UIT University

Department of Computer Science

Batch-2022 – SE

BLACK BOX TESTING REPORT

SOFTWARE QUALITY ENGINEERING

Physiotherapy Guidance System

By

Roha Pathan	(22FA-009-SE)
Laiba Laeeq	(22FA-047-SE)
Manail Ghouri	(22FA-059-SE)

Supervised by

Dr. Muhammad Wasim, Head of department (CS)

Dr. Hafiz Syed Tariq Ali (Assistant Professor), Physiotherapist & Researcher at PILL (Pakistan institute of living and learning) & Remedial Hospital

Software Quality Engineering

Black Box Testing Report using Selenium

Project Title: Physiotherapy Guidance System (PGS)

Course: Software Quality Engineering

Course Code: CSE-303

Faculty name: Talha Ahmed

Testing Technique: Black Box Testing

Testing Tool: Selenium WebDriver

Platform Tested: Web Application

Semester: Fall 2025

Table of Contents

1. Introduction

Software Quality Engineering plays a vital role in ensuring that a software system performs its intended functions reliably and meets user expectations. As modern applications become more interactive and user-centric, systematic testing becomes essential to detect functional issues early.

This report documents the black box testing activities performed on the Physiotherapy Guidance System (PGS) using Selenium automation. The testing focuses on validating system behavior from an end-user perspective, without considering the internal implementation. The primary objective is to verify that key functionalities related to exercise recording, analysis, and comparison work correctly under expected usage scenarios.

The report is prepared in accordance with the course requirements and follows a checklist-based, detailed testing approach to ensure completeness and clarity.

2. Project Overview

The Physiotherapy Guidance System (PGS) assists patients in performing physiotherapy exercises correctly at home. The system compares a patient's posture with reference exercise videos and provides real-time feedback.

2.1 Key Features

- User registration and login
- Exercise selection
- Live posture tracking using camera input
- Real-time visual and audio feedback
- Exercise session completion and progress tracking

3. Testing Approach

3.1 Testing Type: Black Box Testing

Black box testing is a software testing technique in which the internal structure of the system is not examined. Test cases are derived from functional requirements and expected user behavior. Inputs are provided to the system, and outputs are observed to verify correctness.

In this assignment, black box testing was selected to evaluate how end users interact with the Physiotherapy Guidance System through its user interface.

3.2 Checklist Based Testing Strategy

To ensure thorough coverage, a checklist-based testing strategy was followed. A checklist helps verify that all critical functional areas are tested systematically and consistently.

The checklist focused on:

- Page navigation correctness
- Button visibility and click actions
- File upload functionality
- Handling of long-running processes (pose detection)
- Pop-up and confirmation dialog handling
- Successful completion of user workflows

Each Selenium script was designed to satisfy multiple checklist items, ensuring no major functionality was overlooked.

4. Testing Tool: Selenium

4.1 Overview of Selenium

Selenium is an open-source automation testing framework used for testing web applications. It simulates real user interactions such as clicking buttons, entering input, and navigating between pages.

4.2 Why Selenium?

Following are the reasons due to which we used selenium:

- Automates repetitive test cases
- Supports multiple browsers
- Works well with UI-based testing
- Integrates easily with Python

5. Test Environment

Component	**Description**
Operating System	Windows 11
Programming Language	Python
Automation Tool	Selenium WebDriver
Browser	Google Chrome
Framework	PyTest / Unittest
Application Type	Web

6. Test Scope

6.1 In scope

- Login and authentication
- Navigation between screens
- Exercise selection
- Session start and end functionality
- User input validation

6.2 Out of scope

- AI posture accuracy evaluation
- Backend performance testing
- Security and penetration testing

7. Test Scenarios

Based on system requirements and checklist items, the following high level test scenarios were identified:

1. User is able to access the home page successfully
2. User can initiate a new exercise recording session
3. User can upload a valid exercise video file
4. System processes the video and detects pose landmarks
5. User can save the analyzed exercise as a template
6. User can select a saved exercise for comparison
7. User can compare an exercise using webcam or video input
8. System handles confirmation pop-ups correctly

These scenarios reflect real world usage of the application by patients or clinicians.

8. Test Cases

Test cases were derived from the checklist and executed using Selenium automation. Each test case validates a specific functional requirement.

8.1 Test Case – 01: Navigate home page

- **Input:** Access application URL
- **Expected Result:** Home page loads successfully
- **Status:** Pass

8.2 Test Case – 02: Record new exercise

- **Input:** Click "Record New Exercise"
- **Expected Result:** Recording interface opens
- **Status:** Pass

8.3 Test Case – 03: Upload exercise video

- **Input:** Upload valid MP4 file
- **Expected Result:** File uploads without error
- **Status:** Pass

8.4 Test Case – 04: Analyze and save exercise

- **Input:** Click "Analyze and Save"
- **Expected Result:** Pose detection starts and completes
- **Status:** Pass

8.5 Test Case – 05: Compare exercise

- **Input:** Select exercise and click "Compare"
- **Expected Result:** Comparison interface opens
- **Status:** Pass

8.6 Test Case – 06: Test with video or webcam

- **Input:** Upload test video or use webcam
- **Expected Result:** Comparison process initiates successfully
- **Status:** Pass

9. Selenium test implementation and execution

Selenium automation scripts were written in Python to perform black box testing on the Physiotherapy Guidance System. The test cases were designed based on real user workflows, such as recording an exercise, uploading a video, saving a template, and comparing exercises. The scripts interact with the system only through visible user interface elements like buttons, input fields, and pop-ups, without accessing or relying on any internal code or logic. This ensures that the testing strictly follows the principles of black box testing.

