18-T2 ProxMox for CCSE UITS

Final Report

Capstone 4850 Sec 03/02

4/28/2025

Prof. Sharron Perry

Justin Hammitt, Josh Miller, Ian Hopkins

GitHub: github.com/JustinHammitt/pve-portal_flask

Website: http://notarealwebsite.org:5753/

Stats and Statuses:
Code: 300+ Lines
Tooling: Proxmox VE, Python Flask, noVNC, Slapd
Man Hours: 132

Table of Contents

Introduction

This project aims to provide the College of Computing and Software Engineering (CCSE) at Kennesaw State University with a secure, scalable, and cost-effective alternative to its current VMware-based virtualization environment. With the VMware licensing agreement set to expire and the cost of future contracts predicted to exponentially increase, the department sought to transition to Proxmox VE, an open-source hypervisor known for its flexibility and rich feature set. Our team's goal was to build a proof-of-concept Proxmox environment that supports virtual desktop infrastructure (VDI) for 600+ students. This required not only the deployment of a Proxmox cluster, but also the development of a custom web frontend to handle user authentication, VM access, and RDP session integration. The project was developed over the course of the Spring 2025 semester as part of the CS 4850 Senior Capstone. It was broken down into three major phases: infrastructure setup, authentication and access control, and user-facing web interface development. Each milestone was designed to build toward a fully functional and secure system for delivering virtual machines to authenticated users.

Our implementation includes:

- A Proxmox VE backend with Linux and Windows VM templates
- Bridged networking to ensure external access to deployed VMs
- LDAP and PAM-based login via a Flask web portal
- User-specific VM filtering and console access using the Proxmox API
- HTTPS encryption on both the frontend and backend systems

The final deliverable represents a near-complete foundation for future expansion and production use, with core functionality in place and key integrations tested successfully.

Requirements

This project seeks to replace KSU's dependency on VMware with an open-source Proxmox-based infrastructure capable of managing 600+ student virtual machines. Core requirements were driven by both technical objectives and sponsor-defined milestones.

Functional Requirements

- Deploy and manage virtual machines using Proxmox VE.
- Allow VM provisioning from templates (Linux/Windows).
- Authenticate users using LDAP or PAM.
- Allow authenticated users to access only their assigned VMs.
- Enable VM interaction through a secure web interface.
- Enable console and future RDP access via the web portal.

Non-Functional Requirements

- Scalable to at least 600 student VMs.
- Secured via HTTPS (TLS certificates).
- Sessions protected using CSRF tokens and secure cookies.
- API-driven for automation and future integration (Terraform, Ansible).

Sponsor Milestones

Milestone 1: Set Up Proxmox VE and Basic VM Deployment

- Installed Proxmox VE on dedicated hardware
- Created VM templates (Ubuntu and Windows 11)
- Configured bridged networking for external communication
- Deployed test VMs and verified SSH/RDP connectivity

Milestone 2: User Authentication and VM Access Control

- Integrated LDAP authentication via Flask web login
- Enforced session-based access and ticket management
- Configured Proxmox API-based access control and CSRF protection
- Secured the interface via SSL/TLS

Milestone 3: Web Interface for VDI Access and RDP

- Built Flask web portal to display VMs per user
- Integrated “Launch Console” functionality (via noVNC)
- RDP setup for Linux/Windows VMs
- local user creation and login-based RDP redirection
- SSL/TLS configured on both Flask and Proxmox interfaces

Analysis / Design

The ProxMox Project, a CCSE UITS Project, aims to replace VMWare, the current contracted solution. This section of the document aims to outline the design of that replacement and should serve as a outline to the entire project

1. Design Considerations

1.1.Assumptions and Dependencies

The ProxMox Implementation relies on the following components:

- Related software or hardware
    - ProxMox VE - Core virtualization software
    - Server Hardware - (Provided by Ian)
    - Network Hardware - (Provided by Ian)
- Operating systems
    - ProxMox Host Operating System - Deb based
    - Guest OS: Linux, Windows, Others as req.
- End-user characteristics
    - System Administrators - Manages ProxMox Clusters
    - Developers and UITS - Manages VM's does dev and testing
    - End-users - can access vm's through remote clients
- Possible and/or probable changes in functionality
    - Expansion - We will be providing a proof of concept that can be scaled upwards
    - High Availability - Should be implemented before scaling
    - Security Enhancements - IAM Policies, VMAccess Perms (secondary until LDAP can be Tested)

1.2.General Constraints

The ProxMox implementation, as mentioned, is to be implemented as a potentially scalable proof of concept, the following constraints will impact the design of it.

