VirtuManager

Virtualization and Containerization GUI Tool

Developed by:
Enjy Ramadan
Mariam Kandeel
Mazen Ashraf
Sohaila Ashraf
Omar Sherif

Project Demo

Project Overview:

Overview:

VirtuManager is a user-friendly desktop application that simplifies virtualization and containerization tasks using a clean graphical interface. Built with Python's Tkinter and customtkinter, the tool integrates with both VirtualBox and Docker, allowing users to manage virtual machines and build Docker images effortlessly, all from one place.

Phase One: Virtual Machine Management with VirtualBox

In the first phase, VirtuManager focused on simplifying virtual machine lifecycle management using VirtualBox. This included the ability to:

Core Functionalities:

- List Existing VMs: View a real-time list of available virtual machines along with their states (Running, Powered Off, etc.).
- Create VMs Easily: Input VM name, type, RAM size, disk size, and OS ISO image — all through the interface. The app will configure and create the VM accordingly.
- Start/Stop/Delete VMs: Control VM states with a single click, including launching VMs in headless or normal mode.
- Update VM Settings: Modify key VM configurations like RAM, CPU, and disk space.
- Navigation System: Each major function (Create, List, Update, Delete) is accessible through a streamlined and intuitive navigation bar.

The controller logic for all VirtualBox-related operations is abstracted cleanly in the `controllerVM.py` module, which interacts with system commands using the `subprocess` module. The UI files handle only layout and navigation, following the MVC pattern for better scalability and maintenance.

Phase Two: Docker Container Management Integration

Expanding the virtualization scope, Phase Two introduces Docker support — allowing users to build container images and manage them using the same GUI system.

Docker Features Added:

Build Docker Image GUI:

- Users can specify:
    - Image Name and Tag

    - Path to Dockerfile
    - Build Context Folder
- These inputs are handled through a modern UI form with proper labels, spacing, and optional browsing buttons.

Browse Files and Directories:

- Select Dockerfile via filedialog.askopenfilename
- Choose context folder via filedialog.askdirectory
- Automatically updates the entry field upon selection

Build Progress Bar:

- A dynamic progress bar gives visual feedback during the image build process.
- Simulated progress is shown while Docker builds in the background using a threaded function, ensuring the UI remains responsive.

Notifications and Error Handling:

- After the building finishes, success or failure is displayed via messagebox.showinfo() or messagebox.showerror().
- All exceptions and errors during the build are caught and presented clearly.

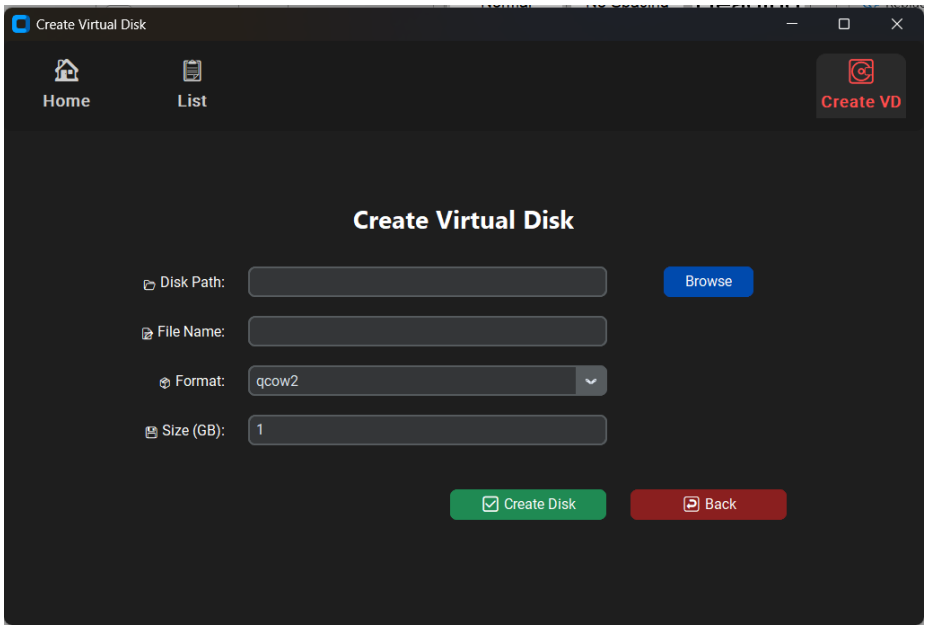
Clean Navigation Flow:

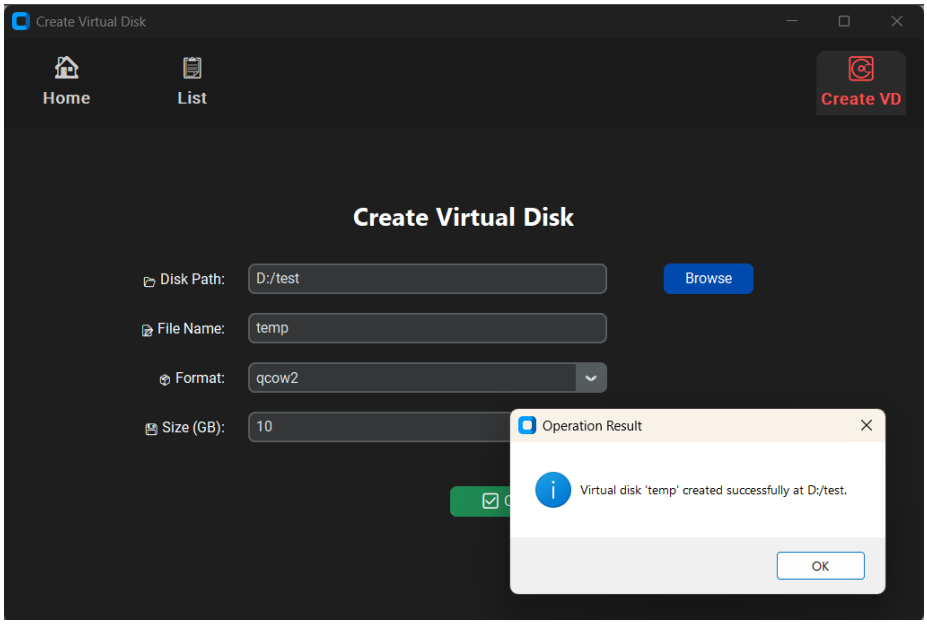
- Users can return to the Dockerfile creation page or main menu using a clear back button.
- The controller logic is delegated to controllerDocker.py, preserving the MVC structure.

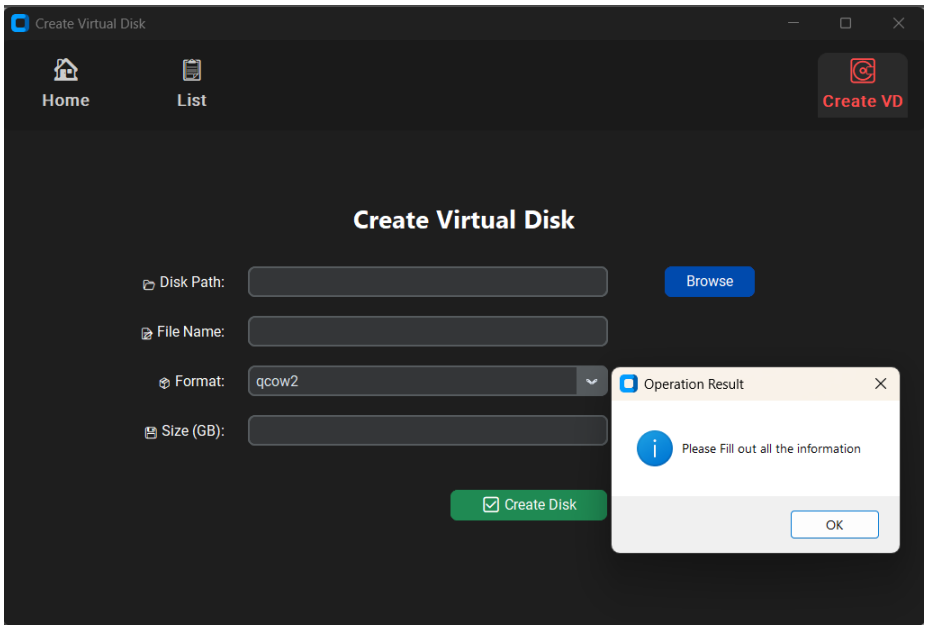
Implemented Features:

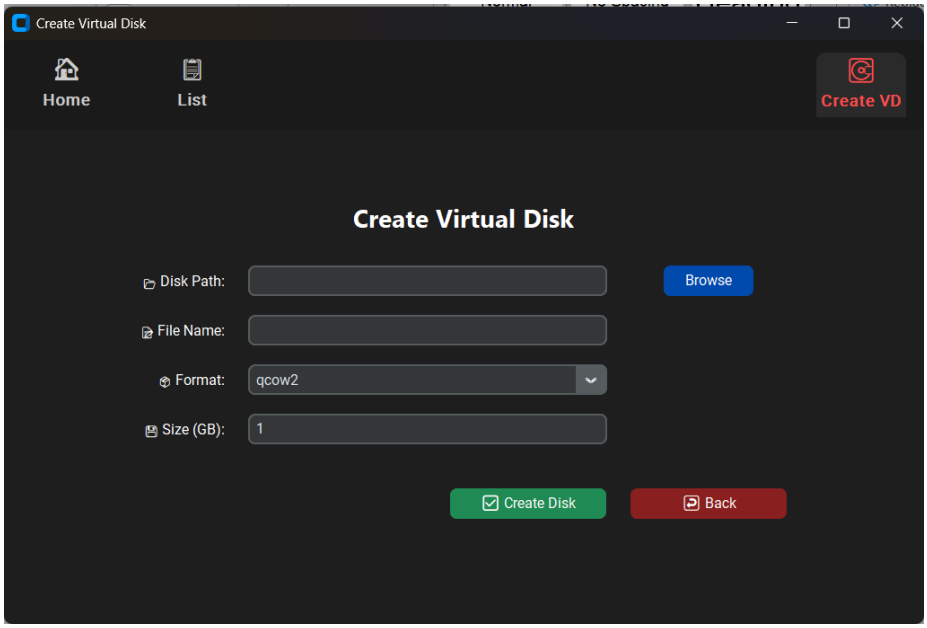
- Virtual Disk Creation: Allows users to create virtual disks with specified parameters such as name, path, format, and size.
- Virtual Machine Creation: Facilitates creating VMs by selecting the required CPU, RAM, disk, and ISO image.
- System Resource Validation: Ensures that the system has sufficient CPU, RAM, and disk space before VM creation to prevent resource shortages.
- Support for Multiple Disk Formats: Includes various virtual disk formats like qcow2, vmdk, vdi, raw, vhd, vhdx, and more, enabling flexibility in virtual machine setups.
- Docker Image Building: Enables users to build Docker images via a graphical interface by selecting image name, tag, Dockerfile, and context folder.
- Progress Monitoring for Docker Build: Includes a dynamic progress bar and real-time feedback for build status.
- Docker Build Error Handling: User-friendly messages inform users about successful builds or any issues during the build process.

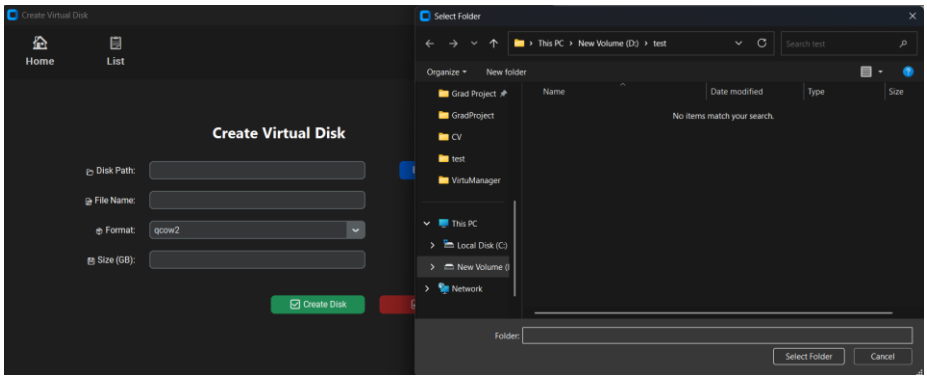
Test Casses:

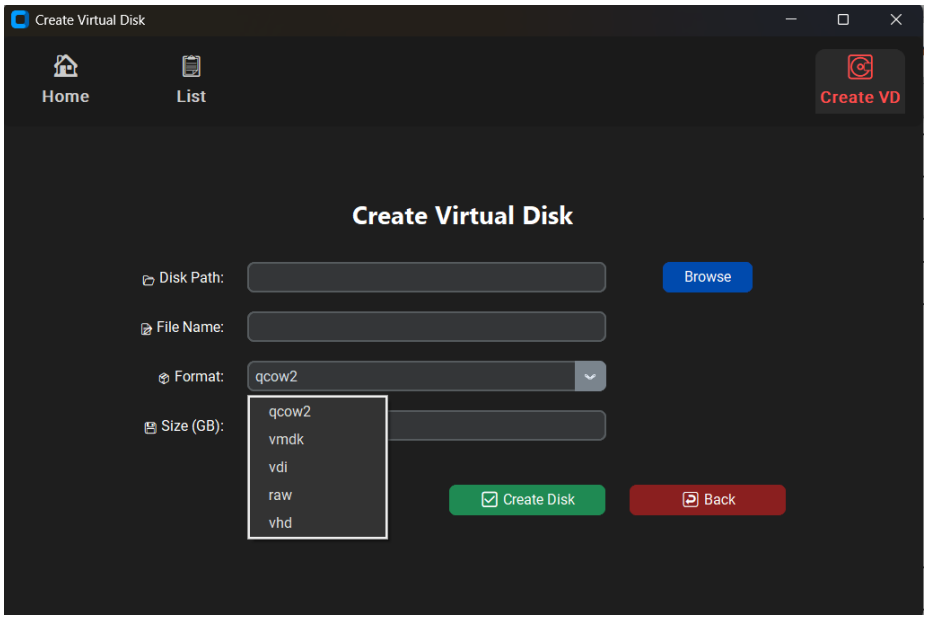
Test Case Name	TVD1: Verify Page Loads Successfully
Test Scenario	Ensure the Create Virtual Disk page opens correctly.
Prerequisites	-
Test Input	Open the Create Disk page
Execution Steps	1. Launch the app. 2. Click on “VD"
Expected Behavior	The Create Virtual Disk form loads without errors
Assumptions	UI and navigation work
Actual Results	UI and navigation work
Status	Pass
Real Life Testing	

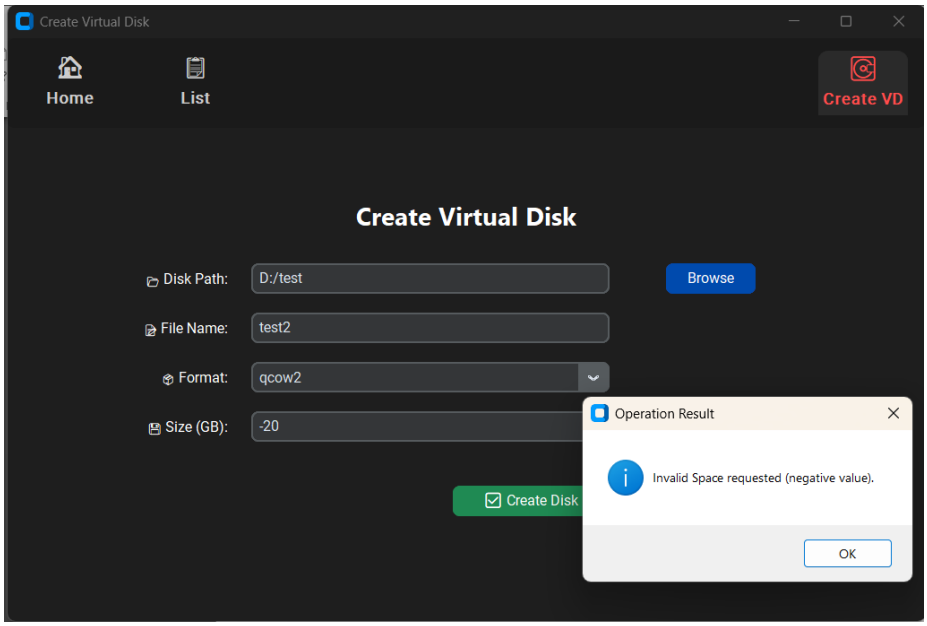
Test Case Name	TVD2: Submit with All Valid Data
Test Scenario	It should create a disk successfully.
Prerequisites	Virtualization engine functional
Test Input	1. Valid path 2. Valid format 3. Valid size
Execution Steps	1. Fill all fields correctly 2. Click “Create Disk”
Expected Behavior	Disk is created
Assumptions	Backend supports operation
Actual Results	Disk is created successfully and ready to function
Status	Pass
Real Life Testing	