The application was tested in a locally hosted environment running on (http://localhost:300). Google Chrome was used as the test browser along with Chrome WebDriver. To automate camera-based features, fake media stream options were enabled, allowing the tests to run without requiring manual camera permissions. Explicit waits and time delays were used to handle dynamic elements and long-running processes such as pose detection and video analysis. Successful execution of each test was verified by observing screen transitions, confirmation messages, and expected system responses.

9.1 Test script structure

The testing logic was divided into multiple scripts to improve clarity and modularity:

- Exercise recording and saving script (Part 1)
- Exercise comparison script (Part 2)
- End-to-end exercise recording and comparison script

Each script represents a complete user journey and validates expected system behavior at every stage.

9.2 Script 1: Record exercise (Part 1)

This script automates the process of recording a new physiotherapy exercise and saving it as a reference template.

Test Flow:

- Navigate to home page
- Click on “Record New Exercise”
- Start recording session
- Upload exercise video file (untitled.mp4)
- Analyze and save exercise
- Wait for pose detection processing
- Save exercise template
- Handle confirmation pop-ups

Expected Output: The system successfully analyzes the uploaded video, detects body pose landmarks, and saves the exercise as a reusable template for future comparisons.

9.3 Script 2: Compare exercise with video (Part 2)

This script validates the comparison functionality between a saved exercise template and a test video or webcam input.

Test Flow:

- Navigate to home page
- Select a saved exercise (Knee Extension)
- Click on “Compare”
- Record exercise using webcam
- Upload test video file (Seated knee extension.mp4)

Expected Output: The system initiates comparison between the reference template and the test input, allowing posture analysis and feedback generation.

9.4 Script 3: End-to-end exercise recording and comparison

An integrated script was also executed to validate the complete workflow from recording a new exercise to comparing it with another session in a single run.

Purpose:

- Validate system stability across multiple dependent features
- Ensure smooth navigation between different modules
- Identify UI synchronization or timing issues

9.5 Test data that was used

We used the following data for testing:

Test Data	Description
untitled.mp4	Reference exercise video for template creation
Seated knee extension.mp4	Test video used for comparison

All test video files were stored locally in a dedicated test-videos directory and uploaded through the UI using Selenium.

9.6 Error Handling

The scripts include:

- Explicit waits (WebDriverWait) for UI elements
- Timeout handling for long-running pose detection
- Conditional pop-up handling (OK dialogs)
- Screenshot capture on critical failures

These mechanisms improved test reliability and reduced false negatives caused by UI delays or asynchronous processing.

9.7 Dependencies

The following dependencies were used for test execution:

- Selenium 4.16.0
- WebDriver Manager 4.0.1
- Google Chrome Browser

Chrome options were configured to use fake media streams to automate camera based interactions without manual permissions.

10. Test Results Summary

This is the summary of the test results:

Test Case ID	Description	Status
TC-01	Valid Login	Pass
TC-02	Invalid Login	Pass
TC-03	Exercise Selection	Pass
TC-04	Start Session	Pass
TC-05	End Session	Pass

Overall, all functional test cases executed successfully.

11. Defects detected

During testing, we did not observe any critical functional defects. Minor UI delays were observed during camera initialization, but they did not affect system functionality.

12. Limitations

Despite thorough checklist based testing, certain limitations exist:

- Selenium tests are limited to UI-level validation
- Accuracy of AI pose detection cannot be fully verified using black box testing
- Real-time webcam behavior is simulated using fake media streams
- Performance under heavy load was not evaluated

The above mentioned limitations are acceptable within the scope of this assignment.

13. Conclusion

This report demonstrates successful application of black box testing using Selenium on the Physiotherapy Guidance System. The testing process verified that major user facing functionalities work as expected. Automated testing improved reliability and reduced manual testing effort. Future work may include performance and usability testing to further enhance system quality.