- Hardware or software environment
    - Server Hardware - Predefined CCSE UITS VM Server Limited CPU, RAM, etc
    - Storage Limitations - Local storage or other methods may be used, but an enterprise solution may utilize Ceph
- End-user environment
    - User Access Control - Permissions to test LDAP will not be easily granted so this may not happen immediately
    - Guest VM's Must support Windows at least and possibly Linux
- Availability or volatility of resources

    - Limited initial resources - limited server hardware
- Standards compliance
    - Security Compliance - Must align with CCSE UITS security policies
    - Data Retention - Backup and Disaster recovery complies with data protection policies
    - Best Practices - Must conform to Terraform, ansible, or other software Best Practices
- Interoperability requirements
    - Networking Compatibility - Must be compatible with current net infrastructure firewalls, VLAN's etc
    - Guest OS compatibility - Must be compatible with current Guest OSes
- Interface/protocol requirements
    - ProxMoxAPI - Must work within ProxMox API for VM provisioning
    - Secure Communications - TLS or other encryption required
    - Remote management - SSH console, Web interfaces etc
- Data repository and distribution requirements
    - Data Redundancy - must ensure data integrity through proper storage replication and snapshots
- Security requirements (or other such regulations)
    - RBAC - Hinges on access to test LDAP
    - Firewall and network isolation - student Vm's should be isolated
    - logging and auditing - Auditing of VM actions, api calls , etc
    - Traffic analysis - allows for identification of unusual traffic
- Memory and other capacity limitations
    - Ram and CPU allocation - CCSE UITS allocated resources only
- Performance requirements
    - VM Performance Benchmarking - Required for Scaling, based on baseline metrics
    - Network Latency and Throughput - Low Latency Network Communication
- Network communications
    - Vlan Configurations
- Verification and validation requirements (testing)
    - User testing - Basic validation of VM Provisioning and automation
    - Scalability Testing - Validation of scaling capability
    - Security Penetration Testing - Security audits
    - Disaster recovery and backup - Validation of snapshots, backups, recovery processes, etc
- Other quality and compliance goals
    - Automation & Self-Healing - Automated monitoring and self-healing script

Proxmox VE Environment

Proxmox VE served as the core virtualization platform, hosting all VMs and exposing the necessary API endpoints for authentication, VM control, and console access. It was deployed with bridged networking and preconfigured Linux and Windows templates. All backend automation and frontend API calls were made through Proxmox's REST interface, supporting a centralized and scriptable management layer for the entire system.

