# TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| S. No. | **TOPIC** | Page No. |
| 1 | INTRODUCTION OF THE PROJECT | 2 |
| 2 | PROJECT SNAPSHOTS | 3 |
| 3 | SELENIUM AUTOMATED TESTING | 12 |
| 4 | UNIT TESTING | 18 |
| 5 | JIRA MANUAL TESTING | 20 |

1. **INTRODUCTION OF THE PROJECT:**

PakCyberGuard is an advanced AI-powered cybercrime reporting and awareness platform built to address the growing digital threats in Pakistan. Many citizens fall victim to online scams, harassment, and identity theft but lack knowledge on how or where to report such crimes. PakCyberGuard bridges this gap by offering a user-friendly platform where people can report incidents from home and get real-time support through an integrated AI chatbot. The chatbot provides legal guidance in both Urdu and English, particularly on sensitive issues such as cyberbullying, impersonation, and digital fraud. The system is especially beneficial for individuals who cannot afford legal support or are unaware of their cyber rights, promoting awareness and empowerment across digital spaces.

**PROJECT PURPOSE:**

The core purpose of PakCyberGuard is to offer a centralized, digital platform where users can report cybercrimes, access basic legal help, and read awareness content to avoid future threats. In Pakistan, victims often don’t know how to initiate formal complaints. This platform guides users through a structured, multi-step reporting form and simultaneously allows interaction with an AI chatbot that answers common legal questions. It also provides access to NGOs and pro bono lawyers for extended support beyond digital means. The goal is to break the psychological and procedural barriers to reporting, ensuring quick action, legal awareness, and support for victims.

**PROJECT OBJECTIVE:**

PakCyberGuard aims to deliver a secure, easy-to-use platform that allows users to file PakCyberGuard aims to create a secure and intuitive platform for users to file cybercrime complaints without needing technical or legal expertise. Built with React and Tailwind CSS, the frontend offers a clean UI, while the backend uses PHP and MySQL to securely store user reports. A multi-step form captures detailed incident data, validated on both frontend (using Formik and Yup) and backend (via PHP). The AI chatbot, developed using FastAPI and integrated with models like LLaMA or Gemma, offers instant bilingual guidance. After form submission, users receive notifications and instructions on next steps, such as contacting legal professionals. An admin panel is also included to review reports and update awareness content regularly.

#### **INFORMATION GATHERING:**

#### The structure of components, functions, and form validations were researched thoroughly and designed according to best practices in web development (ReactJS, PHP, Tailwind CSS).

#### Several research papers, articles, and government guidelines were studied to understand cyber laws in Pakistan, chatbot integration using FastAPI, and form security with DOMPurify.

#### GUI/UX was refined by analysing user needs from platforms like FIA Cybercrime Wing, Digital Pakistan, and other global citizen safety systems.

#### <https://www.nr3c.gov.pk/>

https://www.fia.gov.pk/

1. Project Snapshots

**HOMEPAGE:**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

**LOGIN PAGE:**

A screenshot of a login form

AI-generated content may be incorrect.

**REGISTER PAGE:**

A screenshot of a computer screen

AI-generated content may be incorrect.

**REPORT CRIME FORM:**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a report

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a report

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**AWARENESS PAGE:**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a video game

AI-generated content may be incorrect.

**SECURITY TIPS:**

A screenshot of a computer

AI-generated content may be incorrect.

**WARNING SIGN:**

A screenshot of a computer

AI-generated content may be incorrect.

**ABOUT US PAGE:**  
A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**CONTACT US PAGE:**

A screenshot of a computer

AI-generated content may be incorrect.

**SELENIUM TESTING:**

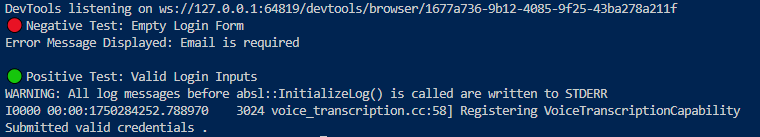
**Single Page Testing:**

**LOGIN PAGE:**

**A computer screen shot of a program

AI-generated content may be incorrect.**

**OUTPUT:**

****

**EXPLANATION:**

The test\_login.py script is a Selenium-based Python test designed to validate the login functionality of a web application.