14. Future Improvements

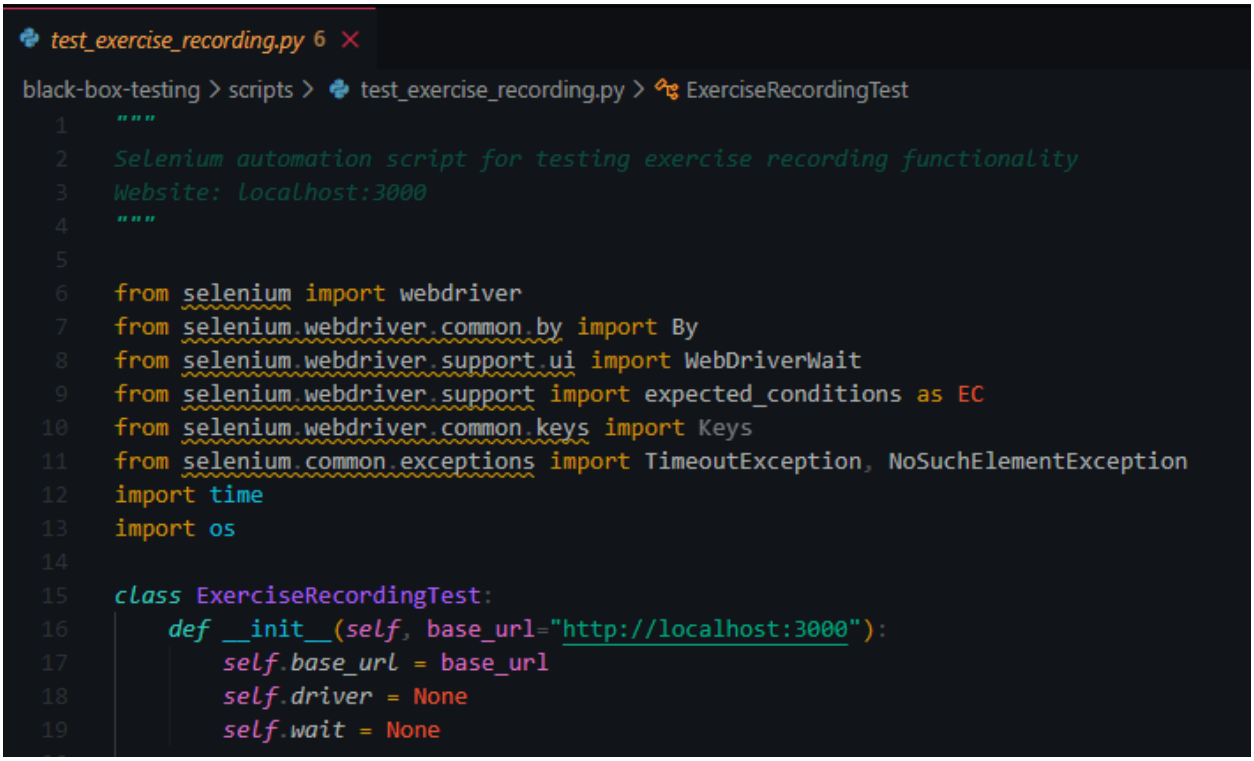
In the future we could enhance on the following:

- Integration of performance testing tools
- Cross-browser testing
- Automated regression testing
- Extended test coverage for doctor portal

15. Appendix

15.1 Appendix A: Selenium Script Screenshots

Testing exercise recording



Selenium test script and library imports

Initialization of class with base URL

```
        def setup_driver(self):
            options = webdriver.ChromeOptions()
            options.add_argument('--use-fake-ui-for-media-stream')
            options.add_argument('--use-fake-device-for-media-stream')
            self.driver = webdriver.Chrome(options=options)
            self.driver.maximize_window()
            self.wait = WebDriverWait(self.driver, 20)
```

Chrome WebDriver config with media stream for recording automation

```
    def click_record_new_exercise(self):
        try:
            record_button = self.wait.until(
                EC.element_to_be_clickable((By.XPATH,
                    "//*[contains(text(), 'Record New Exercise') or contains(text(), 'record new exercise') or contains(text(), 'New Exercise')]"))
            )
            record_button.click()
            time.sleep(2)
        except TimeoutException:
            try:
                record_button = self.driver.find_element(By.CSS_SELECTOR, "button[data-testid*='record'], button[class*='record']")
                record_button.click()
                time.sleep(2)
            except NoSuchElementException:
                raise
```

```
    def start_recording(self):
        try:
            start_button = self.wait.until(
                EC.element_to_be_clickable((By.XPATH,
                    "//*[contains(text(), 'Start recording') or contains(text(), 'Start Recording') or contains(text(), 'Start')]"))
            )
            start_button.click()
            time.sleep(2)
        except TimeoutException:
            try:
                start_button = self.driver.find_element(By.CSS_SELECTOR, "button[data-testid*='start'], button[class*='start']")
                start_button.click()
                time.sleep(2)
            except NoSuchElementException:
                raise
```

```
    def upload_video_file(self, filename="Untitled.mp4"):
        try:
            upload_button = self.wait.until(
                EC.element_to_be_clickable((By.XPATH,
                    "//*[contains(text(), 'Upload Video') or contains(text(), 'Upload File') or contains(text(), 'upload')]"))
            )
            upload_button.click()
            time.sleep(2)
            file_input = self.wait.until(
                EC.presence_of_element_located((By.CSS_SELECTOR, "input[type='file']"))
            )
            current_dir = os.path.dirname(os.path.abspath(__file__))
            test_videos_folder = os.path.join(current_dir, "test-videos")
            file_path = os.path.join(test_videos_folder, filename)
            if not os.path.exists(file_path):
                pass
            file_input.send_keys(file_path)
            time.sleep(3)
        except TimeoutException:
            raise
```

Automated user interaction for exercise recording and video upload

```
244
245     def run_complete_test(self):
246         try:
247             self.setup_driver()
248             self.navigate_to_home()
249             self.click_record_new_exercise()
250             self.start_recording()
251             self.upload_video_file("Untitled.mp4")
252             self.analyze_and_save_exercise()
253             self.wait_for_processing()
254             self.handle_popup()
255             self.click_play()
256             self.save_template_for_comparison()
257             self.click_ok_after_save()
258             self.go_to_home()
259             self.click_knee_extension_compare()
260             self.record_with_webcam()
261             self.test_with_video_file("Seated knee extension.mp4")
262         except Exception as e:
263             import traceback
264             traceback.print_exc()
265         finally:
266             if self.driver:
267                 self.driver.quit()
268
269 if __name__ == "__main__":
270     test = ExerciseRecordingTest(base_url="http://localhost:3000")
271     test.run_complete_test()
272
```