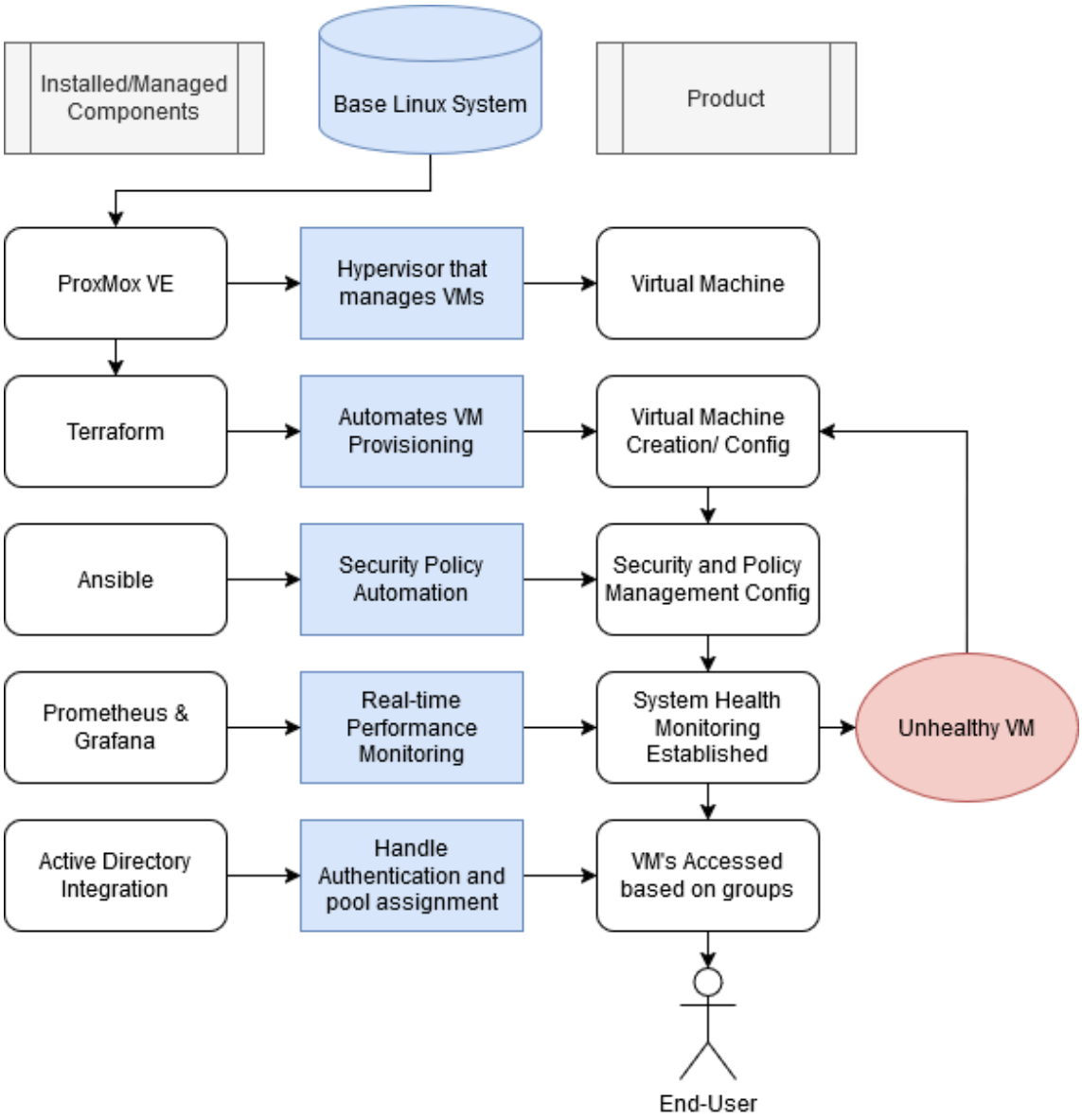

Fig1: Diagram of how VM's should be automatically managed. It's important to note this part of the configuration was decided as out of scope for the time and work constrains we were given. LDAP replaced ADUC and ansible, prometheus, and grafana were also removed and never implemented

LDAP Configuration and Role-Based Access

To simulate Active Directory, we deployed an slapd server that allowed us to create test users and assign them to access groups. LDAP was used as the primary login mechanism, and Proxmox was configured to enforce group-based permissions. For example, `student_a` was added to the `@CS_LDAP` group to test filtered VM access through the dashboard. This setup gave us fine-grained control over access without requiring full AD integration.

Frontend Responsibilities and API Security

The **RDPFrontend** was designed to act as a secure proxy between users and sensitive backend operations exposed by the **Proxmox API**. It handles:

- User login and session management
- API calls for listing, filtering, and displaying VMs
- Secure ticket/token handling for console access
- Enforced filtering via LDAP-based user-to-VM access policies

In most cases, such as login, shutdown, and reboot, the RDPFrontend works as intended, directly interfacing with the backend PVE server, but due to significant issues with cookies and noVNC. We found that there were caveats to relying this heavily on the built-in API's. The major problem is that to access the noVNC RDP Session, the PVE Server unfortunately needs to be exposed, defeating the purpose of the RDPFrontend. The recommended solution to this problem would be to configure noVNC session directly on the guests vm's, but due to the team size and time limitations, we were only able to leverage the API's during this project.

