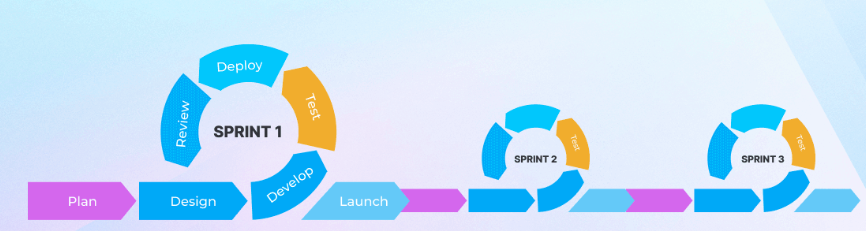
**Aim:** Generate regression test case suite for the software.

**Description:** Regression testing ensures that newly introduced code changes or updates to the software do not adversely affect existing functionality. The test suite will verify that the core functionalities of the software work as expected after updates. This suite will be implemented using Selenium WebDriver, a popular automation testing framework for web applications.

[Selenium](https://www.geeksforgeeks.org/browser-automation-using-selenium/)

* **Open Source**: Selenium is an open-source tool, making it freely available and accessible for developers and testers.
* **Browser Compatibility**: Supports multiple browsers, including Chrome, Firefox, Safari, and Edge, ensuring tests can be run across different environments.
* **Programming Language Support**: Allows writing tests in various programming languages such as Java, Python, C#, Ruby, and JavaScript, providing flexibility for testers.
* **Cross-Platform**: Capable of running on different operating systems, including Windows, macOS, and Linux, which enhances the tool’s portability.
* **Web Application Testing**: Primarily designed for automating web applications, making it ideal for regression testing of web-based systems.
* **Extensive Community Support**: Boasts a large and active community, offering a wealth of resources, plugins, and extensions to aid in test automation.
* **Integration Capabilities**: Integrates well with other tools such as Jenkins for continuous integration and continuous deployment (CI/CD), facilitating automated regression testing in development pipelines.



**When can we perform Regression Testing?**

Here are the scenarios when you can apply the regression testing process.

**New functionality is added to the application:** This happens when new features or modules are created in an app or a website. The regression is performed to see if the existing features are working as usual with the introduction of the new feature.

**In case of change requirement:**When any significant change occurs in the system, regression testing is used. This test is done to check if these shifts have affected features that were there.

**After a defect is fixed:** The developers perform regression testing after fixing a bug in any functionality. This is done to determine if the changes made while fixing the bug have affected other related existing features.

**Once the performance issue is fixed:** After fixing any performance issues, the regression testing process is triggered to see if it has affected other existing functional tests.

**While integrating with a new external system:** End-to-end regression testing process is required whenever the product integrates with a new external system.

**Advantages of Regression Testing**

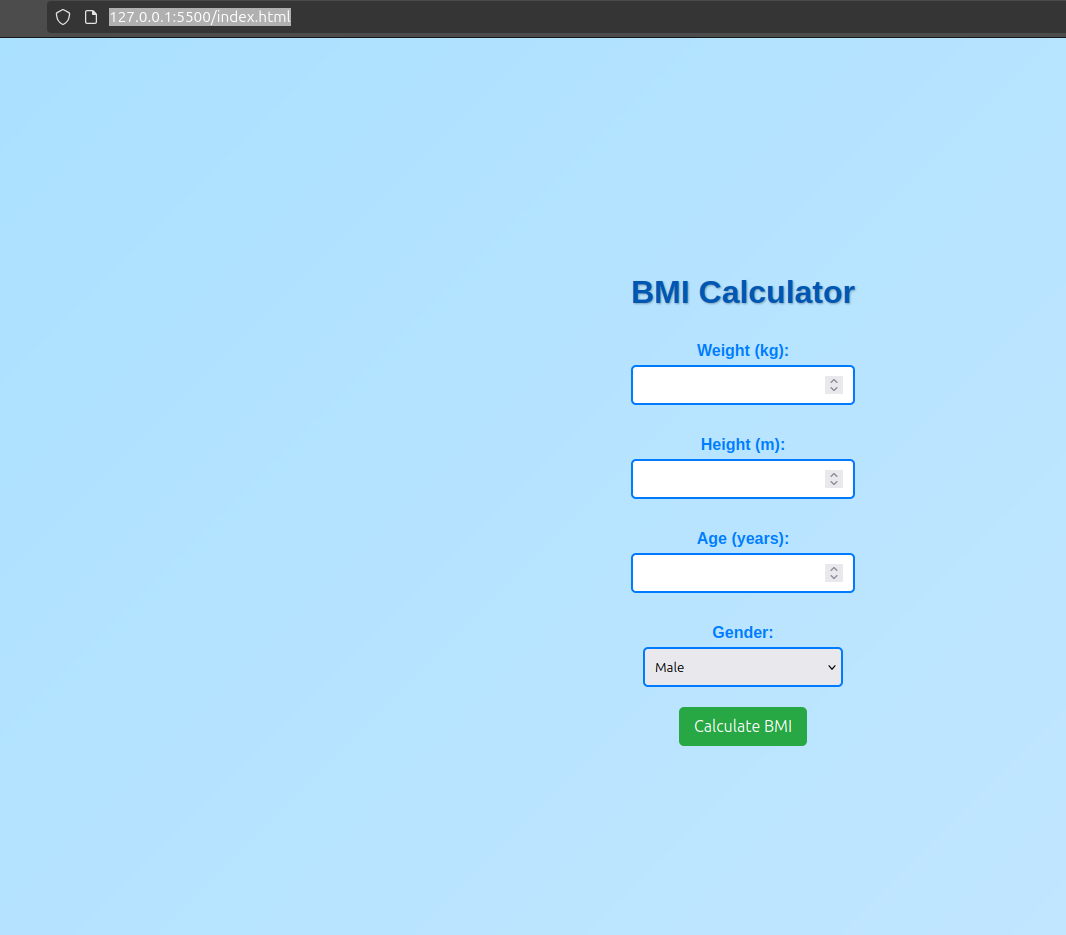
* Automated unit testing
* Comprehensive test coverage
* System integration
* Faster test execution completion
* Improved developer productivity
* Parallel testing
* Reduced costs
* Regression testing improves product quality
* Reusability
* Scalability
* Time efficiency