End-to-end execution flow of exercise recording test

15.2 Appendix B: Selenium Script Screenshots

Record and save

```
"""
Selenium automation script - Part 1: Record Exercise and Save Template
Website: localhost:3000

This script covers:
- Navigate to home
- Record New Exercise
- Start recording
- Upload Video File (Untitled.mp4)
- Analyze and save exercise
- Wait for processing
- Handle popup
- Click play
- Save template for comparison
"""

from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.keys import Keys
from selenium.common.exceptions import TimeoutException, NoSuchElementException
import time
import os

```

Purpose and library imports for exercise recording and template saving automation

```
class ExerciseRecordingPart1:
    def __init__(self, base_url="http://localhost:3000"):
        self.base_url = base_url
        self.driver = None
        self.wait = None

```

Initialization of class with base URL

```python
    def setup_driver(self):
        """Initialize the Chrome WebDriver"""
        options = webdriver.ChromeOptions()
        options.add_argument('--use-fake-ui-for-media-stream')
        options.add_argument('--use-fake-device-for-media-stream')

        self.driver = webdriver.Chrome(options=options)
        self.driver.maximize_window()
        self.wait = WebDriverWait(self.driver, 20)
```

Chrome WebDriver config with media stream

```python
def upload_video_file(self, filename="Untitled.mp4"):
    """Upload video file from Downloads folder"""
    print(f"Uploading video file: {filename}...")
    try:
        upload_button = self.wait.until(
            EC.element_to_be_clickable((By.XPATH,
                "//*[contains(text(), 'Upload Video') or contains(text(), 'Upload File') or contains(text(), 'upload')]"))
        )
        upload_button.click()
        time.sleep(2)

        file_input = self.wait.until(
            EC.presence_of_element_located((By.CSS_SELECTOR, "input[type='file']"))
        )

        current_dir = os.path.dirname(os.path.abspath(__file__))
        test_videos_folder = os.path.join(current_dir, "test-videos")
        file_path = os.path.join(test_videos_folder, filename)

        if not os.path.exists(file_path):
            print(f"Warning: File {file_path} does not exist in test-videos folder.")
            print(f"Please ensure {filename} is in your test-videos folder: {test_videos_folder}")
        else:
            print(f"Found file at: {file_path}")

        file_input.send_keys(file_path)
        time.sleep(3)
        print(f"File {filename} uploaded successfully from test-videos folder")

    except TimeoutException:
        print("Could not find upload button or file input")
        raise
```

```python
    def analyze_and_save_exercise(self):
        """Click on 'Analyze and save exercise' button and wait for pose detection"""
        print("Clicking 'Analyze and save exercise'...")
        try:
            analyze_button = None

            try:
                analyze_button = self.wait.until(
                    EC.element_to_be_clickable((By.XPATH,
                        "//button[contains(text(), 'Analyze and save exercise') or "
                        "contains(text(), 'Analyze and Save Exercise') or "
                        "contains(text(), 'Analyze') or "
                        "contains(text(), 'analyze')]"))
                )
            except TimeoutException:
                analyze_button = self.driver.find_element(By.XPATH,
                    "//*[contains(@class, 'analyze') or contains(@id, 'analyze') or "
                    "contains(text(), 'Save exercise')]")

            if analyze_button:
                analyze_button.click()
                print("'Analyze and save exercise' button clicked")
            else:
                raise Exception("Could not find 'Analyze and save exercise' button")

            print("Waiting for 'Detecting pose' text...")
            self.wait.until(
                EC.presence_of_element_located((By.XPATH,
                    "//*[contains(text(), 'Detecting pose') or contains(text(), 'detecting pose') or "
                    "contains(text(), 'Detecting') or contains(text(), 'Processing')]"))
            )
            print("'Detecting pose' text found - processing started")

            # Check for popup error and click OK if it appears
            try:
                print("Checking for popup error during analysis...")
                ok_button = WebDriverWait(self.driver, 5).until(
                    EC.element_to_be_clickable((By.XPATH,
                        "//button[contains(text(), 'OK') or contains(text(), 'Ok') or contains(text(), 'ok')]"))
                )
```

```python
                ok_button.click()
                print("Popup error OK button clicked")
                time.sleep(2)
            except TimeoutException:
                print("No popup error found during analysis, continuing...")
            except Exception as e:
                print(f"Error checking for popup: {str(e)}, continuing anyway...")

        except TimeoutException as te:
            print(f"Timeout in analyze_and_save_exercise: {str(te)}")
            try:
                self.driver.save_screenshot("analyze_timeout.png")
                print("Screenshot saved as analyze_timeout.png")
            except:
                pass
            raise
        except Exception as e:
            print(f"Error in analyze_and_save_exercise: {str(e)}")
            try:
                self.driver.save_screenshot("analyze_error.png")
                print("Screenshot saved as analyze_error.png")
            except:
                pass
            raise
```