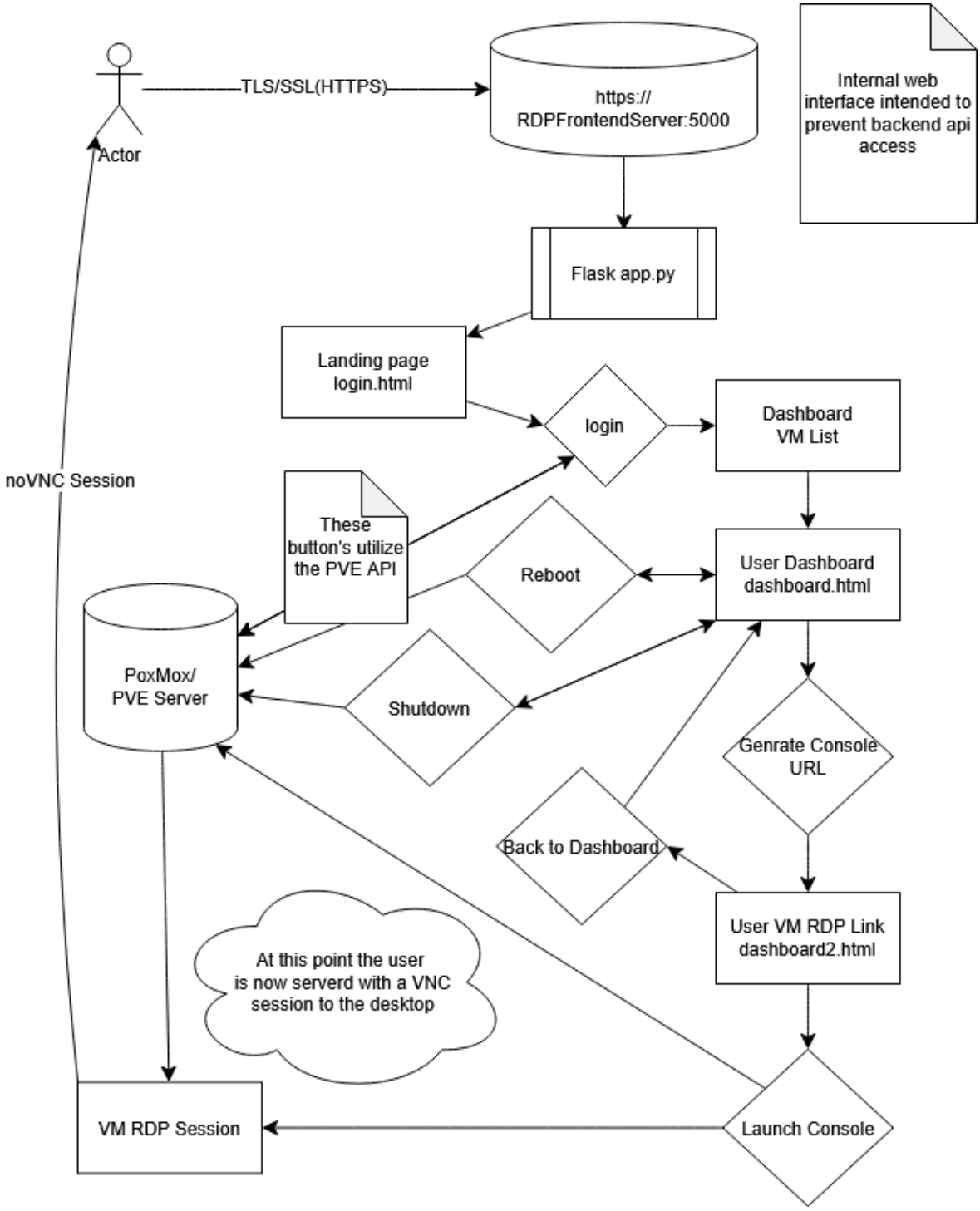

Fig 1: This is the general flow chart for how the whole ProxMox Implementation, and how our RDPFrontend works to server users in conjunction with the PVE backend.

