse des Devices.
  - **Port**
    Hier wird der Port angegeben, auf welchem das Device erreichbar ist.
  - **Password** / **Username** / **Platform**
    Das Passwort, der Benutzername und der Verbindungstyp müssen nicht erfasst werden. Diese Felder werden beim Starten der Tests automatisch erstellt.

Im Bild unten sieht man eine Beispielkonfiguration mit vier Devices.

Listing 10: Pods Beispiel

```
R1:
  hostname: 152.96.11.31
  port: 2041
R1:
  hostname: 152.96.11.31
  port: 2042
R2:
  hostname: 152.96.11.31
  port: 2043
R3:
  hostname: 152.96.11.31
  port: 2044
```

7.2 Testaufbau

Beschreibung

Ein auszuführender Test wird folgendermassen definiert:

Listing 11: einfachen Testbeispiel

```
- name: Example_One_Result:
  command: 'Command'
  device:
  - R1
  expected_result:
  - 'Text of expected result'
  type: testtype
```

Auf Zeile eins wird der Name des Tests angegeben. Dieser ist für die Ausführung des Tests irrelevant, jedoch nicht für das spätere mapping. Auf der zweiten Zeile wird der auszuführende Befehl angegeben (z.B. "show vrf"). Auf Zeile drei wird angegeben, auf welchen Devices der Befehl ausgeführt werden soll. Einzelne Devices oder eine Gruppe von Devices, auf denen der Befehl ausgeführt werden soll, muss als Liste angegeben werden. Sollte ein Befehl auf mehreren Devices ausgeführt werden, muss dies wie folgt angegeben werden:

Listing 12: Test auf mehreren Devices

```
- name: Example_Two_Results:
  command: 'Command'
  device:
  - R1
  - R2
  expected_result:
  - - 'Text of expected result'
    - 'Text of expected result'
    - 'Text of expected result'
  - - 'Text of expected result'
  type: testtype
```

Sobald der Befehl auf mehr als einem Device ausgeführt werden soll, müssen auch entsprechend viele erwartete Ergebnisse angegeben werden. Diese befinden sich auf der letzten Zeile. Das Feld 'Text of expected result' wird ersetzt durch das erwartete Ergebnis, welches anschliessend mit dem effektiven Ergebnis verglichen wird. Im Beispiel oben sieht man, dass drei erwartete Ergebnisse für das Device 'R1' und ein erwartetes Ergebnis für das Device 'R2' angegeben wurden.

In den folgenden Kapiteln werden die Eigenheiten der unterschiedlichen Testtypen aufgezeigt.

Screenscraping

Bei den Screenscraping Tests, wird der Text des erwarteten Ergebnisses eins zu eins im effektiven Resultat gesucht. Wird der Text gefunden, ist der Test erfolgreich.

Beispiel show vrf

Die Konfiguration unten gibt an, dass der Befehl ”show vrf” ausgeführt werden soll und eine Zeile des Resultats, mit der Angabe des erwarteten Resultats übereinstimmen soll.

Listing 13: Beispiel Screenscraping

```
- name: Example_One_Result:
  command: 'show␣vrf'
  device:
  - R1
  expected_result:
  - 'Green␣<not␣set>␣ipv4␣Fa0/0.20'
  type: screenscraping
```

Beispiel ”überspringe Zeitangabe oder ähnliches”

Im Beispiel unten ist ersichtlich, wie eine Zeitangabe, oder ähnliches, übersprungen werden kann. Will man das Ergebnis auf folgenden Text überprüfen: ”10.1.13.3 1 FULL/DR 00:00:32 10.1.13.3 FastEthernet0/1”, muss die Zeitangabe ignoriert werden können, da diese von Testdurchführung zu Testdurchführung variiert. Dies ist möglich, indem man im erwarteten Ergebnis die Zeitangabe durch ein ”*” ersetzt. ”*” entspricht also einem Platzhalter, der beliebig gross sein kann.

Listing 14: Beispiel Screenscraping mit *

```
- name: Example_One_Result:
  command: 'show␣ip␣ospf␣neighbor'
  device:
  - R1
  expected_result:
  - '10.1.13.3␣1␣FULL/DR␣*␣10.1.13.3␣FastEthernet0/1'
  type: screenscraping
```

Countlines

Bei einem Countlines Test wird wie der Name bereits sagt, die Anzahl der Zeilen des effektiven Resultats gezählt und mit dem erwarteten verglichen. Besteht das effektive Resultat aus so vielen Zeilen wie im erwarteten Resultat angegeben, ist der Test erfolgreich. Das erwartete Resultat muss als Zahl angegeben werden.

