System For Pratical Evaluations of Network Administration Course

Diogo Nunes
Mestrado em Engenharia de Redes e Sistemas Informáticos
Departamento de Ciência de Computadores
2025

Orientador
Prof. Rui Prior, Faculdade de Ciências

Todas as correções determinadas

pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, ______/______/________

MSC

Acknowledgements

Acknowledge ALL the people!

Resumo

Este tese é sobre alguma coisa

Palavras-chave: física (keywords em português)

Abstract

This thesis is about something, I guess.

Keywords: Computer Science

Table of Contents

List of Figures

List of Abbreviations

API Application Programming Interface. 6, 8, 17

ASGI Asynchronous Server Gateway Interface. 11, 13– 16

CS Computer Science. 1

DCC Department of Computer Science. 4

GNS3 Graphical Network Simulator-3. 6, 14, 16– 18

HTTP Hypertext Transfer Protocol. 6, 10, 14, 16– 18

IOS Internetworking Operating System. 7

IOU IOS on Unix. 7

JSON JavaScript Object Notation. 6, 15, 17

KVM Kernel-based Virtual Machine. iv, 9

LXC Linux Containers. iv, 9

OAS OpenAPI Specification. 15

Proxmox VE Proxmox Virtual Environment. 8, 9, 14, 16– 18

QEMU Quick Emulator. 7, 9

REST Representational State Transfer. 6, 8

VM Virtual Machine. 8, 9

VPCS Virtual PC Simulator. 7

WSGI Web Server Gateway Interface. 10, 11, 14

1. Introduction

This chapter contextualizes challenges, outlining the limitations of current pedagogical tools and processes while framing the necessity for an automated, scalable solution. By dissecting the shortcomings of existing platforms to manual assessment burdens—we lay the groundwork for a system capable of aiding network administration education.

1.1. Problem Statement

The digital transformation sweeping across industries has created unprecedented demand for skilled computer science professionals. While most, if not all, Computer Science (CS) specializations face growing needs, network administration remains a foundational requirement. This persistent demand reflects the networks's critical role as infrastructure supporting all digital systems.

Modern organizations require professionals who can design, configure, and troubleshoot increasingly complex network environments. Effective education must therefore bridge theoretical knowledge with practical implementation, particularly through hands-on evaluations that simulate real-world scenarios. Yet current assessment methods fail to meet these needs at scale, creating a growing gap between academic preparation and professional requirements.

1.1.1 Need for Automated Evaluation

Practical evaluations are essential to prepare students for real-world challenges, enabling them to apply theoretical knowledge and develop hands-on skills. However, available evaluation methods face significant limitations.

Creating a physical network environment for practical evaluations is sure to be costly and challenging to scale for large student populations. While emulation and virtualization technologies offer cost-effective alternatives for creating flexible practice environments, they lack built-in automated assessment capabilities. Instructors must manually review each student's network topology configuration—a process that is:

- **Time-consuming**: Manual checks grow linearly with class size.
- **Error-prone**: Human reviewers may overlook misconfigurations.

- **Inconsistent**: Subjective grading criteria lead to unfair assessments.

Automating evaluations would reduce instructor workload, freeing time for student support aswell as ensure consistent, objective grading and simultaneously enable immediate feedback for learners.

1.1.2 Limitations of Current Solutions

Existing approaches to network education suffer from two critical gaps:

1. **Single-vendor focus**: Most tools (e.g., Cisco's packet tracer) are designed for specific vendor ecosystems, failing to prepare students for heterogeneous real-world networks where multi-vendor interoperability is essential.
2. **Missing evaluation component**: Some education institutions' curricula, such as our case in Department of Computer Science (DCC), forgo practical assessments entirely, which means:
   - Students complete exercises without validation
   - No measurable feedback on configuration skills
   - Graduates enter industry with possible gaps in their knowledge

1.2. Aims and Objectives

Building upon the foundational work of Santos [1] in automated network topology evaluation, this project has the following technical objectives:

1. **Develop a prototypical automated evaluation environment**:
   - Create a system for assessing network administration exercises without manual intervention
   - Support multi-vendor device configurations (Cisco, Juniper, Linux-based)
   - Validate both connectivity and configuration compliance
2. **Implement a cohesive back-end system**:
   - Transform loose components from Santos [1] into a unified system
   - Develop capabilities to coordinate between:

- Infrastructure provisioning
- Network emulation
- Evaluation automation

2. Background

The main focus of this chapter is to provide the reader with the necessary background to understand the context and technical foundations of this project. The goal of the system is to automatically evaluate network topologies by validating configurations and executing tests across various devices within a virtual network.

Achieving this requires the integration of multiple technologies, as the system must support a wide range of features to deliver an automated solution. Key concerns include not only functionality but also scalability, since multiple students may interact with the platform concurrently, each requiring an isolated working environment.

To support these requirements, this chapter introduces core concepts and tools such as virtualization, web frameworks, administration automation and task processing. These components form the foundation upon which the system is built.

2.1. Overview of Used Technologies

2.1.1 Virtualization

Virtualization is the process of creating a virtual version of physical resources, such as routers, switches, or even entire computers. In the context of this project, it is used to create virtual machines to provide students with a work environment consisting of a virtual network, itself comprised of various types of virtualized devices. This approach enhances scalability and reduces costs, as it allows multiple virtual machines to be run on a single physical machine.

Virtualization can be categorized into **emulation** and **simulation**.

- **Emulation** is the process of creating a virtual version of a physical device in software, replicating its behavior exactly—including any bugs and limitations. This is useful for various things like testing software on different platforms, running legacy software on modern hardware and even running potentially harmful software in a safe isolated environment. Emulation will be used wherever possible to provide students with a work environment that matches the real world as much as possible to best develop their network skills

- **Simulation** models the behaviour of a device, without replicating the underlying hardware or software. This results in a simpler less resource intensive model, though it may not fully capture the real device's behavior. Simulation will be used to simulate the behaviour of certain, simpler and generic, network devices and PCs.

2.1.2 GNS3

Graphical Network Simulator-3 (GNS3) is an open-source graphical network emulator software that allows the user to create complex network topologies and interact with the various devices in it. It is widely used for educational purposes and is often used in preparation for professional network certifications like the Cisco Certified Network Associate (CCNA).

GNS3 employs a simple drag and drop interface to allow users to add new devices, make links between them and even add textual annotations. The software allows users to interact with the devices by way of a console or even a GUI if the device supports it. The software also allows users to export their topologies to be shared with others, which can be useful for teachers to provide students with a pre-configured topology to work on.

