## **Introduction**

This project focused on creating a Virtual Machine and Docker Manager that enables users to efficiently manage virtual machines (VMs) and Docker containers. Key functionalities of the system include creating VMs, generating Dockerfiles, building Docker images, listing available images and running containers, stopping containers, and searching for images locally or on DockerHub.

## **Project Design**

## **System Architecture**

The system is a GUI-based application that provides a user-friendly, menu-driven interface for interacting with various features. It employs QEMU for VM operations and Docker for container management. The modular architecture ensures flexibility for future expansions and feature additions.

### **Technologies Utilized**

- Python: The primary language for scripting and system interaction.

- OS module: To execute command-line operations

- Tkinter: For designing the graphical user interface.

- QEMU: For managing virtual machines.

- Docker: To handle containerization tasks.

### **Challenges Encountered**

- Memory and Disk Input Validation: Ensuring user inputs for memory and disk space were valid numeric values.

- Dockerfile Path Validation: Handling incorrect or non-existent directory paths for saving Dockerfiles.

- Docker Image Building: Managing errors during the image-building process, such as missing Dockerfiles or invalid image tags.

- User Input Validation: Verifying and sanitizing user-provided inputs like container IDs and image names.

## Solutions Implemented

- Enhanced Memory and Disk Validation: Introduced strict input checks to confirm numeric values within acceptable ranges.

- Improved Dockerfile Path Validation: Added pre-checks to ensure directories exist before saving Dockerfiles.

- Streamlined Docker Image Building: Developed robust error-handling mechanisms to provide clear feedback for issues like missing Dockerfiles or incorrect image names.

- Strengthened User Input Validation: Implemented comprehensive input checks for container IDs and image names to prevent errors.

## Testing Methodologies

Unit Testing: Each feature, such as Dockerfile creation and VM management, was tested independently.

Integration Testing: Combined testing of features ensured smooth data flow and navigation within the system. This included tests for menu functionality and inter-feature communication.

## System Performance Evaluation

## Test Cases and Results

- Memory and Disk Validation: Test Case: Enter invalid (non-numerical) values for memory or disk space. Outcome: The system accurately identifies and rejects invalid inputs.

- Dockerfile Path Validation: Test Case: Provide a path to a non-existent directory for saving a Dockerfile. Outcome: The system displays a prompt rejecting the invalid path and prevents file saving.

- Docker Image Building: Test Case: Attempt to build a Docker image with a missing or invalid Dockerfile. Outcome: An informative error message is shown, guiding the user to correct the issue.

- User Input Validation: Test Case: Provide an incorrect container ID or image name. Outcome: The system detects invalid inputs and prevents errors during execution.

## Conclusion

The Virtual Machine and Docker Manager project successfully addressed the identified challenges, implementing solutions that improved reliability and usability. Rigorous testing ensured that individual features worked correctly and integrated seamlessly. This tool serves as a practical resource for managing VMs and Docker containers, fostering a deeper understanding of virtualization and containerization technologies.