**Implementation:**

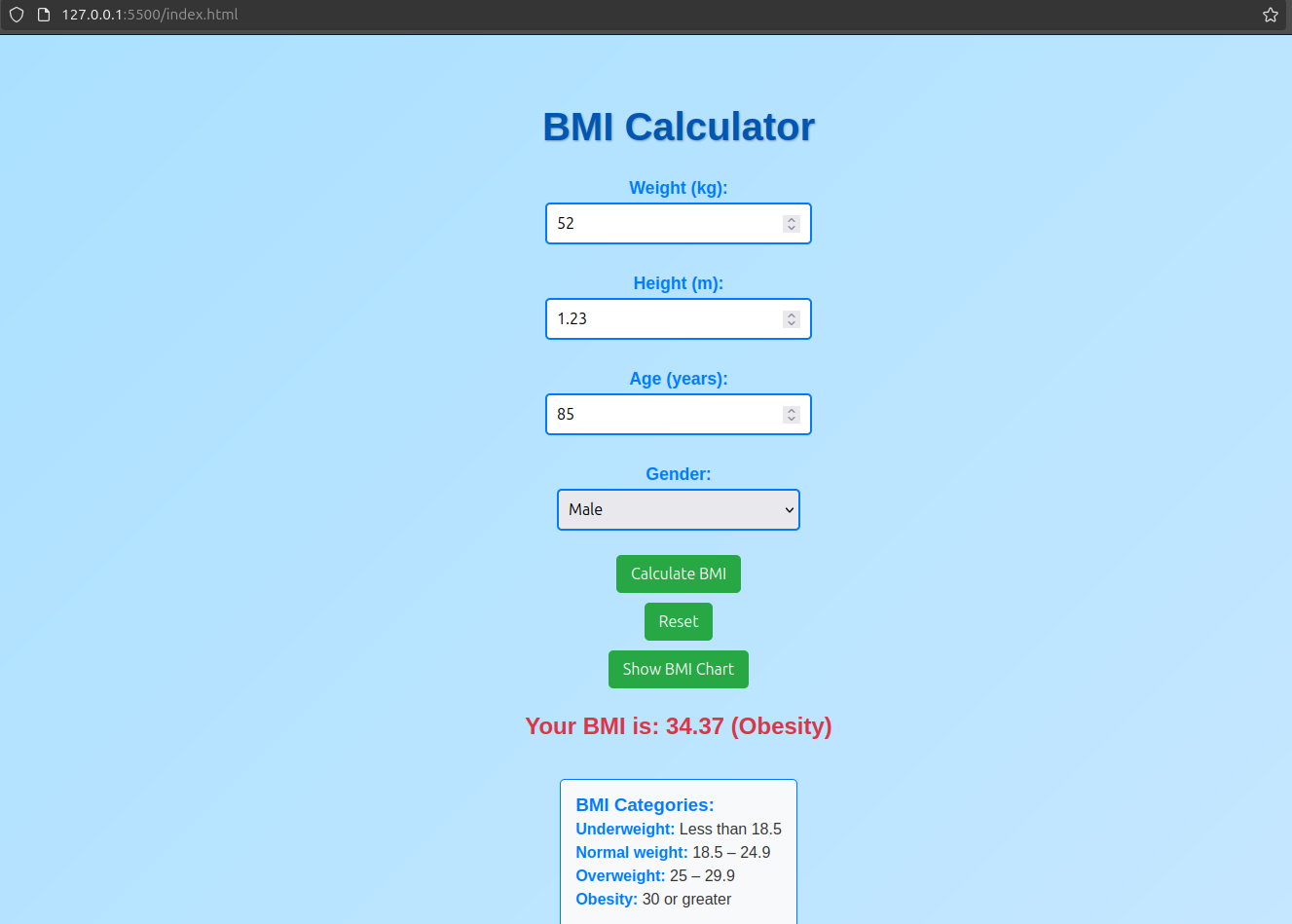
Step 1: Test Environment Setup

1. Prerequisites:
   * Install Selenium WebDriver and its necessary browser drivers (e.g., ChromeDriver, GeckoDriver).
   * Set up the test environment (IDE, project structure). Use tools like TestNG, JUnit, or pytest for test organization.
2. Tools:
   * Programming Language: Java/Python/C#.
   * Browser: Chrome, Firefox, Edge.