It includes two test cases:

**Negative Test (Empty Login Form)**

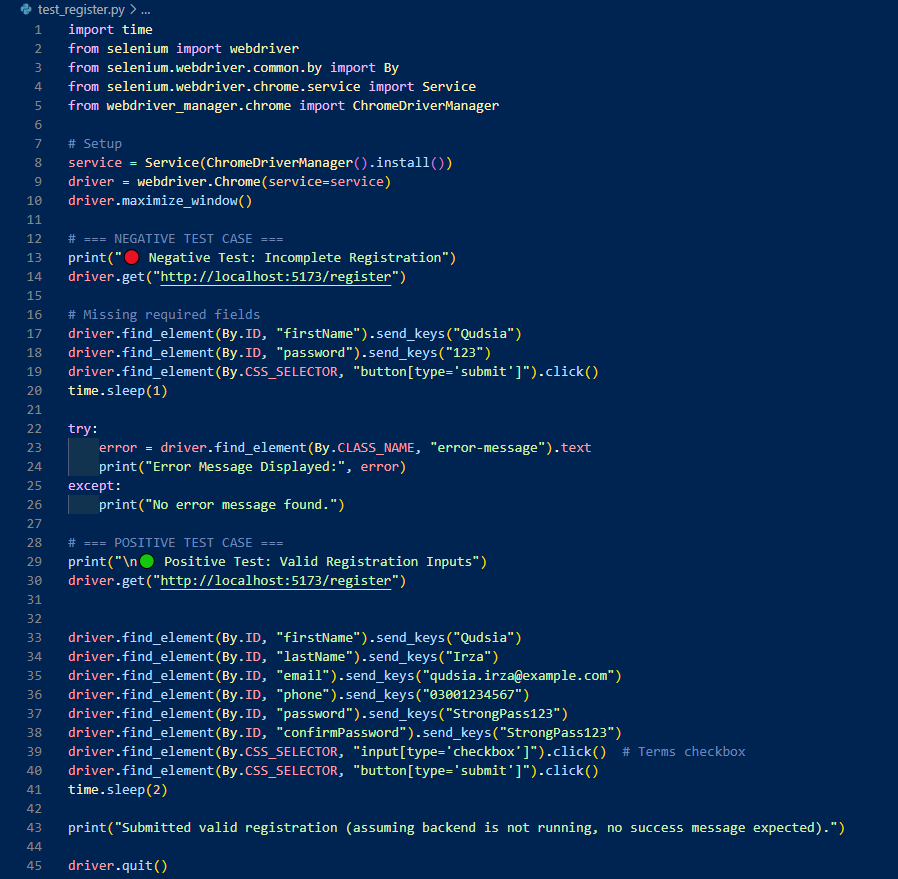
* Opens http://localhost:5173/login.
* Clicks the submit button **without entering credentials**.
* Waits 1 second for frontend validation.
* Tries to find an error message (class="error-message").
* Prints the error if found; else prints no message.

**Positive Test (Valid Login Form)**

* Visits the same login page.
* Enters valid **email and password** using send\_keys().
* Clicks submit and waits 2 seconds.
* Prints confirmation (no backend check due to local dev).

This script effectively checks basic frontend behavior, including validation and input handling.

**REGISTER PAGE:**

****

**OUTPUT:**

A blue screen with white text

AI-generated content may be incorrect.

**EXPLANATION:**

The test\_register.py script is a Selenium-based Python test designed to validate the registration functionality of a local web application (http://localhost:5173/register).