Listing 15: Beispiel Countlines

```
- name: Example_One_Result:
  command: 'show␣vrf'
  device:
  - R1
  expected_result:
  - "4"
  type: countlines
```

Ping

Bei einem Ping Test wird geprüft, ob eine Verbindung zum angegbenen Device aufgebaut werden kann. Für das erwartete Ergebnis gibt es zwei Möglichkeiten:
- Es wird kein Ergebnis angegeben. Sollte dies der Fall sein, wird geprüft, ob mindestens ein Ping beim Ziel angekommen ist.
- Es wird ein Ergebnis angegeben, welches einer Zahl zwischen 0 und 100 entspricht. Sollte dies der Fall sein, wird geprüft, ob ein bestimmter prozentsatz der abgestzten Pings am Ziel angekommen ist.

Listing 16: Beispiel Ping ohne Resultat

```
- name: Example_One_Result:
  command: 'ping␣vrf␣Green␣10.1.12.2'
  device:
  - R1
  expected_result:
  type: ping
```

Listing 17: Beispiel Ping mit Resultat

```
- name: Example_One_Result:
  command: 'ping␣vrf␣Green␣10.1.12.2'
  device:
  - R1
  expected_result:
  - '60'
  type: ping
```

Traceroute

Bei einem Traceroute Test wird geprüft, ob bestimmte Hops, oder bestimmte Hops an bestimmten Positionen gemacht werden. Das erwartete Ergebnis wird als Liste definiert und folgende Punkte sind dabei wichtig:
- Soll ein bestimmter Hop vorkommen, muss die entsprechende IP-Adresse oder der Domainname angegeben werden.
- Kann eine beliebige Anzahl an beliebigen Hops vorkommen, kann man dies durch einen "*" definiert werden.
- Soll eine bestimmte Anzahl an Hops beliebig sein, wird dies durch einen "*" gefolgt von einer Zahl angegeben.

Der nachfolgende Test zeigt, wie angegeben wird, dass der erste Hop "10.1.12.2" und der zweite Hop "12.2.32.1" sein muss. Damit der Test erfolgreich ist, darf zuvor und auch danach kein weiterer Hop mehr vorkommen.

Listing 18: Beispiel Traceroute

```
- name: Example_One_Result:
  command: 'traceroute␣10.1.12.2'
  device:
  - R1
  expected_result:
  - 10.1.12.2
  - 12.2.32.1
  type: traceroute
```

Der nachfolgende Test zeigt, wie angegeben wird, dass eine beliebige Anzahl Hops vorhanden sein darf, jedoch der letzte Hop "10.1.12.2" sein muss.

Listing 19: Beispiel Traceroute mit *

```
- name: Example_One_Result:
  command: 'traceroute␣10.1.12.2'
  device:
  - R1
  expected_result:
  - '*'
  - 10.1.12.2
  type: traceroute
```

Das nachfolgende Beispiel zeigt, wie angegeben werden kann, dass der fünfte Hop ”10.1.12.2” sein muss.

Listing 20: Beispiel Traceroute mit *Zahl

```
- name: Example_One_Result:
  command: 'traceroute␣10.1.12.2'
  device:
  - R1
  expected_result:
  - '*4'
  - 10.1.12.2
  type: traceroute
```

7.3 Testtypen

Ein wichtiger Bestandteil der Studienarbeit ist, dass verschiedene Arten von Tests ausgeführt werden können. Bei der Abgabe der Arbeit sind bereits folgende Typen implementiert:

- **Screenscraping**
  Prüft, ob das erwartete Ergebnis im effektiven Ergebnis eins zu eins vorkommt.
- **Countlines**
  Prüft, ob das effektive Ergebnis die gewünschte Anzahl Zeilen hat.
- **Ping**
  Prüft, ob das gewünschte Device erreichbar ist
- **Traceroute**
  Prüft, ob das gewünschte Device erreichbar ist und ob die angegebenen Hops gemacht werden.

Implementation

Sollten diese Testtypen in der Zukunft nicht ausreichen, können neue Testtypen implementiert werden. Dazu muss im Projekt «NVS-Nornir» die Klasse «Evaluator» erweitert werden. Folgende Schritte müssen gemacht werden:

1. Die Funktion «evaluate test» muss erweitert werden. Hier wird das Mapping zu der Funktion implementiert, in welcher die Logik für die Validierung der Tests implementiert wurde. Hierbei kann man sich am bereits vorhandenen Schema, welches im Bild unten dargestellt wird, orientieren.