Additionally, the software supports packet capturing which is essential for students to develop their debugging and troubleshoting skills. Finally it can also be interacted with via a Representational State Transfer (REST) Application Programming Interface (API) which is of particular interest for this project.

2.1.3 Architecture

The software can be employed in a variety of ways due to its architecture [2] that separates the user interfaces that it offers, namely the locally installed gns3-gui as well as the browser accessible gns3-web, from the gns3-server that runs the emulations and the controller who orchestrates everything.

2.1.3.1 Controller

The controller is integrated in the gns3-server project and is responsible for communicating with all the other components of the software. The controller is a singleton, meaning there should only be one instance of it running at any given time, and it does not support concurrent requests. It is able to control multiple compute instances if so desired, each capable of hosting one or more emulator instances, varying depending on their complexity. The controller also exposes the REST API allowing the ability to interact with the software programatically. All communication is done over Hypertext Transfer Protocol (HTTP) in JavaScript Object Notation (JSON) format and there is support for basic HTTP authentication as well as notifications via websockets.

2.1.3.2 Compute

The compute is also integrated in the gns3-server project and controls the various emulators required to run the nodes in the topology. The list of currently supported emulators is:

- **Dynamips** - Used to emulate Cisco routers and basic switching.
- **IOS on Unix (IOU)** - Used to emulate Cisco Internetworking Operating System (IOS) devices.
- **Quick Emulator (QEMU)** - Used to emulate a wide variety of devices.
- **Virtual PC Simulator (VPCS)** - A basic program meant to simulate a basic PC.
- **VMware/VirtualBox** - Used to run virtual machines with nested virtualization support.
- **Docker** - Used to run docker containers.

2.1.3.3 GUI

The GUI is composed of two separate but with mostly identical functionality, namely the gns3-gui and the gns3-web projects. The gns3-gui project is a desktop application that is used to to interact with a local or remote gns3-server instance. It is written in Python and uses the Qt framework for the graphical interface. The gns3-web is a web interface that is accessible via web browser and even though it is still in a beta stage, it has all the necessary features and stability to be used as a substitute for the gns3-gui.

2.1.4 Proxmox VE

Proxmox Virtual Environment (Proxmox VE) is an open-source platform designed for enterprise-level virtualization [3]. It is based on the Debian distribution of Linux and provides a web-based interface for managing virtual machines and containers. It is widely

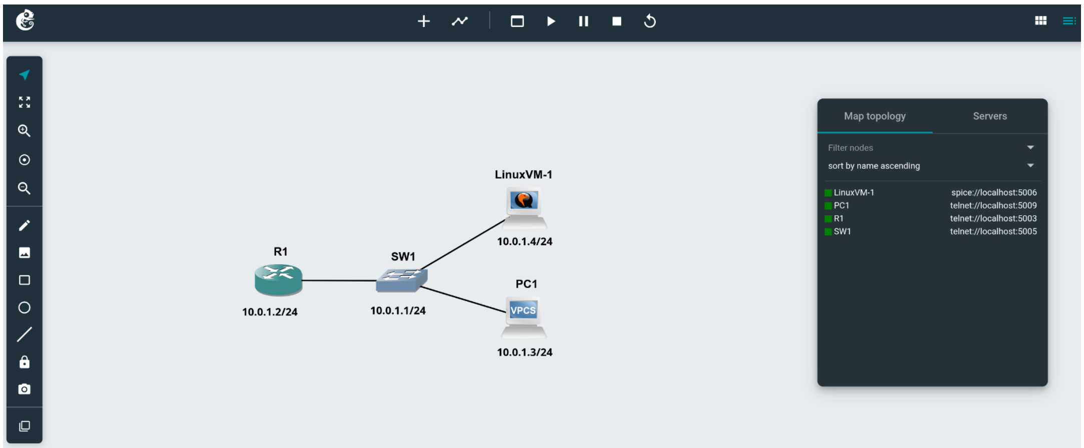

Figure 2.1: A simple network topology example in the GNS3 Web UI

used in data centers and cloud environments, as it provides a scalable and reliable solution for virtualization.

Proxmox VE bundles several core services that can be interacted with via shell commands, a web interface or by using the Proxmox VE REST API. These allow the user to interact with every service provided by Proxmox VE, in a plethora of ways, depending on the user's needs, skills and preferences. The web interface is the most user-friendly way to interact with the platform, as it provides a graphical interface for managing the cluster. The shell commands provide a more direct way to interact with the platform, allowing for more complex operations to be performed and opening the doors to scripting and automation. Finally, the Proxmox VE REST API allows for programmatic and remote interaction with the platform, enabling users to create custom applications that can interact with the platform.

2.1.4.1 Virtualization Technologies

Proxmox VE supports the the deployment and management of two distinct types of virtualization, namely, Kernel-based Virtual Machine (KVM) and Linux Containers (LXC).

Users can interact with these virtualized environments via NoVNC, a simple web-based VNC client or SPICE which is a more feature-rich protocol that provides better performance and more features than VNC. Both of these protocols support the use of a console-based interface, aswell as a full desktop graphical interface.

2.1.4.2 KVM

KVM is a virtualization solution provided by the Linux kernel. It leverages the hardware virtualization extensions of modern processors to provide a full virtualization experience at near-native speeds. Supports a wide range of guest operating systems making it a good choice for general purpose virtualization.

In Proxmox VE, KVM is used as the core component for running virtual machines and is used alongside QEMU.

2.1.4.3 LXC

Containerization is an operating system-level virtualization method that packages an application and its dependencies together into an isolated environment. Contrary to traditional Virtual Machine (VM) solutions, containers dont emulate hardware or require a guest operating system relying instead on the host's kernel. This approach leads to a faster and more lightweight virtualization solution, as they consume less memory and cpu resources.

LXC creates full system containers, capable of simulating a complete Linux distribution providing users with an environment that behaves like a traditional VM but with the speed and efficiency of a container. LXC start much faster than VMs making them ideal for scenarios requiring rapid deployment and/or scaling.

However, it's important to note that while containers offer a degree of isolation, they do not provide the same level of security as VMs. This means that while they may not always be a suitable replacement for VMs.

2.1.5 LDAP

Lightweight Directory Access Protocol (LDAP) is the foundation of user and device management in many institutions. Universities can rely on openLDAP and/or Microsoft's Active Directory (its enterprise implementation) to handle student, faculty accounts and lab computer access, amongst other things.