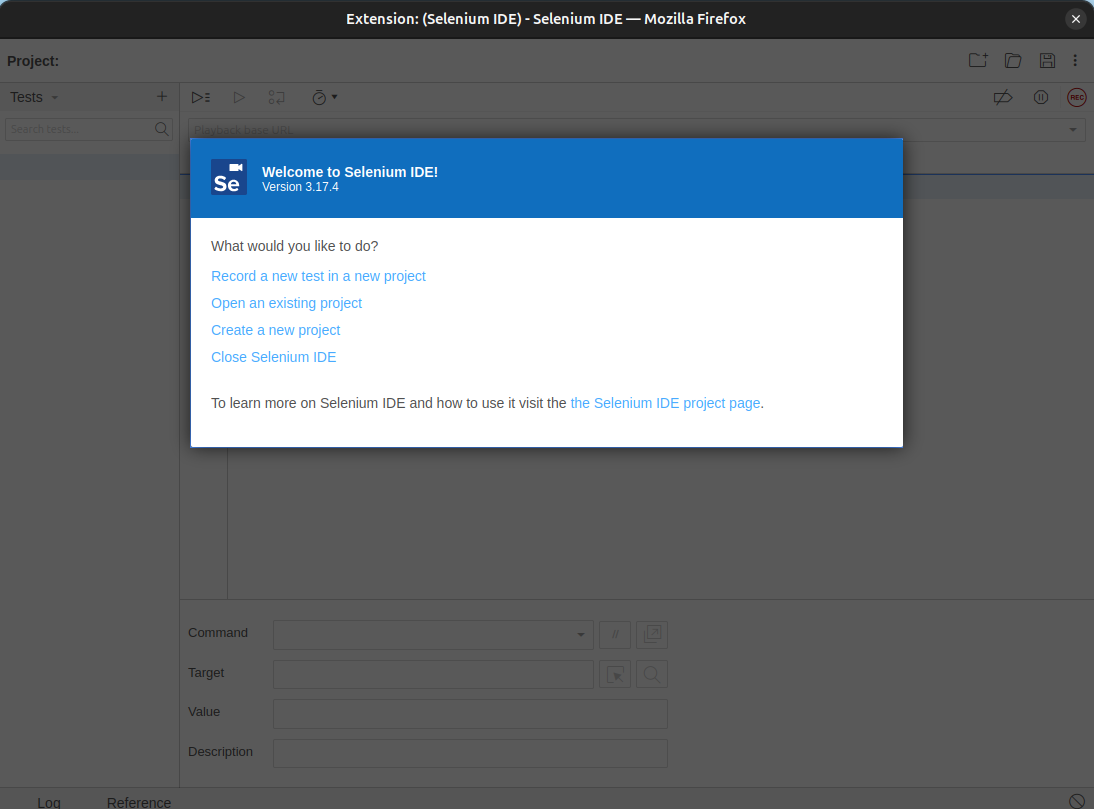
Step 2: Define Test Cases, Develop test cases for each module.