Listing 21: Schema

```
def evaluate_test(self, expected_results, result, type):
    if type == 'screenscraping':
        return self.check_screen_scraping(expected_results, result)
    elif type == 'countlines':
        return self.check_count_lines(expected_results, result)
    elif type == 'ping':
        return self.check_ping(expected_results, result)
    elif type == 'traceroute':
        return self.check_traceroute(expected_results, result)
    else:
        return False
```

2. Anschliessend muss die Funktion implementiert werden, welche für die Validierung der Tests zuständig ist. Es muss darauf geachtet werden, dass es sich beim Rückgabewert um einen Boolean handelt. «True» bedeutet, dass der Test erfolgreich war und «False» wird verwendet, wenn der Test scheiterte.

3. Der neu erstellte Testtyp kann nun beim erstellen neuer Tests verwendet werden.

7.4 Testdurchführung

Im Kapitel unten wird beschrieben, wie der aktuelle Stand der Testdurchführung aussieht. Auch für die Zukunft gibt es bereits Verbesserungsvorschläge, diese sind jedoch im Kapitel ”Ausblick” beschrieben.

Befehlsgruppierung

Die Idee war ursprünglich, dass die Befehle folgendermassen abgesetzt werden:

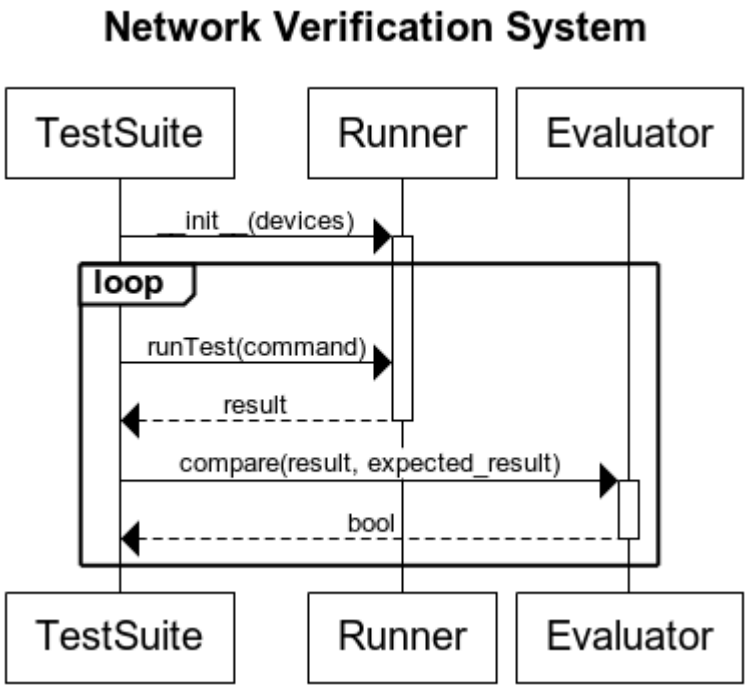


Abbildung 22: ursprünglich geplante Testdurchführung

Das Problem hierbei ist, dass das ganze viel zu ineffizient ist. Bei dieser Lösung wird beim initialisieren des Runners angegeben, auf welchen Devices die Befehle ausgeführt werden sollen und jeder Befehl, der abgesetzt wird, wird auf allen Devices abgesetzt.

Wird bei der Initalisierung des Runner also agegeben, dass insegesamt auf zwei Devices $(z.B.'R1'und'R2')$ Befehle ausgeführt werden und danach der Befehl ’show vrf’ auf dem Device ’R1’ ausgeführt werden soll, wird dieser stattdessen auf beiden Devices abgesetzt. Für die Validierung des Tests spielt dies keine grosse Rolle, da das unnötige Ergebnis einfach ignoriert werden kann. Da jedoch das Ausführen der Befehle der grösste Zeitfresser darstellt, ist dies alles andere als optimal und führt zu viel zu langen Wartezeiten.

Um diesem Problem entgegenzuwirken, wird der Ablauf, wie im Bild unten angezeigt, angepasst:

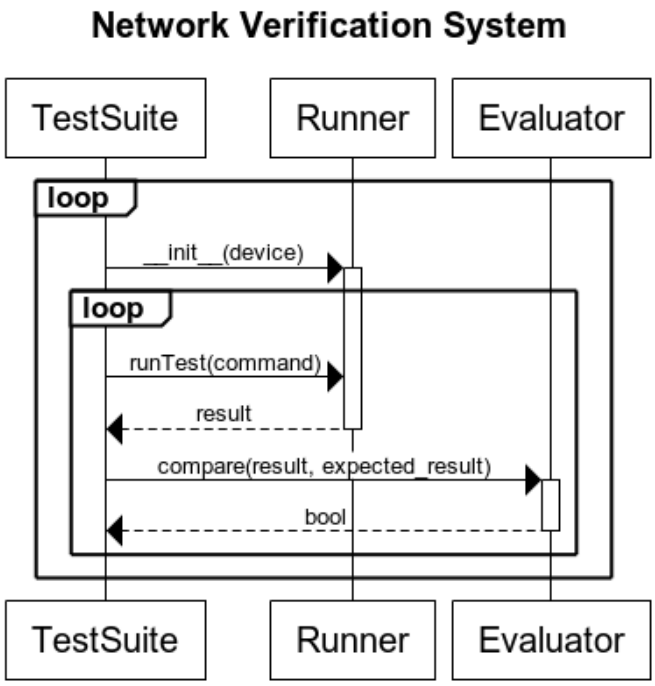


Abbildung 23: verbesserte Testdurchführung

Die Idee ist nun, die abzusetzenden Befehle zu bündeln und zu gruppieren. Alle Befehle, die auf dem Device 'R1' ausgeführt werden sollen, sind beieinander und alle Befehle, die auf Device 'R2' ausgeführt werden sollen, ebenfalls. Um nun die einzelnen Befehle **nur** auf dem dafür vorgesehen Device laufen zu lassen, wird pro Device eine eigene Runner-Instanz kreiert. Auf jeder Instanz werden die dafür vorgesehenen Befehle ausgeführt. Auch wenn dadurch mehrere Runner-Instanzen erstellt werden, werden die Devices weniger stark belastet.

7.5 Konfigurationsdatei Nornir

Bei der Datei ”config.yaml” handelt es sich um die Konfigurationsdatei, die Nornir sagt, auf welchen Devices die Befehle ausgeführt werden sollen. Die Datei sieht folgendermassen aus.

Listing 22: config.yaml

```yaml
inventory:
  options:
    defaults_file: inventory/defaults.yaml
    group_file: inventory/groups.yaml
    host_file:
  plugin: nornir.plugins.inventory.simple.SimpleInventory
```

Die ”config.yaml” Datei muss vorhanden sein damit der Runner richtig funktioniert, jedoch muss diese Datei nicht manuell angepasst werden. Der Runner ist so implementiert, dass er für jedes Device auf dem ein Test ausgeführt werden soll, eine temporäre ”config.yaml” Datei generiert und eine entsprechende Nornir-Instanz konfiguriert. Die erstellten Dateien werden nach Gebrauch wieder gelöscht.

7.6 Konfigurationsdateien

Um das Aufsetzen der Applikationen so einfach wie möglich zu machen, sind alle umgebungsspezifischen Konfigurationen in Konfigurationsdateien ausgelagert. Da es sich um zwei Applikationen handelt, sind ebenfalls zwei Konfigurationsdateien vorhanden, welche vor dem Gebrauch des Tools angepasst werden müssen.

7.6.1 Webserver

Der grossteil der Konfigurationsarbeit muss beim Webserver gemacht werden. Hier müssen verschiedene Dinge konfiguriert werden, wie zum Beispiel die Datenbank, oder der Mailserver, etc..

Verschiedenes

In der nachfolgenden Tabelle werden die allgemeinen Config Variablen beschrieben.

Variable	Beschreibung
DEBUG	Wenn Debug enabled ist, wird der Server im Debug Modus gestartet. Es wird ein Debugger für unbehandelte Exceptions gezeigt und der Server wird automatisch neu geladen, falls etwas am Code geändert wird. Default mässig ist diese Variable auf True gesetzt, wenn ENV ”development” ist, sonst ist diese Variable False [16].
TESTING	Ist Testing enabled, dann werden die Exceptions propagiert und nicht von den Error Handler gehandelt [16].
REGISTER_SECRET	Möchte sich ein neuer Benutzer registrieren, muss ein Secret Key angegeben werden. Dieser Secret Key muss mit REGISTER_SECRET übereinstimmen, damit er sich registrieren kann.
SECRET_KEY	Der Secret Key wird zur Signierung des Session Cookies gebraucht. Der Wert sollte ein langer zufälliger String sein [16].

Datenbank

In der nachfolgenden Tabelle werden die Config Variablen für die Datenbank beschrieben.

Variable	Beschreibung
SQLALCHEMY_DATABASE_URI	Der Datenbank URI welcher für die Datenbank Verbindung verwendet werden soll [15].
SQLALCHEMY_TRACK_MODIFICATIONS	Wenn diese Variable True ist, verfolgt SQLAlchemy Änderungen an Objekten und sendet Signale aus. Dies benötigt jedoch zusätzlich Memory und sollte deaktiviert werden [15].