Uploading exercise video and executing analyze and save workflow

```
    def go_to_home(self):
        """Navigate back to home"""
        print("Navigating to home...")
        try:
            # Try to find home button/link
            home_button = self.wait.until(
                EC.element_to_be_clickable((By.XPATH,
                    "//*[contains(text(), 'Home') or contains(text(), 'home')]"))
            )
            home_button.click()
            time.sleep(2)

        except TimeoutException:
            # Alternative: navigate directly to home URL
            print("Home button not found, navigating to base URL...")
            self.driver.get(self.base_url)
            time.sleep(2)

    def click_play(self):
        """Click on play button - DEPRECATED"""
        print("Skipping play button...")
        pass
```

```
def save_template_for_comparison(self):
    """Save template for comparison, then click OK on popup and go to home"""
    print("Saving template for comparison...")
    try:
        save_template_button = self.wait.until(
            EC.element_to_be_clickable((By.XPATH,
                "//*[contains(text(), 'Save Template') or contains(text(), 'Save') or contains(text(), 'template')]"))
        )
        save_template_button.click()
        time.sleep(2)

        # Wait for and click OK on the "template has been saved" popup
        print("Waiting for 'template saved' popup...")
        try:
            ok_button = WebDriverWait(self.driver, 10).until(
                EC.element_to_be_clickable((By.XPATH,
                    "//button[contains(text(), 'OK') or contains(text(), 'Ok') or contains(text(), 'ok')]"))
            )
            ok_button.click()
            print("Clicked OK on 'template saved' popup")
            time.sleep(2)
        except TimeoutException:
            print("No popup appeared after saving template, continuing...")

        # Navigate back to home
        print("Navigating to home after saving template...")
        try:
            home_button = self.wait.until(
                EC.element_to_be_clickable((By.XPATH,
                    "//*[contains(text(), 'Home') or contains(text(), 'home')]"))
            )
            home_button.click()
            print("Clicked Home button")
            time.sleep(2)
        except TimeoutException:
            print("Home button not found, navigating to base URL...")
            self.driver.get(self.base_url)
            time.sleep(2)

    except TimeoutException:
        print("Could not find 'Save template' button")
        raise
```

Saving exercise template and returning to home page

15.3 Appendix C: Selenium Script Screenshots

Compare exercise

```
"""
Selenium automation script - Part 2: Compare Exercise with Video
Website: localhost:3000

This script covers:
- Click OK after saving template
- Go to home
- Click on knee extension and compare
- Record with webcam
- Test with video file (Seated knee extension.mp4)
"""

from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.keys import Keys
from selenium.common.exceptions import TimeoutException
import time
import os

```

Purpose and library imports for comparing exercises videos automation

```
class ExerciseComparisonPart2:
    def __init__(self, base_url="http://localhost:3000"):
        self.base_url = base_url
        self.driver = None
        self.wait = None

```

Initialization of class with base URL

```python
    def setup_driver(self):
        """Initialize the Chrome WebDriver"""
        options = webdriver.ChromeOptions()
        options.add_argument('--use-fake-ui-for-media-stream')
        options.add_argument('--use-fake-device-for-media-stream')

        self.driver = webdriver.Chrome(options=options)
        self.driver.maximize_window()
        self.wait = WebDriverWait(self.driver, 20)
```

Chrome WebDriver config with media stream

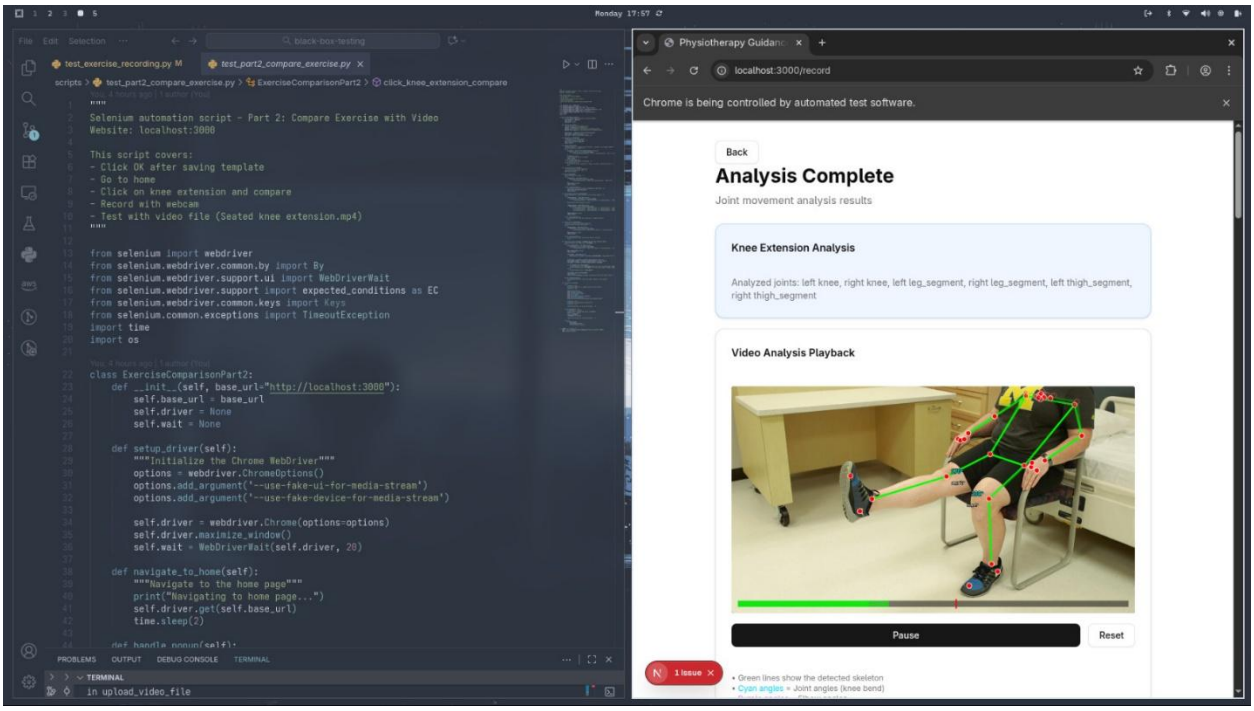
```python
def click_knee_extension_compare(self):
    print("Looking for 'knee extension' and clicking compare...")
    try:
        knee_extension = self.wait.until(
            EC.presence_of_element_located((By.XPATH,
                "//*[contains(text(), 'knee extension') or contains(text(), 'Knee Extension')]"))
        )
        print("Found knee extension entry")

        compare_button = self.wait.until(
            EC.element_to_be_clickable((By.XPATH,
                "//*[contains(text(), 'knee extension') or contains(text(), 'Knee Extension')]//*[contains(text(), 'Compare')] | "
                "//*[contains(text(), 'knee extension') or contains(text(), 'Knee Extension')]//following::*[contains(text(), 'Compare')][1] | "
                "//*[contains(text(), 'knee extension') or contains(text(), 'Knee Extension')]//ancestor::*//button[contains(text(), 'Compare')]"))
        )
        compare_button.click()
        time.sleep(2)

    except TimeoutException:
        print("Could not find knee extension or compare button")
        raise
```