It includes two test cases:

**Negative Test (Incomplete Registration Form)**

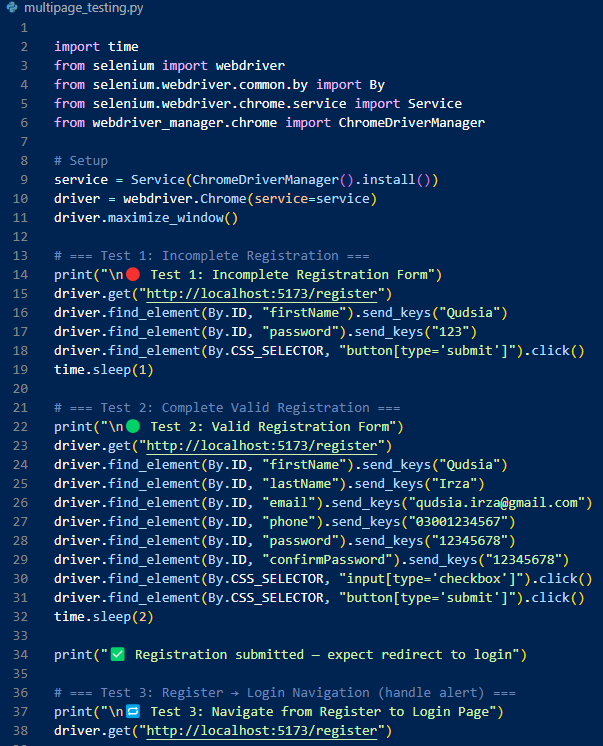
* Opens http://localhost:5173/register.
* Fills only the firstName and password fields.
* Leaves required fields like email and phone empty.
* Clicks the submit button.
* Waits 1 second for frontend validation.
* Searches for error message with class="error-message".
* Prints the error if found; else prints that no message was found.

**Positive Test (Complete Registration Form)**

* Opens http://localhost:5173/register.
* Fills in all required fields: firstName, lastName, email, phone, password, and confirm password.
* Clicks the checkbox (e.g., terms & conditions).
* Clicks the submit button.
* Waits 2 seconds for frontend processes to complete.
* Prints confirmation (no backend logic involved).

This script effectively tests the frontend behavior of the registration form, validating both error handling for incomplete input and proper handling of valid form submissions.

**MULTI -PAGES TEST:**

****

**A computer screen shot of a program

AI-generated content may be incorrect.**

**OUTPUT:  
  
A screenshot of a computer

AI-generated content may be incorrect.**

**EXPLANATION:**

The multipage\_testing.py script is a Selenium-based Python test designed to validate multipage flows (registration and login) of a local web application (http://localhost:5173). It includes six test cases covering both positive and negative scenarios.

**Test 1: Incomplete Registration Form**

* Opens http://localhost:5173/register.
* Fills only the firstName and password fields.
* Leaves required fields like email and phone empty.
* Clicks the submit button.
* Waits 1 second for frontend validation.

**Test 2: Complete Registration Form**

* Opens http://localhost:5173/register.
* Fills in all required fields: firstName, lastName, email, phone, password, and confirm password.
* Clicks the checkbox (e.g., terms & conditions).
* Clicks the submit button.
* Waits 2 seconds for frontend processes to complete.
* Prints confirmation (no backend logic involved).

**Test 3: Navigate from Register to Login Page**

* Opens http://localhost:5173/register.
* (Intended to handle navigation or alert — not implemented in detail.)

**Test 4: Empty Login Form**

* Opens http://localhost:5173/login.
* Clicks the submit button without entering credentials.
* Waits 1 second for frontend validation.