Pfade

In der nachfolgenden Tabelle werden die Config Variablen für die Pfade zu den einzelnen Dateien beschrieben.

Variable	Beschreibung
BASE_FILE_LOCATION	Der Pfad zum Verzeichnis, in welchem die Verzeichnisse der anderen Dateien sind.
HOSTS_FILE_LOCATION	Der Pfad zum Verzeichnis mit den erstellten Host Files.
POD_LOCATION	Der Pfad zum Verzeichnis mit den erstellen Pods.
EXERCISE_LOCATION	Der Pfad zum Verzeichnis mit den erstellen Tests.
PYTHON_PATH	Damit der Runner in der richtigen Python Environment gestartet wird, muss angegeben werden, wo sich diese Environment befindet.
NORNIR_PATH	Hier muss der Pfad zur ”app.py” Datei im Projekt NVS-Nornir angegeben werden. Der Pfad könnte folgendermassen aussehen: ”Ordner/Ordner/NVS-Nornir/app.py”

Mail

In der nachfolgenden Tabelle werden die Config Variablen für die Mail Konfiguration beschrieben.

Variable	Beschreibung
MAIL_SERVER	Die Adresse des Mail Servers, welcher verwendet werden soll. Per Default ist ”localhost” gesetzt [1].
MAIL_USE_TLS	Wenn MAIL_USE_TLS auf True gesetzt wird, wird MAIL_PORT auf Port ”587” gesetzt [9].
MAIL_USERNAME	Der Benutzername, welcher für den Mail Account verwendet wird [1].
MAIL_PASSWORD	Das Passwort, welches für den Mail Account verwendet wird [1].
MAIL_DEFAULT_SENDER	Setzt den Default Sender der Mails [1].
WEBSERVER_ADDRESS	Setzt die Adresse des Webservers im Email, welches nach einer Testdurchführung and die Benutzer versendet wird.

7.6.2 Runner

Der Runner ist einfach zu konfigurieren. Es müssen nur zwei Angaben gemacht werden.

Variable	**Beschreibung**
CONFIG_FILE_PATH	Hier muss der Pfad zur im Kapitel ”Konfigurationsdatei Nornir” beschrieben Datei angegeben werden. Im Normalfall befindet sich die Datei im ”NVS-Nornir” Ordner.
URL_FRONTEND	Damit die Ergebnisse erfolgreich per Webhook an den Webserver übergeben werden können, muss angegeben werden, wie dieser erreichbar ist. Wird der Webserver lokal gehostet, wird die URL sehr wahrscheinlich ”http://127.0.0.1:5000” sein.

8 Ergebnisse

8.1 Network Verification System

Mit dem Network Verification System wurde ein Tool erstellt, mit welchem die Konfiguration von Netzwerk Devices auf ihre Korrektheit überprüft werden kann.

Teilnehmer

Die Teilnehmer eines Praktikas oder einen vom INS durchgeführten Kurs können ihre Netzwerke auf ihre Korrektheit prüfen, ohne das sie die Hilfe eines Betreuers beanspruchen.
Als erstes kann der Teilnehmer entweder einen Pod auswählen, oder die Informationen manuell eingeben. Der Vorteil von Pods ist, dass weniger Informationen eingegeben werden müssen, was angenehmer für den Teilnehmer ist.
Sind alle Informationen angegeben, können die Tests gestartet werden. Sobald die Tests abgeschlossen sind, können sie begutachtet werden. Zu jedem Testcase existiert eine Detailansicht, bei welcher genauere Informationen zu den durchgeführten Tests eingesehen werden können.
Hat der Teilnehmer eine Mail Adresse angegeben, erhält er einen Link zu den Resultaten des durchgeführten Tests. So kann der Teilnehmer auch auf ältere Resultate zurückgreifen.

Betreuer

Den Betreuern steht ein Dashboard zur Verfügung. Auf diesem Dashboard sind alle Exercise Files aufgelistet. Anpassungen an den einzelnen Files können direkt im Dashboard vorgenommen werden.
Diese Exercise Files werden als .yaml Datei im Ordner abgelegt, welcher im Config File spezifiziert wurde. Somit können diese Files auch mit einem anderen Editor angepasst werden.

Fazit

Die Betreuer haben nun mehr Zeit, um den Teilnehmern beim Troubleshooting zu helfen und müssen nicht bei jedem Teilnehmer schauen, ob dieser alle Befehle korrekt eingegeben hat.
Somit ist der Ablauf von Praktikas oder Kursen flüssiger, da die Betreuer mehr Zeit für jene Teilnehmer haben, welche kompliziertere Probleme haben.