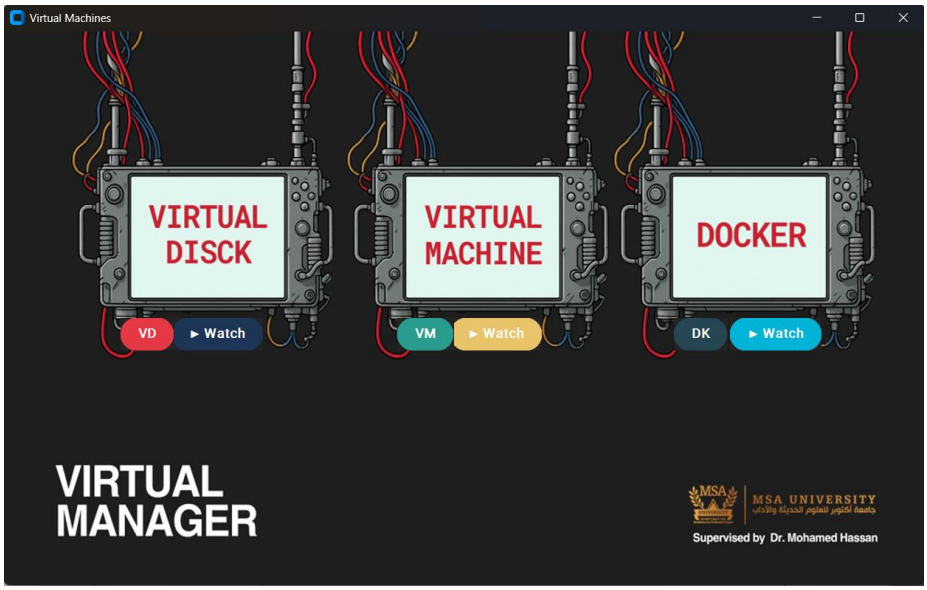
Test Case Name	TVD3: Submit with Empty Fields
Test Scenario	Submit button should validate empty input.
Prerequisites	The form is visible
Test Input	Leave all fields blank
Execution Steps	1. Click “Create Disk” without filling anything
Expected Behavior	Error shown or disk not created
Assumptions	Validation is present
Actual Results	Validation is present
Status	Pass
Real Life Testing	

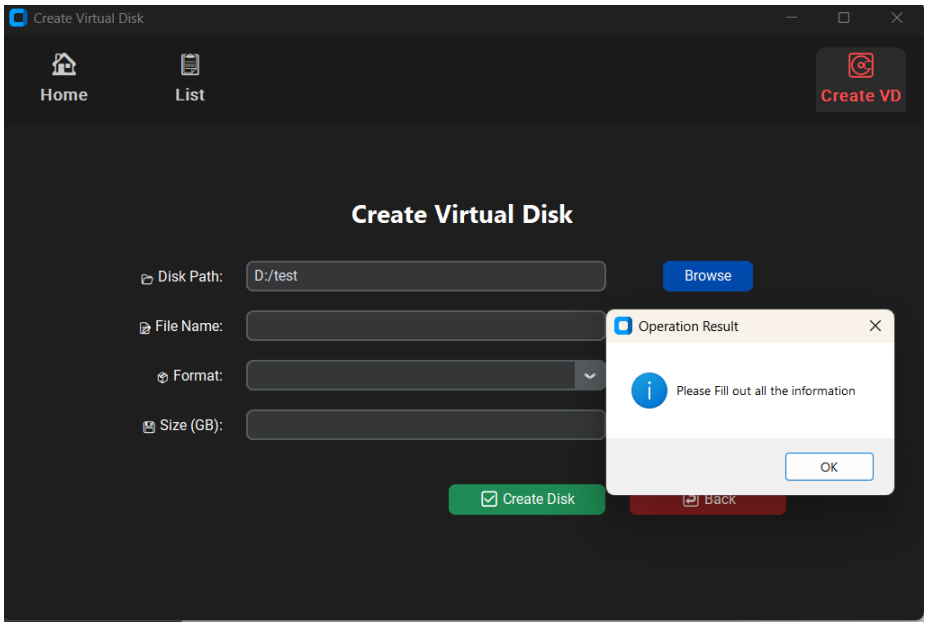
Test Case Name	TVD4: Verify Active States of Buttons
Test Scenario	Ensure that all buttons show appropriate visual feedback when clicked.
Prerequisites	The "Create Virtual Disk" page is fully loaded
Test Input	Interaction clicks with all buttons
Execution Steps	1. Hover and click on the “Browse” button. 2. Hover and click on the “Back” button. 3. Hover and click on the “Create Disk” button.
Expected Behavior	Each button should visually be active
Assumptions	UI framework handles button states
Actual Results	Functioning buttons
Status	Pass
Real Life Testing	

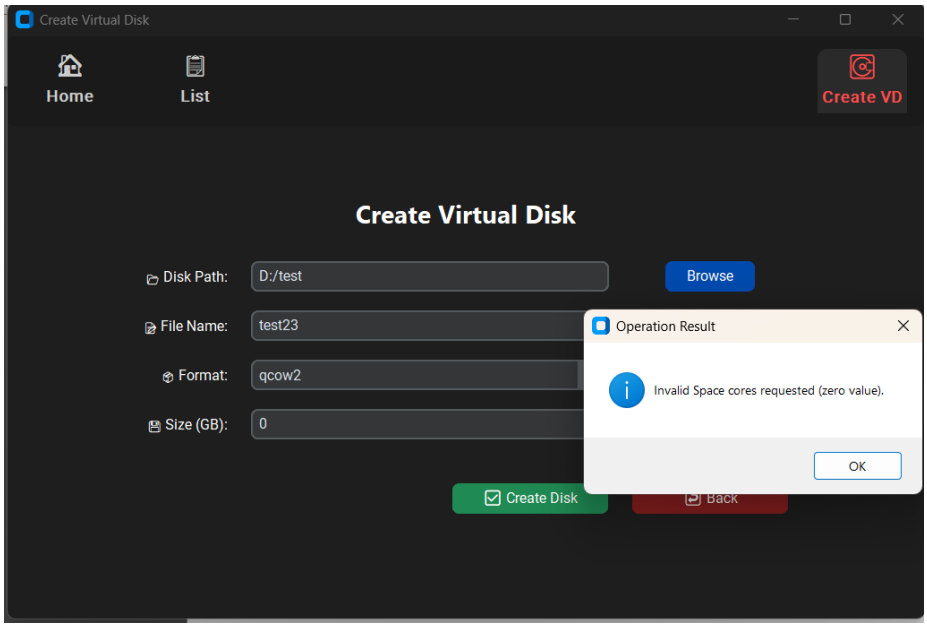
Test Case Name	TVD5: Verify “Disk Path” Browse Button Functionality
Test Scenario	Clicking “Browse” opens laptop search
Prerequisites	The form is visible
Test Input	Click “Browse”
Execution Steps	1. Click on “Browse”
Expected Behavior	File dialog opens for path selection
Assumptions	OS-level file dialog is supported
Actual Results	OS-level file dialog is supported
Status	Pass
Real Life Testing	

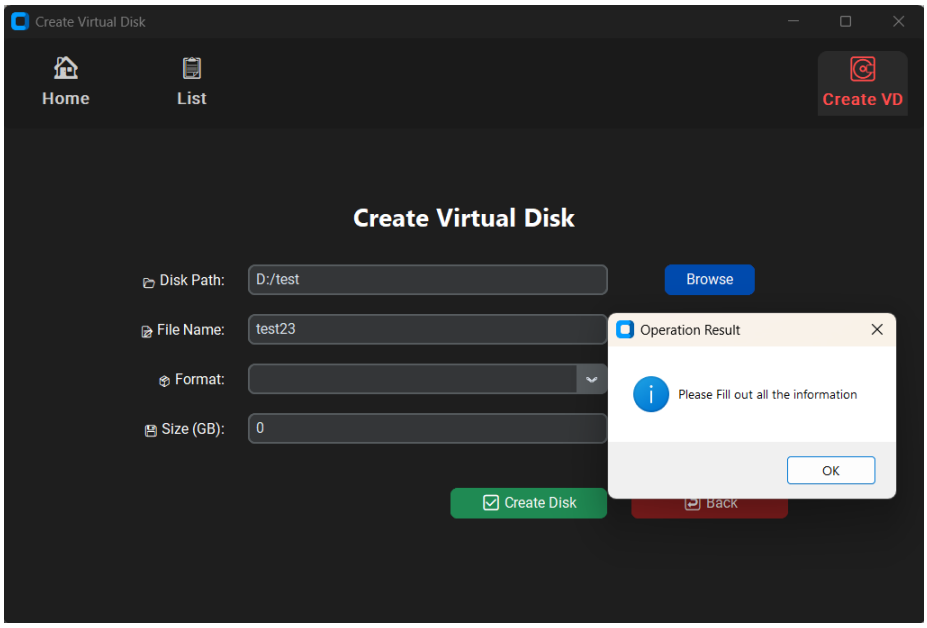
Test Case Name	TVD6: Verify Disk Format Dropdown Options
Test Scenario	Dropdown should list available formats.
Prerequisites	The form is visible
Test Input	Click on the dropdown
Execution Steps	1. Click the “Disk Format” dropdown
Expected Behavior	Shows valid formats like qcow2, vdi, vmdk, raw
Assumptions	Options are hardcoded or loaded correctly
Actual Results	Options are hardcoded or loaded correctly
Status	Pass
Real Life Testing	

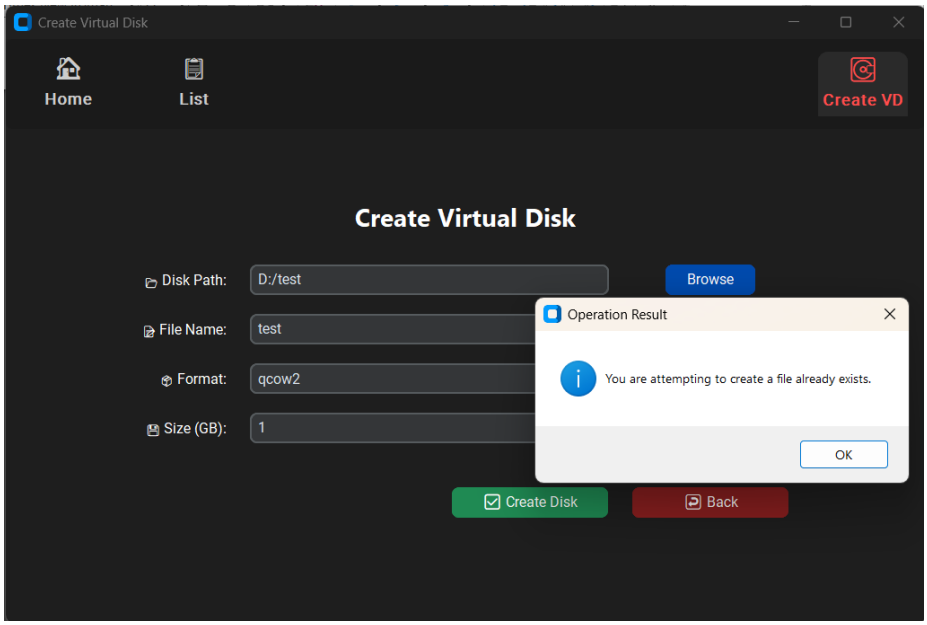
Test Case Name	TVD7: Verify Disk Size Spinner Limits
Test Scenario	Only valid sizes can be selected.
Prerequisites	The form is visible
Test Input	Use arrows to change size
Execution Steps	1. Adjust disk size
Expected Behavior	Only positive integers allowed
Assumptions	Spinner has limits set
Actual Results	Spinner has limits set
Status	Pass
Real Life Testing	

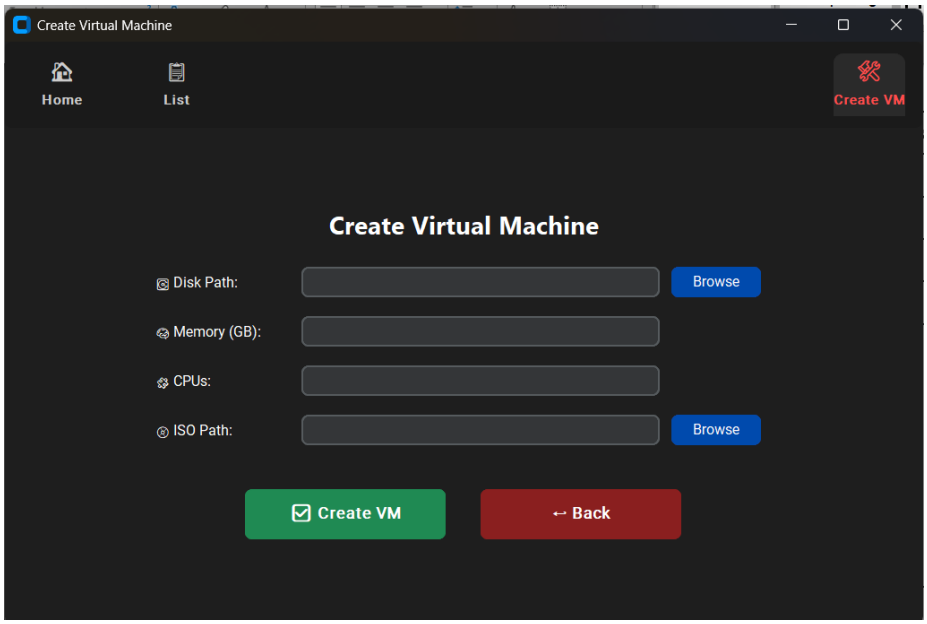
Test Case Name	TVD8: Verify Back Button Functionality
Test Scenario	Clicking “Back” returns to the previous page.
Prerequisites	Page is open
Test Input	Click Back
Execution Steps	1. Click “Back”
Expected Behavior	Redirects back
Assumptions	Navigation handler works
Actual Results	Goes back to the previous page
Status	Pass
Real Life Testing	

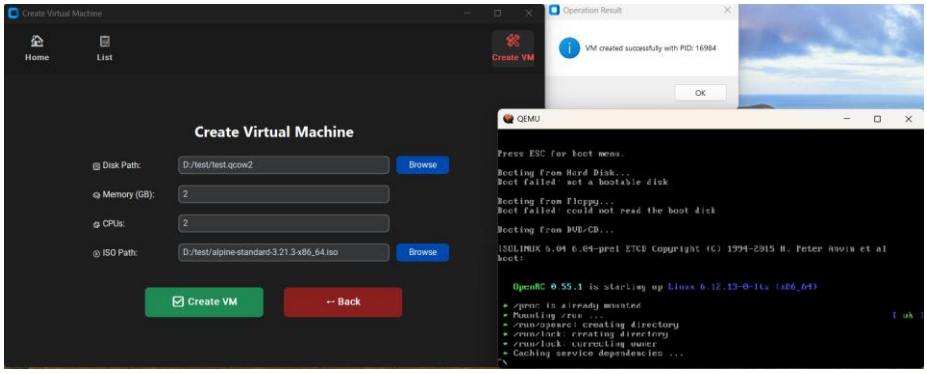
Test Case Name	TVD9: Submit with Only Disk Path Filled
Test Scenario	Incomplete submission should be rejected.
Prerequisites	The form is visible
Test Input	Only browse path
Execution Steps	1. Enter disk path 2. Click “Create Disk”
Expected Behavior	Error due to missing fields
Assumptions	Form requires all fields
Actual Results	Error message box pops up informing user about missing fields
Status	Pass
Real Life Testing	

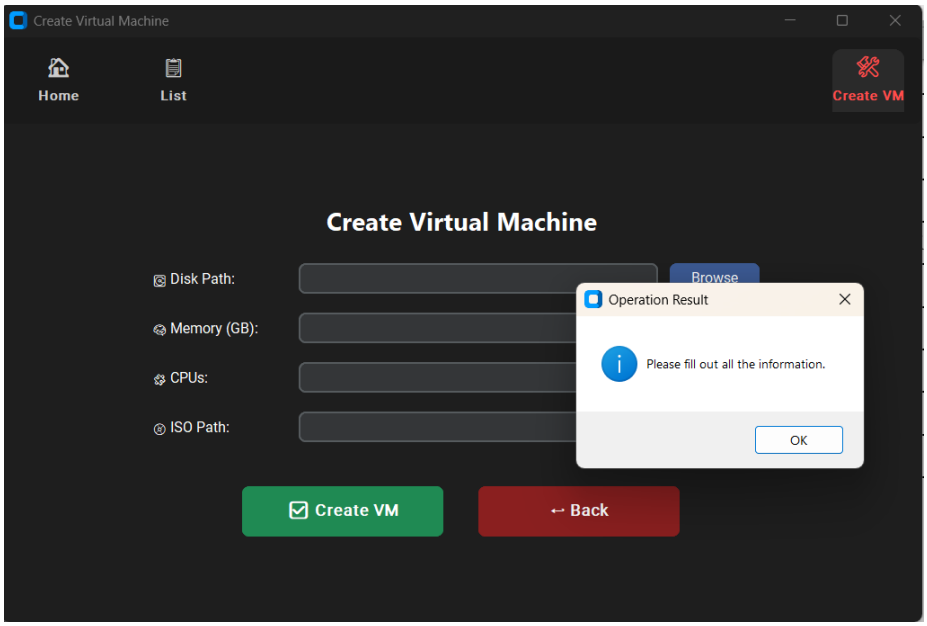
Test Case Name	TVD10: Disk Size Field with Zero
Test Scenario	Zero GB should not be allowed.
Prerequisites	The form is visible
Test Input	0 in Disk Size
Execution Steps	1. Set size to 0 2. Submit
Expected Behavior	Validation error
Assumptions	Size must be >0
Actual Results	Message box pops up informing the user about the disk size error
Status	Pass
Real Life Testing	