Development

The development of our project followed an iterative and milestone-driven approach. Rather than building everything at once, we broke the work into three focused phases—each aligning with a specific milestone set by our sponsor. This allowed us to concentrate on setting up a stable virtualization backend, integrating secure authentication, and finally building a user-friendly frontend for VM access. Each milestone was approached as a standalone unit of progress, with specific tasks, deliverables, and internal testing cycles. As components were completed, we continuously tested and refined the system to ensure it functioned reliably as an integrated whole. This modular approach helped us manage complexity, catch issues early, and deliver functional features step-by-step.

Milestone 1: Proxmox VE Setup and VM Deployment

The initial Proxmox setup is relatively straightforward. Proxmox installs onto a computer/server just like you would install Windows, going through the setup, creating a main account, and installing any updates if necessary. Once the setup is complete, we create basic firewall rules to prevent unwanted access and upload .iso images for the OSs we want to create VMs with, including an iso for Windows, Ubuntu, etc. All that is needed to set up a VM is telling Proxmox how much hardware you want the VM to access, including RAM, storage, CPU, etc., and giving it an iso image to install the appropriate OS. For multiple VMs of the same OS, we can create a template, a base install of Windows, for example, to use to create multiple VMs for multiple users, allowing us to save space while allowing us to automate the process for VM creation, with predetermined hardware specifications with a base image that already has the most recent updates installed from the template.

Milestone 2: LDAP Authentication and Access Control

For the LDAP server, which is similar to Microsoft Active Directory in that it manages user accounts and permissions, we used Ubuntu's built-in LDAP server,slapd (the Stand-alone LDAP Daemon). Once we got it set up, we created an account with full access to a link to the Proxmox server and configured Proxmox to accept users from the LDAP server that is now being hosted. LDAP offers a similar set of roles and permissions to be created through the use of LDAP object classes, allowing for hierarchical authentication. Within the LDAP server various objectclass related to administration, professors, students, and course object classes were created. A set of attributes related to each object class were assigned to each, containing information specific to each object. For testing purposes, the server is being run inside a VM on Proxmox but can be used on any device as long as the Proxmox instance has access to it. Account creation and testing was done with a mix of Apache Directory and the command-line interface, with Proxmox able to update the users it has periodically and automatically. Automation was added using scripts written in either Bash or Python to take advantage of Ubuntu's slapd commands, with many flexible options for configuring users and accounts.

Groups in the LDAP server are directly linked to groups in Proxmox, allowing users of a similar group to share resources and similar VM templates for VM creation while also controlling what permissions they have access to on the Proxmox server itself.

Milestone 3: Web Interface and RDP Integration

During this milestone, we focused on designing and building a secure web-based frontend that allows users to log in, view their assigned virtual machines, and initiate console sessions. This was accomplished by developing a custom Flask application that communicates directly with the Proxmox VE API from a separate server RDPFrontend.

Development Process:

We began by building the **Flask login system**, which handles user authentication using credentials from either **LDAP** or **PAM**, depending on the username format. To authenticate, we made a secure API call to Proxmox's `/access/ticket` endpoint. On success, this returned a **PVEAuthCookie and CSRFPreventionToken**, which we stored in the user's session using Flask's `session` object with secure cookie settings enabled (`Secure`, `HttpOnly`, and `SameSite=Lax`). Once login was functional, we implemented the **dashboard page**, which dynamically queries the Proxmox API using the session's stored ticket and CSRF token. This endpoint (`/nodes/pve/qemu`) retrieves all VMs. The list is filtered so that each user only sees VMs corresponding to their LDAP group membership, which is enforced through group policies already configured in Proxmox's access control system.

The next major component was the **"Generate Console URL" feature**, which uses the Proxmox API's `vncproxy` endpoint to request a one-time session token for noVNC. Once received, we format the session as a full URL that the user can either copy or launch directly. This provides browser-based access to the VM's console via the secure Proxmox proxy. We also implemented **TLS/HTTPS across both the Flask app and the Proxmox backend**. Flask was configured to serve with a provided certificate and private key, and any HTTP traffic is automatically redirected to HTTPS in production. This ensures that all user credentials and Proxmox API calls are encrypted in transit.