8.2 Deviceunterstützung

Diese Applikation wurde nur mit Cisco Devices getestet. Da der NetworkRunner Netmiko verwendet, sollte es möglich sein, dass auch Devices anderer Hersteller verwendet werden können. Dieser Punkt wurde jedoch nicht getestet und kann daher nicht gewährleistet werden. Das grösste Problem wird beim Parsing des Outputs liegen. Zum jetztigen Zeitpunkt ist die Überprüfung der Konfiguration stark an die Ausgabe von Cisco Devices gekoppelt. Bei anderen Herstellern könnte es vorkommen, dass eigentlich korrekte Tests fehlschlagen.

9 Ausblick

Mit dem Abschluss dieser Studienarbeit steht ein Tool zum automatischen Testen von Netzwerk Devices zur Verfügung. Es gibt jedoch noch einige Punkte, welche weiterentwickelt werden könnten.

9.1 Teilnehmerunterstützung

Zum jetztigen Zeitpunkt erhält der Teilnehmer nur eine Übersicht über die Testcases, welche erfolgreich waren, oder gescheitert sind. Zusätzlich kann der Teilnehmer noch einsehen, welcher Befehl ausgeführt wurde und welcher Output generiert wurde. Für den Teilnehmer wäre es aber toll, wenn er auch noch sehen könnte, warum genau die einzelnen Testcases fehlgeschlagen sind und was er machen muss, damit diese Testcases beim nächsten Durchlauf korrekt sind.

9.2 Wiederholte Testdurchführung

Auf der Übersicht ist einzusehen, welche Testcases gescheitert sind. Der Teilnehmer möchte unter Umständen nur einen einzelnen Testcase anschauen, diesen beheben und im Anschluss nur diesen Testcase prüfen. Diese Funktionalität ist momentan nicht gegeben, es können nur alle Testcases zusammen ausgeführt werden. Dies hat den Nachteil, dass je nachdem länger gewartet werden muss, da nochmals alle Tests durchgeführt werden. In Zukunft könnte man auf der Detailansicht eines Testcases einen Button einfügen, mit welchem dieser einzelne Testcase getestet werden kann.

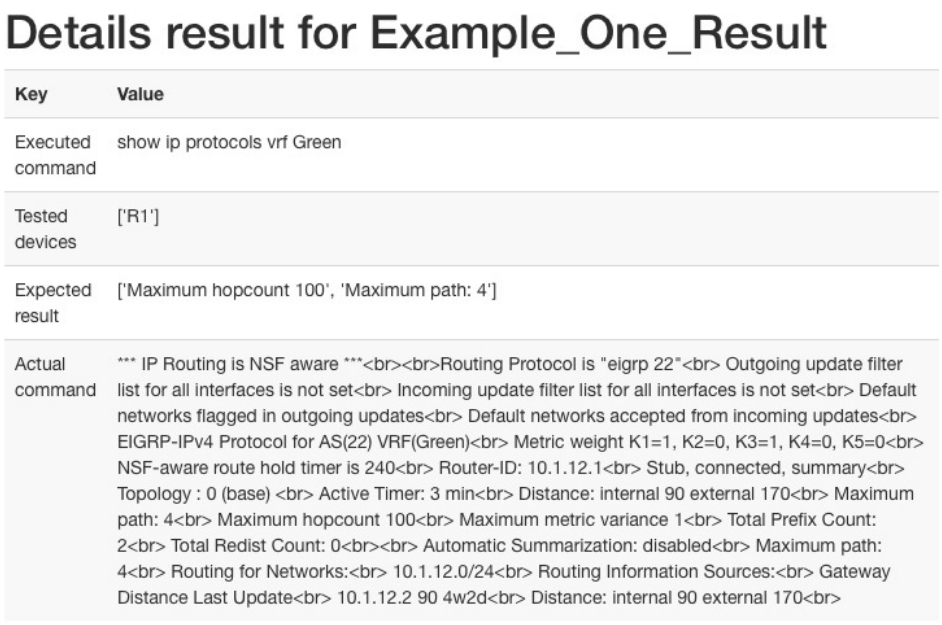
9.3 Verbesserte Testdurchführung

Die Ausgangslage ist folgende: Alle Tests, die ausgeführt werden sollen, werden gruppiert. Sie werden so gruppiert, dass jeder Test nur auf den Devices ausgeführt wird, auf dem er auch ausgeführt werden soll.