One of LDAP's most popular implementations, OpenLDAP, had its initial release in 1998. The protocol's longevity stems from its efficiency at handling large-scale authentication so much so that despite newer alternatives existing, LDAP remains entrenched in academic environments due to its reliability and universal adoption. For our project, LDAP integration enables students access to the system using their existing university credentials.

Two key factors make LDAP particularly valuable for this project: its standardized approach to user management and pre-existing deployment in our target educational environments. Given the extensive use, it's desirable for our system to have the capability to interact with LDAP in order to correctly authenticate users.

2.2. Virtualized Lab Environments

The combined use of Proxmox VE as a virtualization platform and GNS3 for network emulation presents a cost-effective solution for scalable networking education. This approach offers significant benefits over physical lab infrastructures:

- **Resource Efficiency**: Single physical host can support multiple concurrent student environments
- **Operational Characteristics**:
  - Accelerated environment provisioning through templates
  - State preservation via Proxmox VE's snapshot/restore functionality
  - Support for diverse network operating systems through virtualization technologies

2.3. Python Web Frameworks for API-Based Systems

2.3.1 Python

Python is a high-level, interpreted programming language renowned for its readability and versatility. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming, making it suitable for a wide array of applications. In the context of this project, Python serves as the primary programming language as its extensive standard library and supportive community contribute to efficient development and maintenance of the project's codebase.

2.3.2 WSGI

The Web Server Gateway Interface (WSGI) is a pivotal standard for Python web application deployment, defining a consistent interface between web servers and Python web applications/frameworks.

Prior to WSGI's introduction[4], Python web frameworks were typically written against various server-specific APIs such as CGI, FastCGI, or mod_python. This diversity led to compatibility issues, limiting developers' choices of web servers and frameworks, as not all frameworks supported all web servers and vice-versa. To address this fragmentation, WSGI was created as a standardized interface, promoting portability and flexibility in deploying Python web applications.

WSGI serves as a bridge, enabling web servers to communicate with Python applications. It specifies a simple and universal interface for web servers to forward requests to Python applications and for those applications to return responses. This standardization allows developers to choose from a variety of web servers and Python frameworks without compatibility concerns.

Introduced in 2003 as PEP 333, WSGI was later updated to PEP 3333 in 2010 to accommodate Python 3. These specifications outline how web servers and Python applications should interact, ensuring a consistent and reliable deployment environment across different platforms.

The WSGI standard consists of two main components:

- **Server/Gateway Side** - Responsible for receiving HTTP requests from clients and passing them to the Python application. Then receives the response from the application and forwards it to the client.
- **Application** - The Python application that processes requests and returns responses.

Additionally WSGI has support for middleware components. WSGI middleware is a Python callable that wraps another WSGI application to observe or modify its behavior. Middleware can perform various functions, including request preprocessing, response postprocessing, session management, and security checks. This modularity allows developers to add functionality to their applications in a reusable and maintainable manner.

The separation defined by WSGI allows for flexibility and scalability in deploying Python web applications.

Python WSGI applications often use built-in servers, during development, provided by frameworks like Flask. However, these servers typically aren't fully featured and aren't suitable for production environments. In production, WSGI servers act as intermediaries between web servers (e.g., NGINX or Apache) and Python applications, handling incoming requests and serving responses efficiently.

2.3.2.1 Flask

Flask is a web application micro framework written in Python, adhering to the WSGI standard, designed to facilitate the development of web applications by providing essential tools and features. Classified as a microframework, Flask does not require particular tools or libraries, instead choosing to focus on simplicity and extensibility[5].

An example of how easy it is to develop a basic web application with flask is provided in the following small piece of code.

Algorithm 1 Flask Hello World

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

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

2.3.3 ASGI

Asynchronous Server Gateway Interface (ASGI) is an interface specification for Python web servers and applications. It is considered a spiritual successor to WSGI, designed to provide a standard interface for asynchronous communication. ASGI was developed to address the limitations of WSGI, which was primarily designed for synchronous applications. Unlike WSGI, ASGI supports handling multiple requests concurrently, making it suitable for modern web applications that require real-time features such as WebSockets, long-lived connections, background tasks or the use of Python's async features.

As development progressed, asynchronous task handling became a more central requirement, initially addressed by integrating task queues. However, due to resource overhead and deployment complexity, they were phased out. This shift prompted an evaluation of frameworks that offered native support for asynchronous operations.

2.3.3.1 FastAPI

FastAPI is a modern, high-performance web framework adopting the ASGI standard. It leverages open standards, such as OpenAPI Specification (OAS), for defining path operations, parameters, and more, which in turn is based on the JSON schema. FastAPI relies entirely on Python type declarations, making it more intuitive and lowering the barrier to entry to new developers. This approach also simplifies the understanding and maintenance of the codebase.

Built on top of Starlette, a lightweight ASGI framework, and Pydantic, a data validation library. FastAPI combines the strengths of both to provide a powerful and flexible framework for building APIs with automatic data validation, serialization and documentation generation, all of which significantly enhance developer productivity.

Another key feature of FastAPI, being ASGI-compliant, is its built-in support for asynchronous programming, allowing developers to write non-blocking code using Python's *async*/*await* keywords. This is particularly useful for I/O-bound operations, such as database queries or network requests, as it allows the application to handle multiple requests concurrently without blocking the application which is essential in projects such as this one where multiple concurrent HTTP calls are made to interact with multiple devices and services concurrently, such as GNS3 and Proxmox VE.

Another powerful feature of FastAPI is its dependency injection system, that is very easy to use as it is automatically handled by the framework itselft. This allows for a clean and modular codebase, as dependencies can be easily injected into the various components of the application. This is especially useful in larger applications, where managing dependencies can become complex and cumbersome.

A change from Flask to FastAPI laid the groundwork for more efficient handling of I/O-bound operations—such as network interactions with Proxmox VE or GNS3, which will be of importance in future iterations of the project while also streamlining development thanks to FastAPI's built-in request parsing, background task support, and integrated dependency injection system.

2.4. Long running task processing approaches

2.4.1 The need for asynchronous processing in API-heavy applications.

Modern API-driven applications benefit tremendously from asynchronous programming paradigms to handle concurrent operations efficiently. Traditional synchronous execution models, where each request blocks thread execution until completion, prove inadequate for systems requiring high throughput and responsiveness. This limitation becomes particularly apparent in projects like ours, which relies heavily on HTTP calls to various devices and services.