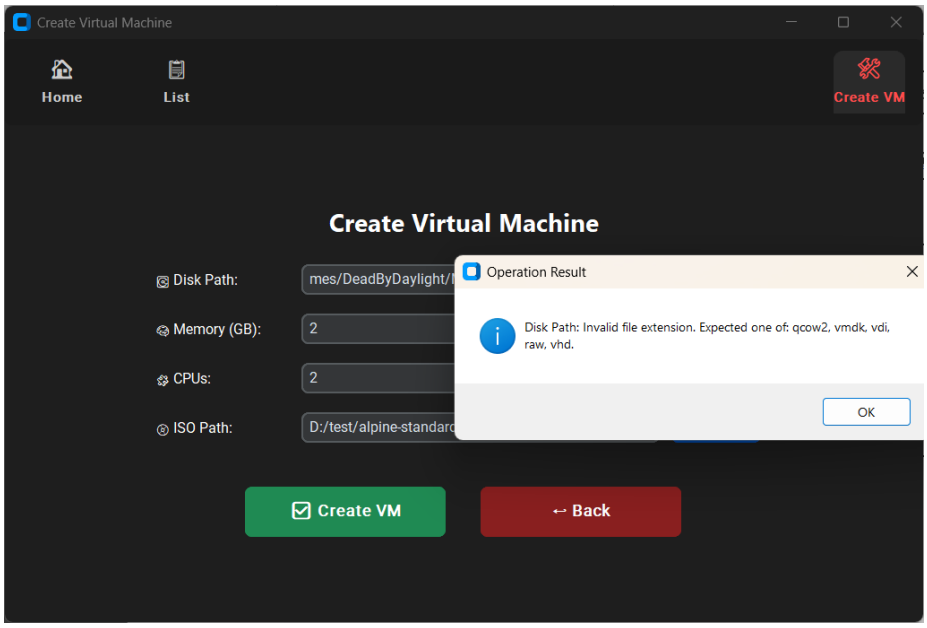
Test Case Name	TVD11: Disk Format Not Selected
Test Scenario	Must require a format.
Prerequisites	Other fields filled
Test Input	Leave format blank
Execution Steps	1. Enter path and size 2. Leave format 3. Submit
Expected Behavior	Validation error
Assumptions	Size must be > 0
Actual Results	Message box pops up informing the user about the missing information error
Status	Pass
Real Life Testing	

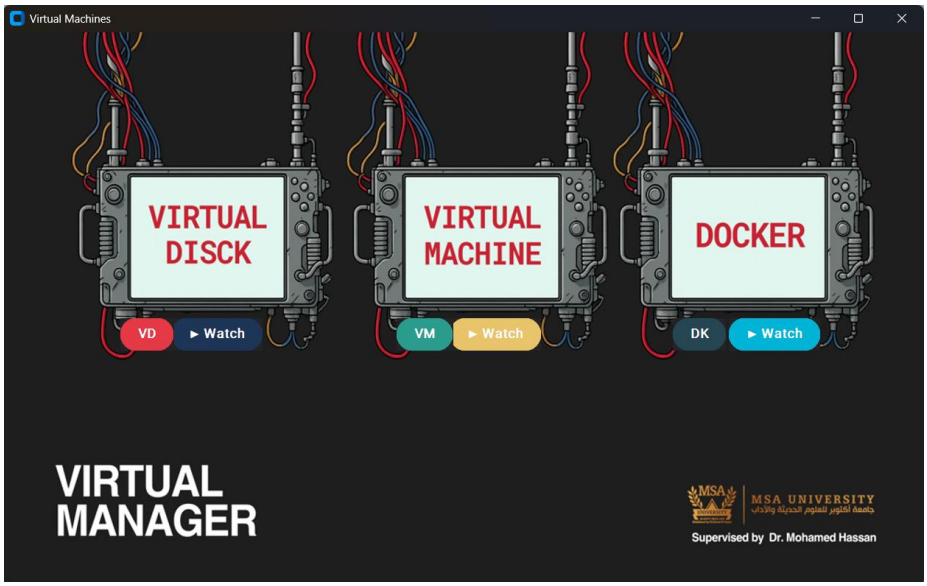
Test Case Name	TVD12: Disk Already Exists at Path
Test Scenario	Duplicate disk path should raise an error.
Prerequisites	File already exists at given path
Test Input	Use the same path as previous disk
Execution Steps	1. Enter existing path 2. Submit form
Expected Behavior	“File already exists” error
Assumptions	System checks for file existence
Actual Results	Message box pops up informing the user about the existing file
Status	Pass
Real Life Testing	

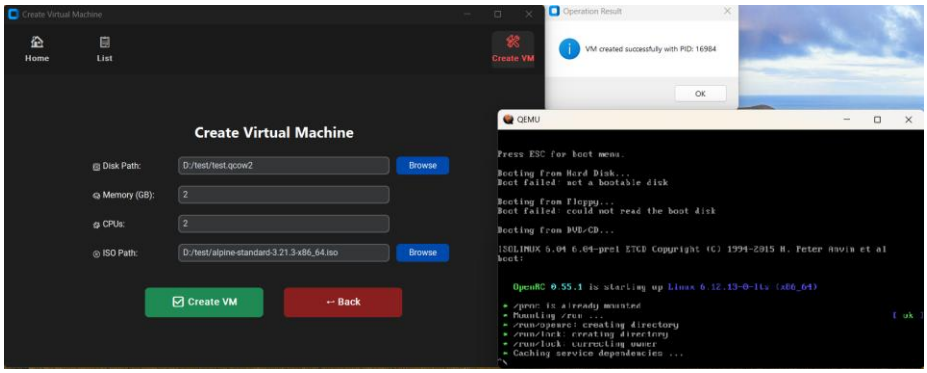
Test Case Name	TVM1: Verify Page Load
Test Scenario	Ensure the "Create Virtual Machine" page loads successfully.
Prerequisites	-
Test Input	Open the "Create VM" tab
Execution Steps	1. Launch the app. 2. Click on “VM"
Expected Behavior	The Create Virtual Machine form loads without errors
Assumptions	UI and navigation work
Actual Results	UI and navigation work
Status	Pass
Real Life Testing	

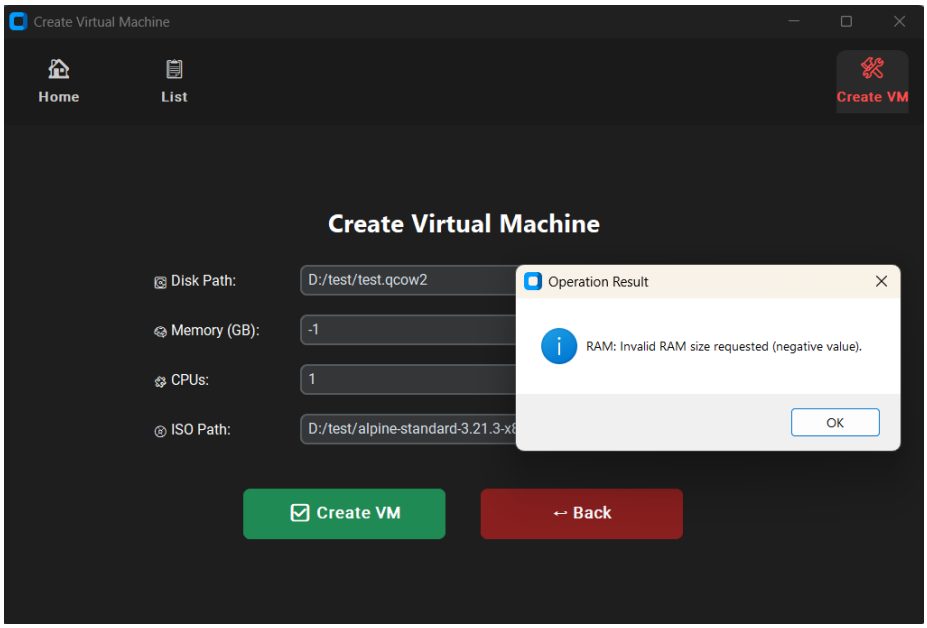
Test Case Name	TVM2: Submit with All Valid Data
Test Scenario	It should create a machine successfully.
Prerequisites	Virtualization engine functional
Test Input	1. Disk input 2. Memory (GB) 3. Number of CPUs 4. ISO file
Execution Steps	1. Fill all fields correctly 2. Click “Create Machine”
Expected Behavior	Machine is created
Assumptions	Backend supports operation
Actual Results	Machine is created successfully and ready to function
Status	Pass
Real Life Testing	

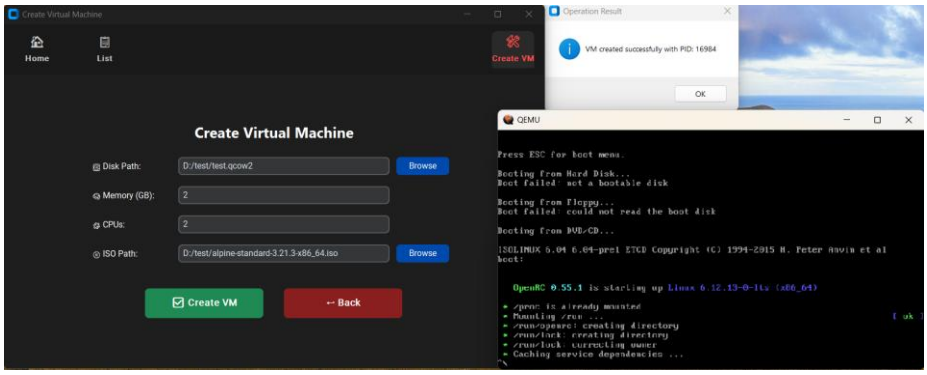
Test Case Name	TVM3: Submit with Empty Fields
Test Scenario	Submit button should validate empty input.
Prerequisites	The Form is visible
Test Input	Leave all fields blank
Execution Steps	1. Click “Create Machine” without filling anything
Expected Behavior	Error shown or machine not created
Assumptions	Validation is present
Actual Results	Validation is present
Status	Pass
Real Life Testing	

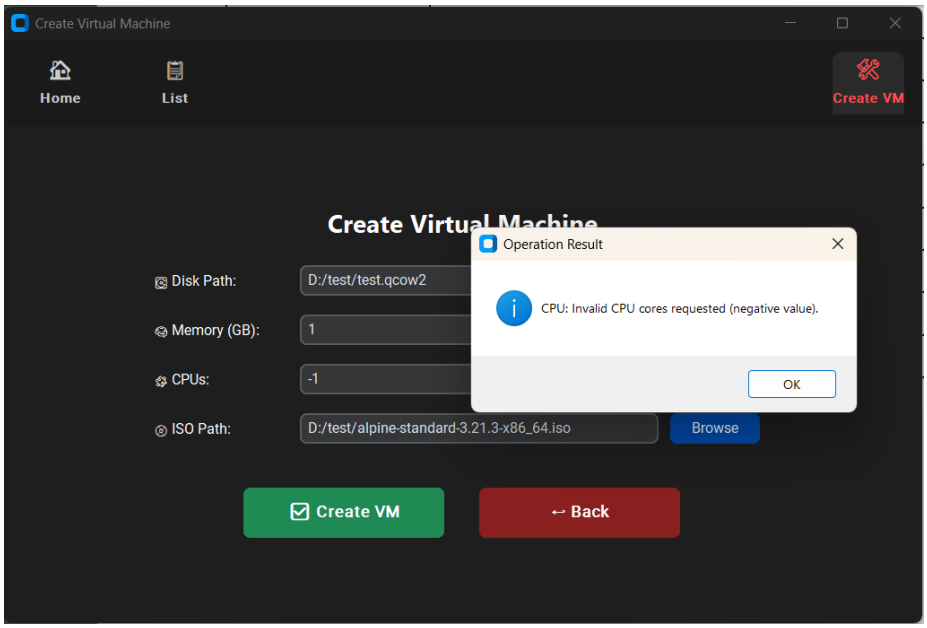
Test Case Name	TVM4: Validate Feedback on VM Creation Failure
Test Scenario	Show clear message if VM creation fails (e.g., missing folder or insufficient disk).
Prerequisites	Known failure scenario exists (like in image: not enough space)
Test Input	Set disk to invalid path ”D:/Games/DeadByDaylight/Manifest_NonUFSFiles_EGS.txt”
Execution Steps	1. Fill form with faulty disk path. 2. Click "Create VM".
Expected Behavior	An error message box pops up informing the user.
Assumptions	Errors returned from backend
Actual Results	Errors returned from backend
Status	Pass
Real Life Testing	

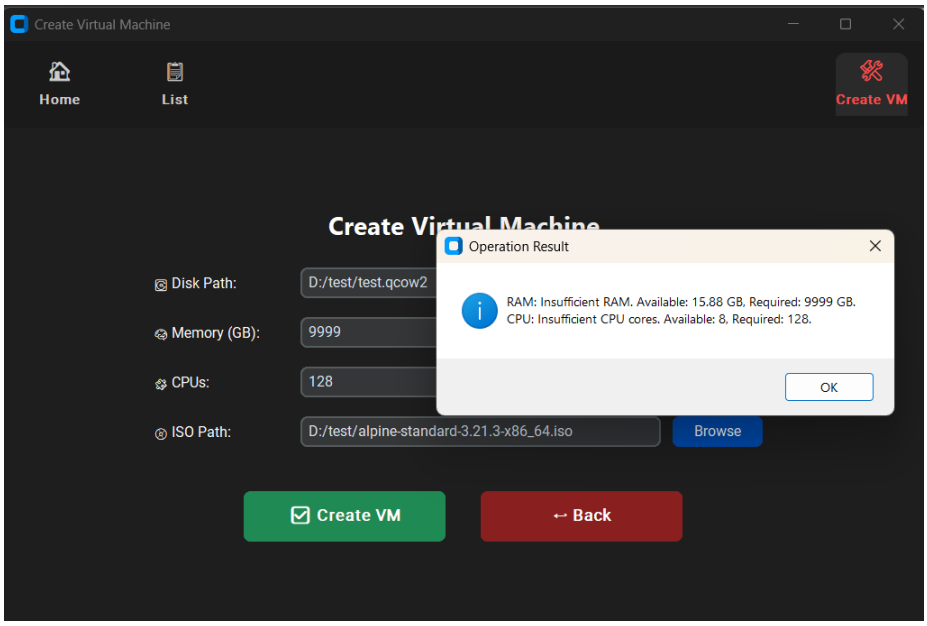
Test Case Name	TVM5: Verify Back Button Functionality
Test Scenario	Clicking “Back” returns to the previous page.
Prerequisites	Page is open
Test Input	Click Back
Execution Steps	2. Click “Back”
Expected Behavior	Redirects back
Assumptions	Navigation handler works
Actual Results	Goes back to the previous page
Status	Pass
Real Life Testing	

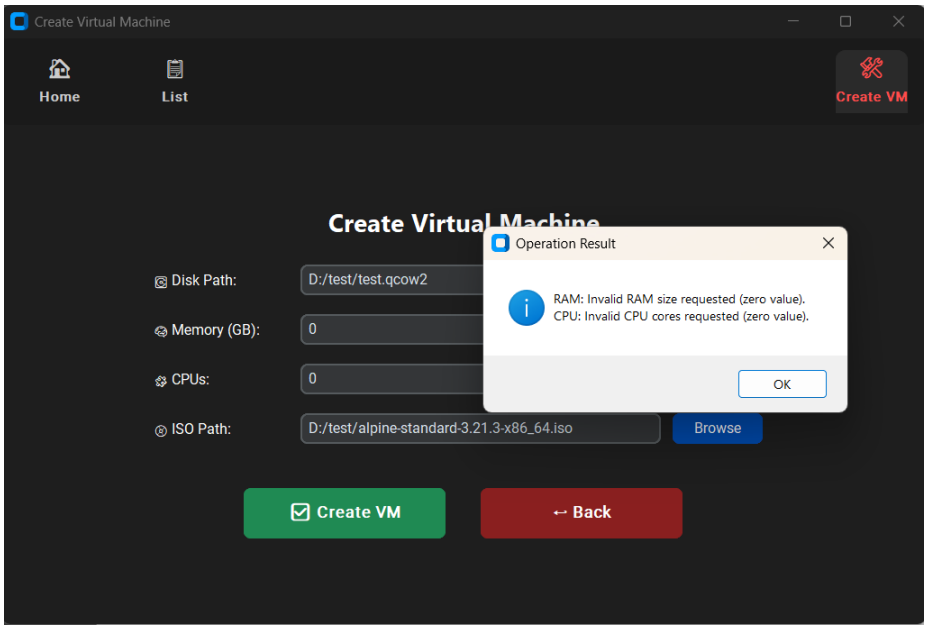
Test Case Name	TVM6: Validate Memory Field (Positive Integer)
Test Scenario	Ensure the user can only enter a positive integer for memory.
Prerequisites	Input box for Memory is active
Test Input	“2”
Execution Steps	1. Click on the Memory field. 2. Enter a value like 2.
Expected Behavior	Field accepts the number and stores it as an integer
Assumptions	Validation is present
Actual Results	Validation is present
Status	Pass
Real Life Testing	