Partially Implemented or Pending Features:

- The **Shutdown** and **Reboot** buttons were added to the dashboard and designed to issue POST requests to the corresponding Proxmox endpoints. However, these buttons are currently **non-functional** and require additional debugging and token verification to complete.

- **Full RDP session redirection and login-to-VM binding** (e.g., launching xRDP sessions and associating them with the current LDAP user) was planned, but not

implemented.

- **Automatic local user provisioning** on first login is a planned feature that would generate a corresponding Linux user on the assigned VM. This is still in the design phase.

Summary:

The key components of the front end—login, session handling, VM filtering, and console access—have been implemented successfully using Flask and the Proxmox API. The system meets the security and access control requirements outlined by the sponsor. The remaining features related to power control and RDP session redirection are in progress and will be the focus of continued work or future iteration.

Test (plan and report)

We used a combination of automated and manual testing, focusing on:

- Unit Testing (White-box):
    - API calls, Proxmox API interactions
    - Authentication mechanisms & servers.
- Integration Testing (Black-box):
    - Ensuring Proxmox, Active Directory, and the web interface work together.
    - Ensuring VMs on both Linux and Windows can be authenticated from LDAP.
    - Ensuring proper roles from LDAP can be assigned to Proxmox users.
    - Performance testing specific VM configurations for both Linux and Windows.
- System Testing (Black-box):
    - System level validation, including VM deployment and RDP access.
    - Testing system level validation on a server level rather than specific VM instances.

    - Testing system level validation on VM groupings targeting course groupings where high-demand is highly isolated and predictable.

- Regression Testing:
    - Validating updates and ensuring they do not break existing functionalities.
    - Implementing large authentication configuration changes to the system.
    - Executing large role modifications to gain insight about real life course registration.

Test Levels

Test Level	Description
Unit Testing	Individual Components
Integration Testing	LDAP Authentication, Terraform Automation, Performance testing
System Testing	Full Sys Functionality, Stress Testing, Performance testing
Regression Testing	Post-update Validation

Test Design

Test Selection

- Functional Testing: VM creation, deletion, performance metrics collection.
- Security Testing: AD integration, access control verification.
- Performance Testing: Stress testing under load conditions.
- Recovery Testing: Ensuring ProxMox Backup Server restores data correctly.

Pass/Fail Criteria

Requirement	Pass Criteria	Fail Criteria
Proxmox VE Setup	VMs can be deployed, network is accessible	VMs fail to boot, network issues occur
LDAP Authentication	Users log in and access assigned VMs	Login failures, access control issues
LDAP Role Assigning	Users are assigned proper roles such as admin, professor, student.	Incorrect role assigning has occurred.
LDAP Course Registration	Users who are specifically students can be en-rolled in one or more courses at a time.	Students are unable to be added into courses in LDAP. Multiple students cannot be registered to a course.
LDAP Professor assigning	Users who are professors need to be assigned as a professor to a course in LDAP.	Users who are professors fail to be assigned to courses as the professor.
RDP Access	Web interface launches RDP sessions correctly	RDP does not connect or fails authentication
VDI Workflow Confirmation	Users can access the RDP Web Front-end, authenticate, and obtain an RDP session with a working VM.	Workflow From WebUI to VM Desktop in some way fails, due to any issues with configuration.

Test Cases & Expected Results

Test Case	Description	Expected Result	Actual Result	Pass/Fail

TC01	Proxmox VE Installation	Proxmox installed and accessible	ProxMox is installed and PVE is Accessible	Pass
TC02	Bridged Networking	VMs communicate externally	VM' do in fact have outside web Access	Pass
TC03	LDAP Authentication	Users log in successfully	User's can Auth using LDAP	Pass
TC04	Web Login	Users access VMs through the web portal	User's can successfully see available VM's	Pass
TC05	RDP Access	Users connect to VMs via RDP	Users can through VE RDP Console access a VM	Pass
TC06	LDAP	Users assigned proper roles. Admin, professor, student	Users are assigned proper roles based on KSU ID	Pass
TC07	LDAP	Proper role assignment for courses.	Roles are implemented and can be assigned individually to students	Pass
TC08	LDAP	Proper role restriction for all roles.	Roles are manually assigned permissions, preventing unauthorized access	Pass