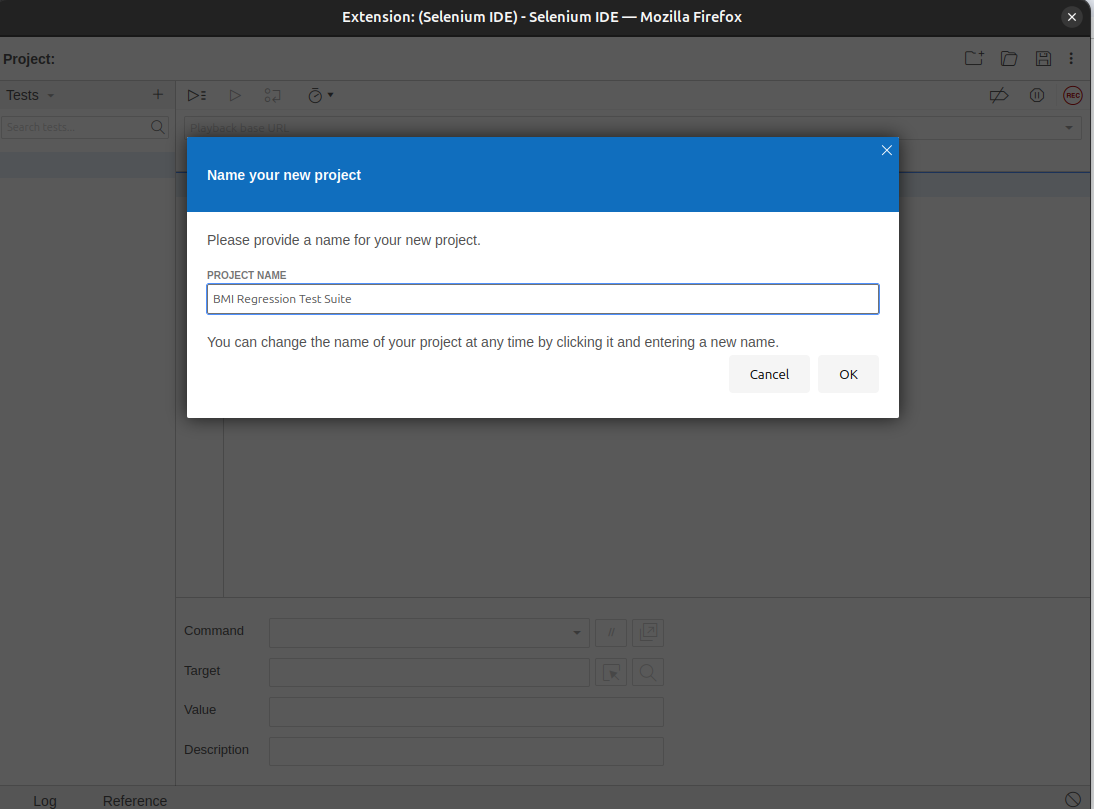


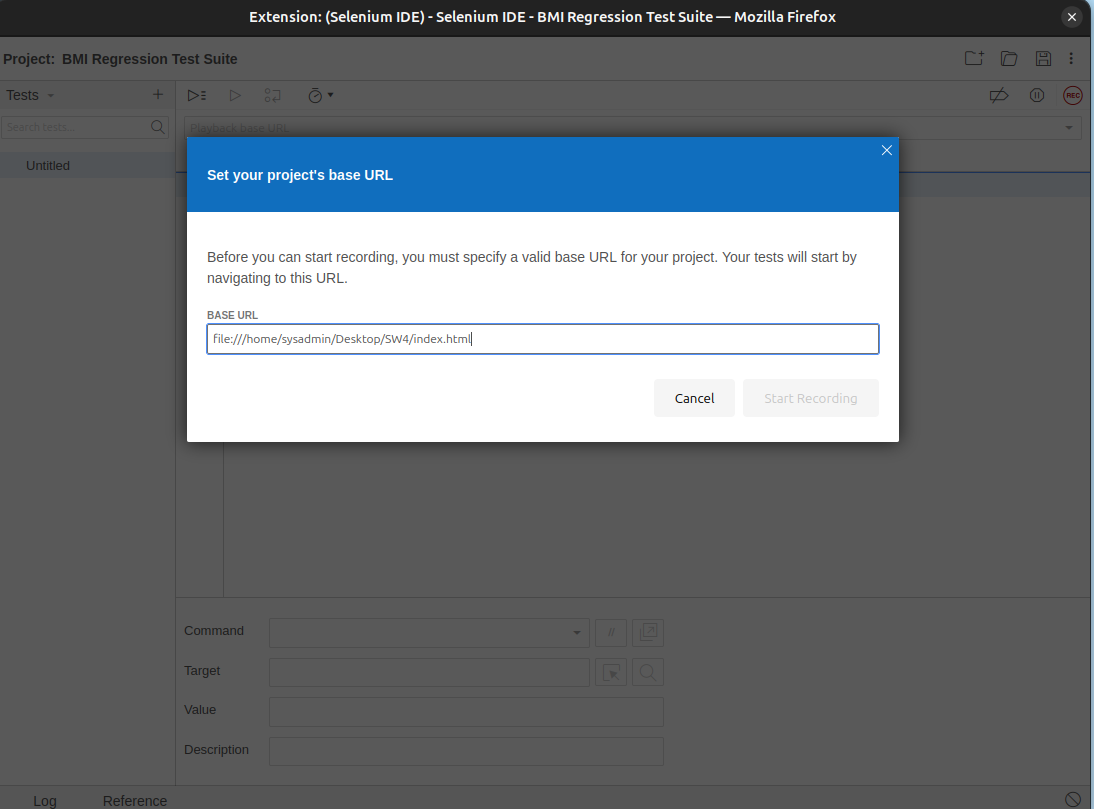
Without Reset button functionality

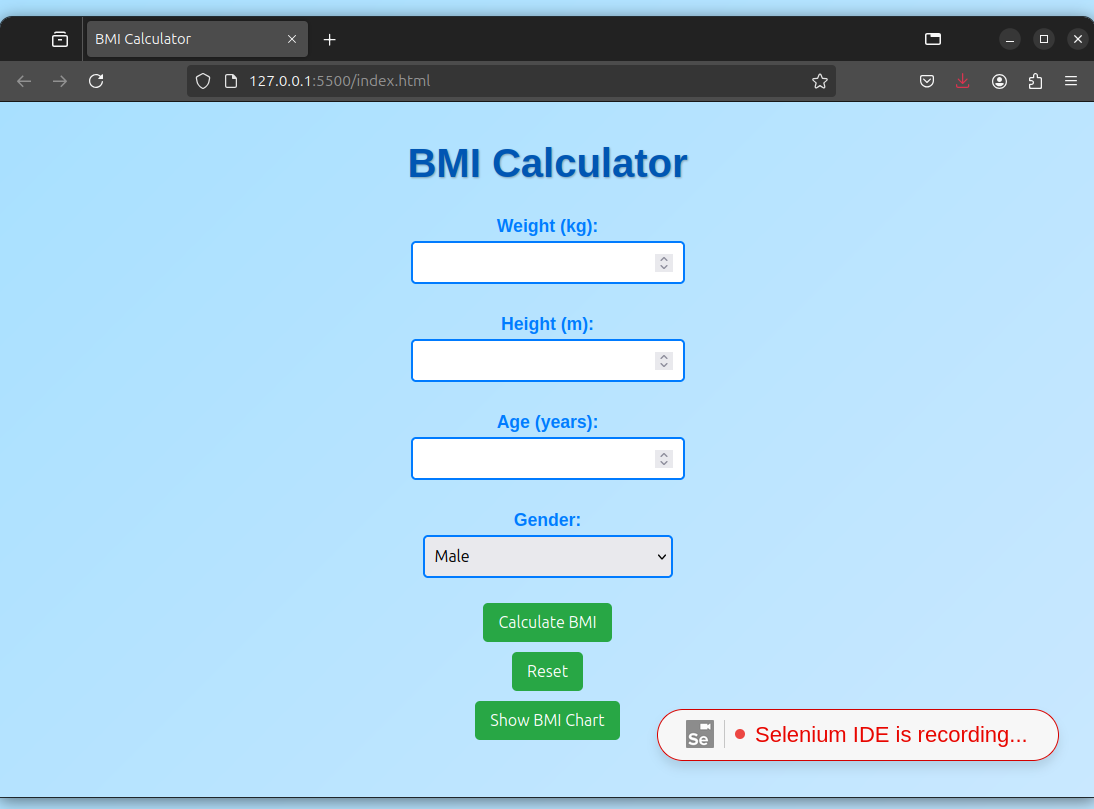


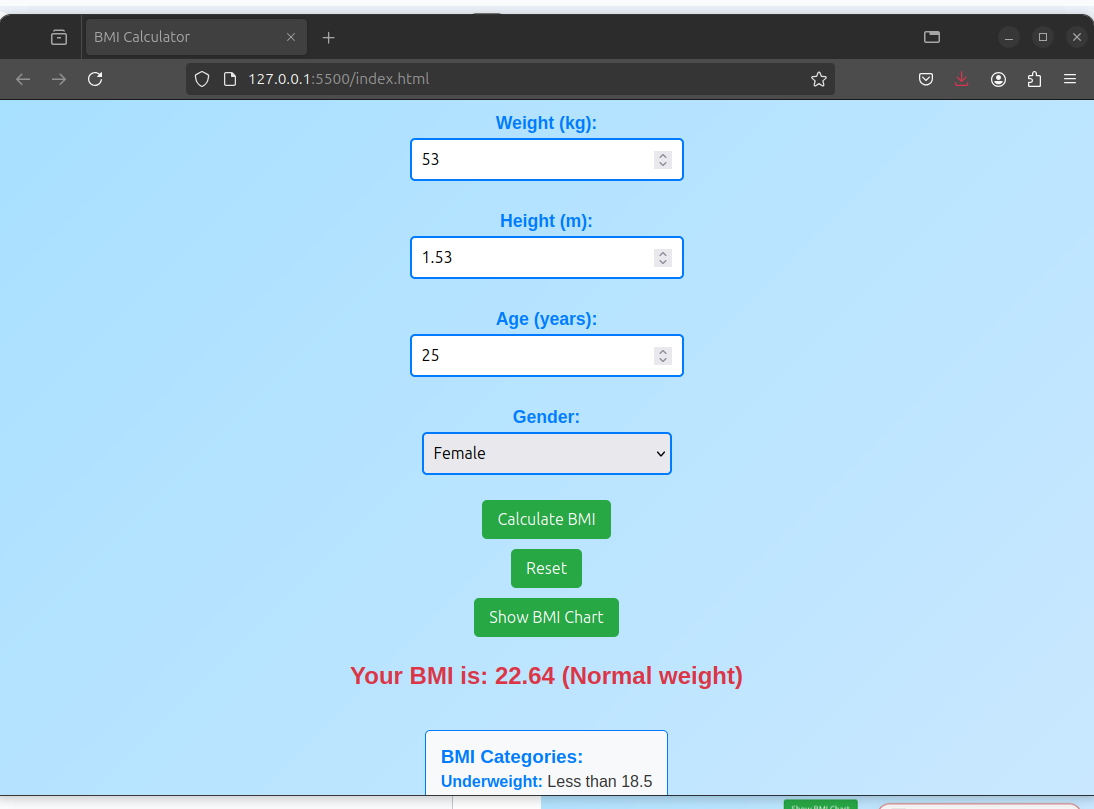
With reset button functionality

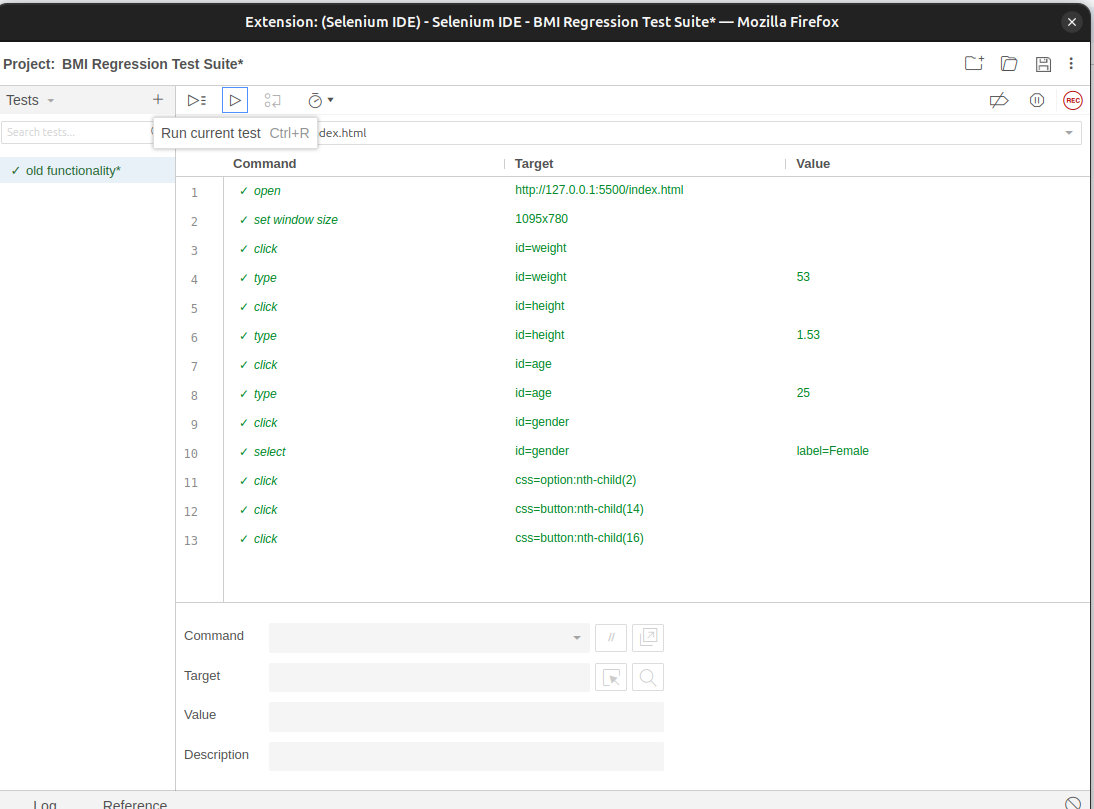


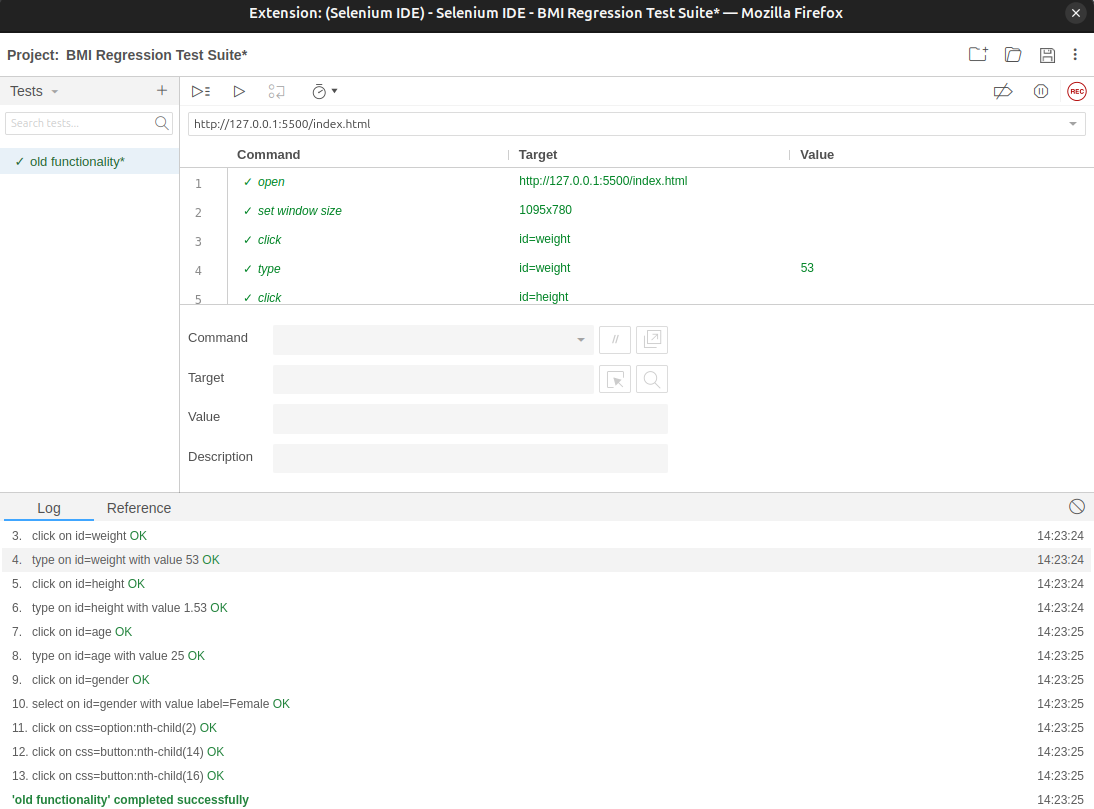




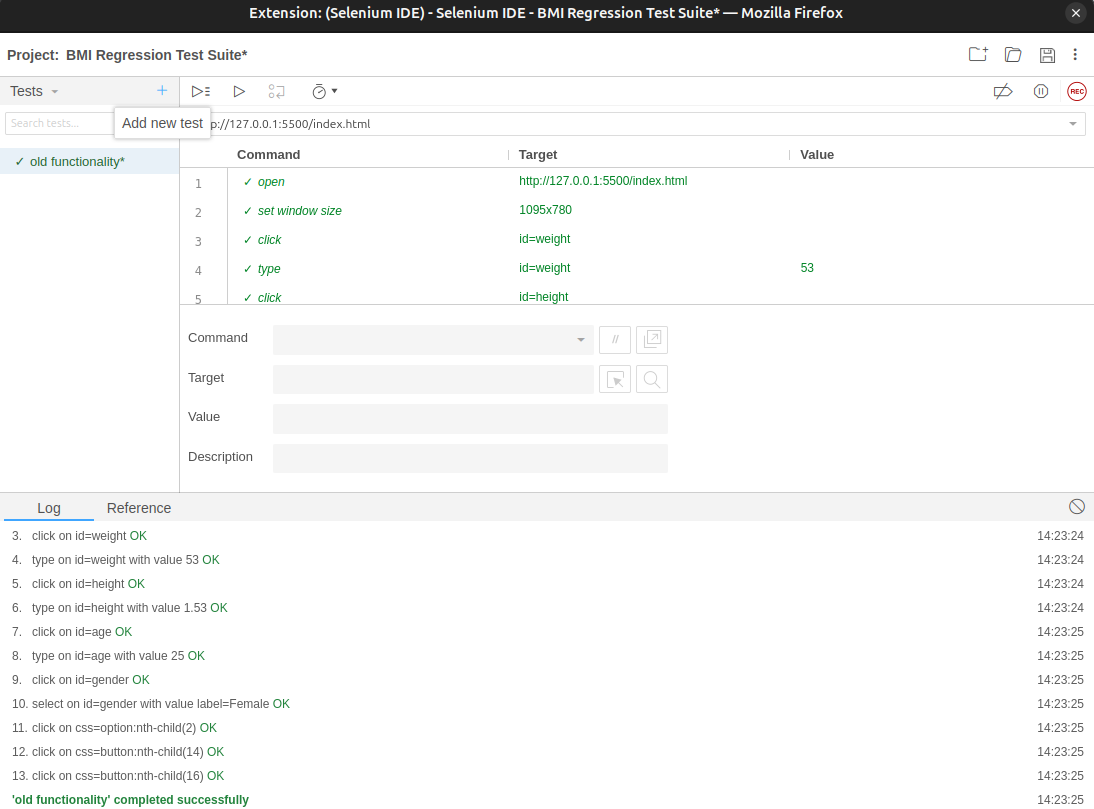




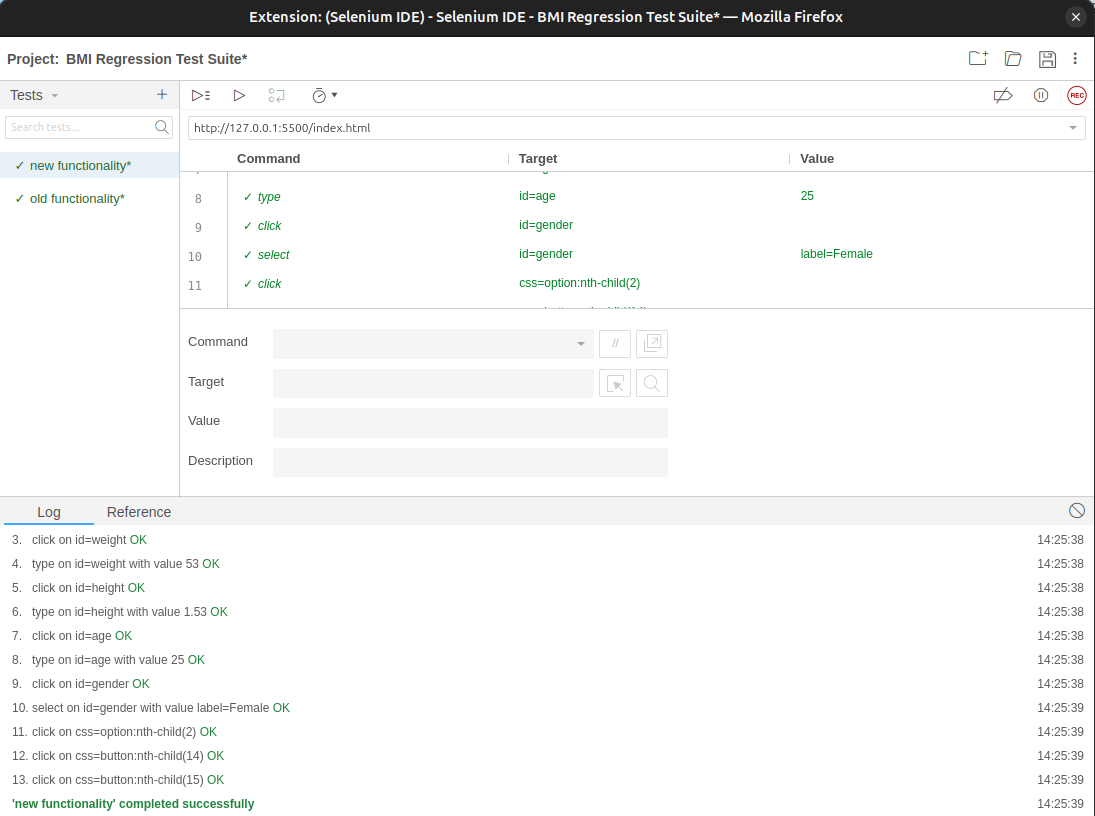




Click on add new test case



Run new test suite



**Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>BMI Calculator</title>

<style>