Der derzeitige Schwachpunkt ist, dass die Tests dennoch seriell durchgeführt werden. Das heisst, auch wenn Tests ausgeführt werden, die sich in keiner weise beeinflussen, werden sie nacheinander gestartet. Hier ist wichtig, dass keine Verwechslung stattfindet. Mehrere Benutzer können ihre Netzwerke paralell testen, ohne sich gegenseitig zu behindern oder zu verlangsamen, jedoch werden die Tests, der individuellen Durchführungen, seriell abgearbeitet.

9.4 Darstellung Detailsansicht

Wenn man einen Test durchführt und danach die Detailsansicht öffnet, kommt es je nach Test vor, dass man vor lauter unstrukturierten Daten überwältigt wird.
Wie man im Bild unten sieht, ist der Text im Feld ”Actual Result” kein bisschen strukturiert. Dies sollte noch angepasst werden, denn so wie es zur Zeit ist, ist es schwierig einen Vergleich mit dem ”Expected Result” zu machen und mögliche Fehler zu indentifizieren.



Details result for Example_One_Result

Key	Value
Executed command	show ip protocols vrf Green
Tested devices	['R1']
Expected result	['Maximum hopcount 100', 'Maximum path: 4']
Actual command	*** IP Routing is NSF aware *** Routing Protocol is "eigrp 22" Outgoing update filter list for all interfaces is not set Incoming update filter list for all interfaces is not set Default networks flagged in outgoing updates Default networks accepted from incoming updates EIGRP-IPv4 Protocol for AS(22) VRF(Green) Metric weight K1=1, K2=0, K3=1, K4=0, K5=0 NSF-aware route hold timer is 240 Router-ID: 10.1.12.1 Stub, connected, summary Topology : 0 (base) Active Timer: 3 min Distance: internal 90 external 170 Maximum path: 4 Maximum hopcount 100 Maximum metric variance 1 Total Prefix Count: 2 Total Redist Count: 0 Automatic Summarization: disabled Maximum path: 4 Routing for Networks: 10.1.12.0/24 Routing Information Sources: Gateway Distance Last Update 10.1.12.2 90 4w2d Distance: internal 90 external 170

Abbildung 24: Detailsansicht

9.5 Traceroute

Es ist zur Zeit nicht möglich, Tests vom Typ Traceroute in Kombination mit Tests anderer Typen auszuführen. Soll einer oder mehrere Traceroute-Tests durchgeführt werden, darf das Testfile keine anderen Testtypen beinhalten.

Problembeschreibung

Um einen Traceroute abzusetzen wird Napalm verwendet, für alle anderen Testtypen Netmiko. Aus diesem Grund muss die Verbindung zu den Devices nach jedem Test wieder abgebaut werden. Sollte dies nicht geschehen, kann es vorkommen, dass Netmiko mit einem Device verbunden ist und Napalm ebenfalls versucht eine Verbindung aufzubauen, was natürlich fehlschlägt.

Im Code unten(Option 1), ist der Runner so implementiert, dass die Verbindungen zu den Devices jeweils nach jedem Test wieder geschlossen werden.

Listing 23: Option 1

```
class Runner(object):
    def __init__(self, config_path):
        self.config_path = config_path

    def run_test(self, test_case):
        with InitNornir(self.config_path) as nr:
            result = nr.run(
                task=netmiko_send_command,
                enable=True,
                command_string=test_case.get_command(),
                on_failed=True
            )
        return result

    def run_traceroute(self, host_data, test):
        target_address = test.get_command()
        target_address = target_address.replace('traceroute', '')
        target_address = target_address.replace(' ', '')
        driver = get_network_driver('ios')
        with driver(host_data['hostname'], host_data['username'],
                host_data['password'], optional_args={'port':
                host_data['port'], 'transport': 'telnet'}) as device:
            result = device.traceroute(target_address, ttl=16, timeout=2)
        return result
```

Aus unbekannten Gründen funktioniert dies jedoch nicht zuverlässig Computerübergreifend. Wird zum Beispiel die Testausführung von verschiedenen Tests von Rechner A gestartet, funktioniert dies einwandfrei. Auch beim Wiederholen treten keine Probleme auf. Wird jedoch die Testdurchführung mit den exakt selben Tests von einem Rechner B gestartet, variieren die Ergebnisse. Die Ergebnisse der einzelnen Tests variieren auch von Durchführung zu Durchführung.

Um dieser Willkür zu entgehen, wurde zu Option 2 (Siehe Code unten) gewechselt.