TC09	LDAP	Users can successfully be removed from the course role.	Scripts have been made to add, modify, and remove users automatically	Pass
TC10	LDAP	Large scale generation executes successfully.	Has not been tested yet	Fail
TC11	RDP - WebUI	Authenticated users see a list of VMs assigned to them in the web UI	User is served a VM list based on Group Permissions	Pass
TC12	RDP - Launch VM	User's can successfully launch an assigned VM	User's form the Dashboard can launch the link to RDP Console	Pass
TC13	RDP - Linux Connections	Linux VM's Accept RDP Connections	RDP Works as intended with linux	Pass
TC14	RDP - Windows Connections	Windows VM's Accept RDP Connections	RDP works as intended with Windows	Pass
TC15	RDP - Access Control	User can only access assigned vm's, and no others	Vm's are not assigned to specific users	Fail

TC16	RDP - Session Binding	RDP sessions should authenticate users based on LDAP	Proxmox API provides LDAP Auth for our app.py	Pass
TC17	Full VDI Workflow	Users can Access RDP WebUI Frontend, Login, Launch an RDP Session, and Access Launched VM	Vm's are not assigned to specific users	Fail
TC18	SSL Web Security	Web interface uses SSL/TLS (HTTPS) properly and blocks insecure traffic	Cert's are self signed, but the connection is indeed encrypted	Pass
TC19	RDP TLS Security	RDP connections use TLS encryption and do not allow unencrypted connections	RDP connections utilize TLS, and are authenticated	Pass

Test Environment

Component	Details
ProxMox Version	8.2.2
Testing Hardware	Ian's Home Server
OS	Linux

Bug Tracking

Severity	Description
Critical	System failure (e.g., VM crashes, LDAP authentication fails)
High	Major feature failure (e.g., RDP does not launch)
Medium	Performance issues
Low	UI/UX inconsistencies

Outstanding Issues / Known Limitations:

- The Shutdown and Reboot buttons are non-functional at this time and will need to be debugged. They appear in the UI but do not successfully trigger Proxmox API actions.
    - Severity: Low

- RDP session integration and user binding have been scoped out but not implemented.
    - Severity: High

- Large-scale LDAP role and course-registration simulation was not feasible within the current test environment.
    - Severity: Low
- Due to limitations in the PVE API RDP Session framework uses direct link's to the proxmox PVE server web console, Meaning that the PVE sever must be open to outside connections for everything to work
    - Severity: Critical (Security failure, PVE server should not be reachable)

Version control

Version control played a limited role in this project. While the majority of the work involved infrastructure configuration and testing the Flask-based web application component of the project was tracked using GitHub. The repository was initialized late in the development process and includes all relevant files for the login system, dashboard interface, and API integration logic. As most of the app was built within a short, concentrated development window, changes were pushed as a single major commit with minor adjustments needed to remove sensitive information before public git upload.

Summary

This project delivers a functional proof-of-concept for transitioning the CCSE Data Center from VMware to a Proxmox-based virtualization environment. The primary focus was to create a secure, scalable system that in the future should support virtual desktop access for over 600 students. Our team developed a custom Flask-based web interface designed as a user login portal that will allow our users easily access VM's. Proxmox VE served as the backend hypervisor, while LDAP integration allowed for secure user authentication and group-based access control. All communication between the frontend, Proxmox, and end users was secured with HTTPS and session-level token management.

The web portal enables users to log in with LDAP credentials, view their assigned VMs, and launch console sessions via Proxmox's noVNC service. Core functionality including session handling, VM filtering, and secure ticketed access has been fully implemented. However, features such as the shutdown and reboot buttons and full RDP session binding are still under development. Despite these remaining tasks, the system meets the core requirements for user-driven virtual machine access. It provides a strong foundation for future enhancements such as automatic user provisioning, course-level RDP access, and broader production deployment within the university's infrastructure.

Appendix