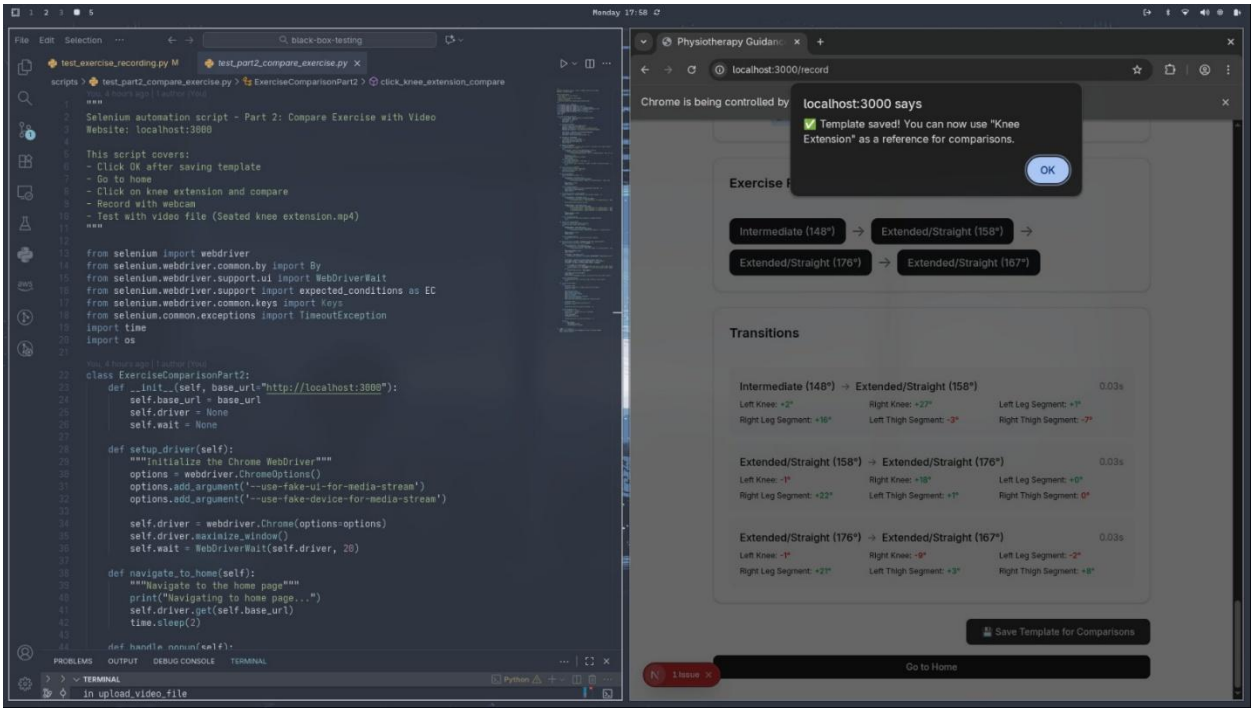
Comparing exercises videos

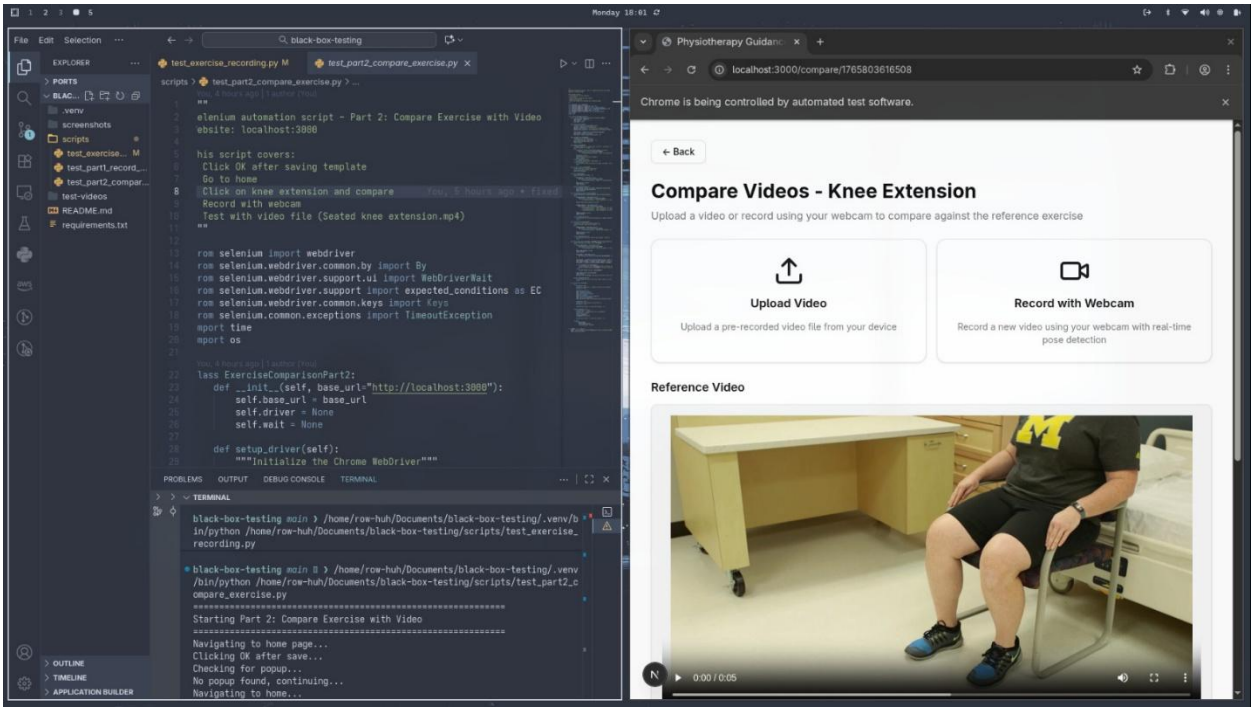
```
    def run_part2_test(self):
        try:
            print("=" * 60)
            print("Starting Part 2: Compare Exercise with Video")
            print("=" * 60)

            self.setup_driver()
            self.navigate_to_home()
            self.click_ok_after_save()
            self.go_to_home()
            self.click_knee_extension_compare()
            self.record_with_webcam()
            self.test_with_video_file("Seated knee extension.mp4")

            print("=" * 60)
            print("Part 2 completed successfully!")
            print("=" * 60)