The asynchronous model, implemented through Python's `async`/`await` syntax, offers several critical advantages:

- **Improved Resource Utilization**: A single thread can manage multiple concurrent I/O operations by yielding control during waiting periods
- **Enhanced Scalability**: Systems can handle higher concurrent user counts with the same hardware resources
- **Responsive Performance**: The application remains reactive even during long-running operations

Although Python has native asynchronous capabilities, libraries must be written with these in mind, meaning some may have limited or even no support for these capabilities.

2.4.2 Asyncio

Asyncio is Python's built-in library for writing concurrent asynchronous code. It serves as the foundation for asynchronous operations in many Python frameworks, enabling high-performance networking, web servers, database connections, distributed task queues, etc..

Asynchronous I/O can be useful in cases of time-consuming operations, as while awaiting the finish of said tasks, it relinquishes control so that other code can run in the meantime. This approach is particularly well-suited for I/O-bound operations—such as network communication, file access, or database queries—where tasks would otherwise spend a significant amount of time waiting for external operations that are outside of our control to complete. Rather than blocking the entire application during such waits, Asyncio allows other tasks to execute in the meantime, leading to more efficient resource utilization and improved throughput.

Overall, Asyncio provides the concurrency model that underpins efficient I/O performance. By embracing this model, the project benefits from improved responsiveness, lower latency, and better scalability—especially under workloads that involve heavy interaction with external services.

Frameworks that leverage these capabilities natively, provide the foundation for building responsive, scalable API services.

2.4.3 Celery

Celery is an open source distributed task queue focused on real-time processing but also offers support for task scheduling. It is implemented in Python, but the underlying protocol can be implemented in any language. Celery requires a message broker to function, such as Redis or RabbitMQ, which are responsible for queuing and distributing tasks from producers (clients) to consumers (workers).

When integrating Celery into their projects developers must mark functions they want to be processed as tasks with Celery provided decorators (e.g. `@app.task`) which allows workers to then execute them asynchronously. The system shines in projects requiring heavy computational or scheduled jobs, but brings with it non-negligible operational, developmental and resouce overhead, doubly so if the project didn't already include the use of message brokers.

2.5. System administration automation tools

Modern system administration increasingly relies on automation tools to manage complex infrastructure while maintaining reliability and reproducibility. In our context, configuration management systems serve as the foundational layer for ensuring well-configured device states across network environments.

2.5.1 Nornir

Nornir is an open-source automation framework written in Python, designed to provide a flexible and efficient approach to network automation tasks[6]. Unlike other automation tools that utilize customized pseudo-languages, Nornir leverages pure Python code, offering developers the full power and versatility of the Python ecosystem.

Nornir supports multi-threaded task execution, allowing operations to run parallel across multiple devices. This capability enhances efficiency and reduces the time required enabling easy scaling to a large number of devices.

The framework provides a robust inventory management system, enabling the organization of devices into groups and the assignment of specific tasks to these groups. This structure facilitates targeted automation and simplifies complex network operations.

Finally, thanks to Nornir's architecture, it is highly extensible through its plugin system, allowing users to create custom plugins for inventory management, task execution, and result processing. This modularity ensures that Nornir can adapt to a wide range of network automation scenarios.

Nornir makes it easy to write reusable tasks for configuration management and state validation which makes it highly desirable in the context of this project. Its ability to handle concurrent operations will also ensure it can scale alongside the rest of the project.

2.5.2 Ansible

Ansible is a widely adopted open-source automation platform that simplifies configuration management, application deployment, and task automation through a declarative YAML-based approach. Unlike imperative scripting solutions, Ansible employs playbooks to define system states, making automation accessible to both developers and operations teams while maintaining robust capabilities for complex workflows.

The platform operates on an agentless architecture, utilizing Secure Shell (SSH) for connectivity, which eliminates the need for persistent software on managed nodes. This design choice significantly reduces deployment overhead while maintaining secure communication through standard protocols. Ansible's push-based execution model allows for immediate task execution across entire device inventories without requiring pre-installed clients.

Ansible remains a valuable tool in task automation and orchestration but, as was already discussed in[1] there are several barriers to the adoption of Ansible in this project, mainly the difficulties encountered by utilizing the Telnet protocol for communications with network devices that dont support the SSH protocol .

3. Related Work

This chapter focuses on placing the current project within the context of existing solutions and related work. The primary goal of this project is to develop a system capable of automatically evaluating network topologies by validating device configurations and executing tests across a virtual network.

While automated assessment systems are well established in the field of programming education—receiving student-submitted code and running it against predefined test cases—equivalent systems for network exercises are far less common. Tools like Mooshak and similar platforms have proven effective for evaluating programming assignments and are widely adopted in academic settings.

At first glance, adapting these approaches to network topologies might seem straightforward. However, network evaluation introduces unique challenges such as the need for per-student virtual environments, real-time communication with multiple devices, and stateful, distributed configurations. This chapter explores existing tools like Mooshak and Packet Tracer, highlighting their capabilities, limitations, and how this project builds upon or diverges from them.

3.1. Programming Evaluation Systems

While not directly related, they are the main inspiration for this project. Programming evaluation systems are widely deployed in universities and other educational institutions. These systems receive, as input, code from students and subsequently run tests on it, outputting a score and even being configurable to provide students the first test case that they failed in, guiding students to the solution without handing it out.

The main differentiator between these systems and the one proposed in this project is the ability to solve a network exercise using multiple configurations across multiple devices, while programming evaluation systems will expect the same output every time, given the same input.

Another key difference is the fact that programming evaluation systems dont always provide a working environment for the students to test their code, owing to the fact that students might prefer to user their own development environment for initial development and testing. This project aims to provide a working environment for students, as setting up a networking lab can be a daunting task for students, especially when they are just starting out. By providing a pre-configured environment, students can focus on learning the concepts and skills they need to succeed in their studies, rather than spending time troubleshooting their setup.

3.1.1 Mooshak and lessons learned

In our context, in the DCC, Mooshak is commonly deployed to be used in the context of classes, exams and even programming contests.

Mooshak is a web-based system for managing programming contests and also to act as an automatic judge of programming contests [7]. It supports a variety of programming languages like Java, C, etc. Under each contest students will find one more problem definitions each containing varying sets of test cases in input-output pairs. After submiting their solution, the system will compile and run the code against the test cases giving a score based on the the amount of test cases passed.