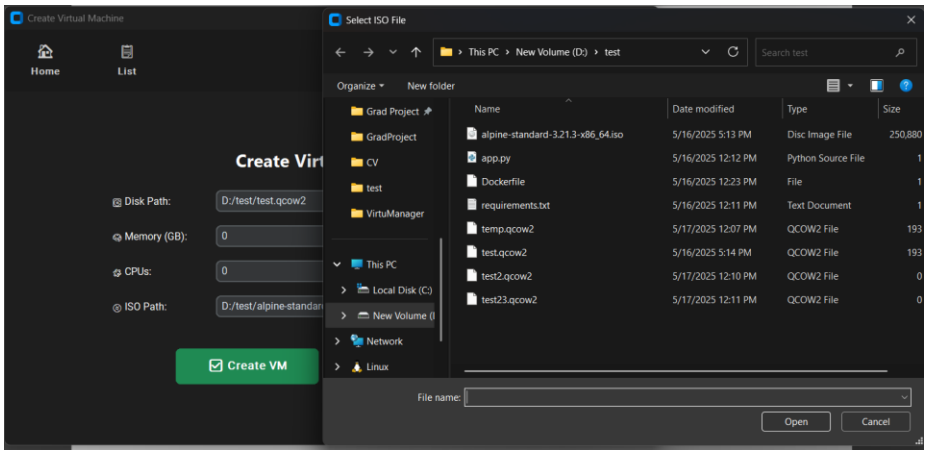
Test Case Name	TVM7: Validate Memory Field (Invalid Input)
Test Scenario	Ensure the field rejects negative values.
Prerequisites	Memory field input not restricted by default
Test Input	“-1”
Execution Steps	1. Click Memory field. 2. Enter invalid input.
Expected Behavior	Field blocks invalid input or displays error
Assumptions	Validation is present
Actual Results	Validation is present
Status	Pass
Real Life Testing	

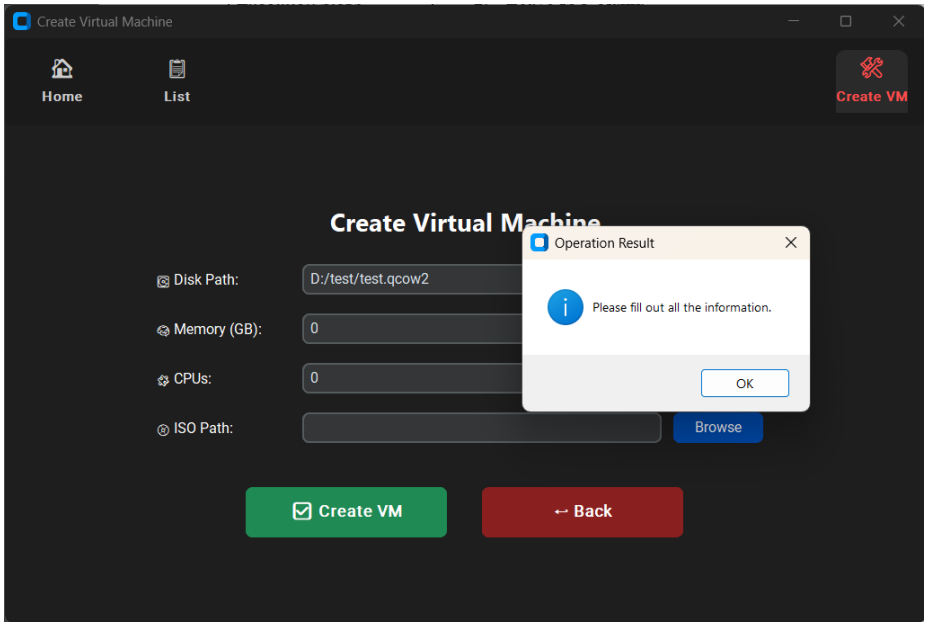
Test Case Name	TVM8: Validate CPUs Field (Positive Integer)
Test Scenario	Ensure the user can only enter valid CPU core counts.
Prerequisites	The CPU field is editable
Test Input	“2”
Execution Steps	1. Enter value in CPUs field.
Expected Behavior	Field accepts integer values
Assumptions	Validation is present
Actual Results	Validation is present
Status	Pass
Real Life Testing	

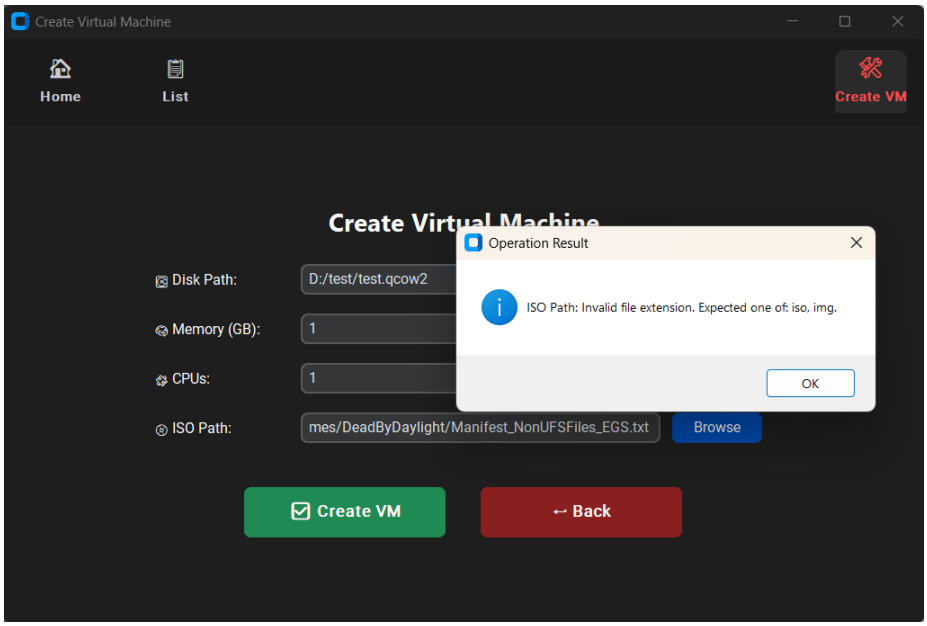
Test Case Name	TVM9: Validate CPUs Field (Negative Integer)
Test Scenario	Ensure the user can only enter valid CPU core counts.
Prerequisites	The CPU field is editable
Test Input	"-1"
Execution Steps	1. Enter value in CPUs field.
Expected Behavior	Field does not accept negative integer values
Assumptions	Validation is present
Actual Results	Validation is present
Status	Pass
Real Life Testing	

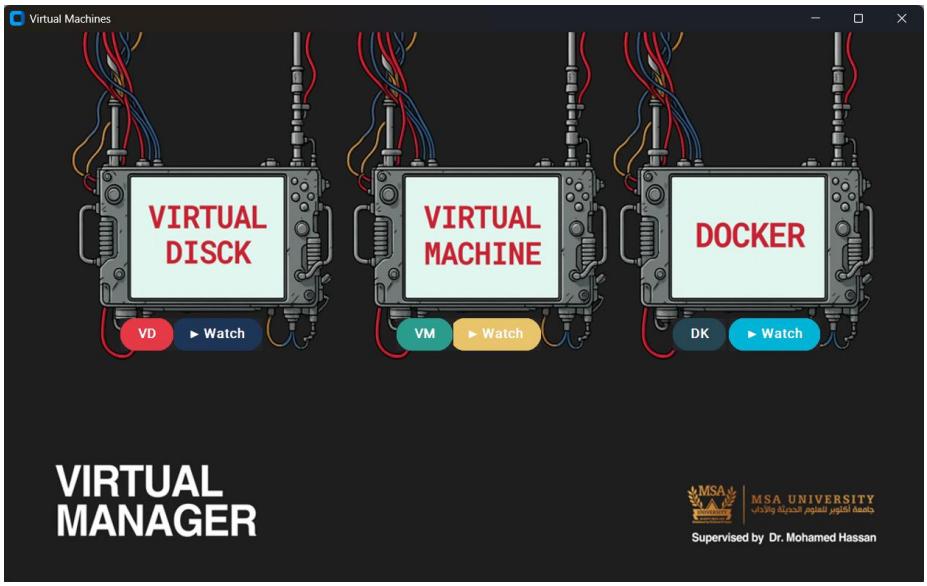
Test Case Name	TVM10: Verify Memory and CPU Limits
Test Scenario	Ensure application restricts memory and CPU input to realistic values.
Prerequisites	Limits defined in code
Test Input	Enter extreme values Memory= “9999”, CPU= “128”
Execution Steps	1. Enter large numbers in both fields.
Expected Behavior	Error message is shown
Assumptions	Validation is present
Actual Results	Validation is present
Status	Pass
Real Life Testing	

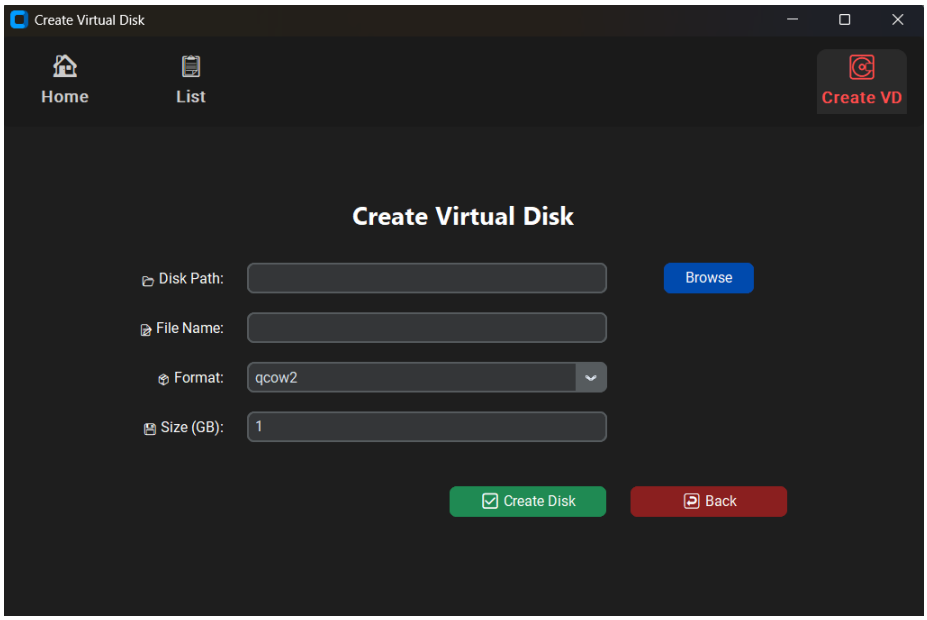
Test Case Name	TVM11: Verify Minimum Memory/CPU Requirement
Test Scenario	Prevent VM creation if memory/CPU value is below minimum allowed.
Prerequisites	Backend enforces min values.
Test Input	Memory: 0 / CPU: 0
Execution Steps	1. Set memory and CPU values to 0. 2. Click Create VM.
Expected Behavior	Error message is shown
Assumptions	Error message informing the user about the incorrect value
Actual Results	Error message informing the user about the incorrect value
Status	Pass
Real Life Testing	

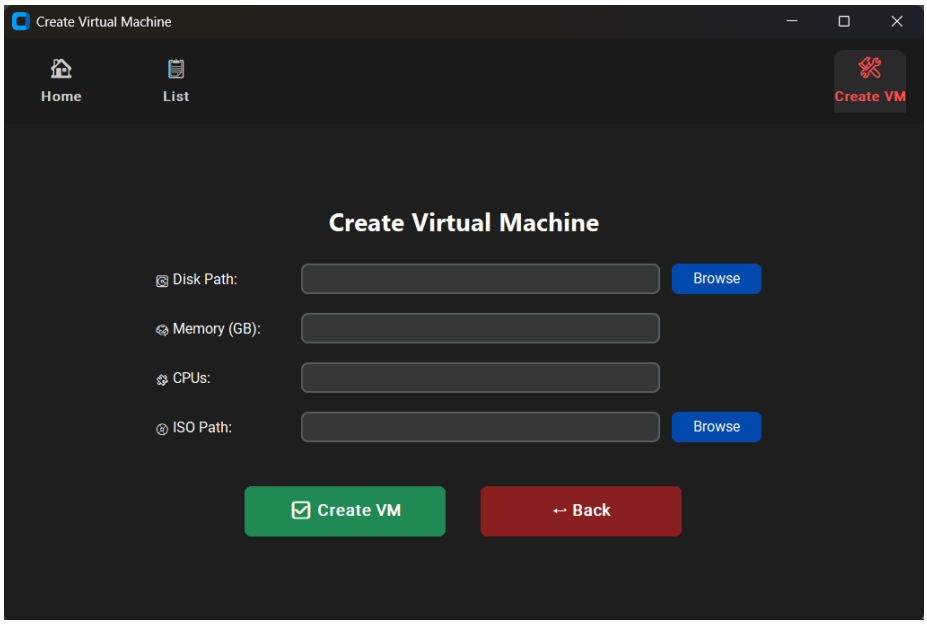
Test Case Name	TVM12: Validate ISO Path Selection
Test Scenario	Ensure users can select ISO file.
Prerequisites	ISO files are available in file system
Test Input	Click "Browse"
Execution Steps	1. Click "Browse" next to ISO Path. 2. Select the ISO file.
Expected Behavior	ISO Path field updates with file location
Assumptions	ISO file format supported
Actual Results	ISO file format supported
Status	Pass
Real Life Testing	

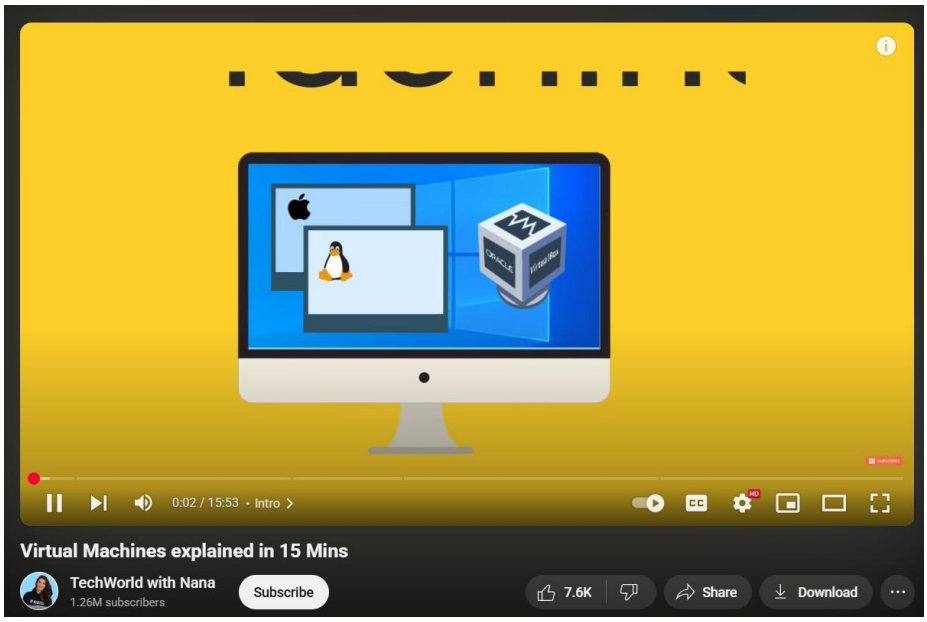
Test Case Name	TVM13: Verify ISO Path is Optional
Test Scenario	Check behavior when ISO path is left blank.
Prerequisites	ISO optional
Test Input	Leave ISO empty, fill rest
Execution Steps	1. Fill required fields. 2. Leave ISO blank. 3. Click Create VM.
Expected Behavior	VM creation succeeds without ISO
Assumptions	Validation is present
Actual Results	Validation is present
Status	Pass
Real Life Testing	

Test Case Name	TVM14: Validate File Format of ISO
Test Scenario	Only .iso files are allowed.
Prerequisites	File system has non-ISO files.
Test Input	Select non-ISO file.
Execution Steps	1. Click Browse. 2. Try selecting .txt or .exe file.
Expected Behavior	System restricts or warns user
Assumptions	Validation is present
Actual Results	Validation is present
Status	Pass
Real Life Testing	

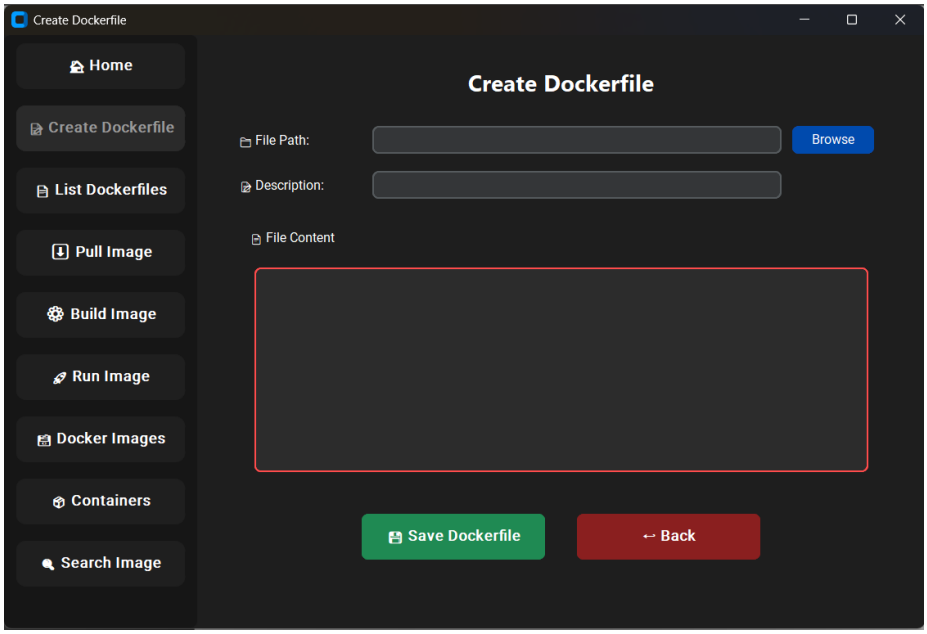
Test Case Name	THP1: Verify Page Load
Test Scenario	Ensure the "HOME" page loads successfully.
Prerequisites	-
Test Input	-
Execution Steps	1. Launch the app.
Expected Behavior	The home page loads without errors
Assumptions	UI and navigation work
Actual Results	UI and navigation work
Status	Pass
Real Life Testing	