/\* Enhanced styles here \*/

body {

font-family: 'Arial', sans-serif;

background: linear-gradient(135deg, #a8e0ff, #c9e8ff);

display: flex;

flex-direction: column;

align-items: center;

justify-content: center;

height: 100vh;

margin: 0;

color: #333;

}

h1 {

color: #0056b3;

text-shadow: 1px 1px 2px rgba(0, 0, 0, 0.3);

font-size: 2.5em;

margin-bottom: 20px;

}

label {

margin: 10px 0 5px;

font-weight: bold;

color: #007bff;

font-size: 1.1em;

}

input[type="number"], select {

padding: 10px;

width: 250px;

border: 2px solid #007bff;

border-radius: 5px;

margin-bottom: 20px;

transition: border-color 0.3s, box-shadow 0.3s;

font-size: 1em;

}

input[type="number"]:focus, select:focus {

border-color: #0056b3;

outline: none;

box-shadow: 0 0 5px rgba(0, 86, 179, 0.5);

}

button {

padding: 10px 15px;

background-color: #28a745;

color: white;

border: none;

border-radius: 5px;

cursor: pointer;

font-size: 1em;

margin: 5px;

transition: background-color 0.3s, transform 0.3s;

}

button:hover {

background-color: #218838;

transform: scale(1.05);

}

h2 {

margin-top: 20px;

color: #dc3545; /\* Red for result text \*/

font-size: 1.5em;

}

.bmi-chart {

margin-top: 20px;

text-align: left;

color: #333;

border: 1px solid #007bff;

border-radius: 5px;

padding: 15px;

background-color: #f8f9fa;

display: none; /\* Hidden by default \*/

}

.bmi-chart h3 {

margin: 0;

color: #007bff;

}

.bmi-chart p {

margin: 5px 0;

}

.bmi-chart span {

font-weight: bold;

color: #007bff;