Mooshak provides a structured approach to test coding and problem solving skills. It begins by offering a problem statement coupled with an optional image and an example test case, normally in the form of input and expected output. Users can submit their proposed solution by uploading a file with their code. The system then evaluates the provided solution against multiple pre-defined test cases, validating the output against the know-good output, and giving feedback in the form of a score based on the number of test cases passed. The system may also be configured to have time and/or memory constraints, to ensure that temporal and spatial complexity are also taken into account.

All of these, serve to provide a thorough evaluation of the student’s solution, which can help guide a student to better their coding and problem solving skills.

The system can also differentiate between differing types of errors, such as not giving the expected output, poorly formatted output, failure to compile or even exceeding the time limits. Mooshak also includes some features designed to drive competition between students, like a real time leaderboard and the ability to have more than 100% of the score for a given contest.

The system however is not without its limitations as it uses plain text files for its test cases and validates the output of student’s code character by character, which can lead to false negatives if the output is not formatted exactly as expected.

3.2. Cisco Packet Tracer

Cisco Packet Tracer is a network **simulation** tool developed by Cisco Systems, widely used in academic environments to teach networking concepts and prepare students for certifications such as the Cisco Certified Network Associate (CCNA). It offers a visual interface for building and simulating virtual network topologies using a variety of Cisco devices, including routers, switches, and end devices.

While Packet Tracer is highly accessible and effective for introducing networking fundamentals, it is a closed-source, proprietary tool limited to simulating Cisco hardware and IOS features. Its functionality is optimized for teaching purposes rather than for flexibility, extensibility, or integration into larger automated workflows.

In contrast, this project aim to allow for a more realistic and extensible lab environment. The use of real operating systems and support of a wide range of vendor platforms for routers and switches, aswell as Linux-based virtual machines is highly desirable. This allows for a more realistic experience, as students will be able to work with the same tools and operating systems that they will encounter in real-world scenarios.

Therefore, while Cisco Packet Tracer remains a valuable educational tool, the needs of this project called for a more flexible and open architecture.

4. System Architecture & Design

Delivering reliable, scalable automated assessment of virtualized networks requires a system built on solid principles of virtualization and automation. This chapter outlines the architecture of the proposed system, detailing the key components and how they interact to enable evaluation of student-submitted network exercises.

The system is designed to provide each student with a working environment where custom virtual network topologies can be deployed, configured, and tested. To achieve this, the platform integrates several technologies—such as GNS3 for network emulation, Proxmox VE for virtualization, and Nornir for configuration testing—alongside an asynchronous web-based API layer for user interaction and system communications.

This section provides a high-level overview of the system, the rationale behind its design choices, and the fundamental components that make up its architecture.

4.1. Functional Use Cases

4.2. System Architecture Overview

The architecture is divided into several key components, each responsible for a specific aspect of the system's functionality. The main components of the system architecture are as follows:

- **Web Application:** The web application serves as the main interface for users to interact with the system. It provides endpoints for, amongst others, evaluation, creation and viewing available exercises. The application is designed to be asynchronous where possible, allowing for efficient handling of multiple requests simultaneously.

- **Proxmox VE:** Proxmox VE is responsible for creating and managing VMs that host the network devices used in the exercises. This component interacts with the web application and all communication is done asynchronously through the Proxmox VE REST API, which allows for efficient communication, keeping the web application responsive, while also keeping the components decoupled.

- **GNS3:** GNS3 is used to emulate all the components of the virtual networks to be configured by students, using various types of virtualization detailed earlier. Communication with GNS3 is done through the GNS3 REST API by the web application during template VM creation and validation

- **Nornir:** This automation framework is used for validating device configurations. It connects to the virtualized devices, executes commands, and compares the output to expected results to determine correctness. Currently this component is integrated into the web application

4.3. Proxmox VE

Proxmox VE functions as the virtualization backbone of the system, enabling the creation and management of Linux-based VMs which in turn host services for use by students. Each VM runs a lightweight Linux-based operating system with a dedicated GNS3 instance, providing a self-contained environment for deploying and configuring virtual networks and their components.

All Proxmox VE-related operations—such as cloning, starting, templating, and deletion—are fully automated and triggered by the web application. Under normal operation ,after doing pre-required setup, no further manual intervention using the Proxmox VE web UI or shell utilities is required; such intervention is only necessary when the system's error handling mechanisms detect failures that cannot be automatically resolved. To securely execute these operations, the application authenticates to the Proxmox VE API using token-based authentication. The required credentials and configuration parameters are securely injected via environment variables , while the time limited token is stored in memory, ensuring that only authorized and properly configured processes can interact with the Proxmox VE infrastructure. this may change

4.3.1 Why Proxmox VE?

Proxmox VE was chosen for several compelling reasons that make it ideal for it to be choosen as our virtualization platform. First, it's completely free to use for all core functionality, with no hidden costs or licensing traps. Unlike proprietary solutions that charge per CPU core or socket, Proxmox VE lets us scale up our infrastructure without worrying about surprise licensing fees.

The platform's support for both containers and VMs within a single management interface gives us tremendous flexibility. We can run lightweight LXC for applications that dont require a full VM while using full VMs where required seamlessly. This hybrid approach would not be as straightforward with other solutions, like VMware ESXi.

We also value storage system's flexibility with LVM-thin provisioning allows efficient snapshotting of student environments while maintaining good performance.

Looking ahead, Proxmox VE's built-in support for emerging technologies like software-defined networking and its robust role-based access control system means our project still has room to grow into Proxmox VE. The active open-source development community ensures continuous improvements without vendor lock-in.

4.3.2 Proxmox VE Limitations

During development, we encountered several challenges when interfacing programmatically with Proxmox VE. One of th most significant issues stemmed from the platform's limited visibility into non-instantaneous operations - particularly for tasks like VM cloning, where the system did not provide task ids in the HTTP responses. This forced us to implement custom polling mechanisms to reliably determine operation completion states where possible.

A more critical limitation emerged in Proxmox VEacrestacapi's resilience characteristics. During stress testing, we discovered that even moderate request volumes done using a single machine running sequential code could overwhelm the single-node cluster's management daemon, triggering frequent HTTP 500 errors. These reliability constraints necessitated the development of protective measures including exponential backoff retry logic and strict client-side concurrent request limiting to maintain system stability.

4.3.3 Proxmox VE Firewall

Proxmox VE comes bundled with a iptables-based firewall implementation that can be enabled and configured at different levels.

The Proxmox host-level firewall provides essential features for securing student work environments, during examination periods, preventing student machines and the virtualized network equipments in them from communicating with outside networks.