            input("Press Enter to close the browser...")

        except Exception as e:
            print(f"\n{'=' * 60}")
            print(f"Part 2 failed with error: {str(e)}")
            print(f"{'=' * 60}")
            import traceback
            traceback.print_exc()

            input("Press Enter to close the browser...")

        finally:
            if self.driver:
                self.driver.quit()
                print("Browser closed")


if __name__ == "__main__":
    test = ExerciseComparisonPart2(base_url="http://localhost:3000")
    test.run_part2_test()
```

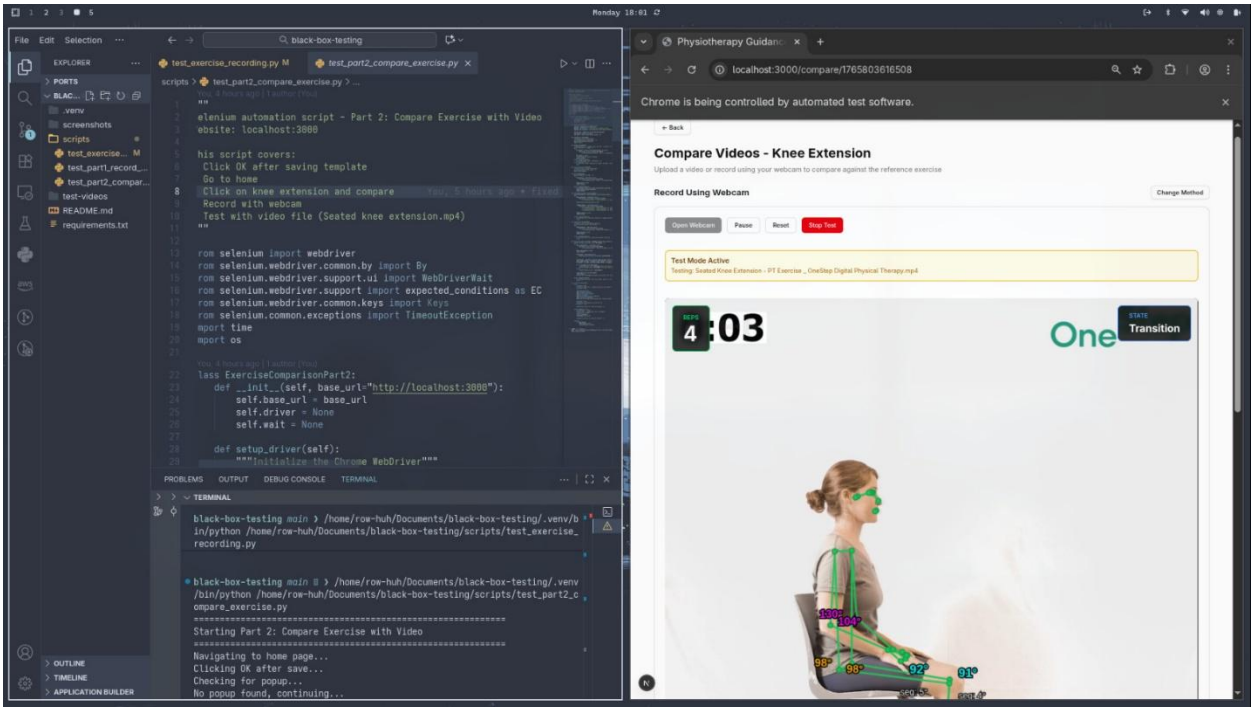
End-to-end execution flow of comparing exercises