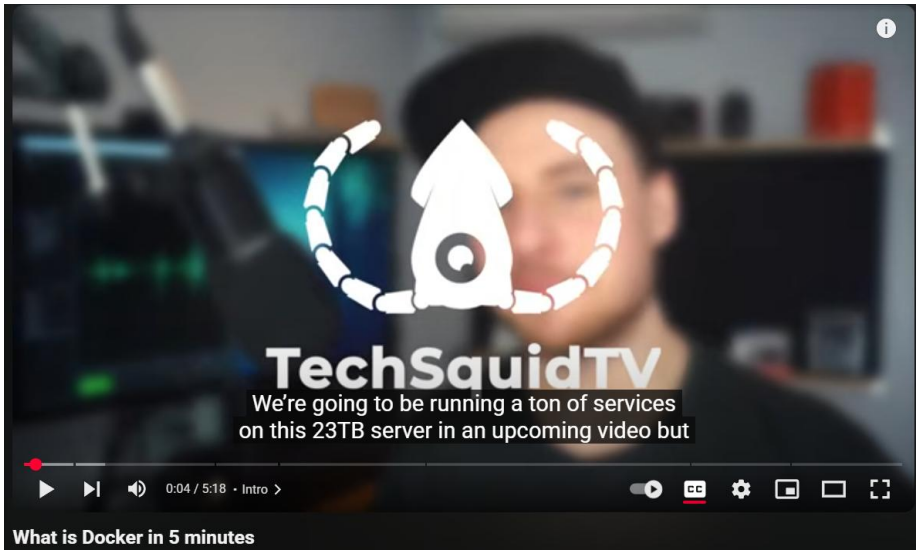
Test Case Name	THP2: Verify “VD” button
Test Scenario	Ensure the “Start VD” button loads the “virtual disk page” correctly.
Prerequisites	-
Test Input	Button input
Execution Steps	1. Press on VD
Expected Behavior	The “virtual disk” page loads without errors
Assumptions	Virtual disk page is loaded
Actual Results	Virtual disk page is loaded
Status	Pass
Real Life Testing	

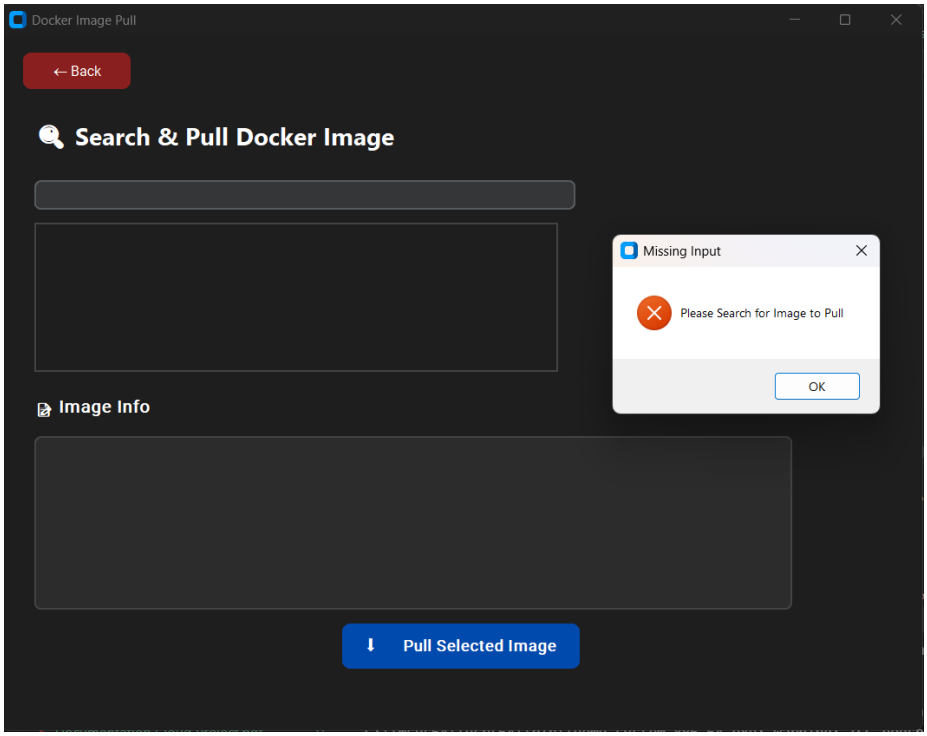
Test Case Name	THP3: Verify “VM” button
Test Scenario	Ensure the “VM” button loads the “virtual machine page” correctly.
Prerequisites	-
Test Input	Button input
Execution Steps	1. Press on VM
Expected Behavior	The “virtual machine” page loads without errors
Assumptions	Virtual machine page is loaded
Actual Results	Virtual machine page is loaded
Status	Pass
Real Life Testing	

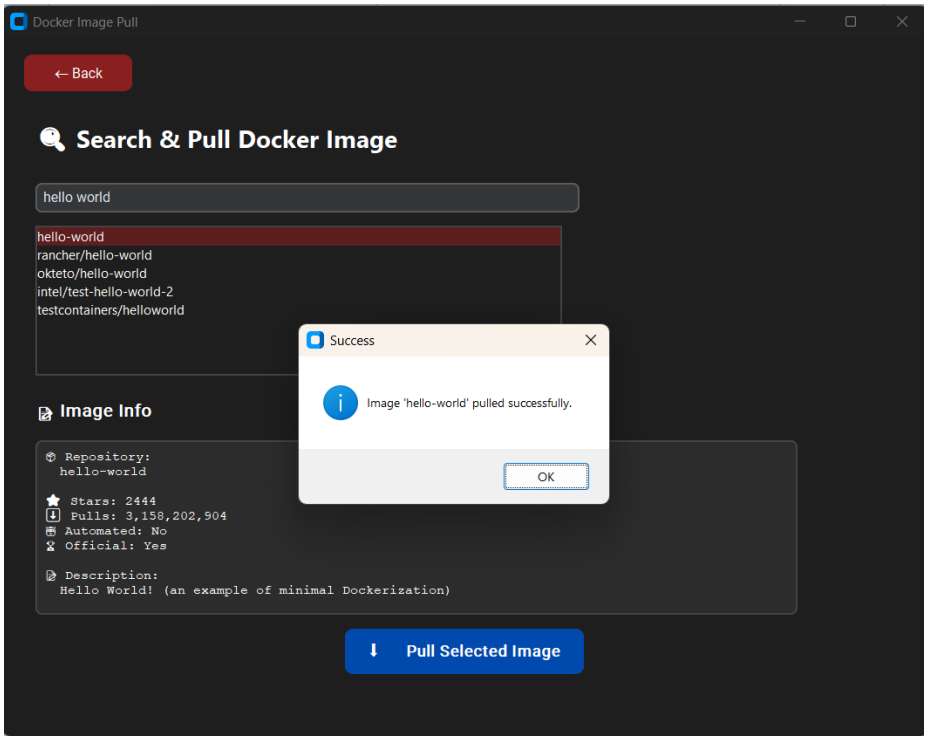
Test Case Name	THP4: Verify the “Watch” button for the VM
Test Scenario	Ensure the “watch” button loads the “YouTube page” correctly.
Prerequisites	-
Test Input	Button input
Execution Steps	1. Press on “watch”
Expected Behavior	The “YouTube” page loads without errors
Assumptions	YouTube page is loaded
Actual Results	YouTube page is loaded
Status	Pass
Real Life Testing	

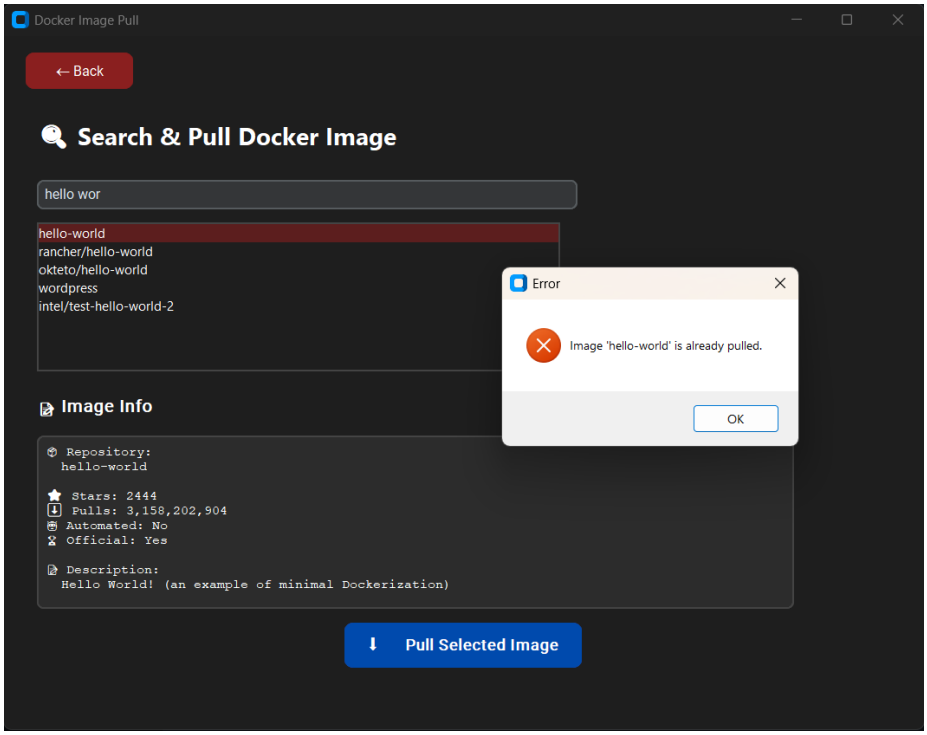
Test Case Name	THP5: Verify the “Watch” button for the VD
Test Scenario	Ensure the “watch” button loads the “YouTube page” correctly.
Prerequisites	-
Test Input	Button input
Execution Steps	2. Press on “watch”
Expected Behavior	The “YouTube” page loads without errors
Assumptions	YouTube page is loaded
Actual Results	YouTube page is loaded
Status	Pass
Real Life Testing	

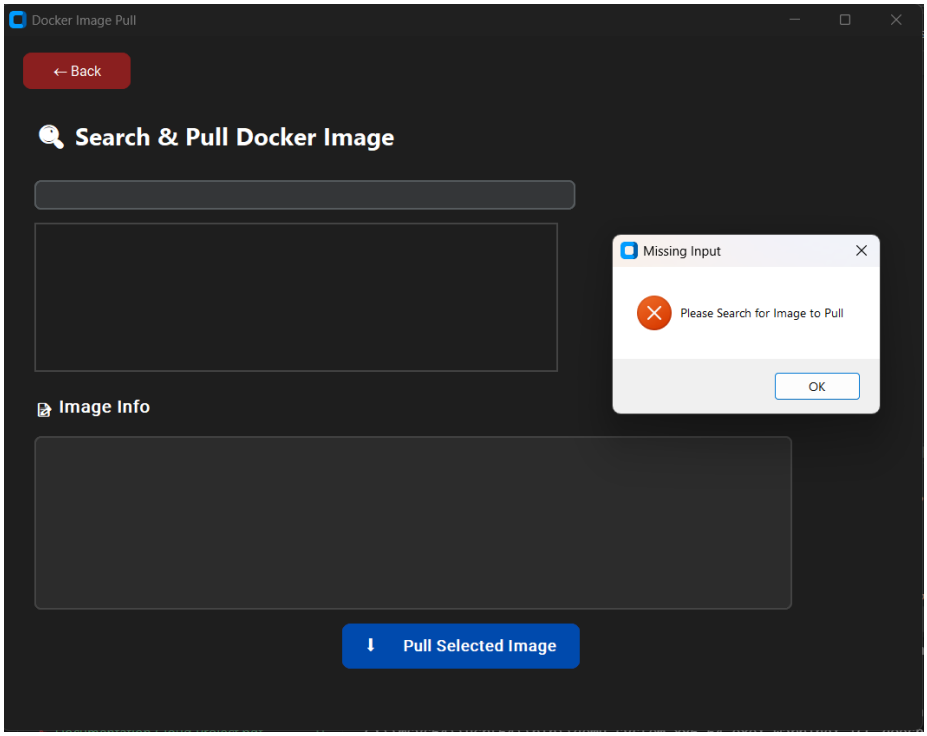
Test Case Name	THP6: Verify “DK” button
Test Scenario	Ensure the “DK” button loads the “docker page” correctly.
Prerequisites	-
Test Input	Button input
Execution Steps	1. Press on start DK
Expected Behavior	The “docker” page loads without errors
Assumptions	Docker page is loaded
Actual Results	Docker machine page is loaded
Status	Pass
Real Life Testing	

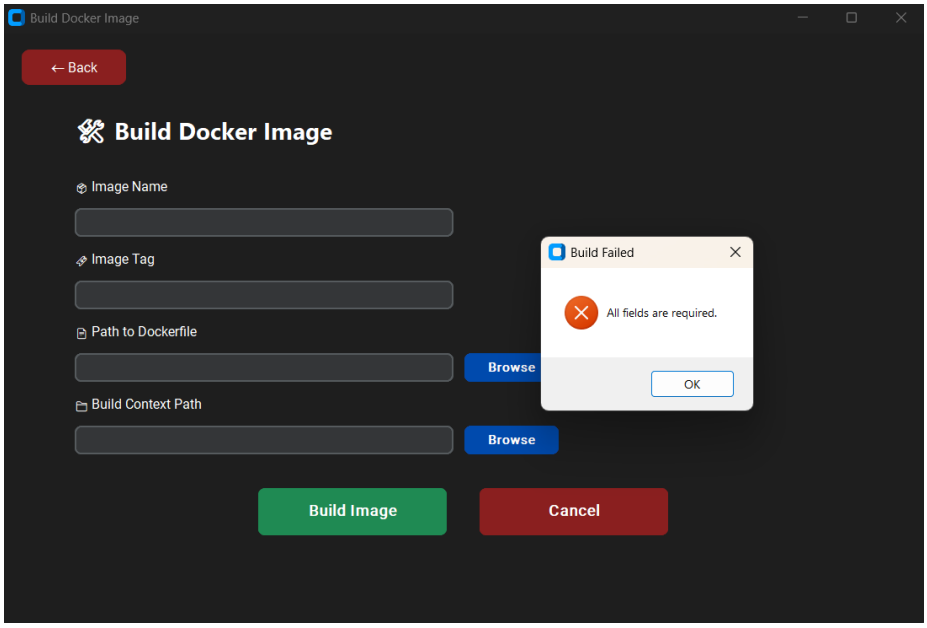
Test Case Name	THP7: Verify the “Watch” button for the DK
Test Scenario	Ensure the “watch” button loads the “YouTube page” correctly.
Prerequisites	-
Test Input	Button input
Execution Steps	1. Press on “watch”
Expected Behavior	The “YouTube” page loads without errors
Assumptions	YouTube page is loaded
Actual Results	YouTube page is loaded
Status	Pass
Real Life Testing	

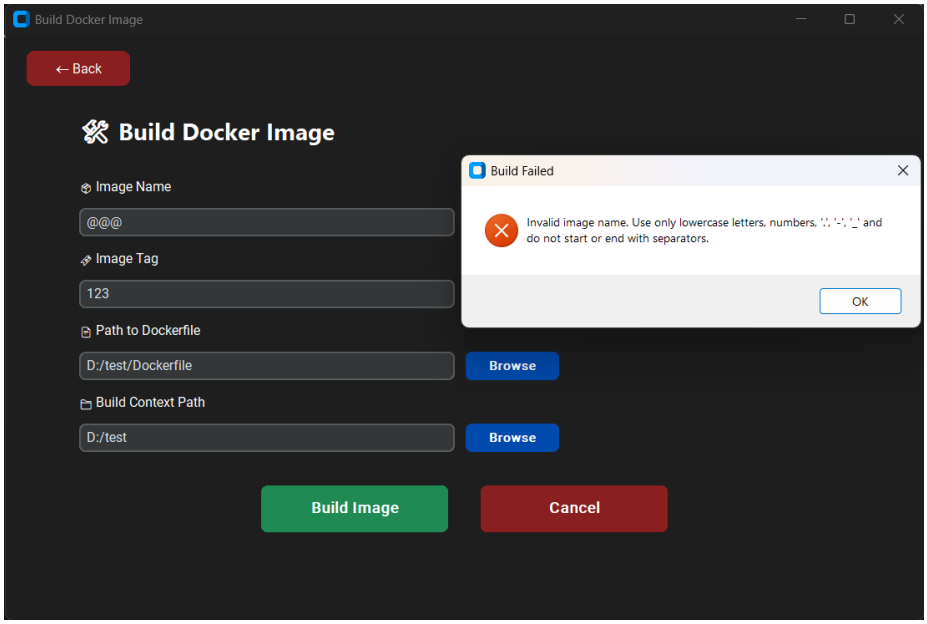
Test Case Name	DK1: Submit with Empty Fields
Test Scenario	Submit button should validate empty input.
Prerequisites	The form is visible
Test Input	Leave all fields blank
Execution Steps	1. Click “Pull Selected Image” without filling anything
Expected Behavior	Error shown or image not pulled
Assumptions	Validation is present
Actual Results	Validation is present
Status	Pass
Real Life Testing	