This is done by adding firewall rules at the host level, meaning to each relevant student VM, that disable communications in both directions, with the exception of the machine that is responsible for configuration validation.

By default this behavior is not active and must be enabled on an as-needed basis, typically when a controlled assessment environment is required for more rigorous situations such as examinations.

In future iterations it may also be valuable to develop this further and making this feature less rigid as it may be interesting to have exercises that communicate with devices on the internet.

4.3.4 Exploration of containers as a full substitute for VMs

During development, we attempted to minimize VM usage where possible to accommodate as much scaling potencial as possible. The introduction of the GNS3 web interface allowed the machines hosting GNS3 instances to operate in a headless manner, removing the need for direct student interaction with the host machine. This eliminated desktop environment requirements and significantly reduced memory overhead, improving scalability.

We further explored replacing VMs entirely with containers, which promised additional resource savings. However, this approach proved unworkable due to fundamental technical constraints. Effective emulation requires KVM acceleration, which presents two problematic scenarios in containerized environments: either running unprivileged containers without KVM access (resulting in unacceptable performance degradation) or configuring privileged containers or containers with KVM passthrough (introducing serious security vulnerabilities).

Given that software emulation without KVM acceleration delivers poor performance for interactive use, we abandoned containerization as a complete VM replacement. The VM-based architecture remains necessary to maintain both performance through hardware acceleration and proper isolation between student environments.

4.3.4.1 VM Lifecycle

The lifecycle of a VM begins when a new exercise is created by a priviledged user through the web application. Upon exercise creation, the platform automatically clones a pre-configured base template VM stored in Proxmox VE. This new instance undergoes a configuration process where the provided GNS3 project file is imported and a series of user-defined commands are executed across the provided network topology. Once the setup is finalized, the configured VM is converted into a new template VM that is tailored to that exercise.

When students are enrolled in an exercise, the system generates individual work environments, by creating linked clones from these exercise-specific templates. Each student receives their own isolated VM instance that precisely mirrors the original template's configuration. This cloning approach ensures both consistency across student environments and rapid provisioning, as linked clones avoid the overhead of full disk copies while maintaining the template's baseline configuration. The use of linked clones significantly reduces both storage requirements and deployment time compared to traditional full cloning methods.

4.4. GNS3

GNS3 serves as the core network virtualization component in our system, providing the capability to emulate various network devices and topologies. The platform was selected for several key advantages: its remote web-based interaction, the intuitive drag-and-drop interface simplifies usage, and its broad device support accommodates both terminal-based and GUI network equipment and full computers. Additionally, GNS3's API allows for programmatic interaction, which proves essential for automation within our environment.

Currently, the system requires manual preparation of the base GNS3 template VM. During initial setup, an administrator must first create and configure a new VM, then proceed to install and set up the GNS3 environment. The final preparation step involves importing all necessary device images, including routers, switches, and other equipment that will be available for student exercises.

This base template then serves as the source for all subsequent student instances through Proxmox VE's cloning functionality. While this manual setup process adds initial configuration overhead, it ensures complete control over the base environment and allows for careful curation of the included device images.

The system achieves scalability through multiple VM instances running in Proxmox VE, each hosting an independent GNS3 environment. This architecture enables concurrent usage by multiple students on a single physical host. For future expansion, the design supports horizontal scaling by adding additional nodes to the Proxmox VE cluster, allowing the platform to accommodate growing numbers of users without requiring complete architectural changes.

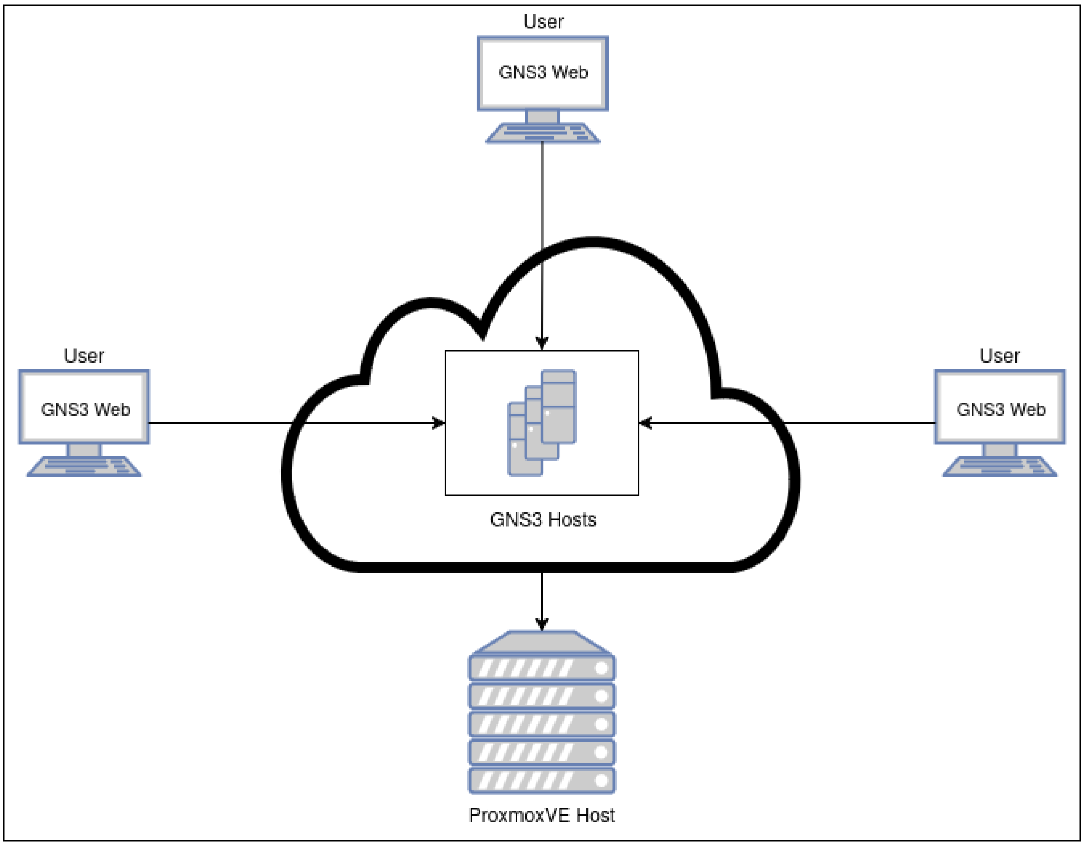

Figure 4.1: A diagram showcasing how users interact with the system's resources on a high level

4.5. High-level architecture

4.5.1 Available Hardware

The current deployment hosts all internal system components (those shown in the architecture diagram excluding the external LDAP instance) on a single physical server with the following specifications.