Listing 24: Option 2

```
class Runner(object):
    def __init__(self, config_path):
        self.nr = InitNornir(config_path)

    def run_test(self, test_case):
        result = self.nr.run(
            task=netmiko_send_command,
            enable=True,
            command_string=test_case.get_command(),
            on_failed=True
        )
        return result

    def run_traceroute(self, host_data, test):
        target_address = test.get_command()
        target_address = target_address.replace('traceroute', '')
        target_address = target_address.replace('␣', '')
        driver = get_network_driver('ios')
        with driver(host_data['hostname'], host_data['username'],
                host_data['password'], optional_args={'port':
                host_data['port'], 'transport': 'telnet'}) as device:
            result = device.traceroute(target_address, ttl=16, timeout=2)
        return result
```

Der Unterschied zu Option 1 ist, dass die Verbindungen zu den Devices nicht nach jedem Test geschlossen werden. Dies hat den grossen Nachteil, dass Tests vom Typ Traceroute nicht zusammen mit anderen Testtypen durchgeführt werden können. Jedoch wird so der Willkür entgangen und beim Fehlschlagen eines Tests kann man sicher sein, dass entweder der Test falsch definiert, oder das Device falsch konfiguriert wurde.

Abbildungsverzeichnis

Akronyme

API Application Programming Interface. 19

CLI Command Line Interface. 20

DSL Domain-Specific Language. 19

HTML Hypertext Markup Language. 32

HTTP HyperText Transfer Protocol. 35

IDE Integrated Development Environment. 19

INS Institute for Networked Solutions. 18

NAPALM Network Automation and Programmability Abstraction Layer with Multivendor support. 19

SSH Secure Shell. 17

TCP/IP Transmission Control Protocol / Internet Protocol. 17

URI Uniform Resource Identifier. 67

WSGI Web Server Gateway Interface. 32

Literatur

[1] Flask-mail. https://pythonhosted.org/Flask-Mail/, —. [Online; accessed 30-May-2019].

[2] —. Flask and django comparison. https://www.educba.com/django-vs-flask/, 2019. [Online; accessed 29-May-2019].

[3] —. Werkzeug documentation. https://werkzeug.palletsprojects.com/en/0.14.x/, 2019. [Online; accessed 28-March-2019].

[4] —. What is wsgi? https://wsgi.readthedocs.io/en/latest/what.html, 2019. [Online; accessed 28-March-2019].

[5] Armin Ronacher. Jinja 2 homepage. http://jinja.pocoo.org/docs/2.10/, 2008. [Online; accessed 28-March-2019].

[6] Armin Ronacher. Flask web development, one drop at a time. http://flask.pocoo.org/, 2019. [Online; accessed 30-March-2019].

[7] K. Byers. Netmiko library. https://pynet.twb-tech.com/blog/automation/netmiko.html, 2019. [Online; accessed 30-March-2019].

[8] David Barroso. Supported devices. https://napalm.readthedocs.io/en/latest/support/, 2016. [Online; accessed 19-May-2019].

[9] davidism. Configure flask-mail to use gmail. https://stackoverflow.com/questions/37058567/configure-flask-mail-to-use-gmail, 2016. [Online; accessed 30-May-2019].

[10] W.-D. Fiege. Why to use django. https://www.hosteurope.de/blog/8-gruende-warum-sie-das-quelloffene-django-webframework-nutzen-sollten/, 2018. [Online; accessed 29-May-2019].

[11] P. Gujar. Django disadvantages. https://www.quora.com/What-are-the-advantages-and-disadvantages-of-using-Django, 2016. [Online; accessed 29-May-2019].

[12] Multiple Github Contributors. Flask companies. https://github.com/rochacbruno/flask-powered, 2018. [Online; accessed 29-May-2019].

[13] Patrick Ogenstad. Introducing nornir - the python automation framework. https://networklore.com/introducing-brigade/, 2018. [Online; accessed 28-March-2019].

[14] J. Protasiewicz. Django companies. https://www.netguru.com/blog/top-10-django-apps-and-why-companies-are-betting-on-this-framework, 2018. [Online; accessed 29-May-2019].

[15] P. Team. Configuration. https://flask-sqlalchemy.palletsprojects.com/en/2.x/config/, 2010. [Online; accessed 30-May-2019].

[16] P. Team. Configuration handling. http://flask.pocoo.org/docs/1.0/config/, 2010. [Online; accessed 30-May-2019].

[17] Wikipedia contributors. Iso/iec 9126 — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=ISO/IEC_9126&oldid=859222671, 2018. [Online; accessed 30-March-2019].

[18] Wikipedia Contributors. Flask (web framework). https://en.wikipedia.org/wiki/Flask_(web_framework), 2019. [Online; accessed 29-May-2019].