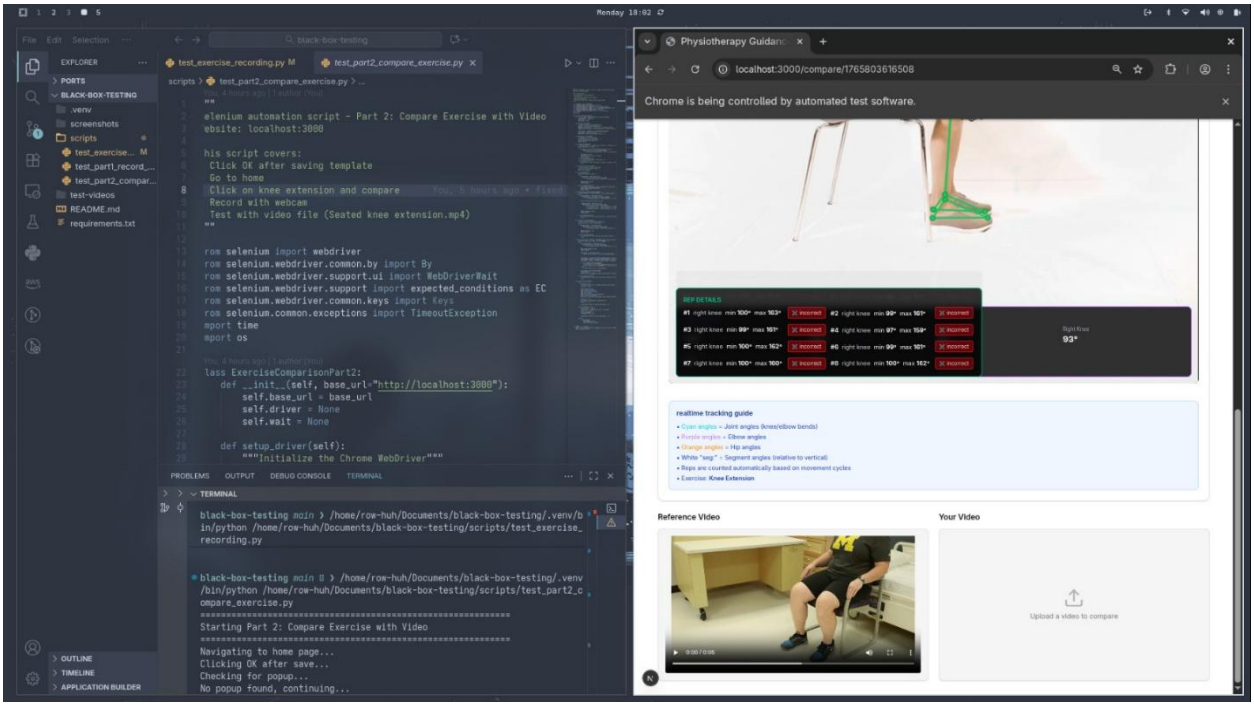
15.4 Appendix D: Selenium test execution screenshots






Chrome is being controlled by automated test software.
Compare Videos - Knee Extension
Upload a video or record using your webcam to compare against the reference exercise
Upload Video
Upload a pre-recorded video file from your device
Record with Webcam
Record a new video using your webcam with real-time pose detection
Reference Video
Starting Part 2: Compare Exercise with Video
Navigating to home page...
Clicking OK after save...
Checking for popup...
No popup found, continuing...


Chrome is being controlled by automated test software.
Compare Videos - Knee Extension
Record Using Webcam
Starting Part 2: Compare Exercise with Video
Navigating to home page...
Clicking OK after save...
Checking for popup...
No popup found, continuing...


Physiotherapy Guidanc
localhost:3000/compare/1765803616508
Chrome is being controlled by automated test software.
realtime tracking guide
Reference Video
Your Video
Upload a video to compare
test_exercise_recording.py M
test_part2_compare_exercise.py
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
Starting Part 2: Compare Exercise with Video
Navigating to home page...
Clicking OK after save...
Checking for popup...
No popup found, continuing...