This machine's specifications, while capable, create inherent memory constraints. With 32GB of available RAM, practical VM allocation becomes the primary bottleneck. For

Table 4.1: System Hardware Specifications

Component	**Specification**
Processor	Intel Core i7-9700K
Memory	32GB DDR4 @ 2666MHz
Storage	1TB Samsung 970 EVO Plus NVMe SSD
Graphics	NVIDIA GTX 1650

instance, when deploying GNS3 instances each configured with 4GB of memory, the system can maintain only seven active VMs simultaneously. This limitation accounts for the Proxmox VE hypervisor's own memory overhead before encountering swap usage, which would degrade performance.

4.5.2 User Interface

The system employs a dual-interface web architecture accessible through standard browsers. For administrative functions and exercise management, users interact with Jinja2-rendered HTML pages delivered by the web application. These templates handle all developed features, like user authentication, VM interaction etc.

When working on networking exercises, users can transition to the GNS3-web interface by clicking a button. This dedicated environment provides direct access to the user's virtual network devices, as required by each exercise scenario. This integration ensures users experience a cohesive workflow from exercise selection to practical implementation without needing multiple authentication steps or application switches.

4.5.3 Web Application

The web application serves as the primary interface through which users interact with the system. It is built using the FastAPI framework, following a migration from an earlier prototype developed using Flask, and follows an asynchronous-first, modular architecture that provides scalable interactions with other system components.

The application exposes a REST API that supports endpoints for user authentication, exercise creation, virtual machine management, and configuration validation. It acts as the coordinator for the entire system, triggering operations in Proxmox VE, GNS3, and Nornir based on user actions.

Wherever possible, asynchronous I/O is employed to prevent blocking during operations such as API calls to Proxmox VE. Multiprocessing is also utilized to handle configuration validation. This keeps the system responsive and performant, especially when handling multiple simultaneous requests from different users.

Internally, the application is designed to be stateless and maintain minimal runtime state. Most essential information—such as user accounts, defined exercises, and student-to-VM mappings—is persisted in a relational database rather than stored in memory. Configuration values such as API tokens, base URLs, and database credentials are injected via environment variables to decouple deployment-specific settings from the application code. This design improves reliability, supports concurrent usage, and enables horizontal scalability if deployed across multiple instances.

To ensure maintainability and modularity, interactions with external services like Proxmox VE and GNS3 are isolated in dedicated modules. These serve as abstraction layers between the application logic and third-party APIs, exposing clean, reusable interfaces while hiding low-level implementation details. For example, Proxmox VE-related operations such as VM creation and deletion are handled in a separate module (e.g services/proxmox.py), as are all GNS3-related tasks. This separation of concerns improves the structure of the codebase and simplifies future maintainability by being more readable.

To help with development and testing, the application automatically generates OpenAPI-compliant documentation, allowing developers to explore and interact with available endpoints. This self-documenting behavior streamlines integration testing and encourages a more agile development process.

Finally, to safeguard user data and infrastructure control points, the application enforces secure authentication mechanisms using JSON Web Token (JWT) ensuring that only authorized users can trigger actions on shared resources.

4.5.4 Virtualization Components

The system employs a hybrid virtualization approach using Proxmox VE as the foundational platform. The usage of containers was explored but it was found unsuitable for our main use case of virtualization, GNS3 instances. However there remains one valid usage for containers for the project, which is hosting the web application. However this component may also be optionally hosted in a separate physical machine.

For network emulation, the system utilizes full KVM-based virtual machines, each hosting a GNS3 instance. These VMs provide the necessary hardware virtualization support for