}

</style>

</head>

<body>

<h1>BMI Calculator</h1>

<label for="weight">Weight (kg):</label>

<input type="number" id="weight" required>

<br>

<label for="height">Height (m):</label>

<input type="number" id="height" step="0.01" required>

<br>

<label for="age">Age (years):</label>

<input type="number" id="age" required>

<br>

<label for="gender">Gender:</label>

<select id="gender">

<option value="male">Male</option>

<option value="female">Female</option>

</select>

<br>

<button onclick="calculateBMI()">Calculate BMI</button>

<button onclick="resetForm()">Reset</button>

<button onclick="showBMIChart()">Show BMI Chart</button>

<h2 id="result"></h2>

<div class="bmi-chart" id="bmi-chart" style="display: none;">

<h3>BMI Categories:</h3>

<p><span>Underweight:</span> Less than 18.5</p>

<p><span>Normal weight:</span> 18.5 – 24.9</p>

<p><span>Overweight:</span> 25 – 29.9</p>

<p><span>Obesity:</span> 30 or greater</p>

</div>

<script>

function calculateBMI() {

const weight = parseFloat(document.getElementById('weight').value);

const height = parseFloat(document.getElementById('height').value);

const age = parseInt(document.getElementById('age').value);

const gender = document.getElementById('gender').value;

if (weight > 0 && height > 0 && age > 0) {

const bmi = weight / (height \* height);

const category = getBMICategory(bmi);

document.getElementById('result').innerText =

`Your BMI is: ${bmi.toFixed(2)} (${category})`;

document.getElementById('bmi-chart').style.display = 'block';

} else {

document.getElementById('result').innerText =

'Please enter valid weight, height, and age.';

}

}

function getBMICategory(bmi) {

if (bmi < 18.5) return 'Underweight';

else if (bmi >= 18.5 && bmi < 24.9) return 'Normal weight';

else if (bmi >= 25 && bmi < 29.9) return 'Overweight';

else return 'Obesity';

}

function resetForm() {

document.getElementById('weight').value = '';

document.getElementById('height').value = '';

document.getElementById('age').value = '';

document.getElementById('gender').value = 'male';

document.getElementById('result').innerText = '';

document.getElementById('bmi-chart').style.display = 'none';

}

function showBMIChart() {

document.getElementById('bmi-chart').style.display = 'block';

}

</script>

</body>

</html>

**test\_bmi\_calculator.py**

from selenium import webdriver # type: ignore

from selenium.webdriver.firefox.service import Service # type: ignore

from selenium.webdriver.common.by import By # type: ignore

import time

# Set up the WebDriver (replace with the path to your driver if needed)

service = Service(executable\_path='/home/sysadmin/Downloads/geckodriver') # Change this to your actual path

driver = webdriver.Firefox()

profile\_path = '/home/sysadmin/snap/firefox/common/.mozilla/firefoz/z59yv691.GeetaSeshapalli'

#driver = webdriver.Firefox(executable\_path='/home/sysadmin/Downloads/geckodriver') # Use 'webdriver.Firefox()' for Firefox

try:

# Step 1: Open the HTML file

driver.get("/home/sysadmin/Desktop/SW4/") # Update with the correct path

# Step 2: Find the weight, height, age, and gender input fields

weight\_input = driver.find\_element(By.ID, "weight")

height\_input = driver.find\_element(By.ID, "height")

age\_input = driver.find\_element(By.ID, "age")

gender\_select = driver.find\_element(By.ID, "gender")

# Step 3: Enter weight, height, age, and select gender

weight\_input.send\_keys("70") # Example weight in kg

height\_input.send\_keys("1.75") # Example height in meters

age\_input.send\_keys("25") # Example age

gender\_select.send\_keys("Male") # Select gender

# Step 4: Click the Calculate button

calculate\_button = driver.find\_element(By.XPATH, "//button[text()='Calculate BMI']")

calculate\_button.click()

# Step 5: Wait for the result to appear

time.sleep(1) # Wait for a moment to ensure the result is displayed

# Step 6: Get the result text

result = driver.find\_element(By.ID, "result").text

print(result) # Output the result to console

# Step 7: Optionally, check if the BMI category is correct

if "Obesity" in result or "Overweight" in result:

print("BMI category is correctly identified.")

else:

print("Check the BMI category.")

finally:

# Close the browser after testing

driver.quit()

**test\_defaultSuite.py**

# Generated by Selenium IDE

import pytest

import time

import json

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.common.action\_chains import ActionChains

from selenium.webdriver.support import expected\_conditions

from selenium.webdriver.support.wait import WebDriverWait

from selenium.webdriver.common.keys import Keys

from selenium.webdriver.common.desired\_capabilities import DesiredCapabilities

class TestDefaultSuite():

def setup\_method(self, method):

self.driver = webdriver.Firefox()

self.vars = {}

def teardown\_method(self, method):

self.driver.quit()

def test\_testcase1(self):

self.driver.get("http://127.0.0.1:5500/index.html")

self.driver.set\_window\_size(550, 691)

self.driver.find\_element(By.ID, "weight").click()

self.driver.find\_element(By.ID, "weight").send\_keys("57")

self.driver.find\_element(By.ID, "height").click()

self.driver.find\_element(By.ID, "height").send\_keys("1.53")

self.driver.find\_element(By.ID, "age").click()

self.driver.find\_element(By.ID, "age").send\_keys("25")

self.driver.find\_element(By.ID, "gender").click()

dropdown = self.driver.find\_element(By.ID, "gender")

dropdown.find\_element(By.XPATH, "//option[. = 'Female']").click()

self.driver.find\_element(By.CSS\_SELECTOR, "option:nth-child(2)").click()

self.driver.find\_element(By.CSS\_SELECTOR, "button").click()

**Conclusion:** By implementing a regression test suite with Selenium, the software's reliability and stability can be ensured post-updates. Automating these test cases reduces manual effort, ensures consistency, and provides quicker feedback during the development lifecycle. Regular execution and maintenance of the suite help to catch defects early and ensure seamless functionality.