**Test 5: Valid Login Form**

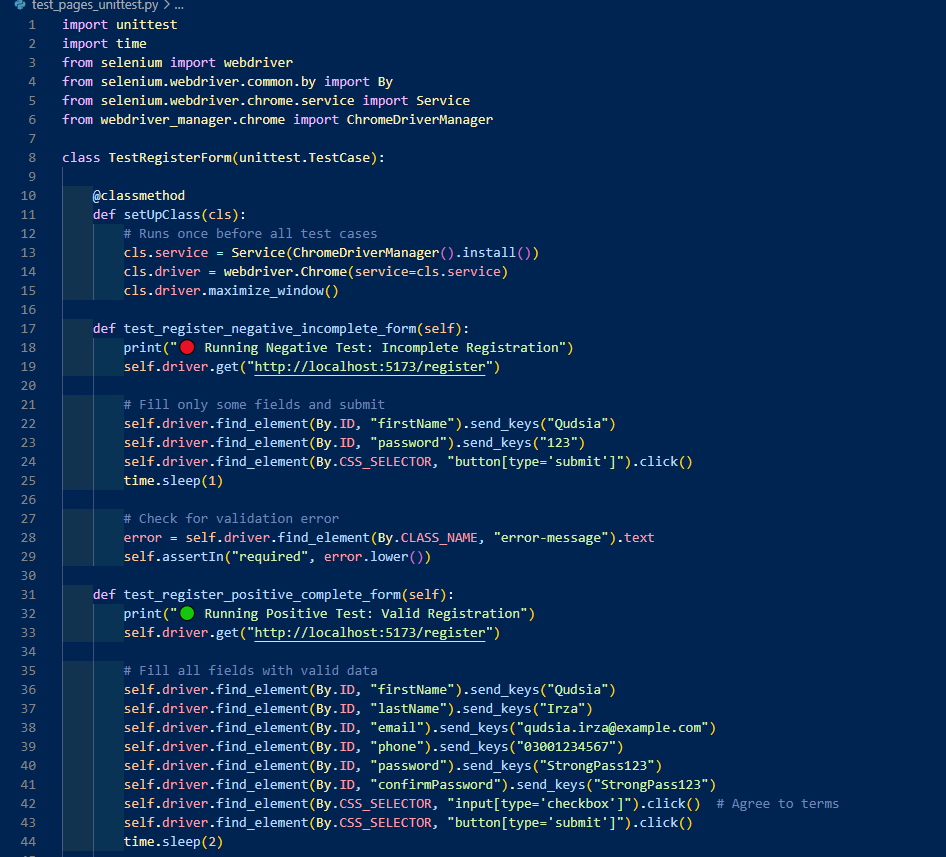
* Opens http://localhost:5173/login.
* Fills in valid email and password.
* Clicks the submit button.
* Waits 2 seconds for response.
* Checks if the URL contains /report.
* Prints success message if redirected, else prints login failure.

**Test 6: Navigate from Login to Register Page**

* Opens http://localhost:5173/login.
* (Intended to test navigation to registration — action not defined.)

The script ends with a message indicating that all tests are completed and closes the browser using driver.quit(). It ensures proper frontend validation, field handling, and basic navigation across registration and login pages.

**UNIT TEST:**



A blue screen with white text

AI-generated content may be incorrect.

**OUTPUT:**

A blue screen with white text

AI-generated content may be incorrect.

**EXPLANATION:**

This script is a Python Selenium Unit Test designed to validate the registration form of a web application using the unittest framework. It includes two test cases to simulate both negative and positive registration flows.

**Test Setup (Executed Once Before All Tests)**

* Imports unittest, selenium, and webdriver\_manager.
* Defines a setUpClass() method to initialize the Chrome WebDriver.
* Uses ChromeDriverManager to install the appropriate driver.
* Maximizes the browser window for clear test visibility.

**Test 1: Incomplete Registration Form**

* Opens http://localhost:5173/register.
* Fills only the firstName and password fields.
* Leaves required fields like email and phone empty.
* Clicks the submit button.
* Waits 1 second for validation messages to load.
* Locates the error element with class="error-message".
* Asserts that the error text contains the word "required" to confirm validation.

**Test 2: Complete Registration Form**

* Opens http://localhost:5173/register.
* Fills in all required fields: firstName, lastName, email, phone, password, and confirmPassword.
* Clicks the terms & conditions checkbox.
* Clicks the submit button.
* Waits 2 seconds for frontend processing.
* Checks that the current URL still contains "register" to confirm frontend handled the form properly (in the absence of backend).

**Test Teardown (Executed After All Tests)**

* Calls tearDownClass() to close the browser with driver.quit().
* Releases browser resources and finalizes test session.

This script is ideal for validating the frontend registration form's behavior under both invalid and valid input conditions, and it is structured for scalability using the unittest framework.

**JIRA TESTING:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**TASK:**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**BUGS:**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**EXPLANATION**

The registration form allows submission without filling the Confirm Password field. While it checks for mismatched passwords, it doesn't enforce that the field is required. This can lead to security issues. The validateForm function should be updated to ensure the Confirm Password field is mandatory and show an error like "Confirm password is required" when left blank.