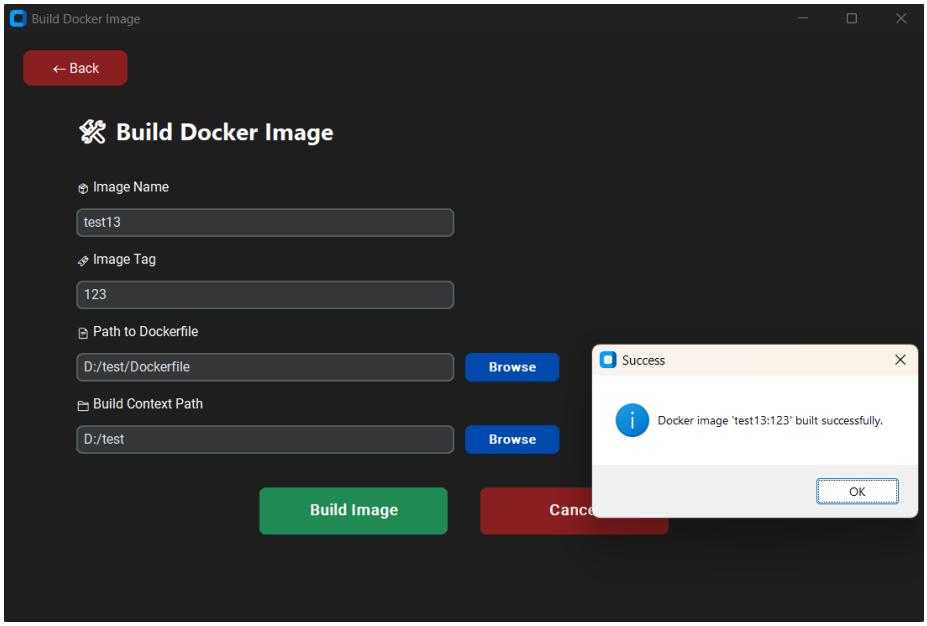
Test Case Name	DK2: Submit with Valid Image Name
Test Scenario	Should pull the image successfully.
Prerequisites	Docker Daemon is running.
Test Input	Valid image name
Execution Steps	1. Enter a valid image name in the search field. 2. Click "Pull Selected Image."
Expected Behavior	Image is pulled successfully.
Assumptions	Validation is present
Actual Results	Validation is present
Status	Pass
Real Life Testing	

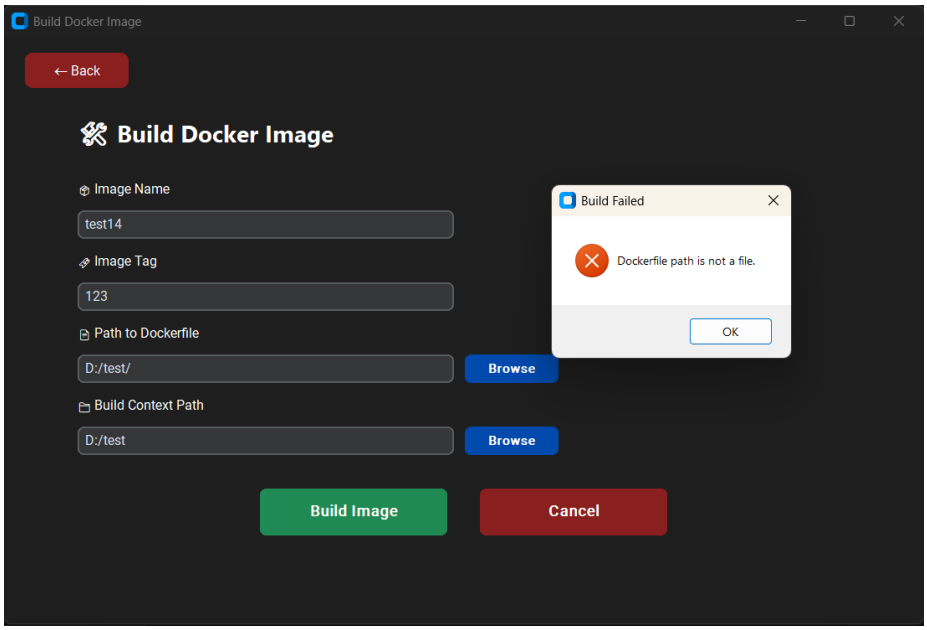
Test Case Name	DK3: Pull Image Multiple Times
Test Scenario	It should handle repeated pulls gracefully.
Prerequisites	Docker Daemon is running.
Test Input	Valid image name
Execution Steps	1. Enter a valid image name in the search field. 2. Click "Pull Selected Image." 3. Click "Pull Selected Image" again.
Expected Behavior	The system should either pull the image again or show a message indicating it's already present.
Assumptions	Validation is present
Actual Results	Validation is present
Status	Pass
Real Life Testing	

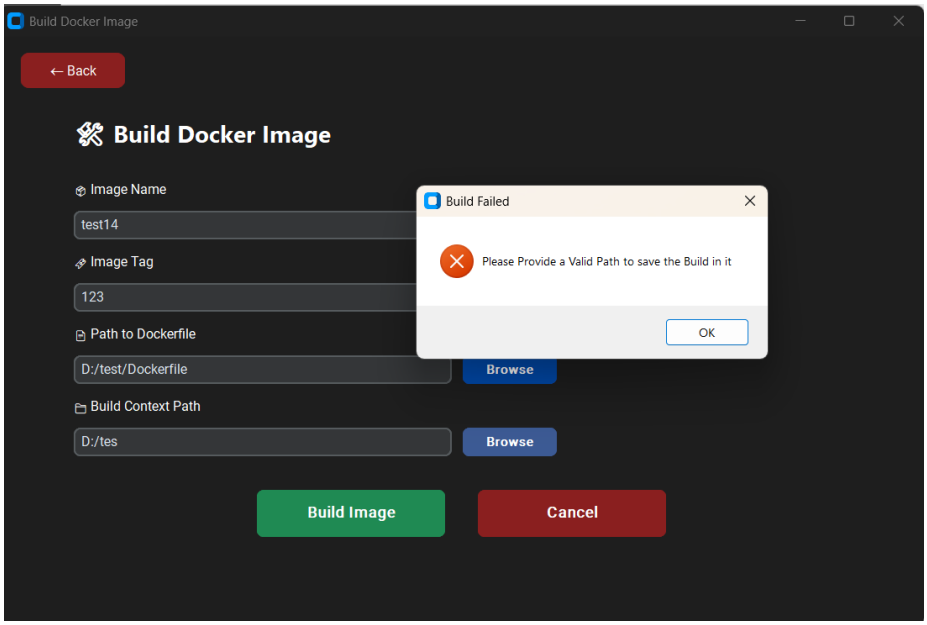
Test Case Name	DK4: Submit with Empty Fields
Test Scenario	Submit button should validate empty input.
Prerequisites	The form is visible
Test Input	Leave all fields blank
Execution Steps	2. Click “Pull Selected Image” without filling anything
Expected Behavior	Error shown or image not pulled
Assumptions	Validation is present
Actual Results	Validation is present
Status	Pass
Real Life Testing	

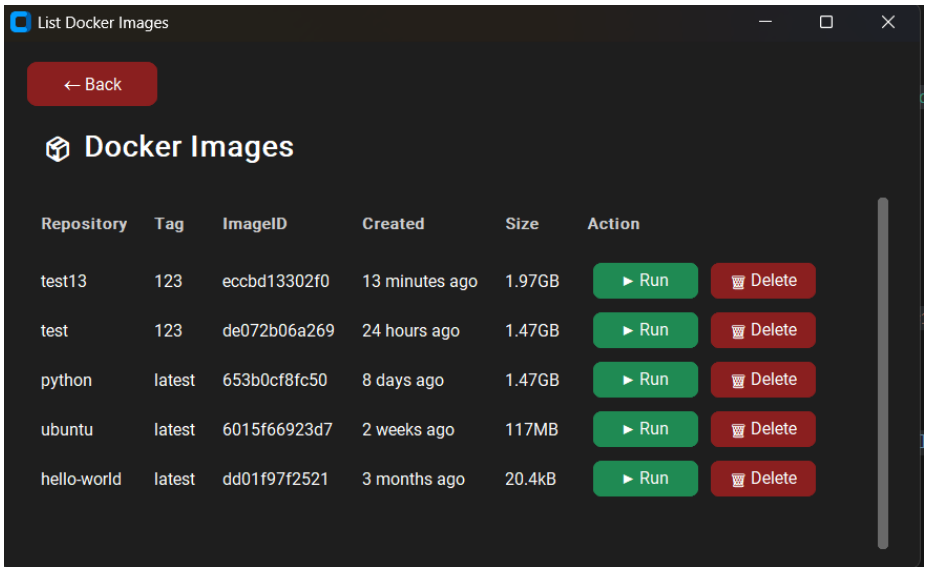
Test Case Name	DKI1: Submit with Empty Fields
Test Scenario	Submit button should validate empty input.
Prerequisites	-
Test Input	Leave all fields blank
Execution Steps	1. Leave all fields empty. 2. Click "Build Image."
Expected Behavior	An error message is displayed indicating that all fields are required.
Assumptions	Validation is present
Actual Results	Validation is present, error message displayed
Status	Pass
Real Life Testing	

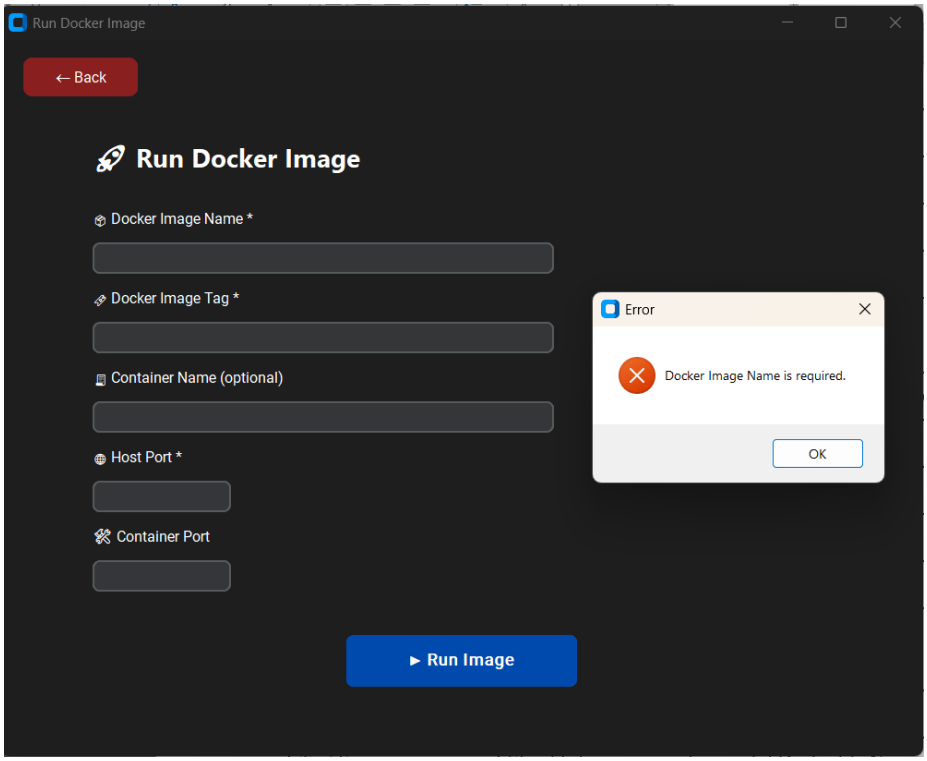
Test Case Name	DKI2: Submit with Invalid Image Name
Test Scenario	Should show an error for invalid image name.
Prerequisites	-
Test Input	Invalid image name (e.g., special characters)
Execution Steps	1. Enter an invalid image name in the search field. 2. Click "Pull Selected Image."
Expected Behavior	An error message is displayed indicating the image name is invalid.
Assumptions	Validation is present
Actual Results	Validation is present
Status	Pass
Real Life Testing	

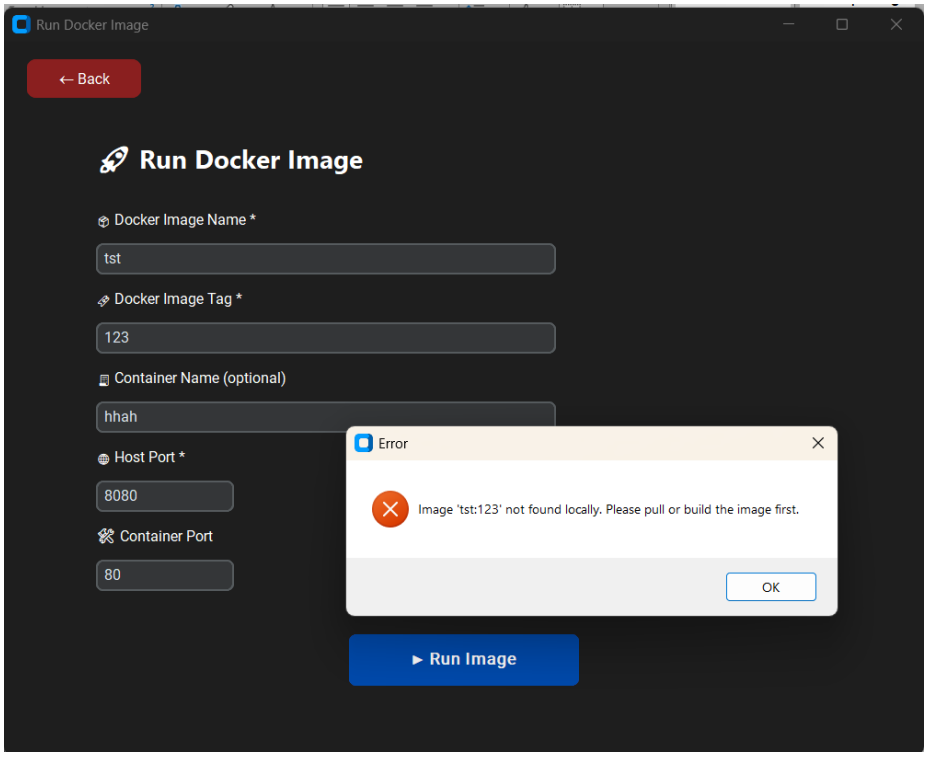
Test Case Name	DKI3: Submit with Valid Data
Test Scenario	Should build the image successfully.
Prerequisites	Docker Daemon is running.
Test Input	Valid image name, tag, and paths
Execution Steps	1. Enter valid data in all fields. 2. Click "Build Image."
Expected Behavior	Image is built successfully.
Assumptions	Validation is present
Actual Results	Image is built and displayed
Status	Pass
Real Life Testing	

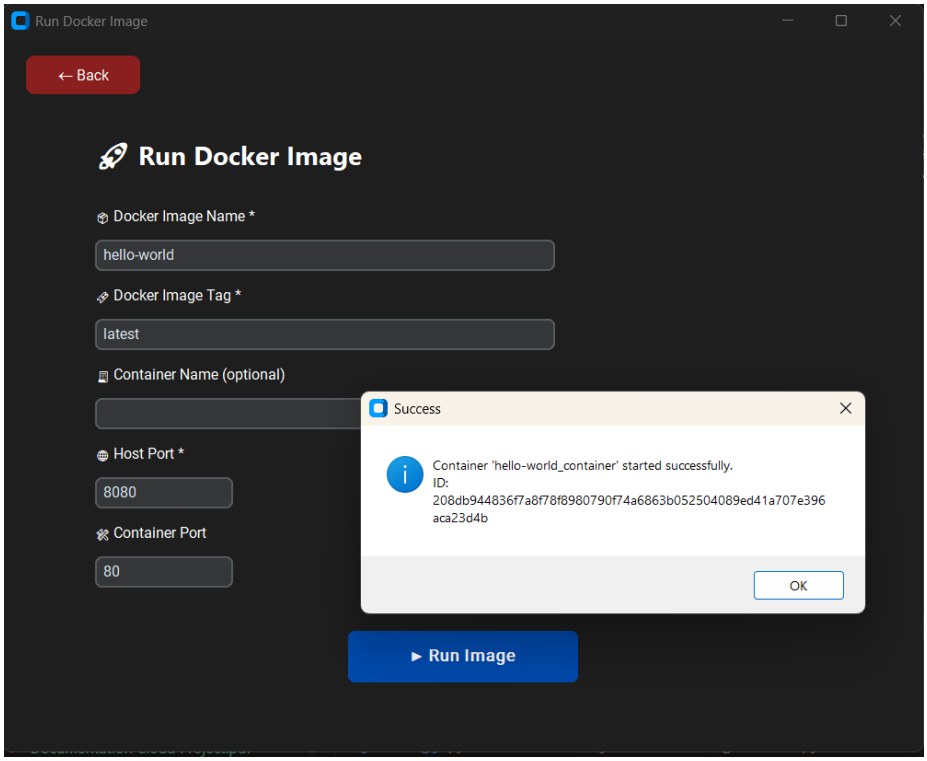
Test Case Name	DKI4: Submit with Invalid Dockerfile Path
Test Scenario	Should show an error for invalid Dockerfile path.
Prerequisites	-
Test Input	Valid image name, tag, and paths
Execution Steps	1. Enter valid data in "Image Name" and "Image Tag." 2. Provide an invalid path for the "Path to Dockerfile." 3. Click "Build Image."
Expected Behavior	An error message indicating the Dockerfile path is invalid.
Assumptions	Backend checks file existence.
Actual Results	Error messages are shown.
Status	Pass
Real Life Testing	