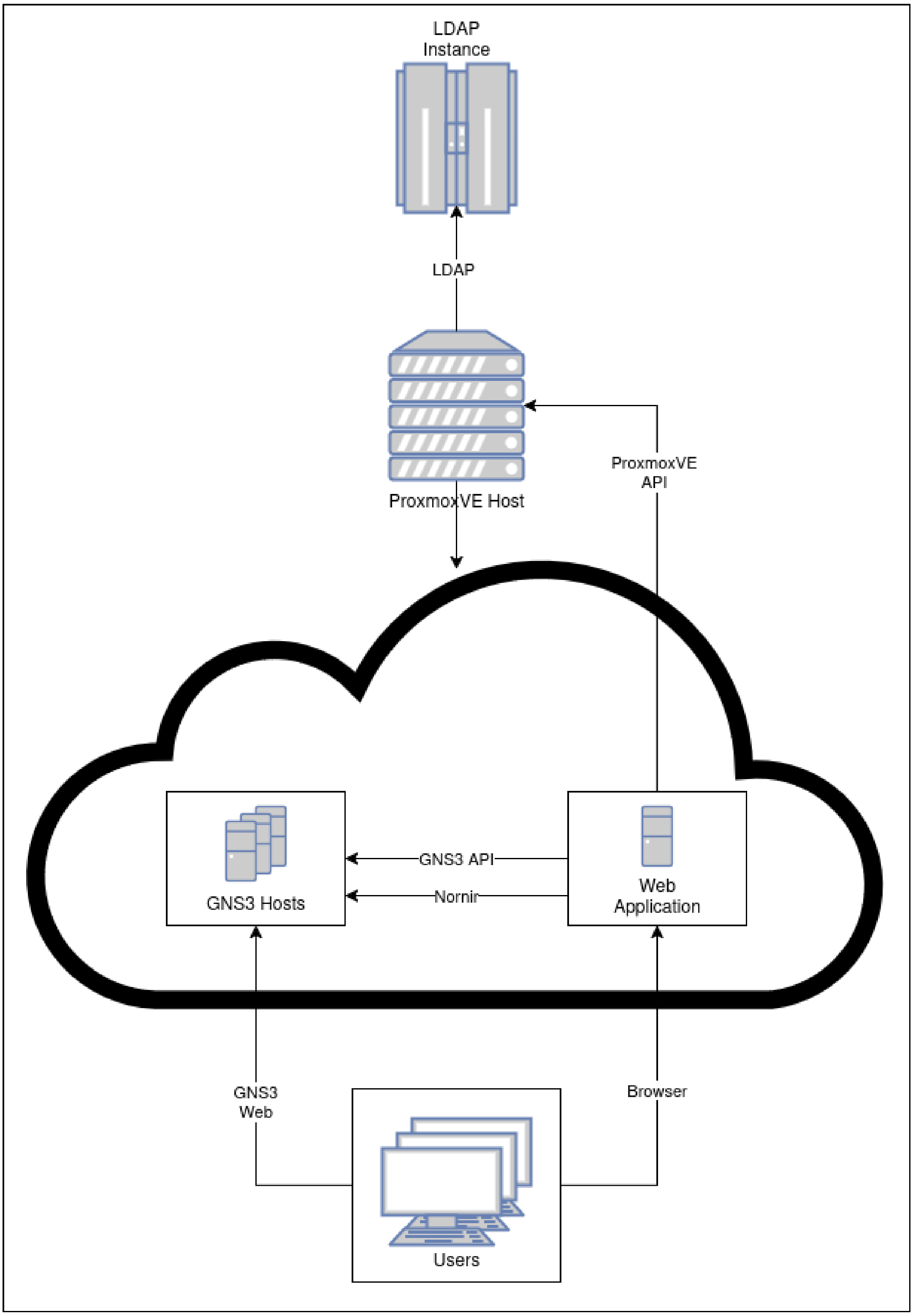

Figure 4.2: A diagram showcasing a high level overview of the system's main components

nested device emulation, particularly crucial for fast virtualization. Finally, by the use of linked clones and storage-efficient backing filesystems, in this case LVM-thin, allows the system to rapidly provision VMs while minimizing storage usage.

4.5.5 Evluation component

The system employs a modular evaluation framework built on Nornir to validate configurations across virtualized network devices. At its core, this component utilizes specialized Python classes called ”modules” that encapsulate platform-specific validation logic. Each module is responsible for three key functions: identifying the target device’s platform (such as Cisco IOS, Linux, or VPCS), executing the appropriate validation commands for that platform, and interpreting the command output using regular expressions to d etermine configuration correctness.

The architecture follows an object-oriented design pattern with a base `CommandModule` class that handles common functionality. This parent class manages the Nornir inventory initialization and provides essential methods like platform detection and command execution. The actual validation logic is implemented in child classes that inherit from `CommandModule`, with each subclass specializing in a particular type of network test. For example, the included `PingModule` implements platform-specific variants of the ping command and corresponding response interpretation methods. This design promotes code reuse while allowing easy extension for new test types, as developers can create additional modules by simply extending the base class and implementing the required platform-specific methods.

Configuration validation occurs through a multi-stage process. When a test is initiated, the system first identifies the target device’s platform through Nornir’s inventory system. It then dispatches the appropriate platform-specific command variant, such as the Cisco IOS-style ping command for routers versus the Linux `ping -c` syntax for Linux hosts. The module captures and sanitizes the raw command output, removing terminal control sequences and other artifacts before applying regular expressions to assess the results. For connectivity tests like ping, the interpretation logic calculates success rates against a configurable tolerance threshold defined in the system constants.

The evaluation framework supports several advanced features to enhance reliability and debugging. Command timeouts are managed to prevent hanging operations, with a default window that can be tuned as needed. Future extensions could incorporate snapshot functionality, allowing the system to capture and compare device states at different points during an exercise, though this capability is not currently implemented in the base version. The modular architecture ensures such enhancements can be added without disrupting existing validation workflows.

4.5.6 Storage component

The system has two main components regarding storage, one for the database needs of the web application and one for the disks of the VMs.

4.5.6.1 Virtual machine storage

For virtual machine storage, the system utilizes LVM-thin provisioning to optimize disk space utilization across student environments. This storage backend enables efficient cloning operations through copy-on-write semantics while supporting advanced features like snapshot preservation. LVM-thin's space-efficient cloning creates a robust foundation for the system's data needs.

4.5.6.2 Web application database

The SQLite database serves as the central repository for all application data, leveraging SQLModel as an ORM layer that combines Pydantic validation with SQLAlchemy's database capabilities. This hybrid approach provides both runtime type safety and efficient database operations, while Alembic handles schema migrations to accommodate evolving data requirements.

The database schema organizes information across several interrelated models. User management builds upon a base `CustomBase` class that automatically tracks creation timestamps, with the `User` model storing authentication credentials, administrative privileges, and relationships to both submissions and virtual machine instances. The `Exercise` model captures lab configuration details, including JSON-serialized validation rules and device configurations stored as text fields due to SQLite's native type limitations. VM provisioning is managed through the `TemplateVm` and `WorkVm` hierarchy, where template instances maintain the base GNS3 project configurations and spawned work environments link back to both users and exercises. The `Submission` model completes the core data structure by tracking student attempts, scores, and evaluation outputs while maintaining referential integrity through SQLAlchemy relationships.

5. Implementation

6. Testing & Evaluation

7. Conclusion & Future Work

References

[1] P. M. C. Santos, "Environment for practical evaluations in network administration," Master's thesis, Faculdade de Ciências da Universidade do Porto, Porto, Portugal, 2024, accessed: March 21, 2025. [Online]. Available: https://hdl.handle.net/10216/164899 [Cited on pages 2 and 15.]

[2] GNS3 Documentation, "Architecture," n.d., accessed: March 15, 2025. [Online]. Available: https://docs.gns3.com/docs/using-gns3/design/architecture/ [Cited on page 5.]

[3] Proxmox Server Solutions GmbH, *Proxmox VE Administration Guide*, 2025, accessed: March 21, 2025. [Online]. Available: https://pve.proxmox.com/pve-docs/pve-admin-guide.html [Cited on page 6.]

[4] P. J. Eby, *PEP 333 – Python Web Server Gateway Interface v1.0*, Online, 2003, accessed: March 21, 2025. [Online]. Available: https://peps.python.org/pep-0333/ [Cited on page 10.]

[5] P. Projects, "Flask documentation," Online, 2025, accessed: March 21, 2025. [Online]. Available: https://flask.palletsprojects.com/en/stable/ [Cited on page 11.]

[6] D. Barroso, "Nornir: The python automation framework," Online, 2025, accessed: March 21, 2025. [Online]. Available: https://nornir.readthedocs.io/en/latest/ [Cited on page 14.]

[7] J. Leal and F. Silva, "Mooshak: a web-based multi-site programming contest system," *SOFTWARE-PRACTICE & EXPERIENCE*, vol. 33, no. 6, pp. 567–581, MAY 2003. [Cited on page 17.]

Appendix Title Here

Write your Appendix content here.