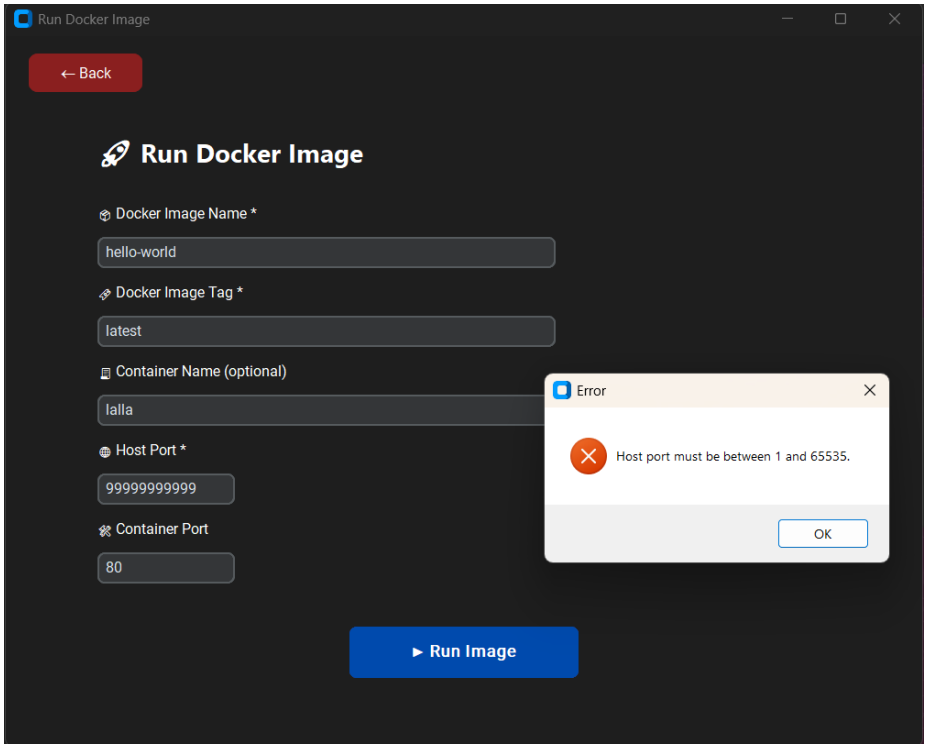
Test Case Name	DKI5: Validate Build Context Path
Test Scenario	Should show an error for invalid build context path.
Prerequisites	-
Test Input	Valid image name, tag, and invalid build context path
Execution Steps	1. Enter valid data in "Image Name" and "Image Tag." 2. Provide an invalid path for the "Build Context Path." 3. Click "Build Image."
Expected Behavior	An error message indicating the build context path is invalid
Assumptions	Backend checks file existence.
Actual Results	Error messages are shown.
Status	Pass
Real Life Testing	

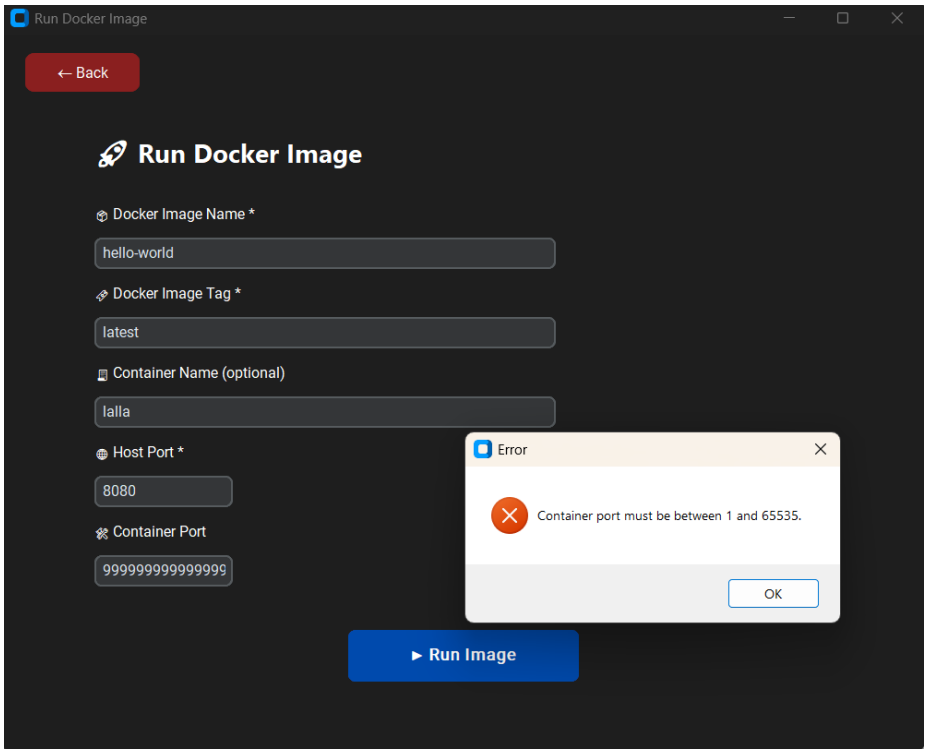
Test Case Name	DKI6: List Image After Build
Test Scenario	Should allow the newly built image.
Prerequisites	Docker Daemon is running.
Test Input	Valid image name of a recently built image
Execution Steps	1. Build an image successfully. 2. Press Back. 3. Click on Docker Images from the Side.
Expected Behavior	The image is shown in the list.
Assumptions	Backend maintains image availability.
Actual Results	Image is created successfully.
Status	Pass
Real Life Testing	

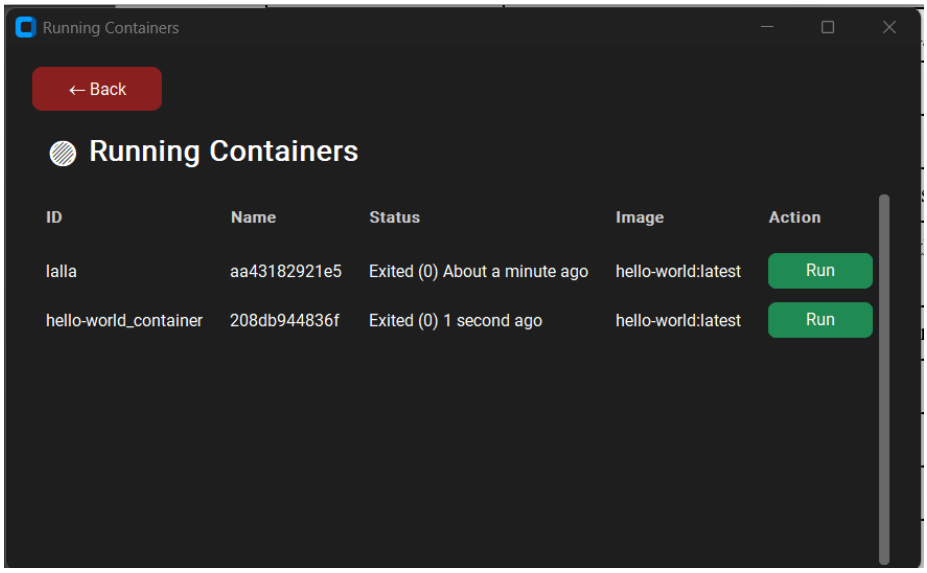
Test Case Name	RDKI1: Submit with Empty Fields
Test Scenario	Submit button should validate empty input.
Prerequisites	-
Test Input	Leave all fields blank
Execution Steps	1. Leave all fields empty. 2. Click "Run Image."
Expected Behavior	An error message is displayed indicating that all fields are required.
Assumptions	Validation is present
Actual Results	Validation is present, error message displayed
Status	Pass
Real Life Testing	

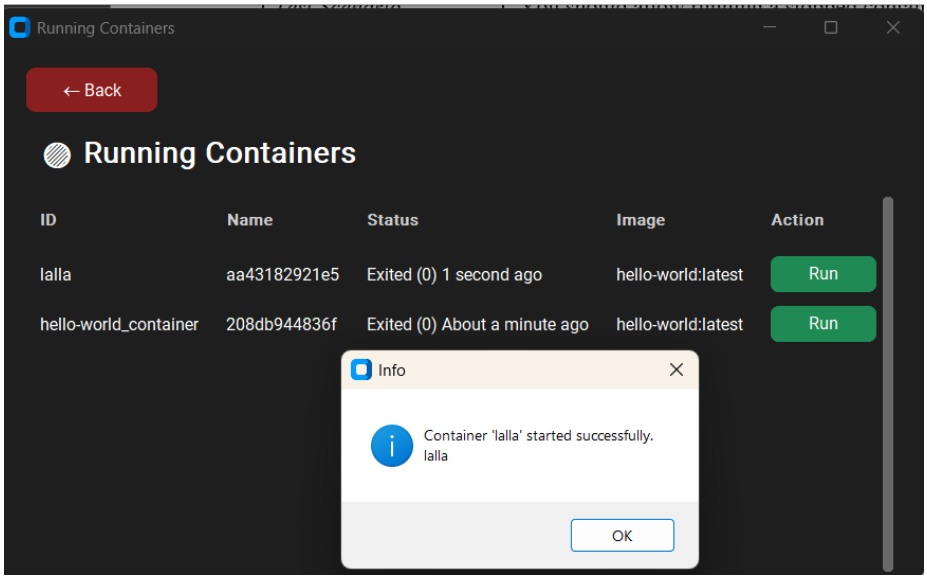
Test Case Name	RDKI2: Submit with Invalid Docker Image Name
Test Scenario	Should show an error for invalid Docker Image Name.
Prerequisites	-
Test Input	Invalid Docker image name (e.g., special characters)
Execution Steps	1. Enter an invalid Docker image name in the "Docker Image Name" field. 2. Fill other fields with valid data. 3. Click "Run Image."
Expected Behavior	An error message indicating the Docker Image name is invalid.
Assumptions	Validation is present.
Actual Results	Error messages are shown.
Status	Pass
Real Life Testing	

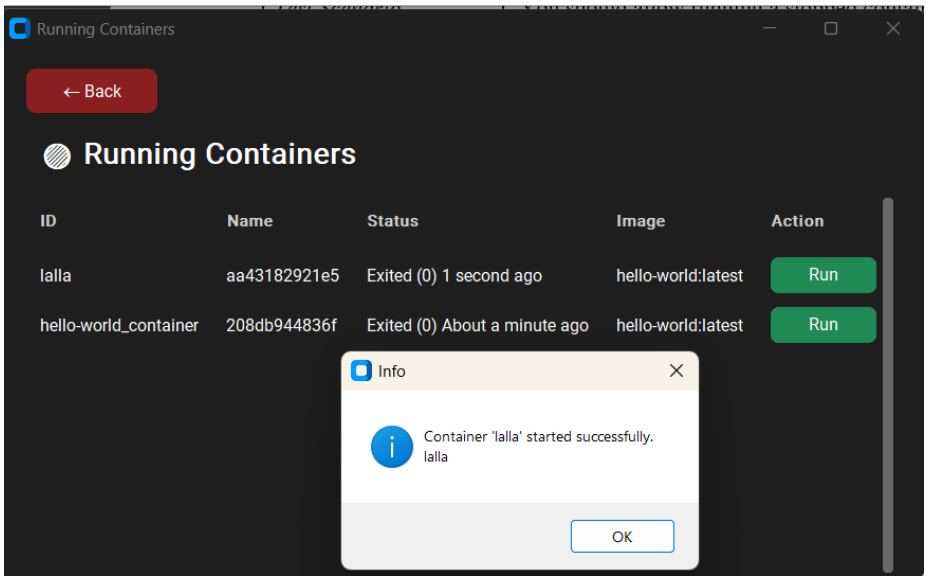
Test Case Name	RDKI3: Submit with Valid Data
Test Scenario	Should run the image successfully.
Prerequisites	Docker Daemon is running.
Test Input	Valid Docker image name, tag, host port, and optional container name
Execution Steps	1. Enter valid data in all fields. 2. Click "Run Image."
Expected Behavior	The image is run successfully.
Assumptions	Backend supports the operation.
Actual Results	Image is running as expected
Status	Pass
Real Life Testing	

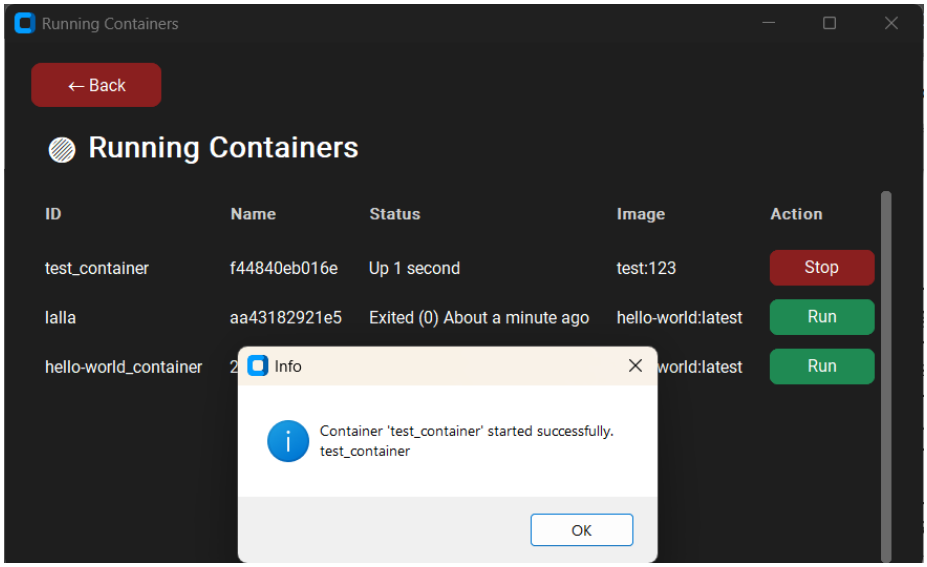
Test Case Name	RDKI4: Submit with Invalid Host Port
Test Scenario	Should show an error for invalid host port.
Prerequisites	-
Test Input	Valid Docker image name and tag, invalid host port
Execution Steps	1. Enter valid data in all fields. 2. Click "Run Image."
Expected Behavior	An error message indicating the host port is invalid.
Assumptions	Backend checks port validity.
Actual Results	Error messages are shown.
Status	Pass
Real Life Testing	

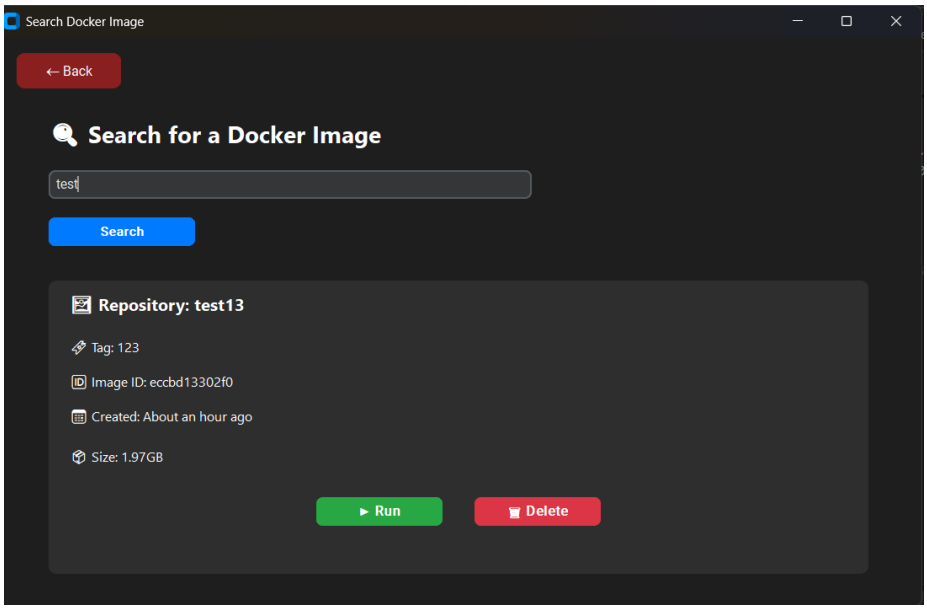
Test Case Name	RDKI5: Submit with Invalid Container Port
Test Scenario	Should show an error for invalid container port.
Prerequisites	-
Test Input	Valid Docker image name and tag, invalid container port
Execution Steps	1. Enter valid Docker image name and tag. 2. Provide an invalid container port. 3. Click "Run Image."
Expected Behavior	An error message indicating the container port is invalid.
Assumptions	Backend checks port validity.
Actual Results	Error messages are shown.
Status	Pass
Real Life Testing	

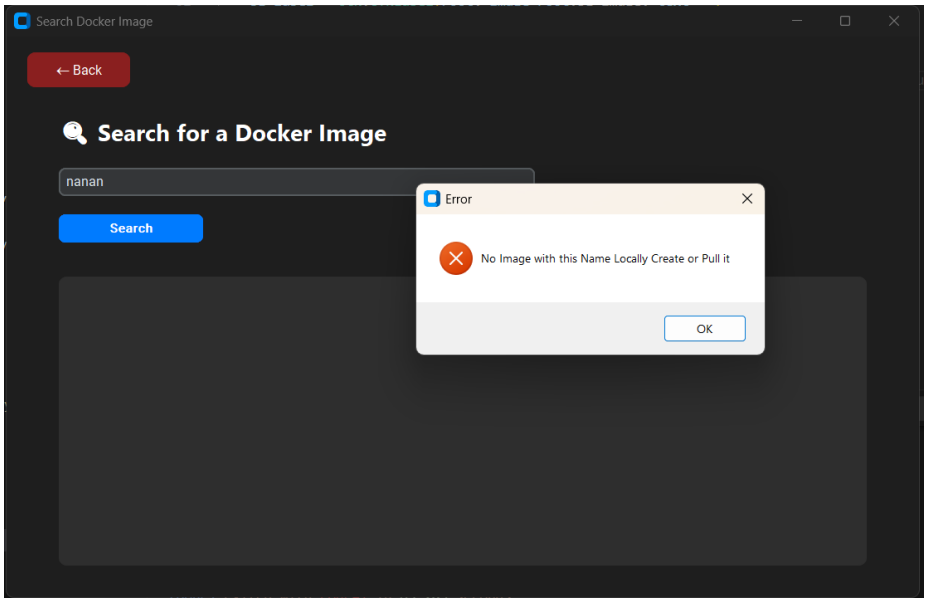
Test Case Name	RCDK1: Display Running Containers
Test Scenario	You should display a list of containers.
Prerequisites	At least one container.
Test Input	-
Execution Steps	1. Navigate to the "Running Containers" page.
Expected Behavior	A list of running containers is displayed with their IDs, statuses, images, and action options.
Assumptions	Backend provides the correct status of running containers.
Actual Results	Running containers are displayed correctly.
Status	Pass
Real Life Testing	

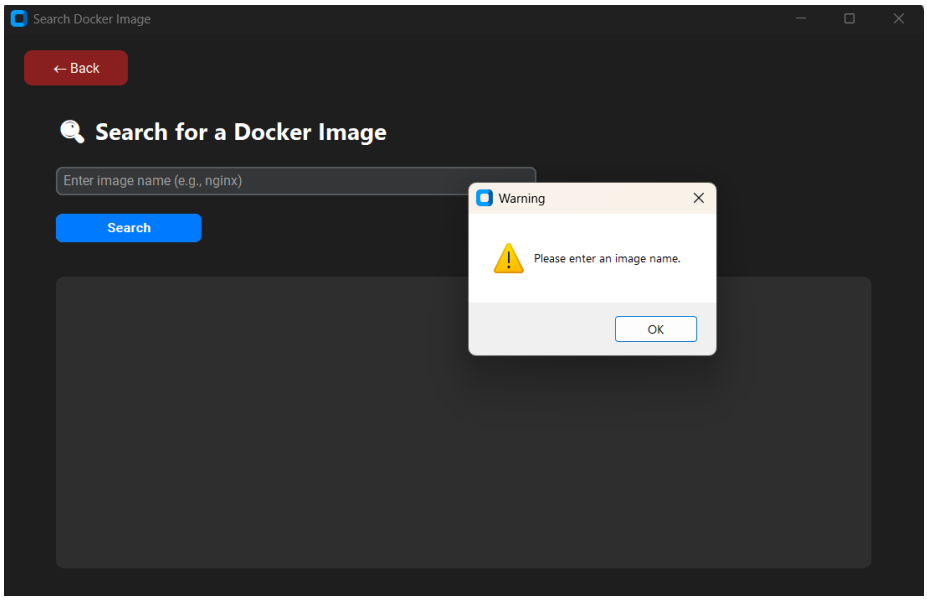
Test Case Name	RCDK2: Run Container from List
Test Scenario	You should allow running a stopped container from the list.
Prerequisites	At least one stopped container is available.
Test Input	-
Execution Steps	1. Navigate to the "Running Containers" page. 2. Click "Run" next to a stopped container.
Expected Behavior	The selected container is started successfully.
Assumptions	Backend supports starting containers.
Actual Results	Container is running as expected.
Status	Pass
Real Life Testing	

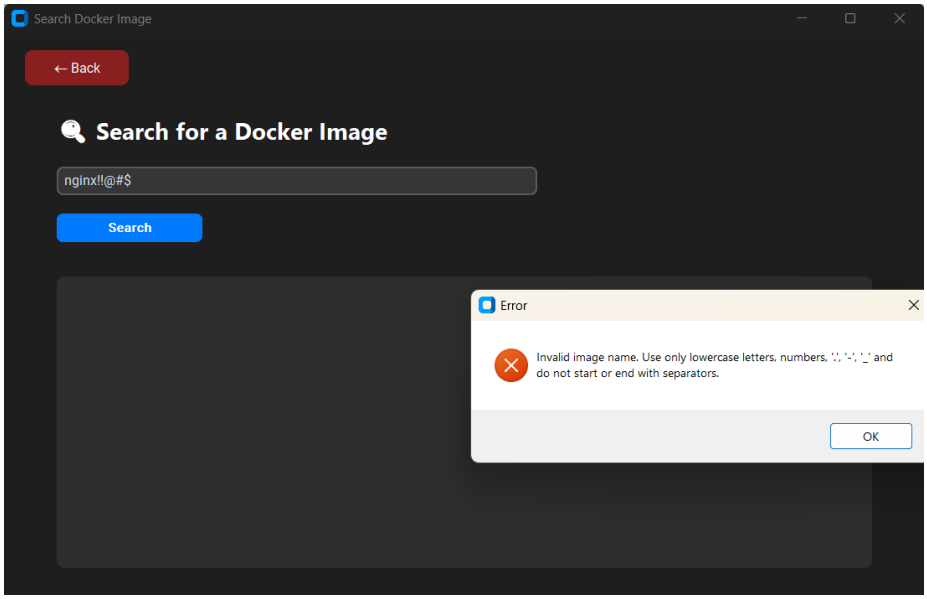
Test Case Name	RCDK3: Check Action Buttons Functionality
Test Scenario	Should ensure action buttons work as intended. (in this case: RUN)
Prerequisites	At least one stopped container is available.
Test Input	-
Execution Steps	1. Navigate to the "Running Containers" page. 2. Click "Run" next to a stopped container. 3. Verify the action completes successfully.
Expected Behavior	The container starts without errors.
Assumptions	Backend processes requests correctly.
Actual Results	Container runs successfully.
Status	Pass
Real Life Testing	

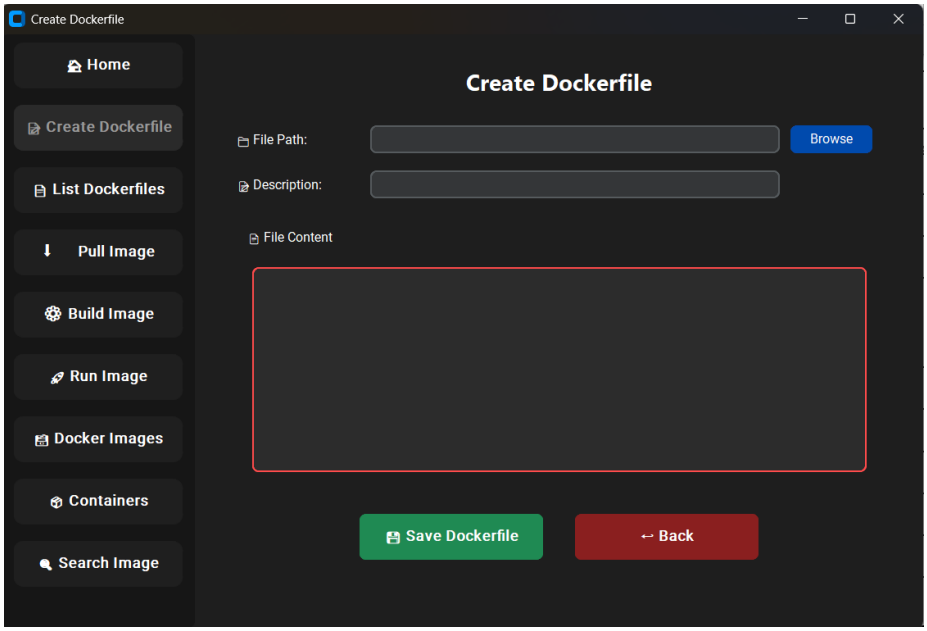
Test Case Name	RCDK4: Refresh Running Containers List
Test Scenario	You should refresh the list of running containers.
Prerequisites	At least one container is running or stopped.
Test Input	-
Execution Steps	1. Navigate to the "Running Containers" page. 2. Click a refresh button (if available).
Expected Behavior	The list of running containers updates to reflect current statuses.
Assumptions	Backend provides updated container statuses.
Actual Results	The list is updated correctly.
Status	Pass
Real Life Testing	

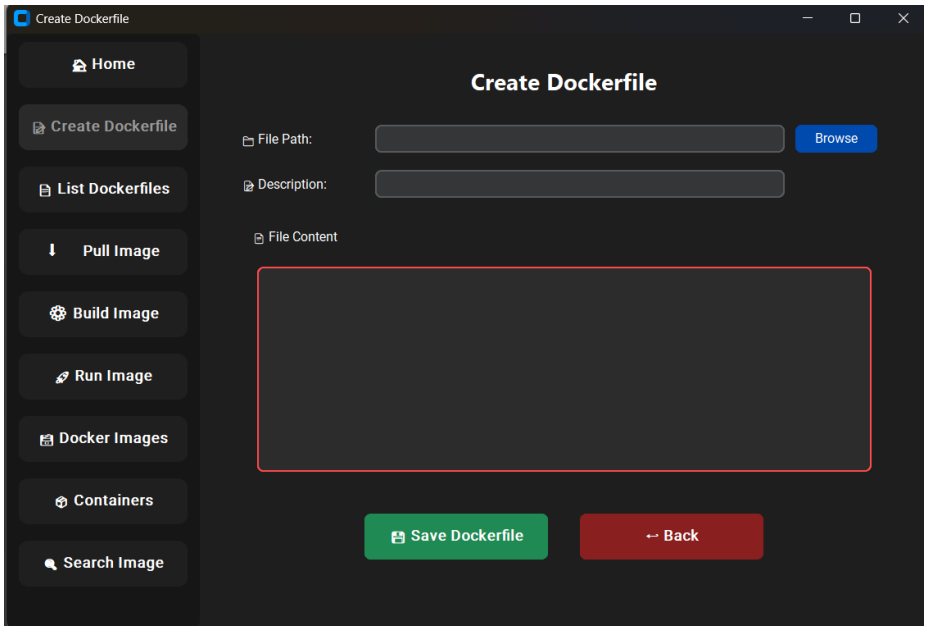
Test Case Name	SDKI1: Search for a Docker Image using valid name
Test Scenario	Users should be able to search for Docker images using valid name.
Prerequisites	Backend is properly integrated with Docker Hub
Test Input	-
Execution Steps	1. Enter a valid image name in the input field. 2. Click on the Search button.
Expected Behavior	A list of Docker images matching the input should be displayed with relevant details.
Assumptions	Backend handles both valid.
Actual Results	Results are displayed correctly.
Status	Pass
Real Life Testing	

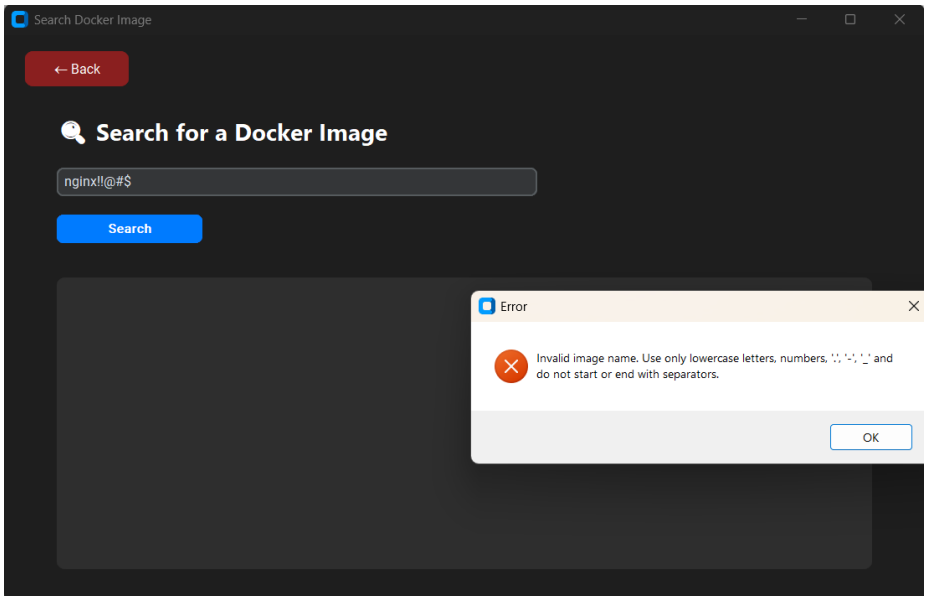
Test Case Name	SDKI2: Search for a Docker Image using invalid name
Test Scenario	Users should be able to search for Docker images using invalid name.
Prerequisites	Backend is properly integrated with Docker Hub
Test Input	-
Execution Steps	1. Enter a invalid image name in the input field. 2. Click on the Search button.
Expected Behavior	An error message or “No results found” should be shown.
Assumptions	Backend handles both invalid.
Actual Results	Error messages or empty results are shown correctly.
Status	Pass
Real Life Testing	

Test Case Name	SDKI3: Search with Empty Input
Test Scenario	Search should not proceed when no input is provided.
Prerequisites	-
Test Input	-
Execution Steps	1. Navigate to the Search Docker Image page. 2. Leave the input field empty. 3. Click the Search button.
Expected Behavior	An error message prompting the user to enter an image name.
Assumptions	Input validation is handled on the backend.
Actual Results	Proper validation message shown
Status	Pass
Real Life Testing	

Test Case Name	SDKI4: Search with Special Characters
Test Scenario	The system should handle invalid characters.
Prerequisites	-
Test Input	nginx!!@#$
Execution Steps	1. Enter special characters in the search bar. 2. Click **Search**.
Expected Behavior	An error message prompting the user to enter a valid name.
Assumptions	Input validation is handled on the backend.
Actual Results	Proper validation message shown
Status	Pass
Real Life Testing	

Test Case Name	GTC1: Navigation Consistency
Test Scenario	Verify that all navigation elements (Back button, side nav) work as expected and consistently.
Prerequisites	-
Test Input	-
Execution Steps	1. Click **Back** from different pages. 2. Use side navigation to jump between sections.
Expected Behavior	Navigation takes the user to the correct page.
Assumptions	navigation consistency
Actual Results	navigation consistency
Status	Pass
Real Life Testing	

Test Case Name	GTC2: Visual Alignment and Spacing
Test Scenario	Check that all components are well-aligned and spaced correctly.
Prerequisites	-
Test Input	-
Execution Steps	1. Inspect each page layout visually. 2. Resize the window and verify alignment holds.
Expected Behavior	- No misalignment, overflow, or awkward spacing. - Text, buttons, and inputs appear balanced.
Assumptions	Consistent Visual Alignment and Spacing
Actual Results	Consistent Visual Alignment and Spacing
Status	Pass
Real Life Testing	

Test Case Name	GTC3: Error Message Clarity
Test Scenario	Users should receive clear, actionable error messages for backend failures or invalid actions.
Prerequisites	-
Test Input	-
Execution Steps	1. Try actions like "Search Image" or "Create Disk" with invalid input.
Expected Behavior	- Error message is clear - No raw error logs or developer traces visible.
Assumptions	Clear error messages
Actual Results	Clear error messages
Status	Pass
Real Life Testing	

Testing:

Testing Approach:

- Unit Testing: Each module, including disk and VM creation, was unit tested to ensure accuracy in functionality.
- System Validation Testing: Ensured that system resource checks (CPU, RAM, disk) work accurately before creating VMs.
- User Interface Testing: Verified the Tkinter GUI for proper user interaction, ensuring inputs are captured correctly.
- Compatibility Testing:
    - Tested with different virtual disk formats (qcow2, vmdk, etc.).
    - Docker image building tested using multiple Dockerfiles and build contexts.

Key Results:

- System resource validation: Prevented VM creation when resources were insufficient.
- Multiple disk format handling: Ensured disk compatibility without failures.
- Docker image build functionality: Verified successful image creation and correct error handling.
- GUI interaction: Inputs were correctly processed, and visual feedback was intuitive across features.

User Manual:

How to Use VirtuManager

1. **Installation:**
   - Install dependencies using the following command:
     $pip\ install\ psutil$
2. **Create Virtual Disk:**
   - Launch VirtuManager.
   - Enter the required parameters (disk name, path, format, and size).
   - Click on "Create Disk" to generate the virtual disk.
3. **Create Virtual Machine:**
   - In the GUI, the CPU, RAM, disk, and ISO image.
   - Click "Create VM." The system will validate available resources before proceeding.
   - If resources are insufficient, a warning will appear; otherwise, the VM will be created.
4. **Build Docker Image:**
   - Navigate to the Docker section from the main menu.
   - Enter the Image Name and Tag.
   - Select the Dockerfile via file browser.
   - Select the Build Context Folder via directory browser.
   - Click "Build" to start image creation.
   - A progress bar will display the build process.
   - After completion, a dialog will show success or error feedback.
5. **Running the Application:**
   - Open the terminal and write $python\ main.py$.
   - The GUI will appear, allowing interaction with the disk and VM creation features.

Troubleshooting:

- **Insufficient Resources (VM Creation):**
  - Close background apps or increase available resources.
- **Invalid Disk Format:**
  - Only use supported formats like qcow2, vmdk, etc.
- **Docker Build Failure:**
  - Ensure the Dockerfile and context folder are valid.
  - Confirm Docker is properly installed and running on your system.